

Evolving Fuzzy Rules for Goal-Scoring Behaviour in a Robot Soccer Environment

A thesis submitted in fulfilment of the requirements for
the degree of Doctor of Philosophy

Jeff Riley
M.App.Sc.

School of Computer Science and Information Technology
RMIT University
December 2005

If we knew what we were doing, it wouldn't be called research, would it?
- Albert Einstein (attrib.)

Abstract

The ability to construct autonomous robots that are able to learn from the environment in which they operate in order to achieve their objectives is a need so far largely unsatisfied, especially for dynamic environments which change quickly and are noisy and uncertain. A method of developing controllers for simple robots that learn, via artificial evolution, how to react in the noisy, uncertain and dynamic environment of simulated robot soccer in order to achieve goal-scoring behaviour is investigated by this thesis.

A rules-based architecture that uses a fuzzy-logic inferencing system is proposed for the simulated soccer player. The set of rules that controls the behaviour of the player is developed by evolving a population of simulated soccer-playing robots that are evaluated in the robot soccer environment. The evolutionary algorithm implemented to evolve the rules is a messy-coded genetic algorithm.

The soccer simulation environment chosen for this work is the RoboCup Soccer Simulation League, which is a dynamic, noisy, real-time environment specifically developed for artificial intelligence research. However, because the RoboCup simulator is a real-time environment all training and testing in the environment takes place in real-time, and this has a significant impact on the capacity of the method to do any real learning. The client-server architecture of the RoboCup simulator further complicates the implementation of the learning process. To overcome these impediments a less complex model of the RoboCup simulator was created.

The new simulator, named SimpleSoccer, is a multi-player capable, dynamic environment that is not noisy, does not operate in real-time, and does not implement a client-server architecture. The simplified environment of SimpleSoccer allows the evolutionary process to run much faster than in the RoboCup environment, so real learning can take place in more reasonable timeframes. Tests are performed to ensure that the SimpleSoccer environment is a

sufficiently good model of the RoboCup environment and that rules learned in the simpler environment are transferable to the RoboCup environment. A method of accelerating the evolutionary search in the RoboCup environment by seeding the population with rules learned in the SimpleSoccer environment is demonstrated.

This thesis also examines the question of how human expertise and expert knowledge affects the evolutionary search. Developing good soccer-playing skills for the robot soccer environment is known to be a difficult problem for evolutionary algorithms, and the problem is often solved by giving players some innate, hand-coded skills to increase the probability that the players will achieve the overall objective set. A well designed fitness function for the evolutionary algorithm can artificially guide the evolutionary process by rewarding incremental and intermediate solutions. Tests are conducted to determine how varying the amount of human help given to the evolutionary algorithm affects the result of the evolutionary process.

Finally, the thesis investigates the underlying cause of the difficulty of the robot soccer problem for evolutionary algorithms. A systematic study of the problem search spaces and fitness landscapes is presented which provides a good understanding of why the problem is difficult, and how injecting human expertise and expert knowledge in various ways can change the relative difficulty of the problem. The study also leads to the conjecture that there is an inherent limit to the amount of learning possible by evolutionary algorithms.

Declaration

I certify that except where due acknowledgement has been made, the work is that of the author alone; the work has not been submitted previously, in whole or in part, to qualify for any other academic award; the content of the thesis is the result of work which has been carried out since the official commencement date of the approved research programme; and, any editorial work, paid or unpaid, carried out by a third party is acknowledged.

Jeff Riley

December, 2005

Acknowledgements

Enormous thanks must go to my wife and children. This thesis could not exist without the incredible support and patience they have shown over the past seven years. I stand in awe of the strength, grace, patience, and peace my wife has exhibited through some very trying times.

Thanks also to my supervisor Vic Ciesielski, whose advice throughout, and particularly during the thesis writing phase, was invaluable (and remains so). James Harland, Xiaodong Li, and my wife Jennifer all provided valuable review comments – thank you all.

Contents

Abstract	iii
1 Introduction	1
1.1 Motivation.....	1
1.2 RoboCupSoccer	2
1.3 SimpleSoccer	3
1.4 Evolutionary Algorithms	4
1.5 Research Questions.....	5
1.5.1 Research Question 1	6
1.5.1.1 Research Question 1 Criteria for Success	7
1.5.2 Research Question 2	8
1.5.2.1 Research Question 2 Criteria for Success	9
1.5.3 Research Question 3	9
1.5.3.1 Research Question 3 Criteria for Success	10
1.6 Contribution.....	10
1.7 Thesis Structure	12
2 Literature Survey	14
2.1 Introduction.....	14
2.2 Genetic Algorithms.....	15
2.2.1 Biological Background.....	16
2.2.2 Representation	17
2.2.3 Initial Population	20
2.2.4 Selection & Reproduction	21
2.2.5 Recombination.....	25
2.2.5.1 Crossover for Fixed Length Chromosomes	26
2.2.5.2 Cut and Splice for Variable Length Chromosomes.....	27
2.2.5.3 Mutation	28
2.2.6 The Schema Theorem.....	28
2.2.7 Current Research in Evolutionary Computation	30
2.3 Fuzzy Logic and Fuzzy Inferencing	31
2.3.1 Linguistic Variables	31
2.3.2 Fuzzy Sets.....	33
2.3.2.1 Fuzzy Set Operations	34
2.3.3 Fuzzy Rules and Approximate Reasoning	34
2.3.4 Fuzzy Inferencing Systems.....	35
2.3.4.1 Fuzzification.....	37

2.3.4.2	Inference/Implication	37
2.3.4.3	Aggregation/Composition and Defuzzification.....	38
2.3.5	Mamdani vs. Sugeno	41
2.3.6	Genetic Fuzzy Systems.....	42
2.3.7	Research Trends in Fuzzy Systems	45
2.4	Fitness Landscapes	46
2.4.1	Landscape Measures.....	50
2.4.1.1	Autocorrelation and Correlation Length	51
2.4.1.2	Information Content	52
2.4.2	Isotropy.....	54
2.4.3	The State of Fitness Landscape Research	55
2.5	Evolutionary Robotics	55
2.6	Robot Soccer.....	56
2.6.1	RoboCup.....	58
2.6.1.1	The RoboCupSoccer Simulation League	58
2.6.2	The SimpleSoccer Environment.....	58
2.6.3	Evolutionary Algorithms and Simulated Robot Soccer	59
2.7	Work Using Evolutionary Algorithms to Solve Monolithic Problems.....	62
2.7.1	Evolving XPilots	63
2.7.2	Brainstormers 2D: Winners, RoboCup 2D-Simulation League 2005	63
2.7.3	GP Produced Competitive Soccer Teams for RoboCup97.....	65
2.7.4	Blondie24: Learning to Play Checkers without Relying on Expert Knowledge	67
2.8	Discussion.....	72
3	Robot Soccer Simulators.....	75
3.1	Introduction.....	75
3.2	The RoboCupSoccer Simulator	75
3.3	The SimpleSoccer Simulator	79
3.4	Player Actions.....	82
3.4.1	Player Default Action	82
4	Evolving Goal-Scoring Behaviour.....	85
4.1	Introduction.....	85
4.2	Chapter Goals	86
4.3	Player Architecture	86
4.3.1	Soccer Server Information.....	88
4.3.2	Fuzzification	89
4.3.3	Implication and Aggregation.....	89
4.3.4	Defuzzification	91
4.3.5	Player Actions	91
4.3.6	Action Selection	92
4.4	Player Learning.....	92
4.5	Representation of the Chromosome.....	93
4.6	Experimental Evaluation.....	96
4.6.1	Trials.....	97
4.6.2	Fitness Evaluation	97
4.6.2.1	RoboCupSoccer Fitness Function	97
4.6.2.2	SimpleSoccer Fitness Function	99
4.6.2.3	Fitness Values	100
4.6.3	Control Parameters	102
4.7	Results.....	103

4.7.1	RoboCupSoccer Initial Trial Results.....	103
4.7.2	SimpleSoccer Initial Trial Results.....	105
4.7.3	SimpleSoccer as a Model for RoboCupSoccer	107
4.7.4	GA Parameter Determination.....	108
4.7.4.1	Maximum Chromosome Length	108
4.7.4.2	Population Size.....	112
4.7.4.3	Selection Methods.....	113
4.7.4.4	Crossover Methods.....	115
4.7.4.5	Mutation Rate.....	116
4.7.4.6	Maximum Generations.....	118
4.7.4.7	Gene Pools.....	120
4.7.5	Performance Comparison – GA vs Random Search	121
4.7.6	Rulesets Evolved.....	123
4.7.6.1	RoboCupSoccer Ruleset.....	124
4.7.6.2	SimpleSoccer Ruleset.....	125
4.8	Summary and Discussion.....	126
5	Accelerating Evolutionary Search for RoboCupSoccer.....	128
5.1	Introduction.....	128
5.2	Chapter Goals	129
5.3	Experimental Overview	130
5.3.1	Control Parameters and Fitness Function.....	130
5.3.2	Seed Rulesets.....	131
5.4	Summary and Discussion.....	135
6	Varying Player Skills and the Fitness Function.....	137
6.1	Introduction.....	137
6.2	Chapter Goals	138
6.3	Available Player Skills.....	139
6.4	Observations from Previous Trials	140
6.5	Varying Player Skills.....	140
6.5.1	Default Action: <i>Hunt Action 1 – Goto Ball</i>	143
6.5.2	Default Action: <i>Hunt Action 2 – Locate Ball</i>	146
6.5.3	Default Action: <i>Hunt Action 3 – Random Turn</i>	149
6.5.4	No Default Action	152
6.5.4.1	Set 1: <i>No Hunt Consequent</i>	153
6.5.4.2	Set 2: <i>Hunt Consequent Allowed</i>	156
6.6	Varying the Fitness Function.....	159
6.7	Summary and Discussion.....	162
6.7.1	Varying Player Skills.....	163
6.7.2	Varying the Fitness Function.....	165
6.8	Conclusions.....	167
7	Analysis of the Search Spaces and Fitness Landscapes	168
7.1	Introduction.....	168
7.2	Chapter Goals	168
7.3	The Robot Soccer Search Space	169
7.4	Which Fitness Landscape?.....	173
7.5	Fitness Landscape Definition.....	176
7.6	Search Space and Fitness Landscape Analysis.....	177
7.6.1	Search Space and Fitness Landscape Representation	178
7.6.2	Fitness Landscapes Measures.....	181

7.6.3	Isotropy.....	181
7.6.4	Analysis.....	182
7.6.4.1	Varying Player Skills	183
7.6.4.2	Varying the Hunt Action.....	191
7.6.4.3	Varying the Fitness Function	197
7.7	Summary and Discussion.....	207
8	Conclusions and Future Work	210
8.1	General Conclusions	210
8.2	Research Question 1	211
8.3	Research Question 2	213
8.3.1	Research Question 3.....	214
8.4	An Evolutionary Quantum?	215
8.5	Future Work.....	216
8.5.1	Robot Soccer	217
8.5.1.1	Evolving Competent Players	217
8.5.1.2	Modelling the Environment	219
8.5.2	Landscape Analysis and Problem Difficulty	219
	Bibliography.....	221
	Appendix A	240
	Appendix B.....	241

List of Figures

Figure 1 The GA Evolutionary Cycle	15
Figure 2 Example of single point crossover.	26
Figure 3 Example of uniform crossover.	27
Figure 4 Example of cut and splice operations	27
Figure 5 Example of single-bit mutation.	28
Figure 6 An example of a linguistic variable	32
Figure 7 Membership function for the fuzzy set “old people”	33
Figure 8 Fuzzy Inferencing System.....	36
Figure 9 Input Variable Fuzzification	37
Figure 10 Correlation methods.....	38
Figure 11 2-rule Mamdani FIS	39
Figure 12 Methods of defuzzification	40
Figure 13 First-order Sugeno FIS	41
Figure 14 Diagrammatic representation of the field of gene combinations	46
Figure 15 An example of a fitness landscape	48
Figure 16 A simplified view of an evolutionary algorithm operating on three landscapes.....	49
Figure 17 Incremental fitness landscapes.....	73
Figure 18 The soccer field and landmarks represented by the RoboCupSoccer server	76
Figure 19 The visible range of a player in the RoboCupSoccer server	77
Figure 20 The soccer field and landmarks in the SimpleSoccer environment.	80
Figure 21 The visible range of a player in the SimpleSoccer environment.	81
Figure 22 Traditional Control Architecture.....	86
Figure 23 Soccer Player Layered Architecture	87
Figure 24 Player Architecture Detail.....	87
Figure 25 Distance, Power and Direction Fuzzy Sets	90
Figure 26 Mean of Maximum defuzzification method.....	91
Figure 27 Messy-coded Genetic Algorithm Example Chromosome Fragment	94
Figure 28 Chromosome and corresponding rules.....	95
Figure 29 RoboCupSoccer: Average Fitness - Initial 20 Trials	103
Figure 30 RoboCupSoccer: Best Fitness - Initial 20 Trials.....	104
Figure 31 RoboCupSoccer: Frequency of Individuals Scoring Goals	104
Figure 32 SimpleSoccer: Average Fitness – Initial 20 Trials.....	105
Figure 33 SimpleSoccer: Best Fitness - Initial 20 Trials.....	106
Figure 34 SimpleSoccer: Frequency of Individuals Scoring Goals	107
Figure 35 Average Fitness: Maximum Chromosome Length Variation	109

Figure 36 Best Fitness: Maximum Chromosome Length Variation.....	109
Figure 37 Average Chromosome Length: Maximum Chromosome Length Variation.....	110
Figure 38 Average Valid Rules per Chromosome: Maximum Chromosome Length Variation....	111
Figure 39 Average Fitness: Population Size Variation	112
Figure 40 Best Fitness: Population Size Variation.....	113
Figure 41 Average Fitness: Selection Method Variation	114
Figure 42 Best Fitness: Selection Method Variation.....	114
Figure 43 Average Fitness: Crossover Method Variation.....	115
Figure 44 Best Fitness: Crossover Method Variation	116
Figure 45 Average Fitness: Mutation Rate Variation.....	117
Figure 46 Best Fitness: Mutation Rate Variation	117
Figure 47 Average Fitness: 100 Generations	119
Figure 48 Best Fitness: 100 Generations.....	119
Figure 49 Population Composition Mean and Standard Deviation: 100 Generations	120
Figure 50 Gene Pools: 100 Generations.....	121
Figure 51 Average Fitness: Random Search Comparison.....	122
Figure 52 Best Fitness: Random Search Comparison	123
Figure 53 Average Fitness: Seeded Trials.....	132
Figure 54 Best Fitness: Seeded Trials	133
Figure 55 Seeded Trials: Frequency of Individuals Scoring Goals.....	134
Figure 56 Average Fitness: Hunt Action 1 – <i>Goto Ball</i> . Experiments 1 - 7.....	144
Figure 57 Average Fitness: Hunt Action 1 – <i>Goto Ball</i> . Experiments 8 – 14.....	144
Figure 58 Best Fitness: Hunt Action 1 – <i>Goto Ball</i> . Experiments 1 - 7	145
Figure 59 Best Fitness: Hunt Action 1 – <i>Goto Ball</i> . Experiments 8 – 14	146
Figure 60 Average Fitness: Hunt Action 2 – <i>Locate Ball</i> . Experiments 1 - 7.....	147
Figure 61 Average Fitness: Hunt Action 2 – <i>Locate Ball</i> . Experiments 8 - 14.....	147
Figure 62 Best Fitness: Hunt Action 2 – <i>Locate Ball</i> . Experiments 1 - 7	148
Figure 63 Best Fitness: Hunt Action 2 – <i>Locate Ball</i> . Experiments 8 – 14.....	149
Figure 64 Average Fitness: Hunt Action 3 – <i>Random Turn</i> . Experiments 1 - 7	150
Figure 65 Average Fitness: Hunt Action 3 – <i>Random Turn</i> . Experiments 8 - 14	151
Figure 66 Best Fitness: Hunt Action 3 – <i>Random Turn</i> . Experiments 1 - 7.....	151
Figure 67 Best Fitness: Hunt Action 3 – <i>Random Turn</i> . Experiments 8 – 14.....	152
Figure 68 Average Fitness: No Default Action – <i>No Hunt Consequent</i> . Experiments 1 - 7...	154
Figure 69 Average Fitness: No Default Action – <i>No Hunt Consequent</i> . Experiments 8 - 14.	154
Figure 70 Best Fitness: No Default Action – <i>No Hunt Consequent</i> . Experiments 1 - 7	155
Figure 71 Best Fitness: No Default Action – <i>No Hunt Consequent</i> . Experiments 8 - 14	155
Figure 72 Average Fitness: No Default Action – <i>Hunt Consequent Allowed</i> . Exp 1 - 7	156
Figure 73 Average Fitness: No Default Action – <i>Hunt Consequent Allowed</i> . Exp 8 - 14	157
Figure 74 Best Fitness: No Default Action – <i>Hunt Consequent Allowed</i> . Experiments 1 - 7 ..	158
Figure 75 Best Fitness: No Default Action – <i>Hunt Consequent Allowed</i> . Experiments 8 - 14	158
Figure 76 Average Fitness: Goals-only Fitness Function	160
Figure 77 Best Fitness: Goals-only Fitness Function.....	161
Figure 78 Goals-only Fitness: Frequency of Individuals Scoring Goals.....	161
Figure 79 Example Ruleset Array: p## is a premise and c## a consequent.....	169
Figure 80 Example fitness graph.....	180
Figure 81 Composite Fitness; All Skills & Directions; Default Action = Hunt Action 1	185
Figure 82 Composite Fitness; All Skills & Directions; Default Action = Hunt Action 1	186
Figure 83 Composite Fitness; Base Skills Only; Default Action = Hunt Action 1	188
Figure 84 Composite Fitness; Base Skills Only; Default Action = Hunt Action 1	189

Figure 85 Composite Fitness; All Skills & Directions; Default Action = Hunt Action 2	192
Figure 86 Composite Fitness; All Skills & Directions; Default Action = Hunt Action 2	193
Figure 87 Composite Fitness; All Skills & Directions; Default Action = Hunt Action 3	195
Figure 88 Composite Fitness; All Skills & Directions; Default Action = Hunt Action 3	196
Figure 89 Goals-only Fitness; All Skills & Directions; Default Action = Hunt Action 1	198
Figure 90 Goals-only Fitness; All Skills & Directions; Default Action = Hunt Action 1	199
Figure 91 Composite Fitness; All Skills & Directions; No Default Action; No Hunt	201
Figure 92 Composite Fitness; All Skills & Directions; No Default Action; No Hunt	202
Figure 93 Goals-only Fitness; All Skills & Directions; No Default Action; No Hunt	204
Figure 94 Goals-only Fitness; All Skills & Directions; No Default Action; No Hunt	205
Figure 95 Fitness Frequency	208
Figure 96 Composite Fitness; Base Skills Only; Default Action = Hunt Action 2	242
Figure 97 Composite Fitness; Base Skills Only; Default Action = Hunt Action 2	243
Figure 98 Composite Fitness; Base Skills Only; Default Action = Hunt Action 3	245
Figure 99 Composite Fitness; Base Skills Only; Default Action = Hunt Action 3	246
Figure 100 Composite Fitness; All Skills & Directions; No Default Action; Hunt Allowed ..	248
Figure 101 Composite Fitness; All Skills & Directions; No Default Action; Hunt Allowed ..	249
Figure 102 Composite Fitness; Base Skills Only; No Default Action; Hunt Allowed	251
Figure 103 Composite Fitness; Base Skills Only; No Default Action; Hunt Allowed	252
Figure 104 Composite Fitness; Base Skills Only; No Default Action; No Hunt	254
Figure 105 Composite Fitness; Base Skills Only; No Default Action; No Hunt	255
Figure 106 Goals-only Fitness; Base Skills Only; No Default Action; Hunt Allowed	257
Figure 107 Goals-only Fitness; Base Skills Only; No Default Action; No Hunt	259

List of Tables

Table 1 Example messy-coded chromosome attributes	19
Table 2 Available Player Actions.....	84
Table 3 Player Default Actions	84
Table 4 Fitness Assignment Summary	101
Table 5 Genetic Algorithm Control Parameters	102
Table 6 Empirically Determined Optimal GA Parameters.....	127
Table 7 Genetic Algorithm Control Parameters: Seeded Trials	130
Table 8 RoboCupSoccer Control Parameters: Seeded Trials.....	130
Table 9 Initial Population Composition: Seeded Trials.....	131
Table 10 Evaluation Times.....	135
Table 11 Genetic Algorithm Control Parameters: SkillSet Variation	141
Table 12 SimpleSoccer Control Parameters: Skillset Variation.....	141
Table 13 Player Skills.....	142
Table 14 Goals-only Fitness Experiments: Player Skills	159
Table 15 Goals-only Fitness: Number of Individuals Scoring Goals.....	162
Table 16 Composite Fitness Function; Varying Player Skills.....	164
Table 17 Simple Goals-only Fitness Function; Varying Player Skills.....	166
Table 18 Number of Consequents per Action	172
Table 19 Experiments Analysed.....	182
Table 20 Composite Fitness; All Skills & Directions; Default Action = Hunt Action 1	187
Table 21 Composite Fitness; Base Skills Only; Default Action = Hunt Action 1	190
Table 22 Composite Fitness; All Skills & Directions; Default Action = Hunt Action 2	194
Table 23 Composite Fitness; All Skills & Directions; Default Action = Hunt Action 3	197
Table 24 Goals-only Fitness; All Skills & Directions; Default Action = Hunt Action 1	200
Table 25 Composite Fitness; All Skills & Directions; No Default Action; No Hunt	203
Table 26 Goals-only Fitness; All Skills & Directions; No Default Action; No Hunt	206
Table 27 GP functions used in the soccer evaluation runs for [14].....	240
Table 28 Composite Fitness; Base Skills Only; Default Action = Hunt Action 2	244
Table 29 Composite Fitness; Base Skills Only; Default Action = Hunt Action 3	247
Table 30 Composite Fitness; All Skills & Directions; No Default Action; Hunt Allowed.....	250
Table 31 Composite Fitness; Base Skills Only; No Default Action; Hunt Allowed.....	253
Table 32 Composite Fitness; Base Skills Only; No Default Action; No Hunt Consequent.....	256

If you wish to make an apple pie from scratch, you must first create the universe.

- Carl Sagan, Cosmos, 1980.

Chapter 1

Introduction

1.1 Motivation

The need for autonomous robots that react in real-time when deployed in complex dynamic environments is becoming increasingly important. The United States Department of Energy (DOE) recently identified the potential for Robots and Intelligent Machines (RIMs) to replace human workers in hazardous environments and by doing so reduce cost and improve worker health and safety [257]. In its long-term vision, the DOE has predicted that RIM capabilities will be as pervasive and indispensable as the personal computer has become. The DOE RIM technology roadmap illustrates how many future challenges will be met effectively through the use of robotics and intelligent machines, and defines a research and development path for RIM continuing through to the year 2020 [63, 258].

In 1992 several researchers, in response to the need for a complex, dynamic and unpredictable, yet standardised and reproducible research environment, proposed the use of the game of soccer as a platform for artificial intelligence research and for promoting science and technology [154, 181]. A series of investigations was carried out to determine the feasibility of implementing a robotic version of soccer as a platform for research, taking into consideration technological, social and financial considerations. The investigations concluded that the proposition was feasible, and indeed desirable, and prototype development of soccer robots and simulator systems were undertaken. In 1993 the Robot J-League, a robot soccer competition,

was launched. The league was later extended as an international joint project and renamed the Robot World Cup Initiative, or RoboCup [151].

RoboCup has since become an international research and education initiative, providing a standard platform and benchmark problem for research in the fields of artificial intelligence and robotics. RoboCup currently consists of three major domains: RoboCupSoccer, RoboCupRescue and RoboCupJunior. The RoboCupSoccer domain includes a simulation league and is the environment used for the RoboCup part of this work.

Fuzzy logic was developed as a means of applying logic to imprecise and uncertain data [283]. Fuzzy logic-based controllers have been applied to a wide range of control problems, but are very difficult to build for embedded robots in situations where the environment changes quickly and there is a lot of uncertainty. One part of the work presented in this thesis is an investigation of a method of creating a rules-based fuzzy controller for a complex, unpredictable and dynamic environment. The method investigated uses a form of genetic algorithm [120] to evolve the rules for a fuzzy logic controller for a robot soccer player.

1.2 RoboCupSoccer

RoboCupSoccer is a competition in which teams of simulated soccer-playing robots compete against each other. It provides a realistic research environment by using a soccer game as a platform for a wide range of research problems including autonomous agent design, multi-agent collaboration, real-time reasoning, reactive behaviour and intelligent robot control [152, 153, 155]. The RoboCupSoccer Server that provides the simulation environment delivers limited and noisy sensor information to players, employs complex physics, real-time dynamics, and allows only limited communication between players [45, 198]. As a result the RoboCupSoccer environment is a very complex real-time, noisy, dynamic domain, and to succeed - indeed to even be competitive - many challenges must be overcome.

The list of skills required to be a successful soccer player is long, and many of the skills on the list are often not considered and go unheralded, as these skills are often innate in human players and so taken for granted. The ability to locate the ball, intercept it if it is moving, defend the goal, dribble the ball, and score goals are just some of the skills a competent soccer player has in his or her arsenal of skills, but all of those skills are built on more innate, less obvious skills and knowledge. For example, goal-scoring behaviour is predicated upon the knowledge that to score a goal the ball must be kicked, and kicked through the goal opening.

This may seem obvious, but when the task is to learn a complex task such as soccer goal-scoring behaviour from scratch, nothing can be considered obvious.

A wide variety of approaches and technologies have been used in attempts to construct good robot soccer players. These include hand-coding, genetic algorithms, genetic programming, reinforcement learning, neural networks, behaviour-based and deliberative agents, and various combinations of those. The combination of messy-coded genetic algorithms and fuzzy inferencing was chosen for this work because fuzzy inferencing suits the uncertain and imprecise environment, the genetic algorithm allows the fuzzy rulebase to adapt to the environment, and the messy-coding allows flexibility in the form and size of the rulebase. In the early years of the RoboCupSoccer competition, though a few intrepid researchers attempted to fine tune some low-level skills using AI machine learning techniques, nearly all entrants used hand-coded skills and strategies [175]. Even today hand-coded players, or players with hand-coded skills, generally continue to outplay players whose skills have been entirely learned or developed automatically [172]. For example, the 2005 RoboCupSoccer 2D simulation league winner used hand-coded strategies which employed a mixture of hand-coded skills and skills developed using machine learning techniques [219]. There has been only limited success when applying standard machine learning techniques to this problem – much on the work to date has been characterised by researchers beginning their work with high expectations, then ratcheting down their expectations as the work progresses, and finally adjusting their goals (and the soccer playing behaviours and skills of the players being developed) to align with the progress being made. A new strategy is required if more progress is to be made.

RoboCupSoccer is described in detail in Chapter 3 of this thesis.

1.3 SimpleSoccer

The RoboCupSoccer simulation league is an important and useful tool for multi-agent and machine learning research which provides a distributed, multi-player environment in which players have an incomplete and uncertain world view. The RoboCupSoccer state-space is extremely large, and the real-time player perception and action cycles in the RoboCupSoccer environment are asynchronous, sometimes resulting in long and unpredictable delays in the completion of actions in response to some stimuli. The large state-space, the inherent delays, and the uncertain and incomplete world view of the players can increase the learning cycle of some machine learning techniques to the point that it becomes prohibitive [175].

A new soccer simulation environment was designed and constructed by the author of this thesis specifically to address this issue. The SimpleSoccer environment [222] is proposed as an environment that reduces complexity and uncertainty sufficiently to increase the viability of machine learning techniques in the simulated robot soccer environment, yet retains sufficient complexity, uncertainty and dynamics to allow learning from SimpleSoccer to be directly transferable to the RoboCupSoccer environment.

SimpleSoccer is described in detail in Chapter 3 of this thesis.

1.4 Evolutionary Algorithms

Inspired by the apparent success of biological evolution over many millennia, engineers and scientists are increasingly turning to simplified computer simulations of natural evolutionary processes to try to solve complex optimisation and search problems – problems which are known to be computationally hard and almost intractable for conventional optimisation and search techniques. Over the past few decades a whole class of so-called evolutionary algorithms has emerged including, for example, evolutionary strategies, genetic algorithms and genetic programming.

Evolutionary algorithms (EAs) are useful optimisation and search techniques but, while they are apparently able to solve some hard problems, it is generally accepted that they are not able to effectively solve problems where the density of solutions in the search space is low and for which there is no way to evaluate the fitness of a potential solution other than as correct or incorrect. Such problems are known as *needle-in-a-haystack* problems [139, 162, 221], and since with these problems there is no gradient information in the fitness landscape, there is no way an evolutionary algorithm can converge on a solution. An evolutionary algorithm requires positive feedback, or a reward, to guide the search, but if the search space is such that the density of solutions – even partial solutions – is very low, then the algorithm has little chance of being rewarded, so no real learning can take place. Similarly, punishment, or negative feedback, for being on the wrong track is not useful for guiding the search in this type of search space. It follows then that for *needle-in-a-haystack* problems an evolutionary algorithm can perform no better than random search: the hay close to the needle is no more informative, or useful in guiding the search, than the hay far away.

The size of the robot soccer state space and the paucity of solutions that lead to good soccer-playing behaviour tend to suggest that the problem resembles a *needle-in-a-haystack* problem. Previous work in the area of evolutionary learning for robot soccer has focussed on the learning and what parameters and controls need to be manipulated in order to produce acceptable soccer-playing behaviour, but there has been no systematic investigation of the difficulty of the problem.

The overall goals of this thesis are to:

- examine whether, and how, the combination of a genetic algorithm and a rules-based fuzzy controller can be used to develop goal-scoring behaviour for a simulated soccer-playing robot in the complex, unpredictable and dynamic environment of robot soccer.
- present a method for systematic investigation of the difficulty of the problem of evolving goal-scoring behaviour in the robot soccer environment, and to undertake and present the results of such an investigation.

The specific research questions addressed by the thesis follow.

1.5 Research Questions

This thesis examines the following three research questions related to the problem of developing goal-scoring behaviour for simulated robot soccer players:

Research Question 1

- a) How can a genetic algorithm be used to evolve a set of fuzzy rules to govern the behaviour of a simulated robot soccer player that produces consistent goal-scoring behaviour?*
- b) Will seeding the initial population for the genetic algorithm with rules evolved in the SimpleSoccer environment improve the quality and speed of the evolutionary process in the more complex RoboCupSoccer environment?*

Research Question 2

How does varying

- a) the skills available to the players, and/or*
- b) the implementation of the fitness function*

affect the result of the evolutionary search?

Research Question 3

Why is evolving goal-scoring behaviour from scratch in a simulated robot soccer environment a difficult problem for evolutionary algorithms?

These research questions are explained in more detail in the following sections.

1.5.1 Research Question 1

The first research question is:

- a) How can a genetic algorithm be used to evolve a set of fuzzy rules to govern the behaviour of a simulated robot soccer player that produces consistent goal-scoring behaviour?*
- b) Will seeding the initial population for the genetic algorithm with rules evolved in the SimpleSoccer environment improve the quality and speed of the evolutionary process in the more complex RoboCupSoccer environment?*

Simulated robot soccer is a complex, dynamic and interesting environment for machine learning, and one which provides many opportunities for exploring very difficult optimisation and search problems. The first part of this question examines the ability of a genetic algorithm to evolve rules to govern the general behaviour of a simulated robot soccer player, and more specifically whether the rules evolved can consistently produce goal-scoring behaviour. Two simulated robot soccer environments providing different complexity were used to examine this question: the RoboCupSoccer environment, and the less complex SimpleSoccer environment. These environments are described in detail in Chapter 3 of this thesis. The RoboCupSoccer environment was initially chosen as the problem domain for the work presented in this thesis,

but because it is a very complex real-time simulation environment it was found to be prohibitively expensive with regard to the time taken for the fitness evaluations for the evolutionary search. To overcome this problem a simpler, non real-time environment known as SimpleSoccer was constructed so as to reduce the time taken for the trials.

The second part of this research question examines the problem of transferring the learning from the simple environment to the complex environment. The possible advantages of seeding the evolutionary process with previously learned individuals are addressed by this question. Because of the real-time nature of the RoboCupSoccer environment, the process of evolving goal-scoring behaviour is very slow. The SimpleSoccer environment on the other hand is not a real-time environment, and is generally much less complex than RoboCupSoccer, and as a result evolutionary search in the SimpleSoccer environment is much faster than in the RoboCupSoccer environment. The early stages of the evolutionary process are typically characterised by the algorithm investigating many poor-performing behaviours as it converges towards goal-scoring behaviours. Since rules which generate reasonable goal-scoring behaviour will have already been evolved in the SimpleSoccer environment, if the SimpleSoccer environment is a sufficiently good, and not too simple, model of the RoboCupSoccer environment, seeding the initial population of the evolutionary search in the RoboCupSoccer environment with the rules evolved in the SimpleSoccer environment could reasonably be expected to accelerate the evolutionary search.

1.5.1.1 Research Question 1 Criteria for Success

The measures of success for research question 1 are:

- The description and implementation of a method which employs a messy-coded genetic algorithm to evolve fuzzy rules that govern the behaviour of a simulated soccer-playing robot.
- The demonstration that the method implemented successfully evolves fuzzy rules which demonstrate consistent goal-scoring behaviour.

- An analysis of the effect on the efficacy of the method of varying the genetic algorithm parameters and, where applicable, a recommendation for the most effective values of those parameters.
- The demonstration of a method of increasing the quality and speed of the evolutionary search in the RoboCupSoccer environment by seeding the process with individuals previously evolved in the SimpleSoccer environment.

1.5.2 Research Question 2

The second research question is:

How does varying

- a) the skills available to the players, and/or*
- b) the implementation of the fitness function*

affect the result of the evolutionary search?

Observation and intuition suggest that improving the quality of the initial population for an evolutionary algorithm will improve the results of the optimisation or search. In the case of simulated robot soccer the quality of the population is a broad measure of the soccer-playing abilities of the members of the population. Specifically with regard to goal-scoring behaviour, and used as a broad definition throughout this thesis, the quality of the population is a measure of the ability of the population to produce goal-scoring behaviour – a high quality population is one in which the individual members possess or exhibit attributes likely to be conducive to producing goal-scoring behaviour. One method of improving the quality of a population is to endow the members of the population with a higher level of innate skills, and by conducting a range of experiments in which the skill level of the initial population is varied, question two examines in some detail the notion that improving the quality of the initial population (in this case by endowing the initial population with more, or better, soccer-playing skills) improves the result of the search.

Whereas a *simple fitness function* rewards a single behaviour – usually the desired outcome - it is a common practice when using evolutionary algorithms to implement a fitness function that rewards a number of incremental behaviours in order to achieve the desired

outcome – such a fitness function being known as a *composite fitness function*. Since the use of a composite fitness function for an evolutionary algorithm is akin to breaking the problem into smaller, more easily solved pieces, it is a contention of this thesis that the composite fitness function is essentially a recipe given to the evolutionary algorithm by humans for solving the problem. Question two also examines the effect that varying the fitness function has on the evolutionary process by conducting a number of experiments with different fitness functions.

1.5.2.1 Research Question 2 Criteria for Success

The measures of success for research question 2 are:

- The presentation of the results of experiments which demonstrate clearly how varying the skills available to members of the initial population changes the outcome of the evolutionary search.
- A similar presentation of the results of experiments which demonstrate clearly the difference in the result of the evolutionary search for a simple fitness function versus a composite fitness function.
- An analysis of the results presented which gives an understanding of how and why changing the initial population and the fitness function affects the result of the evolutionary search.

1.5.3 Research Question 3

The third research question addressed is:

Why is evolving goal-scoring behaviour from scratch in a simulated robot soccer environment a difficult problem for evolutionary algorithms?

Question three addresses the underlying cause of the difficulty of the problem of evolving goal-scoring behaviour from scratch for simulated robot soccer. Evolving goal-scoring behaviour from scratch is considered to be a very difficult problem and is often achieved by endowing

simulated players with an innate set of hand-coded skills, or by decomposing the problem into evolving a set of simpler behaviours which are then aggregated into goal-scoring behaviour.

The work presented in this thesis examines features of the problem search space and fitness landscape to explain the reason for the difficulty of the problem, specifically for the genetic algorithm used, but also for evolutionary algorithms in general.

1.5.3.1 Research Question 3 Criteria for Success

The measures of success for research question 3 are:

- Descriptions of the fitness landscapes defined by the specific combination of the experiments performed and the search algorithm implemented for this work. This will include definitions of the problem search spaces and the functions and relationships used to arrange the points in a search space into a landscape.
- An analysis of the problem search spaces and fitness landscapes that gives an understanding of why evolving goal-scoring behaviour from scratch is a difficult problem for evolutionary algorithms, and how the level of difficulty is manipulated by changing the skillset of the initial population and the implementation of the fitness function.

1.6 Contribution

This thesis makes the following contributions to the fields of evolutionary algorithms and robot soccer:

- 1) A method for evolving reactive robot behaviours in a complex and dynamic environment. The method is shown to be capable of evolving good goal-scoring behaviour with a small population and in relatively few generations when either the quality of the initial population or the fitness function implemented brings the problem within reach of the evolutionary algorithm. The method also produces human-readable rules that govern the behaviour of the players which is a further advantage over many existing methods of learning behaviours where often the learned behaviour is not apparent and not easily extracted.

- 2) The proposition that in order to evolve competent robot soccer players either a rich set of mid- to high-level soccer playing skills needs to be provided, or a composite fitness function used that effectively guides the evolutionary process to developing the skills required to be a competent soccer player. This proposition is supported by a systematic investigation of the effect on the performance of the evolutionary search of varying the amount and nature of human expertise and expert knowledge injected into the algorithm by way of hand-coded innate skills, “smart” default actions and composite fitness function. The results show that the performance of the genetic algorithm implemented is significantly affected by varying the innate skills of the soccer players being evolved and the fitness function with which the players’ performance is measured.
- 3) A method for systematic analysis of problem search spaces and fitness landscapes generated by evolutionary algorithms. The results of an investigation within the framework defined of the search spaces and fitness landscapes for a series of difficult problems in the robot soccer domain are presented, the contribution of which is a better understanding of why the problem is a difficult problem for evolutionary algorithms. The results of the investigation and analyses suggest that evolutionary algorithms may not be well suited to a class of large problems for which the fitness function or reward does not directly map to a solution (called “monolithic” problems throughout this thesis), and the conjecture that an “evolutionary quantum” exists that effectively limits the amount of evolutionary learning possible for such problems.
- 4) The creation of the SimpleSoccer robot soccer simulator which reduces the complexity and uncertainty of the RoboCupSoccer environment sufficiently to increase the viability of machine learning techniques in the simulated robot soccer environment. The SimpleSoccer environment retains sufficient complexity, uncertainty and dynamics to allow learning from SimpleSoccer to be transferable to the RoboCupSoccer environment, either by applying the rules learned in the SimpleSoccer environment directly or by using those rules to seed the evolutionary search in the RoboCupSoccer environment. The SimpleSoccer simulation environment has been, and continues to be, used as research environment by other researchers in the fields of evolutionary computation and robot soccer.

A selection of the work in this thesis has been presented in the following publications:

Book Chapter:

Jeff Riley and Vic Ciesielski. Evolution of fuzzy rule based controllers for dynamic environments. In Kay Chen Tan, Meng Hiot Lim, Xin Yao, and Lipo Wang, editors, *Recent Advances in Simulated Evolution and Learning*, volume 2 of *Advances in Natural Computation*, chapter 23, pages 426-445. World Scientific, 2004.

This book chapter is an expanded and updated version of the SEAL conference paper listed below.

Conference Papers:

Jeff Riley. The SimpleSoccer Machine Learning Environment. In Sung-Bae Cho, Hoai Xuan Nguen, Yin Shan, editors, *Proceedings of the First Asia-Pacific Workshop on Genetic Programming (ASPGP03)*, pages 24-30, Canberra, Australia, December 2003.

Jeff Riley and Vic Ciesielski. Evolving fuzzy rules for reactive agents in dynamic environments. In Lipo Wang, Kay Chen Tan, Takeshi Furuhashi, Jong-Hwan Kim, and Xin Yao, editors, *Proceedings of the 4th Asia-Pacific Conference on Simulated Evolution And Learning (SEAL02)*, pages 124-130, Singapore, November 2002.

1.7 Thesis Structure

Chapter 2 of the thesis follows this introduction, and there an overview of evolutionary and genetic algorithms, search spaces and fitness landscapes, fuzzy logic, fuzzy rule-based inferencing, and the simulated robot soccer environments of RoboCupSoccer and SimpleSoccer is presented. Also presented in Chapter 2 is a literature survey of previous work in those domains relevant to this thesis.

Detailed descriptions of the RoboCupSoccer and SimpleSoccer environments, including the commands available to players, are provided in Chapter 3. An overview of the player architecture is presented, and the list of hand-coded player actions and default actions is also given in Chapter 3.

The work performed to examine the first research question is presented in Chapter 4 and Chapter 5. In Chapter 4 the player architecture is further defined and a representation suitable for a messy-coded genetic algorithm is presented. The results of several experiments to evolve goal-scoring behaviour in both the RoboCupSoccer environment and the SimpleSoccer environment are presented. The reason for the development of the SimpleSoccer environment is also outlined in Chapter 4.

Chapter 5 describes the results of a number of experiments undertaken to examine the question of whether the knowledge learned in the SimpleSoccer environment can be transferred to the RoboCupSoccer environment by seeding the evolutionary process, and the effectiveness of various numbers of seeded individuals in the initial population. Chapter 5 addresses the second part of research question one.

The results of a series of experiments in the SimpleSoccer environment for which the skills available to the players being evolved was varied are presented in Chapter 6. Also presented in Chapter 6 are the results of some experiments in which different fitness functions were implemented. These experiments were performed to examine the second question regarding initial population quality and the fitness function implementation.

Chapter 7 presents definitions and descriptions of the problem search spaces and fitness landscapes for the experiments conducted in Chapter 6 and analyses of those search spaces and fitness landscapes. The analyses were undertaken to address the third research question and to provide explanations for the underlying cause of the difficulty for evolutionary algorithms in evolving goal-scoring behaviour in a simulated robot soccer environment.

Chapter 8 summarises findings of the work and presents the overall conclusions of the thesis. Suggestions for further work are also outlined in Chapter 8.

More gold has been mined from the thoughts of men than has ever been taken from the earth.

- Napoleon Hill (attrib.)

Chapter 2

Literature Survey

2.1 Introduction

The work presented in this thesis is built on three major areas of research: evolutionary algorithms, particularly genetic algorithms; fuzzy logic; and simulated robot soccer. In addition, the research area of problem search spaces and fitness landscapes is drawn upon in order to analyse some of the results produced by this work. This chapter presents some relevant research for each of those areas and details some recent theoretical and experimental work relevant to the contents of this thesis. The remainder of this chapter is ordered into the following five major sections:

Section 2.2 covers genetic algorithms and the parent field of evolutionary computation. An overview of evolutionary computation is presented, and genetic algorithms are described in detail.

Section 2.3 presents an overview of fuzzy logic, fuzzy sets and fuzzy inferencing systems. The two most commonly implemented fuzzy inferencing models are described in this section, as are genetic fuzzy systems.

Section 2.4 presents an overview of fitness landscapes.

Section 2.5 gives a brief overview of evolutionary robotics.

Section 2.6 covers robot soccer, with an introduction and overview of the two simulation environments used in this work.

2.2 Genetic Algorithms

Genetic Algorithms [120] belong to a class of Evolutionary Algorithms inspired by the processes observed in natural evolution. Other examples of evolutionary algorithms are Evolutionary Programming [81], Evolution Strategies [215, 229], and Genetic Programming [158]. Evolutionary algorithms share a common conceptual base of simulating the evolution of a population of individual structures by the use of one or more of the natural evolutionary processes of selection, reproduction and mutation.

A genetic algorithm is an adaptive search technique which maintains a population of potential solutions that evolves over time in accordance with the rules of the genetic operators implemented by the algorithm. Each member of the population has its fitness as a solution to the problem evaluated against some known criteria, and members of the population are then selected for reproduction based upon that fitness, with a new generation of potential solutions being generated from the offspring of (typically) the most fit individuals. The process of evaluation, selection, reproduction, recombination and mutation is iterated until an acceptable solution is found (Figure 1).

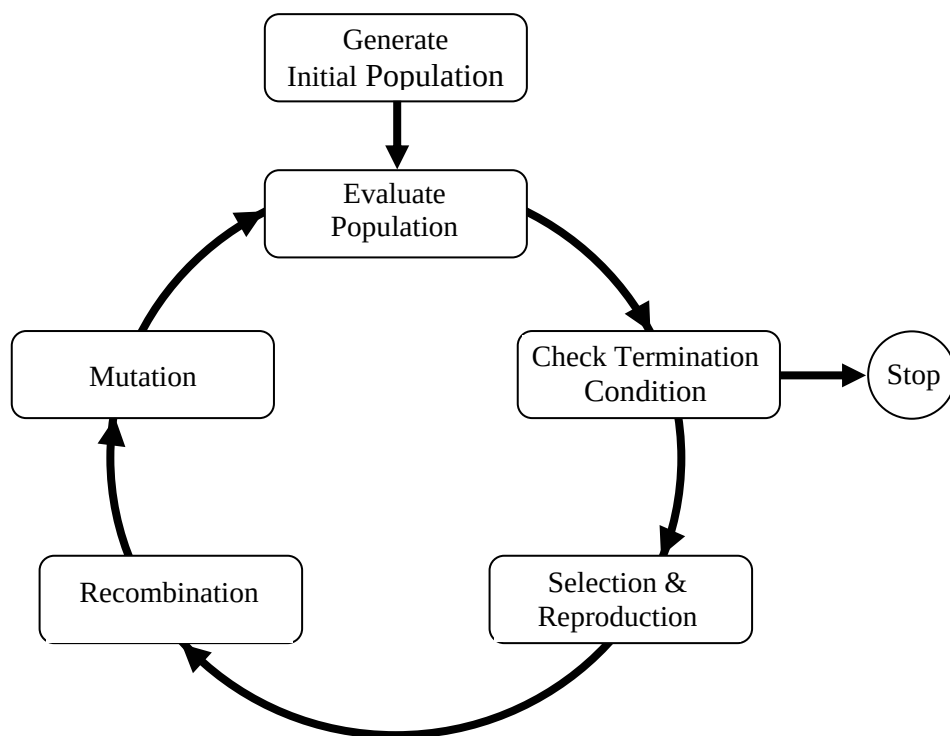


Figure 1 The GA Evolutionary Cycle

The evaluation of the worth of an individual as a solution is achieved by the use of a *fitness function*. The objective of the fitness function is to numerically encode the performance of the individual with reference to the problem for which it is a potential solution. This is an extremely important part of the process, for without a fitness function which accurately evaluates the performance of potential solutions, the search will fail.

In a process known as *exploitation*, the selection and reproduction operators focus on highly fit individuals and enable the genetic algorithm to conduct a local search of promising regions of the search space. Wider *exploration* of untested regions of the search space is achieved by the perturbation of individuals via the recombination and mutation operators.

2.2.1 Biological Background

As mentioned, evolutionary algorithms share a common conceptual base of simulating the evolution of a population of individual structures by the use of one or more of the natural evolutionary processes of selection, reproduction and mutation. It is useful to have an appreciation of the natural evolutionary processes in order to better understand evolutionary algorithms. Following is a brief, somewhat simplified, description of three broad natural evolutionary processes most relevant to genetic algorithms: selection, reproduction and mutation.

All living organisms contain genetic material which defines their makeup. An organism's genetic material is made up of strands of deoxyribonucleic acid, or DNA, and is packaged into one or more chromosomes. A piece of chromosome which defines a particular attribute or trait is a gene, and the various forms of a gene are the gene's alleles. The position of a gene on a chromosome is known as the locus of the gene. Chromosomes are contained within the cells of the organism: the number of copies of chromosomes within the cell determines the cell's ploidy. Haploid cells contain one set of chromosomes, diploid cells two sets, and polyploid cells many sets. Organisms may contain cells of varying ploidy. For example, most human cells contain one set of chromosomes from each parent, and so are diploid, but human sex cells contain a single set of chromosomes, and so are haploid. Genetic algorithms are most commonly implemented as haploid.

In the context of biological evolution, selection is the process by which certain traits or alleles of a species are favoured or disfavoured. Under selection, individuals with advantageous traits tend to be more successful in a reproductive sense than other individuals,

hence they contribute more offspring to the succeeding generation. Since offspring inherit traits from their parents, selection therefore increases the prevalence of the genetic material of those successful individuals. It is important to note here that all of the genetic material of the more successful individuals is increased within the population, not just the advantageous traits. As selection becomes more intense and persistent, traits propagated from successful individuals may become universal to the population or species, and the species is then said to have evolved.

Reproduction is the biological process by which new individual organisms are created, and is achieved through one of two methods: sexual or asexual. Organisms create descendants through sexual reproduction, and this generally requires the involvement of two individuals of a species - typically one of each gender. In some species, individual organisms reproduce without the involvement of other individuals - this is asexual reproduction. Genetic algorithms are most commonly implemented using a mechanism based on sexual reproduction, though there is generally no notion of gender. Sexual reproduction involves the combination of genetic material taken from two, usually different, members of the species to produce one or more new individuals. Offspring so created usually inherit one allele for each trait from each parent, so the genes of the offspring are a recombination of genes from its parents.

Mutations are changes to the genetic material of a cell, and are usually considered random or undirected. There are many causes of mutations ranging from copying errors during cell division to viral exposure. Mutations are always permanent, and sometimes transmissible to progeny. The process of mutation is often considered the driving force of evolution, providing variation in the gene pool. Some mutations will be beneficial, others deleterious but, in nature at least, the overwhelming majority have no known significant effect.

2.2.2 Representation

In Holland's original implementation of the genetic algorithm the candidate solutions were represented as fixed-length, fixed-order bit strings. The representation of the problem is an extremely important factor in the success of any problem solving technique, and genetic algorithms are no exception. Since Holland's original implementation there have been experiments conducted with different representations – some more successful than others, but all indicating that representation should be viewed as problem specific and a representation chosen appropriate to the problem at hand.

The most common representation for genetic algorithms has been Holland’s original binary encoding, and much of the theoretical work with genetic algorithms has as an underlying assumption the fixed-length, fixed-order binary encoding. Examples of variations to the binary encoding scheme are described in [23, 37] (gray coding schemes) and [114] (a diploid implementation).

The use of bit strings to encode problem parameters is somewhat limiting, so various less restrictive representation schemes have been investigated. Real-valued encoding schemes which allow a greater representation flexibility are described or implemented in [74, 104, 111, 136, 188, 193, 203, 224]. A similar encoding scheme is the many-valued encoding scheme described in [150]. Koza’s encoding scheme [158] for genetic programming is an example of a many-valued representation scheme in the form of a tree structure which overcomes the restrictions imposed by a simple binary representation. Several adaptive representations have been suggested: messy genetic algorithms [89, 90], delta coding [275], dynamic parameter encoding (DPE) [227], and ARGOT [231].

The encoding scheme introduced by Goldberg et al. for their so-called messy genetic algorithms is particularly flexible [89]. In the messy representation the chromosome may vary in length, and genes are not fixed at a particular location but are tagged with their “real” locus – thus each gene is essentially assigned a semantic meaning as well as a value (allele). The messy encoding allows *underspecification* and *overspecification* of the chromosome. An *underspecified* chromosome is one for which not all loci have been specified; an *overspecified* chromosome is one which includes loci that have multiple specifications - each possibly specifying a different allele. For example, the following pairs of numbers represent genes in a messy-coded chromosome:

$$\{(2,3),(4,1),(6,1),(3,4),(6,2)\}$$

where the first number of each gene specifies the meaning of the gene (its locus) and the second number is the value of the gene (allele).

The chromosome shown above could specify, for example, various attributes of a motor vehicle if the mapping in Table 1 is applied. Notice that the order of the genes is not significant, and that the chromosome is underspecified in terms of genes with locus 1 and 5, and overspecified in terms of genes with locus 6.

Locus	Meaning	Allele	Meaning
1	Body Style	1	Sedan
		2	Wagon
		3	Hatchback
		4	Convertible
2	Engine Size	1	4 cylinders
		2	6 cylinders
		3	8 cylinders
		4	12 cylinders
3	Exterior Colour	1	Red
		2	Blue
		3	Green
		4	White
4	Paint Type	1	Enamel
		2	Metallic
		3	Colour-Shift
5	Seat Covering	1	Vinyl
		2	Cloth
		3	Leather
6	Entertainment System	1	Radio/Tape
		2	CD
		3	DVD

Table 1 Example messy-coded chromosome attributes

Using the same mapping, the following chromosomes specify two different vehicles:

$$\{(1,1), (4,2), (2,1), (5,1)\}$$

$$\{(2,3), (1,2), (6,1), (6,3), (6,2), (3,3), (5,2), (4,1)\}$$

The operation of a messy genetic algorithm is divided into two distinct phases: a primordial phase which enriches the population with small, promising candidate schemata¹, and a juxtapositional phase which attempts to combine the candidate schemas in useful ways. The genetic algorithm used in this work implements a messy-coding scheme, but does not use the primordial and juxtapositional phases of the standard messy genetic algorithm. The implementation is described in section 4.5 on page 93.

¹ see section 2.2.6 on page 28 for a definition of *schemata*

2.2.3 Initial Population

The initial population for a genetic algorithm is usually generated randomly. However, the performance of a genetic algorithm is often sensitive to the quality of its initial population. Two important factors can affect the performance of the genetic algorithm: the diversity of the population and its average fitness with respect to the objective of the search or optimization task. A lack of diversity in the population can cause early convergence to a locally optimal solution. The quality of the population, measured by the average fitness, can be an important determinant of the likely success of the genetic algorithm - empirical evidence suggests that “better” starting solutions, rather than randomly generated or “poor” starting solutions, usually yield better final solutions, while decreasing the running time of the algorithm [4, 21]. Also, as noted in [73], a randomly generated population may include members which are infeasible as solutions and as a result have poor fitness values, thus yielding poor performance for fitness functions which penalise infeasible solutions. For example, for the fuzzy rules-based representation used in this work an infeasible solution is one for which no valid rule can be extracted from the chromosome, and such infeasible solutions are penalised by being assigned the worst possible fitness value.

Motivated by [218], Hill suggests a problem dependent heuristic be used to generate the initial population for genetic algorithms and concludes that compared to random generation methods, the heuristic performs especially well [113]. Furthermore, Hill notes the expectation that the near-feasible solutions produced by the heuristic will be especially attractive to penalty-based fitness functions often implemented for of genetic algorithms. A similar method is used to generate the initial population of neural networks in [126] where it is noted that the heuristic improves both the quality of the result and the computational efficiency of the algorithm.

Similar results have been reported by other researchers – the empirical evidence suggests that a problem dependent heuristic for initial population generation is advantageous for evolutionary algorithms [84, 109, 171, 261, 271, 281]. The advantage reported is an improvement in both the quality of the final solution and computational efficiency – so better

solutions are found more quickly by the evolutionary search. Brusey reports similar results for seeding a reinforcement learning algorithm [35].

The majority of the experiments reported in this thesis are performed with a randomly generated population. The results of a number of tests for which the initial population was seeded with previously evolved solutions are presented in Chapter 5.

2.2.4 Selection & Reproduction

The principle which underpins natural evolutionary theory [59] is that which refers to the process of *natural selection*, sometimes described as *survival of the fittest* [60, 239]. Often misunderstood and misused, the phrase “survival of the fittest” is essentially a tautology. There is no evidence to support the notion that natural evolution is a directed process: natural evolution has not been shown to be a process for which the objective is known or for which a fitness function is defined. First, the term survivors is something of a misnomer in itself, since individuals can only survive a single generation – they do not survive the evolutionary process. In reality, survivors by definition live longer than non-survivors, and as a result of their relative longevity are in a position to mate more often and so produce more offspring. What tends to survive the evolutionary process is some of the genetic makeup of the individuals producing the greatest number of offspring, so in any objective analysis of a natural evolutionary process the survival of the fittest reduces to the survival of the most prolific progenitors – essentially the *survival of the survivors*.

A better interpretation of the phrase “*survival of the fittest*” is that it conveys the expectation that variations of attributes in individuals which are most appropriate or most suitable for the environment in which they are situated will survive the evolutionary process. This is supported by the following excerpt from Darwin’s writings:

"I have called this principle, by which each slight variation, if useful, is preserved, by the term natural selection, in order to mark its relation to man's power of selection. But the expression often used by Mr. Herbert Spencer, of the Survival of the Fittest, is more accurate, and is sometimes equally convenient." [61], p.99.

Darwin further asserted that variations which are injurious would be destroyed by the process of natural selection, and variations which were neither useful nor injurious would not be affected.

Selection and reproduction are important processes for evolutionary algorithms. Individuals from the population are selected according to some criteria to be reproduced for the next generation. An important distinction here is the difference between the biological reproduction process described in section 2.2.1 on page 16, and genetic algorithm reproduction. GA reproduction is essentially a cloning operation in which the individuals selected for reproduction are copied, and it is during the recombination process that the copies are mated to form new individuals, whereas the biological reproduction process combines the GA processes of reproduction and recombination. For genetic algorithms, selection and reproduction alone cannot introduce new individuals into the population: the genetically-inspired recombination operators of crossover and mutation perform that task (section 2.2.5, page 25). The purpose of selection and reproduction is to favour fitter individuals on the basis that the fitter an individual the more likely it will produce even more fit offspring. It is important however that a balance between the genetic operators be achieved so that the evolutionary process is kept moving at a reasonable rate and in the direction indicated by the fitter individuals, while the diversity of the population is not compromised by selecting too strongly for fitness.

In the case of genetic algorithms, the selection of individuals which will have some part of their genetic material propagated through to the next generation of potential solutions is achieved by using one of several different selection methods. Some of the selection methods most commonly implemented are:

- A ***Fitness-Proportionate*** method of selection [15, 86, 120] was implemented by Holland for his original GA, in which the number of times an individual could be expected to be selected to reproduce was the ratio of the individual's fitness to the average fitness of the population. Probably the most common method for the implementation of this selection method can be likened to a biased roulette wheel, where each individual in the current population has a slot on the roulette wheel proportional to that individual's fitness. The roulette wheel is spun once for each parent required, with the winning individuals being paired for reproduction. The diversity of the population is to some extent retained because individuals with a low fitness still have a chance, albeit small, of being selected for reproduction, the diversity of the population is to some extent retained. Because of

the relatively small population sizes used in typical GA implementations, the actual rate of selection of an individual is often very different from the expected rate of selection for that individual. *Stochastic Universal Sampling* [15] is a variant technique suggested by Baker in an attempt to minimize this difference between the expected and actual rate of selection of individuals. A significant problem with fitness-proportionate selection is that the rate of evolution is dependent upon the variation in fitness of the population, and as a result the selection method can encourage the premature convergence of the population towards the more highly fit individuals which may emerge early in the evolutionary process.

- **Fitness Windowing** [98] and **Sigma Scaling** [82, 86] are examples of a number of modified fitness-proportionate methods developed to overcome the problem of premature convergence. These methods attempt to normalise, or scale, the expected number of offspring for an individual so as to de-emphasise highly fit individuals early in the evolutionary process, and emphasise highly fit individuals later in the evolutionary process when fitness variation across the population is low. The sigma scaling method calculates the expected number of offspring for an individual as a function of the individual's fitness, the mean fitness of the population, and the standard deviation of the fitness values of the population.
- The **Boltzmann Selection** method [69, 70, 72, 87] is a further adaptation of the fitness-proportionate method. Whereas the sigma scaling method typically maintains the rate at which individuals of a particular fitness produce offspring (the selection pressure), Boltzmann selection implements a method by which the selection pressure varies from low to high with time during the evolutionary process.
- **Rank Selection** [14, 62, 277] is a commonly employed method to overcome focus on extreme individuals. For rank selection, the population is ranked according to the raw fitness values of the individuals, and the expected number of offspring for an individual is determined by its rank rather than its fitness value. This has a similar effect to the fitness scaling methods, but rank selection further ensures a regular spread of individuals and as a result a small number of individuals with extreme fitness values will have little effect on the selection process.

- **Tournament Selection** [32, 88] is another commonly used selection method used to overcome focus on extreme individuals, and avoids the sensitivity to fitness scaling suffered by the scaling methods. In this method a number of chromosomes, commonly two, are chosen at random from the population and compared in a probabilistic “tournament”, with the better chromosome being more likely to win. The chromosomes chosen are returned to the population, and the process of sampling and comparing is continued until the required number of parents has been chosen.
- **Steady-State Selection** [62, 68, 253, 276, 277] is a selection method used to replace a small number of individuals from the population rather than, as is more common, the entire population. In the “generational” model of genetic algorithms, the percentage of the population which is replaced in each generation, known as the *population gap* [66], is 100 - the entire population [99]. In steady-state selection and replacement only a few, typically two, individuals are selected for replacement in each generation. The individuals selected for replacement may be selected in a variety of ways: e.g. fitness (selecting the *least fit* individuals for replacement), randomly, etc.; as may be their replacements: e.g. inverse fitness (selecting the *most fit* individuals as replacements), randomly etc.
- **Elitism** [66, 188, 192] is often used as an adjunct to other selection methods. The objective of elitism is to ensure the genetic material of the fittest individuals is passed on to the next generation, so the GA is compelled to retain some of the most fit individuals at each generation.

Some techniques and measures have been developed to characterise and classify different selection techniques. Goldberg and Deb [88] introduced the notion of *takeover time* as a measure of selective pressure. *Takeover time* is the number of generations required for the single best individual to permeate the entire population if selection is the only genetic operator used (i.e. no recombination), with the assumption being that selective pressure is inversely proportional to takeover time. Bäck [11] uses the takeover time measure to analyse and compare several selection techniques. Hancock [105] extended Goldberg and Deb’s work by evaluating a broader range of selection strategies and measuring some stochastic effects not considered by Goldberg and Deb, and by considering the rate of elimination of weaker strings

as well as the takeover time. Blickle and Thiele [24] conducted a comprehensive analysis of a number of selection schemes using several techniques developed to measure the way a population of normally-distributed fitnesses is transformed by selection: *reproduction rate*, *selection intensity*, and *loss of diversity*.

More recently, new selection techniques and extensions to the traditional selection techniques have been proposed. Some examples are:

- *Offspring Selection*, introduced by Affenzeller and Wagner [2, 3], is an extension to standard selection methods and attempts to guide selection by measuring the success of generated offspring and metering selection pressure accordingly. Affenzeller and Wagner claim results which show that for some problems offspring selection is an improvement over the more traditional selection techniques.
- *Fitness Uniform Selection Strategy (FUSS)* (Hutter, [130]) is a selection technique which dynamically adjusts selection pressure based on the proportion of the population considered “fit”. Hutter claims that *FUSS* is more effective than “standard” selection schemes.
- *Entropy-Boltzmann Selection* (Lee, [168]) applies the concepts of entropy and importance sampling to the Boltzmann selection method in an attempt to balance exploitation and exploration. Lee claims better performance in the tests performed than more conventional selection methods.

2.2.5 Recombination

Recombination is achieved by the use of one of a number of genetically-inspired recombination operators. Central to Holland’s genetic algorithm is the implementation of the *crossover* recombination operator. In earlier works, evolutionary algorithms relied solely on *mutation* of individuals in the population of potential solutions in order to evolve a good solution. The crossover operator approximates sexual reproduction in biology, and is the combination of genetic material from two individuals to produce one or more offspring. Potential solutions are encoded onto chromosomes, and it is those chromosomes which are the genetic material manipulated by the crossover and mutation operators.

2.2.5.1 Crossover for Fixed Length Chromosomes

Many crossover operators have been developed by genetic algorithm practitioners. Single point, two point, and uniform crossover are just three examples. Single point crossover is achieved by randomly choosing a single point at which to separate, swap, and rejoin the chromosome. Figure 2 is an example of single point crossover for chromosomes represented as fixed-length bit strings. Two point crossover is an extension of single point crossover. In two point crossover, two points are chosen at random and the segment of the chromosome between those points is exchanged.

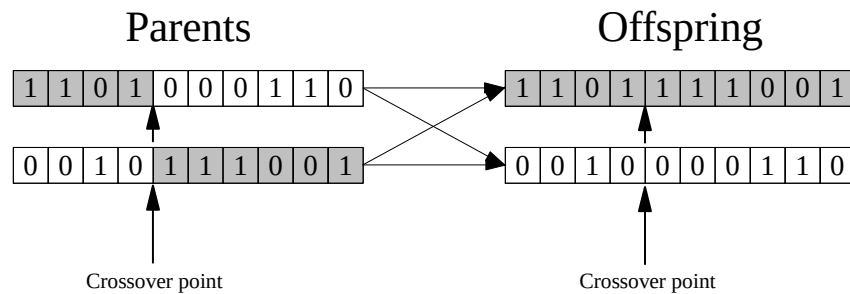


Figure 2 Example of single point crossover.

Single point crossover suffers from several problems. Arguably the most serious of these is the propensity for schemata² with long defining lengths to be destroyed when the chromosome of which they are part undergoes single point crossover. A similarly serious problem for single point crossover is its inability in some circumstances to combine all possible schemata. For example, single point crossover is not able to combine the parent schemata $*1*11***1$ and $1****11**$ to produce the offspring $11*1111*1$. This phenomenon is known as positional bias because the schemata which can be created or destroyed depend largely on the location of the bits on the chromosome. Two point crossover is less likely to destroy schemata with large defining lengths, and while it also is unable to combine all possible schemata, it can combine more than single point crossover.

Uniform crossover is implemented by generating a bit mask equal in length to the number of genes on the chromosome being manipulated, with the value of each bit on the mask being determined with some arbitrary probability. For each bit of the mask which is on (or a “1”) the corresponding gene of the parent chromosomes is swapped (or crossed over) before propagation to the offspring; and for each mask bit which is off (or a “0”) the corresponding

² see section 2.2.6 on page 28 for a definition of *schemata*

parent genes are propagated to the offspring unchanged. Figure 3 is an example of uniform crossover for chromosomes represented as bit strings.

Because each gene on the chromosome has some probability of being exchanged, uniform crossover suffers from no positional bias. Uniform crossover is able to combine all possible schemata, but it can be extremely disruptive of schemata of any length.

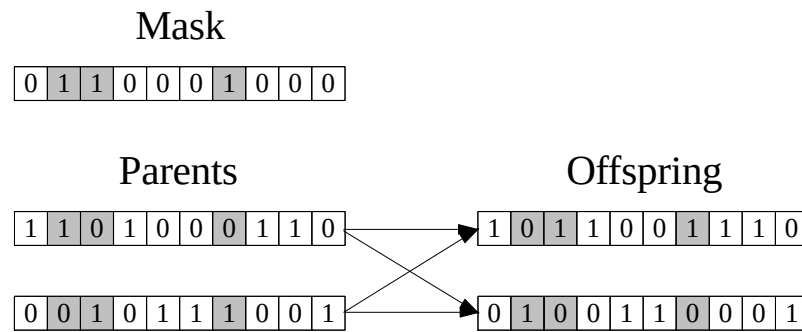


Figure 3 Example of uniform crossover.

2.2.5.2 Cut and Splice for Variable Length Chromosomes

Since the messy-coding implemented allows chromosomes of different lengths the crossover operation needs to be modified. For messy-coded genetic algorithms the crossover operation is considered in its two distinct steps: the *cut* operation and the *splice* operation. The *cut* operator cuts each chromosome at a randomly chosen position, and since the chromosomes may be of different lengths, the resultant fragments may also be of different lengths. The *splice* operator concatenates the fragments produced by the *cut* operator, resulting in two new chromosomes of possibly different lengths from the original chromosomes. The cut and splice operators implemented in this work guarantee the operations will not result in out-of-bounds data in the resultant chromosomes. Figure 4 is an example of the cut and splice operations for messy-coded chromosomes.

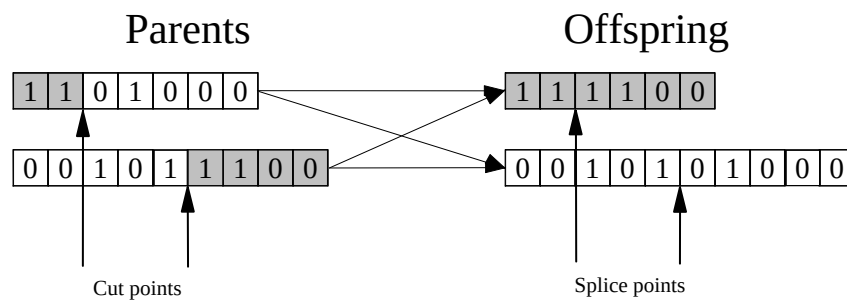


Figure 4 Example of cut and splice operations

2.2.5.3 Mutation

Mutation, which helps to maintain diversity in the population, is the arbitrary modification of individuals. As for crossover, there are many mutation operators and implementation variations of those operators, including random single-bit mutation, uniform random mutation, distribution-based mutation schemes, and complex, adaptive mutation schemes [12, 31, 86, 115, 120, 144, 191, 256, 282]. Perhaps unsurprisingly, the general result is that the best balance of mutation scheme and the rate and probability at which individuals undergo mutation is very dependent upon the nature of the problem and other parameters, such as the representation used and fitness function implemented.

The mutation scheme implemented in this work is a variation of random single-bit mutation – actually random single-allele mutation since the genes encoded in this work are integer values rather than single bits - in which a single allele is chosen randomly for modification to a random value. The mutation operator implemented in this work guarantees mutations will not result in out-of-bounds data in the resultant chromosome. Figure 5 shows an example of a single-bit mutation.



Figure 5 Example of single-bit mutation.

2.2.6 The Schema Theorem

Holland's *Schema Theorem* [120] and the concept of *schemata*, or *similarity templates* [86], offer an explanation of the search heuristics of genetic algorithms. In terms of bit strings defined over the alphabet $\{0,1\}$, a schema is simply a template string defined over the expanded alphabet $\{0,1,*\}$, where $*$ is a wild-card or don't-care character. Using this extended alphabet, bit strings can be said to be instances of many different similarity templates, or schemata.

For example, the bit string 010 is an instance of each schema in the set

$$\{***, 0**, *1*, **0, 01*, 0*0, *10, 010\},$$

and conversely the schema $00*1*$ describes the set of bit strings

$$\{00010, 00110, 00011, 00111\}$$

For any schema H , the order of H , $o(H)$, is defined by Holland to be the number of defined bits in the schema. Holland further defines the defining length of H , $d(H)$, to be the distance (number of bits) between the first and last defined bits of H .

For example, the schema

$$H = 0*01*011**$$

has $o(H) = 6$ and $d(H) = 7$

An important insight provided by Holland is that as the genetic algorithm explicitly searches through bit strings defined over the alphabet $\{0,1\}$, it *implicitly* searches the much larger space of schemata defined over the alphabet $\{0,1,*\}$. That is, knowledge of the fitness of a particular bit string implies some knowledge of the fitness of *all* the schemata of which that bit string is a member. Thus, the genetic algorithm can be said to be inherently parallel and so enable very large solution spaces to be searched in a reasonable time frame.

Holland derived a mathematical expression which predicts the number of copies of a given schema expected in a population after undergoing selection, crossover and mutation. Analysis of this expression suggests that schemata of lower order and defining length have a greater probability of survival after undergoing selection, crossover and mutation.

The schema theorem described by Holland states that the expected number of copies of very fit, short (in terms of defining length), low order schemata increases exponentially. Conversely, the expected number of copies of short, low order schemata of low fitness decreases exponentially. The Building Block Hypothesis [86] further states that the propagation and recombination from generation to generation of small, highly fit schemata is fundamental to the power of genetic algorithms. According to the schema theorem and building block hypothesis, it is this recombination of short, low order, highly fit schemata forming even more highly fit, higher order schemata which gives the genetic algorithm its power.

There has been, and continues to be, some debate regarding the usefulness of the schema theorem. Many researchers believe that, at best, the schema theorem has a limited usefulness or that it does not completely explain the search heuristics of genetic algorithms (e.g. [6, 46,

78-80, 100, 182, 195, 214, 266]), while others believe the theorem is basically sound or can remain relevant by being extended or varied (e.g. [56, 103, 189, 211, 243, 244]). In particular, extensions to the schema theory to cover the variable length chromosome of messy and messy-code genetic algorithms have been proposed (e.g. [212, 213]).

2.2.7 Current Research in Evolutionary Computation

The field of evolutionary computation is such a broad field and attracts so many active researchers that it is difficult to list all the different avenues of current and future research. Some of the more interesting areas are:

Artificial Immune Systems: The human immune system has evolved into a very robust, adaptive and distributed detection mechanism. Researchers in this new field are experimenting with the simulation of the natural immune system, including processes such as hypermutation, clonal selection, negative selection and immune network theory. Artificial immune systems are being applied to problems such as computer virus detection, fault diagnosis, intrusion detection and fraud detection.

Evolvable Hardware: Researchers are experimenting with the use of evolutionary algorithms to evolve the design of electronic circuits. Artificial evolution is applied to this task in one of two ways:

- an offline, or *extrinsic* model, in which a design is evolved by an evolutionary algorithm using hardware simulation software to evaluate fitness. Once an acceptable design is developed the physical hardware is constructed or configured using the evolved design.
- an inline, or *intrinsic* model, in which real hardware (usually a Field Programmable Gate Array (FPGA)) is configured and evaluated *in situ*.

Creative Evolutionary Systems: A growing area of research for evolutionary computation is the application of evolutionary algorithms to creative areas such as architecture, art, music, and design. The ability of artificial evolution to explore possibilities and find novel and creative solutions is one of its strengths, and researchers are using that strength to investigate how artificial evolution can generate novelty for (previously) unconventional applications in the creative arts.

2.3 Fuzzy Logic and Fuzzy Inferencing

Fuzzy logic was introduced by Lofti Zadeh [283] as an approach to handling the vague and uncertain method of human reasoning. Prior to Zadeh's work with fuzzy logic and fuzzy sets, set theory dictated that objects were either a member of a set or not, and if not were excluded from the set. Zadeh introduced the notion of fuzzy sets, where membership is a matter of degree and objects might exist to some degree in what may in classical, or crisp, set theory be conflicting sets. The essence of the issue is captured quite succinctly by the following excerpt:

"One seed does not constitute a pile, nor two nor three ... from the other side everybody will agree that 100 million seeds constitute a pile. What therefore is the appropriate limit? Can we say that 325,647 seeds don't constitute a pile but 325,648 do?" [27].

At the centre of the issue is the question of when, and how abruptly, an object ceases to belong to one category and begins to belong to another. For example, rather than a person being arbitrarily removed from the set of "children" at age 16 and placed in the set of "young adults", he or she might at age 14 be considered to be a child to some degree and also a young adult to some, probably lesser, degree. Similarly, rather than moving to the set of "adults" at 21, the set of "middle-aged people" at 45, and finally to the set of "old people" at 65, he or she might transition smoothly between sets by at times being a member, to some degree, of two or more sets. An important point is that classic set theory does not differentiate between members of a set. This means, for example, that two people whose ages are 65 and 95 may both be members of the crisp set of "old people", but while they are clearly "old" to different degrees they cannot be differentiated. Fuzzy set theory allows members of fuzzy sets to be differentiated by their degree of membership.

2.3.1 Linguistic Variables

The notion of linguistic variables is fundamental to fuzzy set theory. Following is a brief definition of the concept. A linguistic variable is a variable whose values are words or sentences that indicate fuzzy numbers.

More formally, a linguistic variable is characterised by a quintuple denoted by [285, 286]:

$$\langle X, T(X), \mathbf{X}, G, M \rangle$$

where

X is the name of the variable

$T(X)$ is the term set of X whose elements are labels of linguistic values of X

G is a grammar for generating the names of X

M is a semantic rule for associating with each label $L \in T(X)$ its meaning $M(L)$, which is a fuzzy set on the universe $\mathbf{X}$ whose base variable is x .

For example [207]:

Consider a linguistic variable named *temperature*, that is, $X = \text{temperature}$, with $\mathbf{T} = [0, 50]$ and base variable $t \in \mathbf{T}$. The term set associated with *temperature* could be

$$T(\text{temperature}) = \{\text{very low}, \text{low}, \text{medium}, \text{high}, \text{not low and not very high}, \text{very high}, \dots\}$$

where each term in $T(\text{temperature})$ is a label of a linguistic value of *temperature*. The meaning $M(T)$ of a label $T \in T(\text{temperature})$ is define to be the constraint $T(t)$ on the base variable t imposed by the name of T . Therefore $M(T)$ is a fuzzy set of $\mathbf{T}$ whose membership function $T(t)$ conveys the semantics of name T . Figure 6 illustrates the concept.

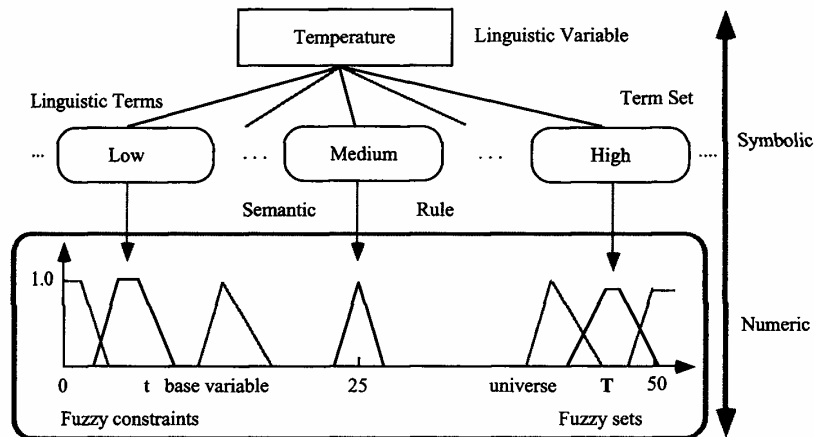


Figure 6 An example of a linguistic variable
(Reproduced from [207])

2.3.2 Fuzzy Sets

In fuzzy set theory [142, 156, 159, 283] a fuzzy set is defined as a collection of objects each of which is assigned a membership value which expresses *the degree to which the object is compatible with the properties or features distinctive to the collection*. Fuzzy sets can either be discrete or continuous, and membership values range from 0 to 1, where 0 indicates complete exclusion from the collection and 1 indicates complete inclusion [207].

The membership functions of fuzzy sets are usually represented graphically. For example, Figure 7 is a graphical representation of the membership function for the fuzzy set “old people” (showing ages 0 to 100). In this example, a person who is aged 40 years is considered to be “old” to degree 0.28, and a person aged 75 “old” to degree 0.93.

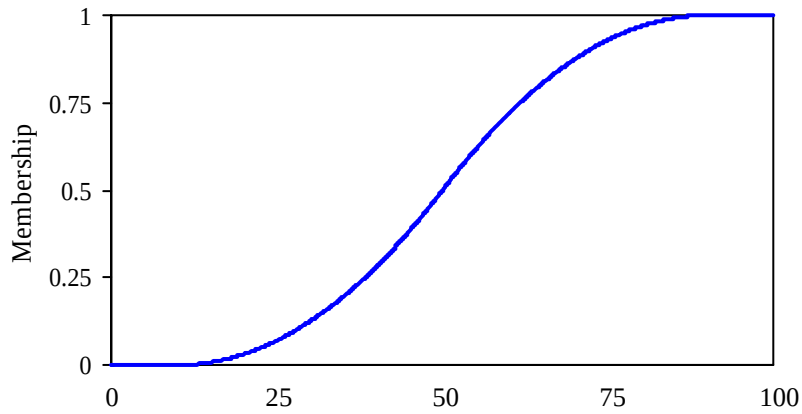


Figure 7 Membership function for the fuzzy set “old people”

The membership function shown graphically in Figure 7 is an S-function defined by Zadeh as

$$\mu_A(x, \alpha, \beta) = \begin{cases} 0 & \text{if } x \leq \alpha \\ 2\left(\frac{x - \alpha}{\beta - \alpha}\right)^2 & \text{if } \alpha < x \leq \left(\frac{\alpha + \beta}{2}\right) \\ 1 - 2\left(\frac{x - \beta}{\beta - \alpha}\right)^2 & \text{if } \left(\frac{\alpha + \beta}{2}\right) < x \leq \beta \\ 1 & \text{if } x > \beta \end{cases},$$

but membership functions can be any function of the form $\mu_A(x): X \rightarrow [0,1]$, and the suitability of a function is usually determined by the context of the circumstance in which it is to be used. Some of the more commonly implemented membership functions are triangular functions, trapezoidal functions, gaussian and sigmoidal functions.

The method used to define the shape of membership functions is most often dependent on the application. For most fuzzy logic control problems the assumption is that the membership functions are linear triangular functions. However, triangular functions may not accurately represent the linguistic terms being modelled, so for some applications triangular membership functions may not be the most appropriate membership functions. For most applications there is no definitive method for the determination of the membership function μ_A , and as a result various manual and automatic methods are often used.

2.3.2.1 Fuzzy Set Operations

Analogues for traditional crisp set operations have been defined for fuzzy sets. Some of the more common operations defined are [283]:

- Union

$$\mu_{A \cup B}(x) = \max(\mu_A(x), \mu_B(x))$$
- Intersection

$$\mu_{A \cap B}(x) = \min(\mu_A(x), \mu_B(x))$$
- Complement

$$\mu_{X-A}(x) = 1 - \mu_A(x)$$

2.3.3 Fuzzy Rules and Approximate Reasoning

Zadeh proposed the following *max-min* composition operation for combining fuzzy relations [283, 284]:

Let R_1 and R_2 be two fuzzy relations defined on $X \times Y$ and $Y \times Z$ respectively, then the *max-min* composition of R_1 and R_2 is a fuzzy set defined by:

$$R_1 \circ R_2 = \left\{ \left[(x, z), \max_y \min(\mu_{R_1}(x, y), \mu_{R_2}(y, z)) \right] \mid x \in X, y \in Y, z \in Z \right\}$$

which can be written as

$$\mu_{R_1 \circ R_2}(x, z) = \vee_y [\mu_{R_1}(x, y) \wedge \mu_{R_2}(y, z)]$$

where $\vee$ represents *max*, and $\wedge$ *min*.

Other methods of composition have been proposed. One of the most commonly used is the *max-product* composition operation, which can be written as follows:

$$\mu_{R_1 \circ R_2}(x, z) = \vee_y [\mu_{R_1}(x, y) \mu_{R_2}(y, z)]$$

Fuzzy inferencing operates on the basis of rules which are expressed in the form of **if-then** constructs. The fuzzy rules are written as

if premise then consequent

where the rule premise and consequent are fuzzy propositions. A fuzzy proposition is of the form "*temperature is hot*", "*Brian is young*" etc. The rule premise, or antecedent, may be a compound fuzzy proposition – multiple atomic propositions joined with fuzzy connectives such as AND, OR and NOT. The rule consequent may also be a compound fuzzy proposition, but in this work only simple, or atomic, consequents are considered.

2.3.4 Fuzzy Inferencing Systems

Fuzzy Inferencing Systems (FIS) use fuzzy sets and reasoning techniques relevant to fuzzy sets to make decisions or draw conclusions about imprecise, incomplete or vague information. Due to the multidisciplinary nature of fuzzy inferencing systems they are sometimes also known as *fuzzy expert systems* and *fuzzy rule-based systems* [143], *fuzzy associated memory* [157], *fuzzy logic controllers* [166, 167, 184], and just plain *fuzzy models* [254]. One of the largest areas of application for fuzzy inferencing systems relevant to this thesis has been, and continues to be, control systems [54, 55, 132, 166, 167, 184, 228, 233] including the control of mobile robots [19, 20, 58, 225].

Fuzzy inferencing systems are usually comprised of three main conceptual components:

- A *database* containing the membership functions for the input variables
- A *rulebase* containing fuzzy rules
- A *reasoning mechanism* that performs the inference procedure

so the operation of the FIS depends on the execution of three or four major tasks, depending upon the type of output required:

- *Fuzzification*: determines the degree of membership for crisp input variables
- *Inference/Implication*: evaluates the fuzzy rules and produces an output for each rule
- *Aggregation/Composition*: combines the outputs of all the rules
- *Defuzzification*: computes the crisp output

The defuzzification task is not performed if crisp output is not a requirement. These tasks are reflected in the FIS shown in Figure 8.

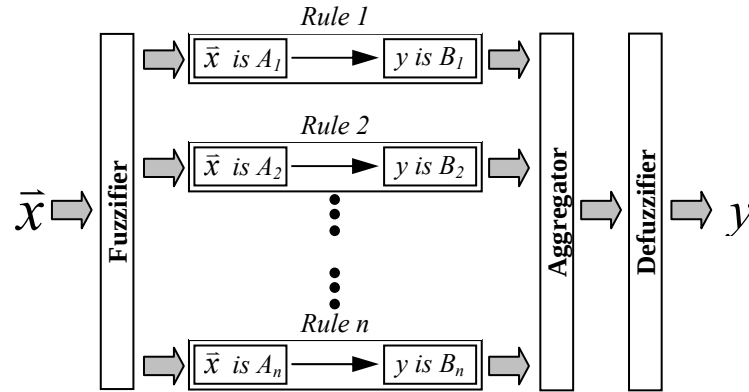


Figure 8 Fuzzy Inferencing System

Two of the most commonly adopted fuzzy inferencing systems are those described by Mamdani [184] and Sugeno [252, 254]. The difference between the two methods is in the consequents of the fuzzy rules, and thus their aggregation and defuzzification methods differ accordingly. In the following sections the common tasks of fuzzification and inferencing are described, followed by separate descriptions of aggregation and defuzzification in the Mamdani and Sugeno models.

2.3.4.1 Fuzzification

Fuzzification of input variables is the process of determining the degree of membership of fuzzy sets for each of the crisp input variables. An example of input variable fuzzification is shown in Figure 9. In this example the crisp input variable x has a degree of membership of both fuzzy sets A_1 and A_2 .

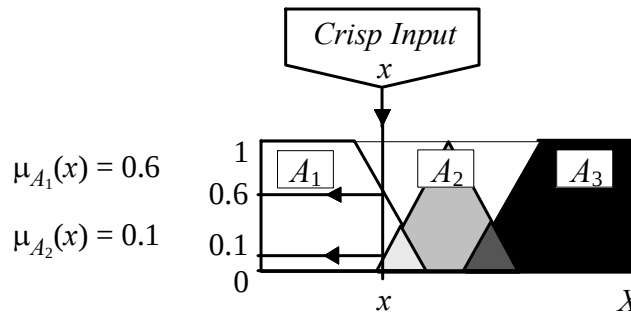


Figure 9 Input Variable Fuzzification

2.3.4.2 Inference/Implication

After the input values are fuzzified they are applied to the antecedents of the fuzzy rules. For fuzzy rules with multiple antecedents, the fuzzy operators AND and OR are used as appropriate to obtain a single number that represents the result of the antecedent evaluation. This value is the degree to which the rule is true and is then applied to the consequent membership function. The evaluation of the antecedents is as follows:

- for the *disjunction* of rule antecedents, the fuzzy operator *OR* is defined by the fuzzy set operation *union*:

$$\mu_{A \cup B}(x) = \max[\mu_A(x), \mu_B(x)]$$

- for the *conjunction* of rule antecedents, the fuzzy operator *AND* is defined by the fuzzy set operation *intersection*:

$$\mu_{A \cap B}(x) = \min[\mu_A(x), \mu_B(x)]$$

Two of the more common methods of correlating the result of the antecedent evaluation with the membership function of the consequent fuzzy set are *Correlation Minimum* and *Correlation Product*:

- **Correlation Minimum** is the most commonly used method. This method clips the consequent membership function at the level of the antecedent truth and as a result the consequent fuzzy set loses some information, but it is straight forward to calculate and provides an aggregated output surface that is relatively easy to defuzzify.
- **Correlation Product** scales the consequent fuzzy set preserving its original shape, therefore loses less information than the correlation minimum method. The method adjusts the original membership function of the rule consequent by multiplying all its membership degrees by the truth value of the rule antecedent, so is more complex to calculate and allows an aggregated output surface which is more difficult to defuzzify.

A visual comparison of the two methods is shown in Figure 10.

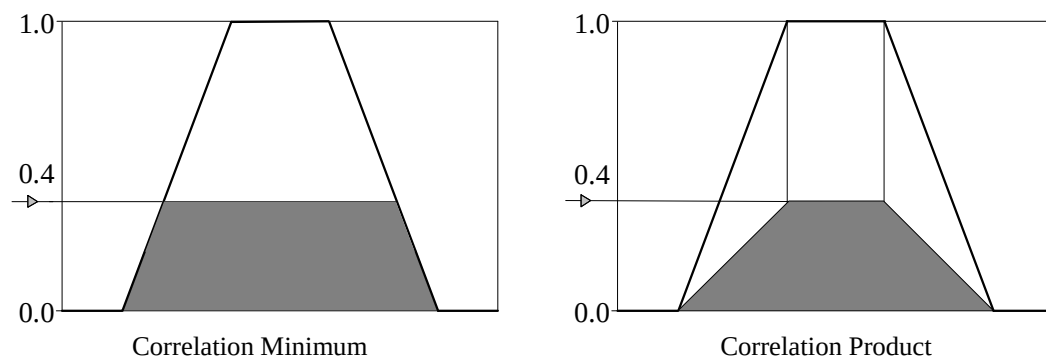


Figure 10 Correlation methods

2.3.4.3 Aggregation/Composition and Defuzzification

Where correlation involves action taken on one fuzzy set at a time, the aggregation process acts on multiple fuzzy sets. Aggregation is the process of combining the correlated fuzzy sets to produce a composite fuzzy region that represents the solution variable. The solution fuzzy region is then defuzzified if a crisp solution is required.

These are the tasks for which the Mamdani and Sugeno models differ. While the two models are similar, there is a difference in the way each model defines rule consequents, thus requiring different aggregation and defuzzification methods. In fact, since the Sugeno model implements rule consequents as functions of the input variables which produce crisp outputs, there is no need for defuzzification at all, and the aggregation process, which technically operates on fuzzy sets, is replaced by a simple weighted average of the crisp outputs.

Mamdani Fuzzy Inferencing Systems

There are two commonly implemented forms of aggregation: the *min/max* method, and the *additive* method.

The *min/max* method ORs the correlated consequent fuzzy set for each rule with the contents of the solution variable's output fuzzy region. This process takes the maximum of the consequent fuzzy set and the solution fuzzy set at each point along their mutual membership functions. Figure 11 is an illustration of a two-rule Mamdani Fuzzy Inferencing System which implements the correlation minimum implication method and the min/max method of aggregation.

The *additive* method on the other hand ADDs the correlated consequent fuzzy set for each rule with the contents of the solution variable's output fuzzy region. This process adds the truth membership values of the consequent fuzzy set and the solution fuzzy set at each point along their mutual membership functions. The addition is a bounded sum operation so that the composite membership value cannot exceed 1.0.

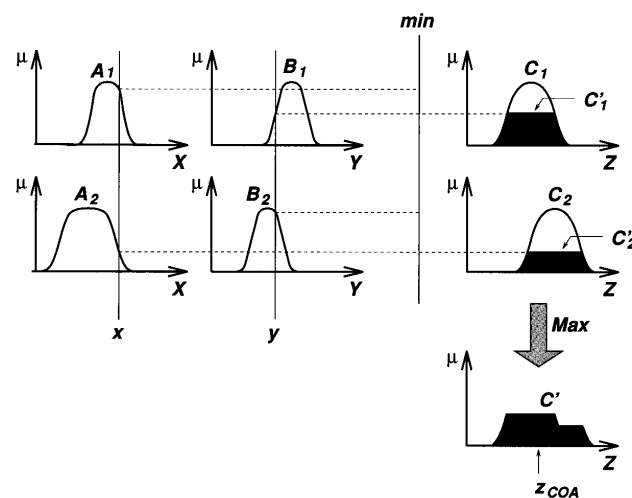


Figure 11 2-rule Mamdani FIS using Correlation Minimum implication and min/max aggregation
(Reproduced from [133])

There have been many methods for defuzzification proposed in the literature. A selection of those proposed are:

- *Smallest of Maximum* is the smallest value at which the membership function μ reached the maximum μ^* .
- *Largest of Maximum* is the largest value at which the membership function μ reached the maximum μ^* .
- *Mean of Maximum* is the average of the values at which the membership function μ reached the maximum μ^* .
- *Centroid of Area* is the fuzzy “centre of mass” of the output fuzzy set.
- *Bisector of Area* is the value which partitions the output fuzzy set into two regions of equal area.

These methods are illustrated in Figure 12.

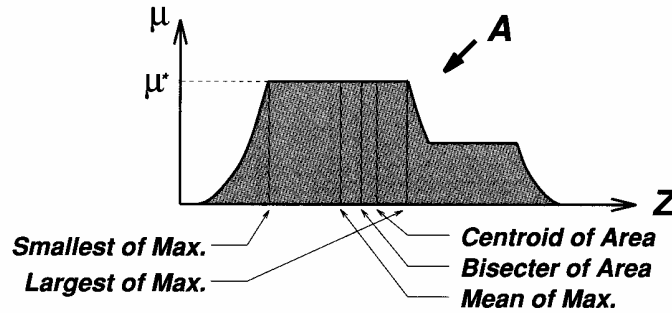


Figure 12 Methods of defuzzification
(Reproduced from [133])

Sugeno Fuzzy Inferencing Systems

Rather than implement the rule consequent as a fuzzy set, Sugeno used a mathematical function of the input variable. For example, a rule in the Sugeno model is written as:

$$\text{if } x \text{ is } A \text{ and } y \text{ is } B \text{ then } z \text{ is } f(x, y)$$

where x , y and z are linguistic variables; A and B are fuzzy sets on universe of discourses X and Y , respectively; and $f(x, y)$ is a mathematical function.

The Sugeno model (also known as the *TSK fuzzy model* after Takagi, Sugeno and Kang) [252, 254] was originally proposed with the output function implemented as a first-order

polynomial, and a Sugeno model so implemented FIS is known as a *first-order Sugeno FIS* (Figure 13). Sugeno fuzzy models are most commonly implemented with the output of each fuzzy rule being a constant (i.e. $f(x, y) = k$, where k is a constant). A Sugeno model implemented in this way is known as a *zero-order Sugeno FIS*.

Since each rule in a Sugeno FIS has a crisp output, no defuzzification is required. The rule outputs are aggregated by calculating the weighted average of the rule outputs, or sometimes in order to reduce computation overhead, the weighted sum of the rule outputs.

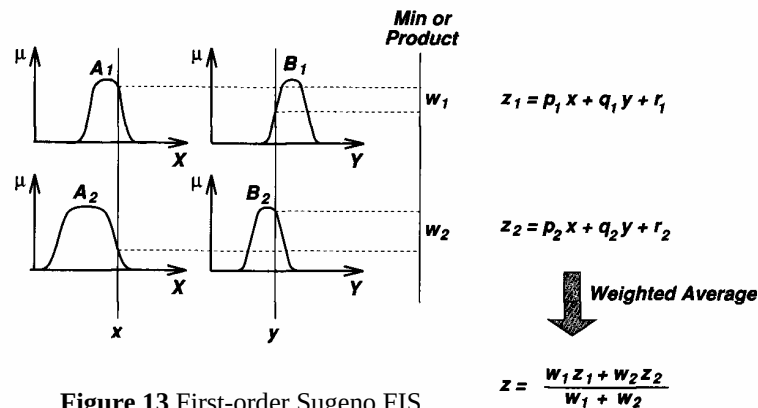


Figure 13 First-order Sugeno FIS
(Reproduced from [133])

2.3.5 Mamdani vs. Sugeno

The main benefit of the Sugeno model is the computational simplicity of the output when compared to the Mamdani model. However there is no straight-forward method of determining the coefficients for the polynomials when implementing a first-order Sugeno FIS. For such cases, and for the constant output for the zero-order model, these parameters could be learned with the use of an evolutionary algorithm. The Sugeno model does not translate as easily to human reasoning models as does the Mamdani model – the solution fuzzy sets and associated linguistic variables are more easily interpreted and understood by humans than the polynomial or constant output of the Sugeno model. In some sense the Mamdani model seems to fit more intuitively with the robot soccer problem – the ability to respond to an uncertain environment with fuzzy actions seems to be more desirable than responding with a crisp polynomial or constant measure for some action.

2.3.6 Genetic Fuzzy Systems

In recent years many researchers have investigated the use of evolutionary algorithms as tools for aiding in the design and construction of fuzzy systems. This hybridisation is commonly referred to as *Fuzzy Evolutionary Computation* or *Genetic Fuzzy Systems*, and ranges from parameter optimisation to learning the entire set of rules for fuzzy rule-based systems [53, 208, 226].

The particular type of genetic fuzzy system used in this work is a *Genetic Fuzzy Rule-Based System* – a system in which an evolutionary algorithm is used to generate or optimise one or more of the components of a fuzzy rule-based system. There has been a great deal of work in this field in the past ten years or so, with many researchers contributing in many areas. Some work has focussed on using the evolutionary algorithm to optimise the fuzzy rulebase only, some focusses on optimising the parameters of the fuzzy system, particularly the fuzzy set membership functions, while still other work endeavours to optimise the complete system – rules and parameters.

Previous research on the evolutionary optimisation of fuzzy rulebases using genetic algorithms can be divided into three main sub-categories based upon the way in which the genetic algorithm is applied:

- the *Pittsburgh* approach [36, 117-119, 131, 169, 183, 190, 209, 236, 280, 287], where each chromosome is a complete set of rules.
- the *Michigan* approach [25, 26, 92, 121, 206, 260, 264], where each chromosome a single rule.
- the *Iterative Rule Learning* approach [51, 96, 97, 265], similar to the Michigan approach in that each chromosome is a single rule, but with the Iterative Rule Learning approach only the best individual from a run is kept (and the remainder of the population discarded), thus each run of the genetic algorithm provides a partial solution to the problem, requiring multiple runs to generate a complete solution.

As well as genetic algorithms some work with using of genetic programming to develop the rulebase for fuzzy rule-based systems has been done. Some example can be found in [5, 85, 116]. Various types of evolutionary algorithms have been investigated for use in the evolutionary optimisation of the fuzzy set membership functions: e.g. evolution strategies [51],

genetic algorithms [51, 112, 145, 149, 169, 183, 186, 190, 230, 280, 287], genetic programming [22, 122, 123].

The approach used in the work presented in this thesis is to use a messy-coded genetic algorithm to optimise only the fuzzy rules using the Pittsburgh approach: the complete rulebase is encoded onto a single chromosome. The fuzzy set membership functions used in this work are predefined and fixed.

Since Smith first used the Pittsburgh method to successfully evolve a co-operative set of fuzzy rules for playing poker [236] other researchers have used the method to evolve the rules and parameters for fuzzy logic controllers in various forms. Following is a review of some selected work from the field, focussing generally on work of most relevance to this thesis – the use of genetic algorithms to evolve the fuzzy rulebase using the Pittsburgh approach.

In [169] Lee and Takagi report the use of a genetic algorithm and penalty-based evaluation strategy to determine the shape and position of the fuzzy set membership functions, the number of fuzzy rules and the rule consequent parameters for a fuzzy-logic controlled pole-balancing system. The fuzzy inferencing system implemented in the work was a TSK fuzzy model (see section 2.3.4.3 on page 38). The fuzzy set membership functions evolved are triangular and characterized by left base, right base and centre-line distance from the centre-line of the previous membership function. The rule antecedents are pre-defined and the coefficients for the consequent polynomials are evolved. The chromosome was implemented as a multi-part chromosome with sections of the chromosome representing the membership functions and consequent coefficients. A small population size of 10 was used, with an elitist selection strategy (see section 2.2.4 on page 21) and a maximum number of generations of 5000. The method produced reasonably small rulebases of just four rules. Lee and Takagi found that the method produced similar results to a method developed by Jang [134, 135] which used a neural network to determine the parameters for the fuzzy logic controller.

Zhou and Lai [287] describe the use of a genetic algorithm to evolve the membership functions for the fuzzy sets as well as the consequent variable for a fixed set of fuzzy control rules (implemented as a fuzzy control table) for a fuzzy PID³ controller. The chromosome of the genetic algorithm was implemented as a two-part chromosome: the first part being the integer-coded consequent variables on the fuzzy control table, and the second the real-coded

³ PID (Proportional, Integral, Derivative) controllers are automatic controllers governed by a feedback loop from the controller output to the input.

parameters of the fuzzy set membership functions. The fuzzy set membership functions were triangular and characterized by the values of the two base vertices. Zhou and Lai implemented a dynamic mutation rate and performance-based stopping conditions. The maximum number of generations was set at 50, but good results were usually achieved before the maximum. Zhou and Lai report results which indicate that the method is effective and efficient, especially in comparison to the (then) conventional trial-and-error design process for fuzzy logic controllers.

In [117-119] Hoffmann and Pfister use a messy-coded genetic algorithm (see section 2.2.2 on page 17) to evolve the fuzzy rules for an autonomous mobile robot. The robot is trained to navigate through a series of corridors from an initial location to a selected target location, and avoid objects placed in its path. In this method Hoffmann and Pfister employed a variable length messy-coded chromosome coding the clauses for the fuzzy rules. Overspecification was resolved by simply OR-ing like terms, otherwise clauses were assumed to be connected by the AND operator. The method uses a second genetic algorithm to dynamically update the training environment parameters, such as object locations, corridor sizes, robot start and end points). The method implements a dynamic fitness function where the reward changes depending upon previous results of various subtasks – for example, once the “avoid objects” subtask is successfully learned, the reward for other subtasks is increased on the basis that it is more important to learn subtasks which have not yet been mastered. Hoffmann and Pfister report promising results – the robot was able to learn and perform well in real world situations if those situations were not significantly more difficult than the training environment.

Magdalena and Monasterio-Huelin [183] use a genetic algorithm to evolve new gait patterns for a bipedal walker using existing patterns as the starting point. In this method the fuzzy rulebase and the parameters for the fuzzy controller are evolved. The genetic algorithm chromosome is a combination of integers representing the parameters of the fuzzy logic controller (e.g. number of inputs, number of outputs, number of fuzzy sets), real numbers for the fuzzy set membership functions, and a bit string that represents a variable number of rules. The results reported indicate that the method performs well, and that the evolved gait patterns outperformed the initial patterns.

The work presented in [280] by Wu and Liu describes the use of a genetic algorithm to evolve both the input and output fuzzy set membership functions for a fuzzy controller for the cart-centring problem. The membership functions are triangular in shape, and characterised by

left and right bases plus location of centre-line. The rule antecedents are all possible combinations of fuzzy input variables, and only the membership functions for the output fuzzy sets are evolved. The genetic algorithm chromosome is real-coded, representing the membership function parameters for the input and output fuzzy sets. Parents and offspring compete for selection using a rank selection scheme (see section 2.2.4 on page 21). A population of 50 individuals is evolved over 100 generations. While no comparisons were made with results from other methods, the method was shown to be feasible for the cart-centring problem under various starting conditions.

Mirzaei et al. use an island model parallel genetic algorithm – a model in which separate, isolated sub-populations (islands) evolve in parallel with occasional migration of genetic material between the sub-populations – to evolve the rulebase and membership functions for a TSK/Sugeno fuzzy controller (see section 2.3.4.3 on page 38) for an anti-lock braking system. The parallel genetic algorithm is implemented with two sub-populations of 50 individuals each. Elitist selection is used to ensure a number of the fittest individuals are propagated to subsequent generations, with the remainder of the population being selected by the roulette wheel method (see section 2.2.4 on page 21). The resultant fuzzy controller is tested on a vehicle model under various road conditions. Very good performance is reported for different road conditions, with wheel skip being kept to the desired level and stopping distance significantly lower than for wheel lock-up.

A review of a small selection of the hundreds of publications in this area [52] was presented in this section. A comparison of the Pittsburgh and Michigan approaches is presented in [210], and an overview of genetic fuzzy systems is presented in [52].

2.3.7 Research Trends in Fuzzy Systems

Fuzzy logic and related fields encompasses so much research that it would not be possible to discuss it all in this thesis. Instead a brief discussion of the research trends in an area of research directly related to this thesis - genetic fuzzy systems – is presented.

The major areas of continuing research in genetic fuzzy systems are:

- the application of different evolutionary techniques to creating the fuzzy systems - for example, genetic programming is regarded as a technique well suited to generating fuzzy rules, and that continues to be a popular avenue for research.

- techniques to evolve the fuzzy system parameters - the number of inputs and outputs of the fuzzy system and fuzzy set memberships - as well as the fuzzy rules.
- rule selection and rulebase reduction techniques – the size of the fuzzy rulebase is sometimes an issue, especially for high-dimensional problems. Machine learning techniques including neural network and evolutionary algorithms are being applied to address this issue.

2.4 Fitness Landscapes

The concept of fitness landscapes, and the idea that the process of evolution could be studied by visualizing the distribution of fitness values across the population as a landscape, has been long-established in the field of evolutionary biology, having been first proposed by Wright in 1932 [279]. Later the landscape analogy was revived with the development of formal methods to handle optimization problems in complex physical systems [83].

Wright’s original diagram, reproduced in Figure 14, is a two-dimensional depiction of the relative “adaptiveness” of all possible combinations of allelic states for genes on a particular chromosome. The diagram is effectively a contour map with the contour lines representing levels of adaptiveness. This original diagram included no labels for the axes, and no indication was given as to how the various gene combinations should be arranged on the landscape - no notion of “neighbourhood” was defined or discussed by Wright. In landscape terms, the neighbourhood relation defines which points, or individuals, are arranged as immediate neighbours on the landscape, and so is extremely important in defining the landscape.

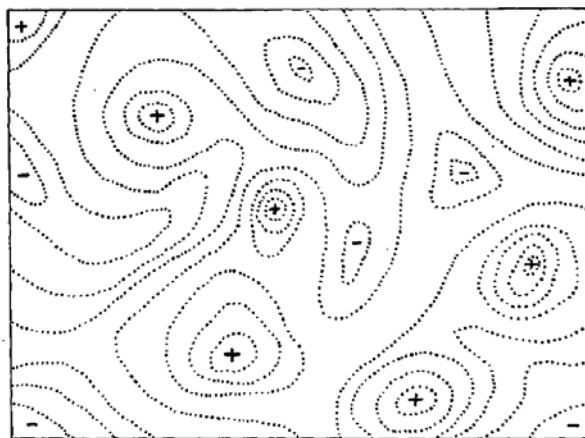


Figure 14 Diagrammatic representation of the field of gene combinations in two dimensions instead of many thousands. Dotted lines represent contours with respect to adaptiveness.
(Reproduced from [279])

Much of the work involving fitness landscapes avoids a rigorous definition of the landscape under analysis [137], and where it is mentioned or implied at all the landscape is usually assumed to be the *single-bit mutation landscape*: the landscape generated by arranging all single-bit mutations of a chromosome represented as a string of binary digits such that chromosomes that differ by only a single bit are neighbours, then plotting the fitness of each chromosome on a separate axis. On such landscapes, genetic operators such as crossover are assumed to take *hypersteps* over the fitness landscape described by mutation. There have been attempts to overcome this deficiency, and following is a description of some of major recent work in the field.

The *NK fitness landscape model* [146, 147] was proposed by Kauffman as a means to study how epistasis - the dependency of fitness upon the interaction of the allelic state of multiple genes [180] - affects the ruggedness of the fitness landscape. An NK fitness landscape is defined by a fitness function which can be tuned in order to alter the ruggedness of the resultant fitness landscape. The fitness function is defined by two parameters: the number of genes (N), and the number of epistatic links, or interactions, between genes (K). In the most common implementations each gene has two possible alleles, thus the genotype can be represented by a bit string of length N . Each gene in a chromosome contributes to the fitness of the chromosome based on $K+1$ values: its own and those of the K genes to which it is linked. The three-dimensional fitness landscape is constructed by arranging chromosomes in two dimensions on the landscape such that bit strings that differ in the value of only one bit are neighbours, then using the fitness of the chromosomes as the third dimension. The NK landscape is widely used by the evolutionary computation community to generate epistatic landscapes as test functions for search and optimisation techniques [67, 110, 234, 235].

Weinberger [268, 270] proposed a fitness landscape model in which the landscape is represented as a graph on which the vertices correspond to individuals and have associated fitness values, and traversing the edge of the graph corresponds to the action of a genetic operator (mutation, crossover etc.) and so taking a step on the landscape. Jones [137, 140], Culberson [57], and later Hordijk [124] describe similar fitness landscape models in which the landscape is represented as a graph (e.g. Figure 15 on page 48). Reidys and Stadler [217] analyse a similar fitness landscape topography, focussing on the geometry of the moves from one vertex to another and provide a mathematical treatment of the edge traversals and the resultant complex topographies.

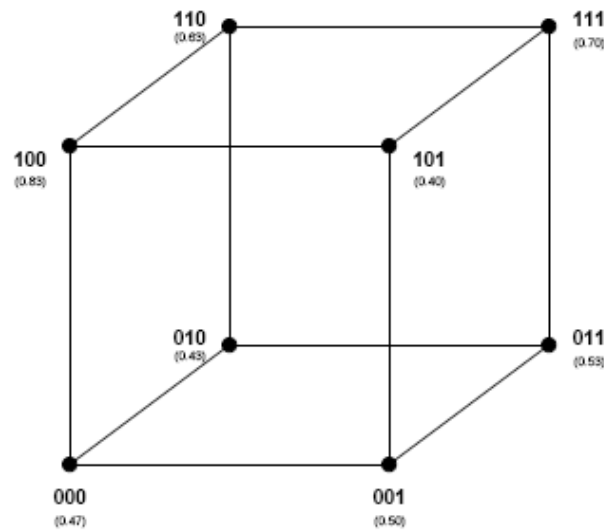


Figure 15 An example of a fitness landscape for bit strings of length three, where the neighbourhood relation is defined by point mutation. Fitness values (randomly assigned) are shown in parentheses.
(Reproduced from [124])

Culberson's model described a landscape defined by a crossover operator rather than mutation, and in which the graph vertices represented a population of points. Jones presents a similar model, and expands it further to include the concept that each genetic operator defines its own separate landscape [137]. In Jones' "one operator, one landscape" model, an evolutionary algorithm which implements the genetic operators selection, mutation and crossover makes transitions on three separate landscapes (Figure 16). According to Jones' model the evolutionary algorithm takes steps on the mutation landscape, after which individuals are paired to form vertices on the crossover landscape and further steps are then taken on that landscape, and then the population is gathered into a vertex on the selection landscape for a further step there. The neighbourhood relation in Jones' model is therefore simply defined by the genetic operator for each landscape.

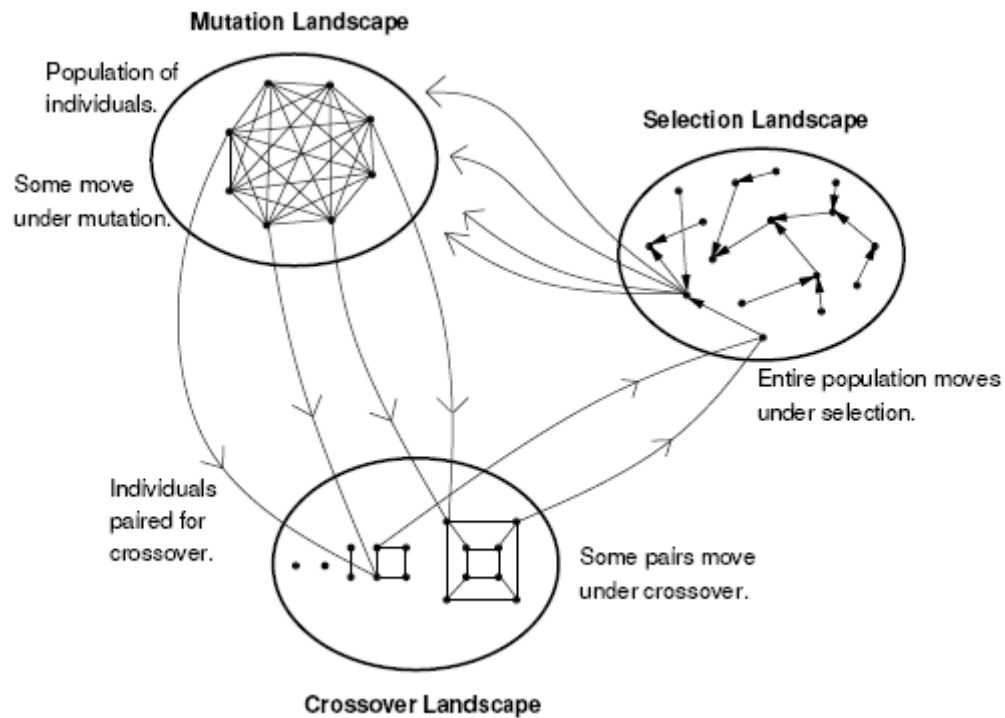


Figure 16 A simplified view of an evolutionary algorithm operating on three landscapes.
(Reproduced from [137])

More recently Moraglio and Poli [194] present a new topological framework for evolutionary algorithms and as part of that framework redefine the mutation and crossover operators to be more tightly linked to the fitness landscape. In the model proposed by Moraglio and Poli the genetic operators are defined by the fitness landscape upon which they operate – the genetic operators are a natural consequence of the neighbourhood relation and distance metric of the fitness landscape. For example, under what Moraglio and Poli call *topological uniform crossover*, offspring are defined as those individuals that lie between the parents on the fitness landscape. Similarly, under *topological ε -mutation*, offspring are defined as those individuals that lie ε steps away from the parent on the fitness landscape.

In essence, the fitness landscape is a *metaphor* – a metaphor in which the landscape is often thought of as a 3-dimensional terrain, where the peaks (hills or mountains) represent areas of high fitness and the valleys between the peaks low fitness. A fitness landscape which depicts the fitness values of a population that varies greatly in fitness will likely display many local high peaks surrounded by deep valleys. Such a landscape is said to be *rugged*. Similarly, if many individuals in the population have a similar fitness, the landscape will vary little and is said to be *flat*. Search on some landscapes is notionally easier than search on others – search

on a predominantly flat landscape is likely to be difficult, as is search on a rugged landscape with peaks and valleys randomly distributed.

In evolutionary biology the proliferation of an individual's genetic material is considered the ultimate objective of life, and the success or failure of a particular genotype, or its phenotypic expression, is most often measured by the number of progeny it produces – so for evolutionary biology, the fitness of an individual is a function of the number of progeny produced by that individual. For evolutionary optimisation the objective is usually less nebulous, and the success or failure of a particular genotype, or phenotype, is measured by a well-defined fitness function, and typically the number of progeny produced by an individual is a function of the fitness of that individual. Fitness landscapes are used in evolutionary biology and evolutionary optimisation to show the correlation between genotypes, or phenotypes, as a measure of success or failure and search difficulty [138, 140, 146, 147, 163, 237, 263].

2.4.1 Landscape Measures

Several methods for measuring and analysing landscapes for search algorithms have been proposed in previous work. The methods proposed can be categorised into two broad streams: statistical measures [6, 124, 138, 173, 185, 267, 269] and information measures [28-30, 262, 263]. Borenstein and Poli have extended the information measure to include a measure of the performance of the algorithm and so define a “performance landscape” which may prove useful but does not yet have sufficient history to gauge its efficacy. All the methods proposed have in common the notion that the points in the search space are arranged according to some neighbourhood relationship, and a measure of fitness, performance or information content associated with the points defines the ruggedness of the landscape.

The methods used to measure and analyse the structure of fitness landscapes in this thesis are the *autocorrelation* method suggested by Weingberger [267, 269], and the *information content* approach suggested by Vassilev [262, 263]. These methods were chosen because they are different methods of measuring the structure of landscapes and, while there seems to be no generally accepted standard approach, both methods have gained some favour and are commonly cited as reasonable landscape characterisation methods [124, 187, 237, 242]. Jones' *Fitness Distance Correlation* [138] is an interesting landscape measure but requires that the solution be known in order to calculate the metric, so is not applicable to the work presented in this thesis.

2.4.1.1 Autocorrelation and Correlation Length

Correlation measures are the most commonly used means of characterising fitness landscapes and are seen as good indicators of ruggedness. Autocorrelation refers to the correlation of a time series with its own past and future values, and is also known as serial correlation, referring to the correlation between members of a series of numbers arranged in time. Autocorrelation is a correlation coefficient, but instead of measuring the correlation between two different variables, autocorrelation measures the correlation between two values of the same variable at times X_i and X_{i+k} , where k is the number of time steps, or lag, between the two values.

Autocorrelation definition [124, 267, 269]:

Given measurements, $Y_1, Y_2, \dots, Y_N$ at time $X_1, X_2, \dots, X_N$, where N is the number of measurements, and

$$\bar{Y} = \frac{1}{N} \sum_{i=1}^N Y_i, N > 0$$

the time lag k autocorrelation function r_k is defined as

$$r_k = \frac{\sum_{i=1}^{N-k} (Y_i - \bar{Y})(Y_{i+k} - \bar{Y})}{\sum_{i=1}^N (Y_i - \bar{Y})^2}, N > k$$

Equation 1 Autocorrelation Function

If $|r_k| \approx 1.0$ there is much correlation between the points k steps apart in the series, whereas if $|r_k| \approx 0.0$ there is little correlation.

Weinberger proposed that a random walk be generated on the fitness landscape, where each step on the walk is taken between neighbouring points but the neighbour to which the step is taken is selected randomly, and the fitness values for each point visited during the random walk form a time series of numbers. The autocorrelation function can then be used as a measure of the ruggedness of the landscape described by the random walk. The landscape

measures used in this thesis are calculated using the random walk described in section 7.5 on page 176.

The correlation length of a series of numbers is the largest distance, or time lag, between points for which some correlation exists. Hordijk [124] defines the correlation length of a time series as one less than the first time lag for which the autocorrelation falls inside the region bounded by the two-standard-error bound (i.e. one less than the first time lag at which the autocorrelation becomes statistically equal to zero, making the correlation length the largest time lag for which the correlation between two points is still statistically significant). This is the method used for calculating the correlation length in this thesis. The two-standard-error bound e is defined as

$$e = \pm \frac{2}{\sqrt{N}}$$

so the correlation length ℓ is defined in this thesis as the first lag k for which

$$|r_k| < \left| \frac{2}{\sqrt{N}} \right|$$

Equation 2 Correlation Length

2.4.1.2 Information Content

An alternative to the statistical autocorrelation measure proposed by Weinberger is Vassilev's *information content* method, based on both classic and algorithmic information theory [39, 232]. Vassilev et al. propose three information measures that characterise the structure of a fitness landscape from a series of points generated by a random walk over the landscape [262, 263]:

- *Information Content* – characterises the ruggedness of the landscape.
- *Partial Information Content* – measures the modality of the landscape.
- *Information Stability* – the sensitivity of the information content measures.

These measures are calculated by generating a random walk of length n on the fitness landscape, with the aim being to extract information by characterising the series of points as an ensemble of objects. To calculate the information content, a string

$$S(\varepsilon) = s_1 s_2 \dots s_n, \text{ where } s_i \in \{\bar{1}, 0, 1\}$$

representing a group of objects generated from a random walk over the fitness landscape. $S(\varepsilon)$ is enumerated according to the function

$$s_i = \psi_{f_i}(i, \varepsilon), \text{ for } i = 1..n$$

and

$$\psi_{f_i}(i, \varepsilon) = \begin{cases} \bar{1} & \text{if } f_i - f_{i-1} < -\varepsilon \\ 0 & \text{if } |f_i - f_{i-1}| \leq \varepsilon \\ 1 & \text{if } f_i - f_{i-1} > \varepsilon \end{cases}$$

Thus the string $S(\varepsilon)$ defines a sequence of objects where each object is represented by a substring $s_i s_{i+1}$ being a sub-block of length two of the string $S(\varepsilon)$.

The parameter ε is a real number taken from the interval $[0.0, 1.0]$ (in this case) which defines neutral fitness and determines the accuracy with which the string $S(\varepsilon)$ is defined. If the absolute fitness difference between neighbouring points is less than ε , the points are considered to be of equal fitness. This means that as ε increases from 0.0 to the maximum possible fitness difference between points along the walk (1.0), the amount of fitness change (entropy), and the sensitivity of ψ_{f_i} , decrease to zero.

The **information content** is defined as the entropic measure of the group of sub-blocks of length two of string $S(\varepsilon)$, and is given by

$$H(\varepsilon) = - \sum_{p \neq q} P_{[pq]} \log_6 P_{[pq]}$$

Equation 3 Information Content

$P_{[pq]}$ are frequencies of the possible blocks pq of elements from the set $\{\bar{1}, 0, 1\}$ given by

$$P_{[pq]} = \frac{n_{[pq]}}{n}$$

where $n_{[pq]}$ is the number of occurrences of pq in $S(\varepsilon)$ and n the length of $S(\varepsilon)$.

The **partial information content** is a measure of the modality (the number or frequency of local optima) of the landscape, and is calculated by filtering out elements of the string $S(\varepsilon)$ which are not essential for measuring modality to create a new string $S'(\varepsilon)$, then measuring the length μ of the new string. The string $S'(\varepsilon)$ is defined as

$$S'(\varepsilon) = s_{i_1} s_{i_2} \dots s_{i_j}, s_{i_j} \neq 0, s_{i_j} \neq s_{i_{j-1}}, j > 1$$

Thus the string $S'(\varepsilon)$ has the form " $\bar{1}1\bar{1}\dots$ ", representing the slopes of the path taken by the random walk over the landscape, and so is empty if $S(\varepsilon)$ contains only 0s. The partial information content is given by

$$M(\varepsilon) = \frac{\mu}{n}$$

Equation 4 Partial Information Content

Note that when $M(\varepsilon)$ is 1, the path taken by the random walk over the landscape is considered to be maximally multimodal, and when $M(\varepsilon)$ is 0, the path is flat.

The **information stability** ε^* is defined as the smallest value of ε for which the landscape becomes flat (i.e. for which $S'(\varepsilon)$ is empty). Since ε governs the sensitivity of the information content and partial information content measures, ε^* is a measure of the difference in fitness between neighbouring points on the random walk.

2.4.2 Isotropy

A body is said to be isotropic at a point if each rotation about that point brings the body into an indistinguishable position, and anisotropic otherwise. So for an isotropic body, all directions at

a point of examination are indiscernible. Furthermore, a body for which a difference between directions cannot be discerned *by available means* is said to be *statistically isotropic* irrespective of any fine structure anisotropy. For any landscape measures to be generalised to the entire landscape, the landscape over which the measures are taken must be statistically isotropic. That is, for a random walk over part of a landscape to be representative of the entire landscape, and so for the measures of the random walk to be regarded as representative of the entire landscape, the landscape must look the same everywhere [124, 262].

2.4.3 The State of Fitness Landscape Research

The major area of concern with fitness landscapes is that there is no generally accepted definition of what constitutes a fitness landscape. There is not much agreement in the field as to what a fitness landscape is and whether a neighbourhood relation is required to describe it, and much less agreement as to what the neighbourhood relation should be. Jones did make a big impact and his “one operator one landscape” approach does have a core following. However, work continues to try to present a coherent, consistent view of fitness landscapes and the neighbourhood relations that define them. Related work includes research into information landscapes and performance landscapes, and various landscape measures continue to be developed even in the absence of a consensus of opinion on the definition of fitness landscapes.

2.5 Evolutionary Robotics

Evolutionary robotics is the field of applying computational models of evolution to the design and control of robots [93-95, 129, 170, 201]. While this thesis is primarily concerned with robots simulated in software, some of the techniques applied to physical robots and the lessons learned are of interest. A review of research specifically related to evolutionary algorithms applied to simulated robot soccer appears in the following section.

The idea of using evolutionary algorithms to aid or improve the design of robot control systems dates back to the late 1980s, with the first use of the term *evolutionary robotics* generally attributed to Cliff, Harvey and Husbands in the early 1990s [50, 107, 108]. Experiments by Cliff, Harvey and Husbands, and Floreano and Mondada [75] led to a heightened interest in the application of evolutionary algorithms to the design and control of

robots. Since then evolutionary algorithms have been applied to various aspects of physical robotics such as control, vehicle navigation and routing, vision, movement etc. (e.g. [125, 128, 161, 179, 202, 238, 255, 288]). The cost and complexities involved in the engineering of physical robots have shifted attention somewhat towards the investigation of the theoretical aspects of evolutionary robotics using software robots rather than their physical counterparts.

Software robots are commonly implemented and referred to as *software agents*. A software agent is a piece of autonomous, or semi-autonomous computer software that operates either reactively or proactively without continuous direct supervision or control. A player in the context of simulated robot soccer can be described as a software agent.

There is a large body of work in the area of intelligent autonomous agents, but in recent times some researchers have moved away from modelling intelligent behaviour by designing and implementing complex agents. While the traditional single, complex agent approach in which the agent's behaviours are governed by a chain of interrelated and interdependent processing modules (for example, Figure 22 on page 86) has been shown to be successful in specialized domains such as game playing, reasoning, and path planning [197], other approaches need to be considered. One such approach is the simple, reactive agent approach [33, 34] which replaces the notion of a central locus of control with independent, reactive modules (or layers) that govern the agent's behaviour (for example, Figure 23 on page 87). With this approach a group of simple, reactive agents co-operate to achieve some objective. Several variations of this approach have been investigated by different researchers, from *on-the-fly* co-operation amongst a group of agents [278] to the complexity that arises from the interaction between agents and their environment [18]. The simple reactive agent approach would seem to be a reasonable one, and one for which the machine learning techniques described may work well. In the work presented in this thesis, the focus is on using those techniques to create a simple reactive player which exhibits reasonable goal-scoring behaviour, rather than a complex, quasi-intelligent software robot.

2.6 Robot Soccer

In 1992 Alan Mackworth, from the University of British Columbia in Canada, asserted that traditional “good old fashioned” artificial intelligence (AI) was deficient, and that the basic assumptions underlying problem solving with computer agents were limiting progress in the

field [181]. Mackworth held that the main assumptions of traditional AI had always been that [181], p.2:

- *All that is useful for an agent can be described in terms of individuals and relations amongst individuals.*
- *An agent's beliefs about the world are true and justified.*
- *An agent's knowledge of the world is definite and positive.*
- *The agent's knowledge of the world is complete.*
- *The environment is static unless an agent changes it.*
- *There is only one agent in the world.*
- *Given a complete and definite description of the world the agent can predict all the effects of an action.*
- *Actions are discrete and they are carried out sequentially.*

While these assumptions may hold for some, mostly artificial, situations, the complexity, dynamics, and uncertainty of real world situations causes many, if not all, of the assumptions listed above to fail. Mackworth proposed “*To realize the force of these assumptions let us consider a world in which they are all violated. Suppose we want to build a robot to play soccer*” [181], p.3. Such a robot, by virtue of the fact that it is situated in a world in which all the traditional assumptions of AI are violated, cannot assume that all its beliefs are true and justified; its knowledge of its world (i.e. the soccer game) will be indefinite and incomplete – the robot will not be able to assume something is false just because it is not known to be true. The robot's environment will contain other robots (players), and they will interact with each other and the environment, changing their environment in unpredictable ways. The game will be played with multiple players (robots) taking (unpredictable) actions concurrently; each modifying the environment as a result. The robot's world will therefore not be deterministic – nothing can be known, or predicted, with precision. In short, robot soccer as proposed by Mackworth is an ideal platform for AI research: it provides a complex, dynamic and unpredictable, yet standardised and reproducible environment.

Also in 1992, and independently of Mackworth, a group of Japanese researchers organised a Workshop on Grand Challenges in Artificial Intelligence. At this workshop, held in Tokyo in October of 1992, the possibility of using the game of soccer for promoting science and technology was discussed [154]. Subsequently a series of investigations were carried out to

determine the feasibility of the suggestion from technological, social and financial standpoints. The proposition was considered feasible and desirable, prototype development of soccer robots and simulator systems were undertaken, and rules of the competition were drafted. In 1993 the Robot J-League, a robot soccer competition, was launched. The league was later extended as an international joint project and renamed the Robot World Cup Initiative, or RoboCup [151].

2.6.1 RoboCup

The Robot World Cup Initiative, RoboCup, has become an international research and education initiative, providing a standard platform and benchmark problem for research in the fields of artificial intelligence and robotics. It provides a realistic research environment by using a soccer game as a platform for a wide range of research problems including autonomous agent design, multi-agent collaboration, real-time reasoning, reactive behaviour and intelligent robot control [152, 153, 155].

RoboCup currently consists of three major domains: RoboCupSoccer, RoboCupRescue and RoboCupJunior. The RoboCupSoccer domain includes a simulation league and is the environment used for the RoboCup part of this work.

2.6.1.1 The RoboCupSoccer Simulation League

The RoboCupSoccer Simulation League provides a simulated but realistic soccer environment which obviates the need for robot hardware and its associated difficulties, allowing researchers to focus on issues such as autonomous agent design, learning, planning, real-time multi-agent reasoning, teamwork and collaboration. The RoboCupSoccer simulator has been in continual development since its inception in 1995, and allows researchers to study many aspects of machine learning techniques and multi-agent systems in a complex, dynamic domain [198-200]. The RoboCupSoccer environment is described in detail in Chapter 3 of this thesis.

2.6.2 The SimpleSoccer Environment

The RoboCupSoccer simulation league is an important and useful tool for multi-agent and machine learning research which provides a distributed, multi-agent environment in which agents have an incomplete and uncertain world view. The RoboCupSoccer state-space is extremely large, and the agent perception and action cycles in the RoboCupSoccer environment

are asynchronous, sometimes resulting in long and unpredictable delays in the completion of actions in response to some stimuli. The large state-space, the inherent delays, and the uncertain and incomplete world view of the agents can increase the learning cycle of some machine learning techniques onerously.

There is a large body of work in the area of the application of machine learning techniques to the challenges of RoboCupSoccer (e.g. [47-49, 172, 174, 175, 177, 219, 220, 247, 250, 259]), but because the RoboCupSoccer environment is so large, complex and unpredictable, the extent to which such techniques can meet these challenges is not certain. The SimpleSoccer environment was designed and developed to address the problem of the complexity of the RoboCupSoccer environment inhibiting further research, and is described in detail in Chapter 3 of this thesis.

2.6.3 Evolutionary Algorithms and Simulated Robot Soccer

Evolutionary algorithms in various forms have been applied to the problem of robot soccer. Some representative examples are described briefly in the following paragraphs.

Two early attempts to learn competent soccer playing skills from scratch via genetic programming are described in [177] and [7]. Both set out to create complete, cooperative soccer playing teams, but neither achieved that objective. The objective in [177] was scaled back to attempt to evolve co-operative behaviour over hand-coded low-level behaviours. The players in [7] developed some successful individual behaviours with the use of a sophisticated composite fitness function, but the objective of collaborative team behaviour was not realised.

In [175] a team of soccer players with a rich set of innate soccer-playing skills was developed, using genetic programming and co-evolution, that worked through sub-optimal behaviour described as “kiddie-soccer” (where all players chase the ball) to reasonable goal-scoring and defensive behaviour.

A layered learning technique was introduced in [246] and [251], the essential principle of which is to provide the algorithm with a bottom-up hierarchical decomposition of a large task into smaller sub-tasks, or layers, and have the algorithm learn each sub-task separately and feed the output of one learned sub-task into the next layer.

The layered learning technique is based upon the following four principles [251]:

- A mapping directly from inputs to outputs is not tractably learnable.
- A bottom-up, hierarchical task decomposition is given.
- Machine learning exploits data to train and/or adapt. Learning occurs separately at each level.
- The output of learning in one layer feeds into the next.

Stone and Veloso used the layered learning technique to produce good results when training robot soccer players for RoboCupSoccer [251].

Keepaway Soccer [249] is a sub-domain of robot soccer in which the objective is not to score goals but to gain and maintain possession of the ball. There are two teams in keepaway soccer: the *keepers* and the *takers*. The task of the keepers is to maintain possession of the ball, while the objective of the takers is to take the ball away from the keepers. The keepaway soccer field is generally smaller than the RoboCupSoccer field, and no goal areas are required. The keepaway soccer teams are usually smaller than a full team in RoboCupSoccer, and are often numerically unbalanced (e.g. 3 vs 2 Keepaway Soccer [160]).

Gustafson [101] and Gustafson and Hsu [102, 127] applied genetic programming and the layered learning technique of Stone and Veloso to keepaway soccer. For this method the problem was decomposed into smaller sub-problems, and genetic programming applied to the sub-problems sequentially - the population in the last generation of a sub-problem was used as the initial population of the next sub-problem. The results presented by Gustafson and Hsu indicate that for the problem studied layered learning in genetic programming outperformed the standard genetic programming method.

Di Pietro et al. [71] reported some success using a genetic algorithm to train 3 keepers against 2 takers for keepaway soccer in the RoboCup soccer simulator. Players were endowed with a set of high-level skills, and the focus was on learning strategies for keepers in possession of the ball.

Three different approaches to create RoboCup players using genetic programming are described in [48] – the approaches differing in the level of innate skill the players have. In the initial experiment described, the players were given no innate skills beyond the actions provided by the RoboCupSoccer server. The third experiment was a variation of the first experiment. Ciesielski et al. reported that the players from the first and third experiments –

players with no innate skills - performed poorly. In the second experiment described, players were given some innate higher-level hand-coded skills such as the ability to kick the ball toward the goal, or to pass to the closest teammate. The players from the second experiment – players with some innate hand-coded skills – performed a little more adequately than the other experiments described. Ciesielski et al. concluded that the robot soccer problem is a very difficult problem for evolutionary algorithms and that a significant amount of work is still needed for the development of higher-level functions and appropriate fitness measures.

Using keepaway soccer as a machine learning testbed, Whiteson and Stone [274] used neuro-evolution to train keepers in the Teambots domain [17]. In that work the players were able to learn several conceptually different tasks from basic skills to higher-level reasoning using “concurrent layered learning” – a method in which predefined tasks are learned incrementally with the use of a composite fitness function. The player uses a hand-coded decision tree to make decisions, with the leaves of the tree being the learned skills.

Whiteson et al. [272, 273] study three different methods for learning the sub-tasks of a decomposed task in order to examine the impact of injecting human expert knowledge into the algorithm with respect to the trade-off between:

- making an otherwise unlearnable task learnable
- the expert knowledge constraining the hypothesis space
- the effort required to inject the human knowledge.

Coevolution, layered learning, and concurrent layered learning are applied to two versions of keepaway soccer that differ in the difficulty of learning. Whiteson et al. conclude that given a suitable task decomposition an evolutionary-based algorithm (in this case neuroevolution) can master difficult tasks. They also conclude, somewhat unsurprisingly, that the appropriate level of human expert knowledge injected and therefore the level of constraint depends critically on the difficulty of the problem.

Castillo et al. [38] modified an existing RoboCupSoccer team – the 11Monkeys team [148] – replacing its offensive hand-coded, state dependent rules with an XCS genetic classifier system. Each rule was translated into a genetic classifier, and then each classifier evolved in real time. Castillo et al. reported that their XCS classifier system outperformed the original 11Monkeys team, though did not perform quite so well against other, more recently developed, teams.

In [196] Nakashima et al. describe a method for learning certain strategies in the RoboCupSoccer environment, and report some limited success. The method uses an evolutionary algorithm similar to evolution strategies, and implements mutation as the only evolutionary operator. The player uses the learned strategies to decide which of several hand-coded actions will be taken. The strategies learned are applicable only when the player is in possession of the ball.

Bajurnow and Ciesielski used the SimpleSoccer environment to examine genetic programming and layered learning for the robot soccer problem [13]. Bajurnow and Ciesielski concluded that layered learning is able to evolve goal-scoring behaviour comparable to standard genetic programs more reliably and in a shorter time, but the quality of solutions found by layered learning did not exceed those found using standard genetic programming. Furthermore, Bajurnow and Ciesielski claim that layered learning in this fashion requires a *“large amount of domain specific knowledge and programmer effort to engineer an appropriate layer and the effort required is not justified for a problem of this scale.”* [13], p.7.

Other examples of research in this or related areas can be found in, for example, [178] where breeding and co-ordination strategies were studied for evolving teams in a simple predator/prey environment; [160, 247, 248] where reinforcement learning was used to train players in the keepaway soccer environment; [165] in which genetic programming was used in a specific training environment to evolve goal-keeping behaviour for RoboCupSoccer; [9] where genetic programming was used to develop a team of players for RoboCupSoccer; [127] in which the incremental reuse of intermediate solutions for genetic programming in the keepaway soccer environment is studied.

2.7 Work Using Evolutionary Algorithms to Solve Monolithic Problems

Presented in the following sections are analyses of four major or recent works, some of which have generated interest in the research community as implementations in which evolutionary algorithms have successfully solved large, complex or monolithic problems from scratch or with no human help. As defined earlier in this thesis, a monolithic problem is one for which the fitness function or reward does not directly map to a solution. The works reviewed in the following sections are directly related to the second and third research questions being addressed by this thesis – all four use some combination of a rich initial skillset and a

composite fitness function to guide the evolutionary search (research question 2), and the analysis of why the initial skills or the composite fitness function is needed helps to shed some light on why this problem is a difficult one for evolutionary algorithms (research question 3). Following the analyses a brief summary is presented.

2.7.1 Evolving XPilots

In [204] and [205] Parker et al. use a genetic algorithm to evolve autonomous control and survival strategies for the internet game of XPilot [240, 241]. In [204] a genetic algorithm is used to evolve the consequents for a set of rules to control a robot in the XPilot game. Expert domain knowledge is injected into the learning process by defining the fixed number of rules, and by predefining the rule antecedents – only the consequents are learned via the genetic algorithm. In [205] the weights for a neural network are evolved to control a robot in the XPilot game. The neural network has a fixed architecture with 22 inputs, 3 outputs and no hidden layer. Expert domain knowledge is injected into the learning process by defining the meaning of the neural network inputs and providing sophisticated mechanisms for discovering the values to be input. For example, the inputs include the distance and direction to the “most dangerous bullet”, and the change in those values – requiring expert knowledge not only to determine which is the “most dangerous bullet”, but also to “know” that these are important inputs to the control of the robot. In addition, as noted by the Parker et al. “*Many attempts were made to find a near-optimal fitness function.*” [205], p.4, thus using expert knowledge to further guide the evolutionary search.

2.7.2 Brainstormers 2D: Winners, RoboCup 2D-Simulation League 2005

The winner of the 2005 RoboCup 2D-Simulation League⁴ was the Brainstormers 2D team from the Neuroinformatics Group at the University Of Osnabrück [219]. The Brainstormers 2D team uses reinforcement learning to train neural networks some basic soccer-playing skills, and then to learn the overall team strategies. Not all of the player’s skills are learned this way – some are implemented as semi-complex hand-coded behaviours. The architecture of the player is quasi-hierarchical, with a skills ranging from very simple to very complex. Since

⁴ The RoboCup Simulation League was originally implemented as a 2-dimensional simulation; recently a 3-dimensional version was implemented which adds more complexity to the model.

the team managed to win the 2005 competition it is a good quality solution which outperformed many other competitors.

The basic skills learned by the neural networks include ball interception, navigation to a specific field position, kicking the ball with desired velocity and direction, dribbling, passing, and scoring. Interestingly Riedmiller et al. note that while it is scientifically appealing to develop machine learning techniques to realise player behaviours, the imperatives of competition necessitate using all available means to search for nearly optimal, highly competitive solutions. As a consequence Riedmiller et al. developed an analytic, hand-coded method for intercepting the ball which was found to outperform their own reinforcement learning-developed solution by about 8.5% (as measured by the average number of steps to intercept the ball) [219].

Motivated by this result Riedmiller et al. revisited the reinforcement learning approach for ball interception and determined some areas with high potential for improvement, and managed to supersede the previously learned solution by about 6.0%. While the new method does not equal the quality of the hand-coded solution, it is close. Riedmiller et al. also suggest the result supports the following arguments [219]:

- using reinforcement learning, it is not too difficult to yield good, or even high quality behaviour, but
- in many cases, particularly when tackling real world problems of high complexity, significant effort is necessary in order to obtain learned solutions that come near to the optimal solution, hence which, in the case of robot soccer, are competitive.

The focus of the Brainstormers 2D team is now on learning team strategies – including goal-scoring behaviour (as a group) – as well as improving the quality of the learned basic behaviours in an effort to equal or surpass the hand-coded skills [219].

The success of the Brainstormers 2D team at the 2005 RoboCup 2D-Simulation League competition is of interest for the following reasons:

- A team using some skills and strategies learned via a machine learning technique outperformed other teams which used varying levels and combinations of AI techniques and hand-coding to define their players and strategies.

- The team developers found that hand-coded skills outperformed learned skills – possibly indicating that the success in the competition was at least in part reflecting the quality of the hand-coded skills of the Brainstormers 2D team.

2.7.3 GP Produced Competitive Soccer Teams for RoboCup97

In 1998 Luke reported that he had used genetic programming (GP) to evolve a team of simulated robot soccer players that were competitive with teams hand-coded by humans in the RoboCup97 trials [175]. The team won its first two games against hand-crafted teams, an achievement significant enough for Luke to be presented with the RoboCup Scientific Challenge Award for demonstrating the utility of the genetic programming approach for developing simulated soccer players.

The technique used is similar to the Pittsburgh approach for genetic algorithms: the GP genome was an entire team, so a team of players was evolved rather than individual players [175, 178]. The standard GP genome was modified to allow each player to be represented by two trees rather than just a single tree, where one tree was responsible for generating *kick* behaviour and the other *move* behaviour. Each tree was evolved separately - there was no crossover of genetic material between *kick* and *move* trees.

An additional layer of software was developed which simplified the RoboCupSoccer environment, making the domain less complex and more orthogonal and eliminating many of the boundary conditions the players would otherwise encounter.

Actions were implemented in an hierarchical fashion, with a default action defined. During play the program executing the player's moves followed the instructions of one, and only one, of the *kick* and *move* trees, then acted based on the following rules:

- if the player could see the ball and was close enough to kick it the *kick* tree was used. The player would kick the ball as directed by the *kick* tree, then turn in the direction the ball was kicked.
- if the player could see the ball but was not close enough to kick it the *move* tree was used. The player would turn and dash as directed by the *move* tree, or dash in reverse if by doing so the ball would be kept in sight.
- if the player could not see the ball it would turn in the direction it last turned until the ball was in sight.

A set of hand-coded functions was developed and made available to the GP to utilise in evolving the kick and move trees⁵. Luke describes these functions as “low-level ‘basic’ behaviour functions” [175], but there is not much that is low-level or basic about most of them. The functions range from providing simple integer constants to a complex function that calculates a vector to the closest point on the line segment between the ball and the nearest known opponent – somewhat analogous to ball interception, which Luke describes as “a surprisingly complex task” [175, 245].

Teams were evolved in a competitive environment where an individual team’s fitness evaluation was based on competition with other members of the population [8]. To minimise the impact of the fitness evaluation process on the performance of the algorithm, teams were paired at random with members of the population and the fitness assessment was based on the result of a single game of soccer between the two teams. The number of games played per generation for fitness evaluation was therefore equal to half the size of the population.

Fitness evaluation was initially based on a number of factors, such as the number of goals, time in possession of the ball, average position of the ball, number of successful passes, etc., but it was found in early tests that the entire population would converge to very poor solutions. The fitness assessment was changed to goal difference alone and the population avoided such convergence. It was discovered that initial game scores were very high and quite variable, hence the success of the coarse fitness function. Luke concludes that this was in part due to vectors to the ball and to the goal being fundamental parts of the function set, allowing teams to perform well offensively.

Experiments were performed to evolve both homogeneous and heterogeneous teams. The genome for an homogeneous team is a single pair of *kick* and *move* trees, and each player in the team uses the same *kick-move* tree pair during play. By contrast, the genome for an heterogeneous team consists of multiple *kick-move* tree pairs (one for each player in the team), and each player therefore uses a different *kick-move* tree pair during play. Luke found that the homogeneous teams outperformed the heterogeneous teams. This is a surprising result since the environment is one in which heterogeneity could be expected to be advantageous.

⁵ see Table 27 in Appendix for a complete description of the functions implemented.

In summary, though a team was evolved that was competitive with teams hand-coded by humans and that was indeed a significant achievement, in order to evolve that team players were afforded the benefit of human expert knowledge and experience by being given:

- the absolute positions of themselves and all visible players, the ball and the goals (via the additional layer of simplifying software).
- a semi-intelligent default action - turn until the ball is located. Luke explains the significance of this default action: “... *by doing everything reasonable to keep the ball in view, we were able to eliminate many of the boundary conditions which occur when ball suddenly disappears due to arbitrary player movement (a big problem in our early tests).*” [175], p. 3.
- a set of high-level innate skills in the form of human-developed, hand-coded function. Luke refers to the significance of this: “*Ideally, we would have liked to produce soccer players out of a variety of very low-level, generic vector functions. This would have allowed us to say that in no way did we bias the function set to produce certain kinds of strategies. But our early tests suggested that the domain was so complex that there was little hope of evolving a team with this kind of function set. Instead, we designed a large set of functions (some generic, some specialized) we thought would have particular utility in the soccer domain.*” [175], p.3.

The real significance in the rich set of functions given to the players is not simply in the innate skills that they were endowed with as a result, but in the knowledge and understanding that those skills and actions would be required. Not only did these players not need to learn *how* to perform those skills, they did not need to learn *what skills* would be useful in order to play soccer competitively. This is an extremely important and significant point in relation to *learning from scratch*: often learning from scratch isn’t really learning from scratch, but learning from a pre-existing skillset, either hand-coded or previously learned.

2.7.4 Blondie24: Learning to Play Checkers without Relying on Expert Knowledge

While not robot soccer, Blondie24, developed by Chellapilla and Fogel, is an example of an evolutionary algorithm used to solve what would seem to be a large, monolithic problem:

learning to play checkers⁶ from scratch [40-44, 76, 77]. Blondie24 is a checkers player which learned its strategy using a co-evolutionary approach in which a genetic algorithm trained a neural network to assess board positions for checkers. Blondie24 was not provided with any overt domain knowledge, and was rewarded (or penalised as appropriate) based only on whether it won, lost, or drew games. Blondie24 was pitted against human players and managed to perform competitively. The various claims which have been made with regard to the performance of Blondie24 are encapsulated in the titles of ChellaPilla and Fogel's papers: "*Evolving a Neural Network to Play Checkers without Human Expertise*", "*Evolving an Expert Checkers Playing Program without Using Human Expertise*", "*Evolving Neural Networks to Play Checkers without Relying on Expert Knowledge*", and "*Co-Evolving Checkers Playing Programs using only Win, Lose or Draw*". The conclusion drawn by Chellapilla and Fogel is that Blondie24, using a co-evolutionary genetic algorithm and with no expert knowledge, learned to play checkers at an expert level from scratch. The implication is that Blondie24 has achieved some sort of "true" artificial intelligence ("*Did Fogel create artificial intelligence? I think the answer again is yes.*" [106], p.27) and met the *Samuel-Newell Challenge* [76] of a program learning the game of checkers from scratch without using human expertise and using nothing more than win/lose feedback.

In the context of the current state of evolutionary computation the results reported by Chellapilla and Fogel with regard to Blondie24 are a significant achievement, but the conclusions drawn are not sustainable. Following is a brief analysis of the method and commentary on its significance and conclusions drawn.

The method uses a genetic algorithm and a co-evolutionary approach to evolve the connection weights and biases for a population of fixed architecture neural networks. A population of 15 neural networks is constructed, each initialised with random weights. The neural network's task is to assess any board position presented to it.

Input to the network is a vector representing the board position. The vector consists of 33 values, representing the 32 playable squares on the board plus a single value which is the sum of the 32 values representing the squares. Each of the 32 inputs representing the squares is assigned a value from the set $\{K, 1, 0, -1, -K\}$ depending upon what piece occupies the corresponding square, where K represents a king, 1 a regular piece, and 0 an empty square.

⁶ A board game known as "checkers" in the U.S.A and "draughts" many other places in the world. For information see <http://www.acfcheckers.com/>. For consistency the nomenclature used by Chellapilla and Fogel in their publications is adopted in this thesis.

The sign, positive or negative, for the kings and pieces denotes opposing teams. The value of K is initialized to 2.0, but is evolved along with the network connection weights and biases.

Output from the network is a scalar representing how “good” the board position is – taken as an indication of the quality of potential future positions – with 1.0 indicating a “good” position, -1.0 a “bad” position, and somewhere in between being a relative judgement somewhere between “good” and “bad”.

A standard gaming algorithm - a fail-soft alpha-beta search [141] of the game tree - is used to generate moves into the future (with depth 4, or more if forced moves are involved). Each board position is then presented to each network in the population for evaluation and the most advantageous move chosen. After all the networks in the population are evaluated a number of the "best" are chosen for reproduction. Each member of the population's fitness is evaluated by having it play one game against each of 5 randomly selected members of the population (with replacement). The fitness function implemented is not complex: a network is assigned a fitness value f according to the function:

$$f = \begin{cases} -2 & \text{if } \text{loss} \\ 0 & \text{if } \text{draw} \\ +1 & \text{if } \text{win} \end{cases}$$

where a draw is called arbitrarily after 100 moves if there is no other result. The evolutionary process is allowed to run for 250 generations, and the best performing network after 250 generations is then evaluated against human players.

A first glance would seem to indicate that Blondie24 is being endowed with no innate skills and the fitness function is simple, but upon further consideration it is apparent that this is not so. Despite ChellaPilla and Fogel's claim that they are really not giving the networks any extra information by including the sum of the board position, they are. ChellaPilla and Fogel state:

“The only feature that could be claimed to have been offered is a function of the piece differential between a player and its opponent, owing to the sum of the inputs being supplied directly to the output node. The output essentially sums all the inputs which in

turn offers the piece advantage or disadvantage. But this is not true in general, for when kings are present on the board, the value K or $-K$ is used in the summation, and as described below, this value was evolvable rather than prescribed by the programmers a priori. Thus the evolutionary algorithm had the potential to override the piece differential and invent a new feature in its place.” [41], p.10.

This raises two points:

1. The existence of the potential to override an advantageous input feature does not mitigate the advantage that the feature confers, and certainly doesn't guarantee it being overridden - why would the algorithm invent a new feature in place of an advantageous one already supplied?
2. The value of K and the fact that it is evolvable is only relevant after a king has been crowned - until that time the summed input is just the piece differential between players and possibly a useful input to the network. Furthermore, by the time a king is crowned one of the players may have already gained a game winning advantage. Moreover, even after K becomes evolvable its value is constrained to lie in $[1.0, 3.0]$, so the least information provided by the extra input (when $K = 1.0$) is the simple piece differential, and if $K \neq 1.0$ then the input is also a measure of the difference in the number of kings each player has had crowned. Regardless of the value of K the extra input is always a measure of the relative strengths of the opposing sides.

ChellaPilla and Fogel explain why they believe that this extra input is not really a useful feature being offered to the network beyond just the state of the board, but its very existence raises the question of the reason for its inclusion as an input. Equally importantly, what happens if it is removed? A comparison of tests conducted with and without the extra feature would be useful, perhaps instructive.

The extra input to the network is an example of *human expertise or expert knowledge* used to help the network learn the task of assessing the board position. It is not just a matter of the summed input being supplied – in essence that is nothing that the network could not have done for itself from the other inputs supplied. But there is a great deal of expert knowledge and experience in *knowing* first that the summation of the inputs is an indication of the relative

strengths of the opposing sides, and second that this information could be useful to learning how to assess the board position.

The fitness function, though not very complex, is a composite fitness function since it rewards networks on a sliding scale for different quality solutions. This has the effect of introducing gradient information to the fitness landscape that would not be present if the fitness assigned was, for example, simply a count of the number of games won. Furthermore, the scale of the reward (-2 for a loss, 0 for a draw, and $+1$ for a win) is skewed against losing, which “*reflected a desire to avoid networks that would tolerate losing*” [41], p.22. This is again injecting expert knowledge and experience into the method – endowing the methods with the *innate knowledge* that losing should be avoided means that it does not have to be learned. Chellapilla and Fogel themselves acknowledge that “*It would be interesting to observe any difference in performance that would result from setting the payoff to be zero sum.*” [41], p.22.

The achievement of Chellapilla and Fogel with regard to Blondie24 is related to the work done for this thesis to evolve goal-scoring behaviour, and the conjecture that an evolutionary quantum exists which affects the ability of evolutionary algorithms to solve large, monolithic problems. In the context of robot soccer, learning goal-scoring behaviour requires that the players learn basic skills, such as finding the ball, kicking the ball, and kicking the ball towards the goal, from which they can build the more complex skill of scoring goals. Learning goal-scoring behaviour *from scratch* means that the players must first learn that they need to learn basic skills and what they are, and only then can they proceed to learning those basic skills and assembling them into more complex behaviours. Trying to learn that with no innate expert knowledge and a simple fitness function seems to be difficult for evolutionary algorithms.

Winning at checkers on the other hand, in the context of Blondie24, while a difficult problem seems to require only one skill - determining the “quality” of a checkers board position. Blondie24 is not learning that in order to win it needs to jump opponents’ pieces – that is innate knowledge existing in the gaming algorithm that determines the possible moves. Blondie24 is not even learning that it can only move a piece diagonally - again, that is innate knowledge. The fitness function implemented for Blondie24 is not complex, but neither is it a simple binary function – the feedback is varied according to the result and in fact skewed to avoid a particular result, so encapsulates expert domain knowledge.

Blondie24 is endowed with some innate, expert domain knowledge and thus is not learning to play checkers from scratch. What it really learns, by using a genetic algorithm to train a neural network, is how to recognise a good checkers board position from a less good one.

2.8 Discussion

All of the methods reviewed in sections 2.6 and 2.7 of the thesis use either high-level hand-coded skills, a composite fitness function which guides evolution, or some other method of injecting human expert knowledge - or a combination of all those - in order to elicit competent soccer skills from the players being evolved.

Techniques have been developed to overcome a perceived limitation of artificial evolution to solve difficult problems. Stone and Veloso's layered learning [246, 251] is one such technique:

"The basic idea behind layered learning is that some problems are so complex that they cannot be learned directly from the inputs provided by the system. But if the problem is decomposed into several layers of behaviors, each behavior can be learned and used in the learning of other behaviors. In this way, a complex problem can be solved." [101], p.15.

Another technique, known as *Incremental Evolution*, was introduced particularly for problems where the percentage of the search space that constitutes a solution is very small and the fitness landscape very rugged [91]. In such problems the probability of producing individuals in the initial random population that are viable material for solutions is low and evolution is unlikely to make progress because the population gets trapped in suboptimal regions of the fitness landscape during the early stages of evolution. Incremental evolution attempts to address this issue by decomposing the task into a family of sub-tasks, starting with a less difficult task and with each successive task becoming more difficult (Figure 17).

As the sub-tasks become more difficult the solution set becomes smaller, but because the evolutionary algorithm is being directed to the area of the search space containing the solution set by solving the preceding, simpler tasks the problem difficulty is reduced. The success of incremental evolution is heavily dependent upon the generation of the sub-tasks and the ability of their solutions to position the population in a fruitful region of the search space for solving the next sub-task. Sub-tasks are not generated automatically, but are predetermined by the experiment designer using expert knowledge and experience.

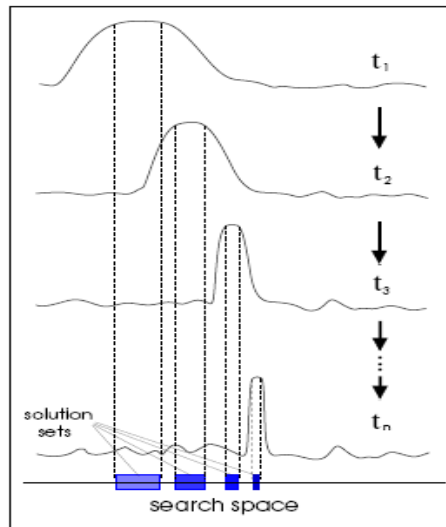


Figure 17 Incremental fitness landscapes

The figure illustrates, for a 1-dimensional search space, how incremental evolution works by gradually reshaping the fitness landscape to guide the population toward a solution to the goal task. The initial task, t_1 , provides an easy target for evolution, which positions the population in the correct region to approach t_2 . Successive tasks do the same until the goal task t_n is reached. (Reproduced from [91])

A technique often used to overcome problems when using evolutionary algorithms to solve difficult problems is to “tune” the algorithm parameters to better suit the nature of the problem. In [223] Rothlauf et al. describe the technique of configuring a genetic algorithm specifically for a particular problem in the following way:

“Users are often confronted with the situation that a GA does a good job for a small or easy problem, but when scaled up to bigger or more complicated problems, the traditional GA degrades or even breaks down. The user is frustrated and starts knob twiddling until he gets good solutions (or not! ... The process of matching the encoding and the GA often draws the line between failure or success of optimisation.” [223], p.2.

De Jong and Pollack note that *“Designing an adequate fitness function requires substantial knowledge of a problem and of features that indicate progress towards a solution.”* [64], p.1. The implication is that the fitness function should guide the evolutionary search by indicating incremental progress towards a solution. This is further confirmation that, perhaps at times unintentionally, many researchers are using their own expert knowledge to assist the evolutionary algorithms to find solutions for difficult problems. This conjecture is further

explored in [1] where Abbott et al. claim that genetic programming is inherently flawed because by specifying the fitness function “*the user must provide the GP engine with an evolutionary pathway toward a solution.*” [1], p.1. While the claim is perhaps a little strong, especially since there is no requirement that the fitness function for genetic programming or any other evolutionary algorithm be implemented as a composite function that indicates incremental progress towards a solution, the underlying notion that the evolutionary process can be guided by a composite fitness function is sound.

It is clear from the review of the literature presented in this chapter that the robot soccer problem is a very difficult problem and is often solved by injecting human expertise and expert knowledge. It is also clear that injecting expert knowledge is a technique often used when implementing evolutionary algorithms to solve other very difficult problems. The work conducted for this thesis seeks

- to solve a particularly difficult problem – that of evolving goal-scoring behaviour in the robot soccer environment, and
- to present a systematic analysis of the problem search space and fitness landscape in order to better understand why the problem is difficult and gain some insight as to whether difficult problems such as this can be solved by evolutionary algorithms without the need for injecting human expertise and expert knowledge.

*Le bon Dieu est dans le détail.
- Gustave Flaubert (attrib.)*

Chapter 3

Robot Soccer Simulators

3.1 Introduction

Two robot soccer simulators were used for the work conducted for this thesis. The RoboCupSoccer simulator is an existing simulator provided by the RoboCup project [45, 198]. The SimpleSoccer simulator was designed and created specifically for this work. This chapter describes the environment architecture and composition, player capabilities, commands and actions for both simulators. The additional mid-level hand-coded skills available to the players, and the player default actions, are also described in this chapter.

3.2 The RoboCupSoccer Simulator

The RoboCupSoccer simulator is implemented as a client-server environment, with the soccer server accepting network connections from client programs representing individual simulated soccer players. Once connected and registered, clients communicate with the server via a defined set of commands and responses, as well as periodically receiving certain visual and state information transmitted from the server to the client. The environment provided by the RoboCupSoccer simulator is comprised of

- the soccer field
- fixed landmarks, including the goals
- the ball
- a referee
- up to two teams with a maximum of eleven players each
- an optional coach for each team

In the RoboCupSoccer environment the playing field is represented as a 2-dimensional grid with the field, and surrounding area, containing fixed virtual landmarks to help players determine the location of the ball, goals, other players and themselves (Figure 18).

Players have a limited view of their surroundings, provided to them in the visual information received from the server. Small random perturbations are introduced by the soccer server to the visual information and certain state information provided by the server to each client (player), as well as to the actions of players and the motion of the ball after it is kicked. Due to the uncertain information provided by the server, and the unpredictable actions of the ball and other players on the field, players have an uncertain knowledge of the positions of landmarks, the ball, goals, and other players.

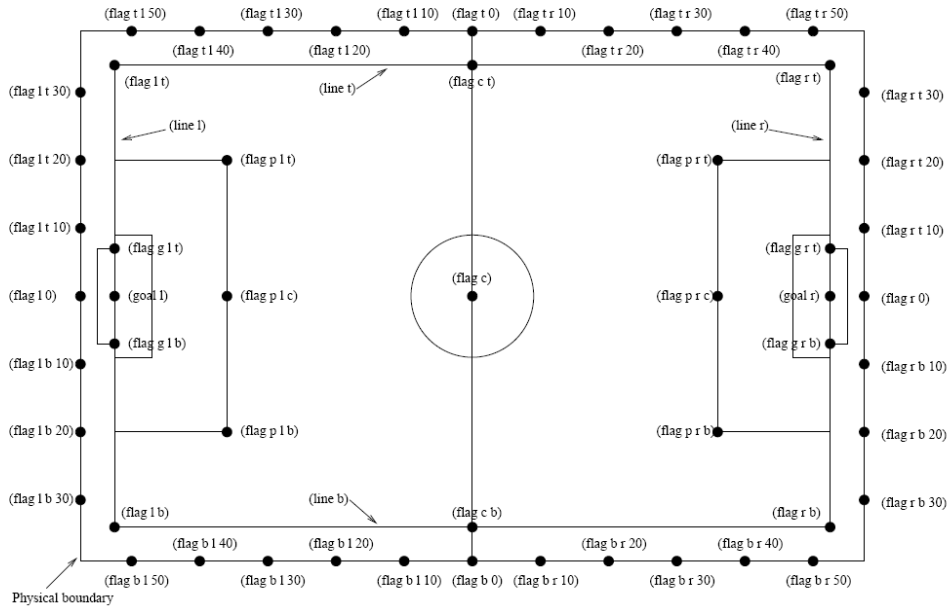


Figure 18 The soccer field and landmarks represented by the RoboCupSoccer server.
(Reproduced from [45])

The orientation of players on the field is determined by a player's grid coordinates and the direction in which it is facing. The RoboCupSoccer server records both a body angle and a neck angle for each player: a player may turn its neck independently from its body so that it may look in a direction different from that which its body is facing. Figure 19 shows the range of a player's vision – players can see objects in a cone-shaped area in the direction the player is facing. A player's viewing cone is determined by the view angle and visible distance, and only

objects within a player's viewing cone can be seen by the player. The black circles shown in Figure 19 represent objects on the field – only one is visible to the player in the diagram. The detail of the visual feedback delivered to a player by the RoboCupSoccer server depends on the player's viewing angle, which can be changed dynamically by the player. The amount of detail varies inversely with the viewing angle: a wide viewing allows the player to see a greater number of objects, but it sees them with less detail than if the viewing angle was narrow and fewer objects were seen. A player is also able to sense objects, including players, that are close, but not visible to the player. In such cases, only partial information about the object is made available to the player.

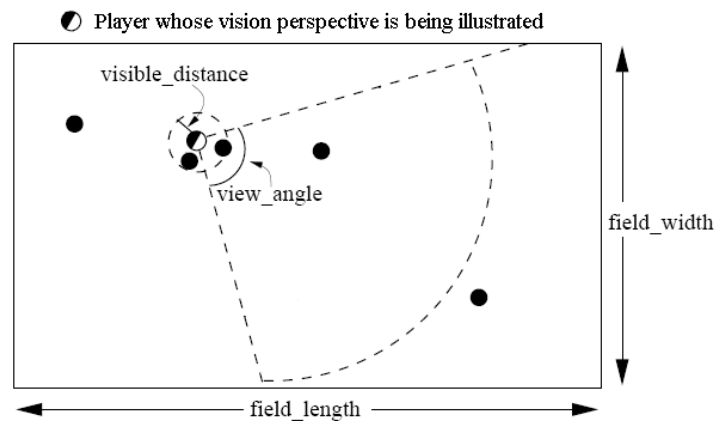


Figure 19 The visible range of a player in the RoboCupSoccer server.
(Adapted from [246])

The soccer server provides a set of commands to allow clients to control the actions of players – or more precisely, to allow clients to cause players to endeavour to perform some action. Players' limited knowledge, skills, capabilities and resources, the number of players, and the complex, dynamic nature of the environment and its inherent uncertainty all pose challenges for reasoning, collaboration and the execution of intelligent and coordinated actions. Most commands provided by the server require parameters for further qualification, so client programs (players) must determine values for parameters for each command according to their current situation and the desired action.

The subset of RoboCupSoccer commands used in this work are:

- *turn(direction)*: the player turns through the angle specified by *direction*.
- *dash(power)*: the player dashes in the direction it is facing with the power specified.
- *kick(power, direction)*: if the ball is within a kickable distance from the player, the player kicks the ball in the direction specified with the power specified.

For each of these commands:

- *direction* is specified in degrees (-180° to $+180^\circ$) relative to the direction the player is facing.
- *power* is specified as a percentage of maximum power and determines the distance the player or ball will travel as a result of the action.

The RoboCupSoccer environment provides players with three different sensors:

- the aural sensor detects messages sent by the referee, the coaches and the other players.
- the visual sensor detects visual information about the field, such as the distance and direction to objects in the player's current field of view. The visual sensor also works as a proximity sensor by "sensing" objects that are close, but behind the player. In order to introduce some "real world" uncertainty, random "noise" is introduced to the visual sensor information provided to the player – thus the information may be incomplete and/or uncertain.
- the body sensor detects the current status of the player, such as its stamina, speed and neck angle. The body sensor also delivers statistical information such as the number of times the player has kicked the ball, the number of times the player has dashed etc.

Together the sensors provide a reasonably detailed picture of the player's environment. The aural sensor is not used in this work, and the body sensor is used only for synchronisation with the server. The Soccer Server and RoboCupSoccer is described in detail in the RoboCup Soccer Server Users Manual [45, 198]. RoboCup Soccer Server version 7.07 was used as the base server for this work.

3.3 The SimpleSoccer Simulator

As previously discussed (see section 2.6.2 on page 58), there has been much work in the area of applying machine learning techniques to the challenges of robot soccer, but more progress could be made more quickly if the complexity and uncertainty inherent in the RoboCupSoccer environment could be reduced. The SimpleSoccer environment was designed, and is proposed, as an environment that reduces complexity and uncertainty sufficiently to increase the viability of machine learning techniques in the simulated robot soccer environment, yet retains sufficient complexity, uncertainty and dynamics to allow learning from SimpleSoccer to be directly transferable to the RoboCupSoccer environment.

The primary objective when creating the SimpleSoccer environment was to create an environment complex and dynamic enough that while low-level tactics may differ due to the removal of systematic uncertainty, high-level strategies directly applicable to the RoboCupSoccer environment could be developed, thus providing a simple yet sufficiently accurate model of the RoboCupSoccer environment that allows rapid evolutionary learning.

The design objective was achieved by modelling only the attributes of the RoboCupSoccer environment necessary to allow ball and player interaction with the provision of basic player actions, and by not modelling the client-server environment provided in RoboCupSoccer. The SimpleSoccer environment is comprised of :

- the soccer field
- fixed landmarks - the goals
- the ball
- up to two teams with a maximum of eleven players each

While the SimpleSoccer environment was inspired by simplicity of the Ascii Soccer environment [16], it is a more complex environment which more closely models the RoboCupSoccer environment. The soccer field in SimpleSoccer is represented by a two-dimensional grid with the goal markers being the only landmarks available to players (Figure 20 on page 80). Whereas for Ascii Soccer the goal area is the entire width of the playing field at each end, the goal area for SimpleSoccer is a defined area at each end of the field, more in keeping with the RoboCupSoccer field and goals. The boundaries in SimpleSoccer, except for the goal areas, are hard barriers which impede movement of the ball and players: the ball does not rebound from the boundaries.

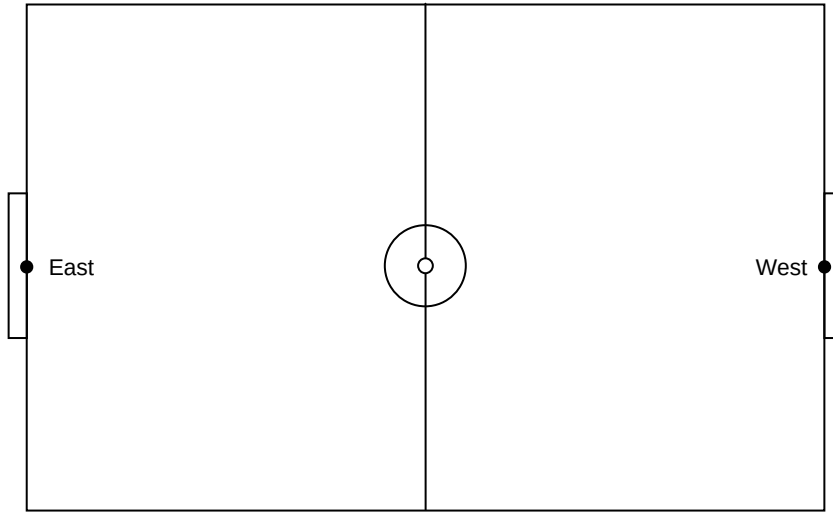


Figure 20 The soccer field and landmarks in the SimpleSoccer environment.

Player movement and sensory capabilities in SimpleSoccer are more aligned with those of RoboCupSoccer than Ascii Soccer. Whereas for Ascii Soccer player movement is limited to eight discrete directions, in SimpleSoccer players may move in any direction, specified by a real-valued angle from 0.0° to 360.0° relative to the player's current facing direction. Similarly, the ball can be kicked in any direction. In the SimpleSoccer environment, player and ball locations are specified by discrete grid co-ordinates, or cells. While movement and other actions can be in any direction, at the completion of an action, player and ball final locations are quantized to discrete cells. Unlike Ascii Soccer, in which a player's sensory capability is limited to the cells immediately adjacent to the player, players in SimpleSoccer have a field of vision similar to that of RoboCupSoccer.

Figure 21 shows the range of a player's vision in the SimpleSoccer environment – players can see objects in a diamond-shaped area in the direction the player is facing. A player's viewing diamond is determined by the view angle and view length, and only objects within a player's viewing diamond can be seen by the player. The black circles shown in Figure 21 represent objects on the field – only one is visible to the player in the diagram. At each time interval, or tick, players are presented with the cell co-ordinates, direction of (relative to the player's facing direction) and distance to (number of cells) for any object of interest (ball, player or goal) in the player's field of vision.

The detail of the visual feedback delivered to a player in the SimpleSoccer environment is the same irrespective of the player's vision parameters – only the size of the viewing diamond changes, the amount of detail does not. Unlike the RoboCupSoccer environment, players are

not able to sense objects that are close but not visible to the player – the only sense available to players in the SimpleSoccer environment is visual.

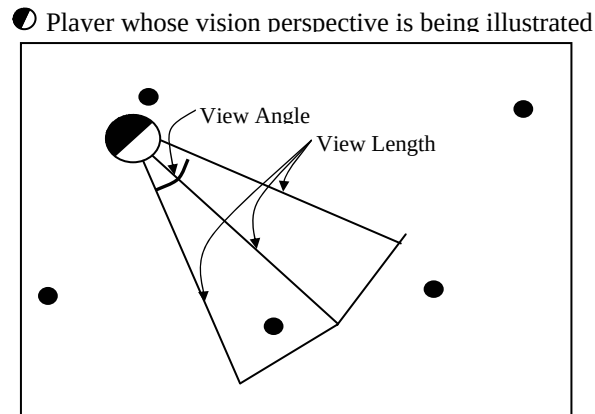


Figure 21 The visible range of a player in the SimpleSoccer environment.

The SimpleSoccer environment accommodates up to two teams, each with a maximum of eleven players. The “East” team starts the game on the right-hand (or east) side of the playing field (as viewed by the observer) and kicks towards the East Goal (Figure 20). Similarly, the “West” team starts the game on the left-hand (or west) side of the playing field and kicks towards the West Goal. At the start of play the ball is placed at the centre of the field, and only the team “kicking-off” may enter the centre circle (Figure 20) until contact is made with the ball.

The specific commands provided by the SimpleSoccer environment are:

- *turn(direction)*: the player turns through the angle specified by direction.
- *dash(direction, power, face)*: the player dashes in the direction specified with the power specified.
- *kick(direction, power, face)*: if the ball is within a kickable distance from the player, the player kicks the ball in the direction specified with the power specified.

For each of these commands:

- *direction* is specified in degrees in a clockwise direction relative to the direction the player is facing.
- *power* is specified as a percentage of maximum power and determines the number of cells the player or ball will travel as a result of the action.
- *face*, where specified, if true causes the player to turn to face in the direction specified after the completion of the action performed.

A SimpleSoccer unit of time is a single tick corresponding to one iteration of the program's main loop. At each tick the ball and players are moved, if necessary, a single cell (as a result of a previous action) and each player is presented with their new (visual) view of the state of the game, whereupon each player determines what action, if any, is to be taken and that action is begun (any previous action still in progress is superseded by the new action).

There may be a maximum of eleven players per team in SimpleSoccer, and team sizes may be uneven. There is no referee, thus there are no free kicks for offside or other rule violations. The ball is never out of bounds; the boundaries (except for the goal areas) are hard barriers which impede movement of the ball and players. There is no concept of player momentum and stamina as implemented in the RoboCupSoccer environment.

The SimpleSoccer environment provides players with a single sensor that detects visual information about the field, such as the distance and direction to objects in the player's current field of view – no other information is provided to the player. In contrast to the RoboCupSoccer environment, no random “noise” is introduced to the visual sensor information provided to the player – thus the information provided is complete and certain, and there is no loss of clarity of vision over distance.

3.4 Player Actions

The actions available to players in both simulation environments are built upon the commands provided by each simulator – some are simple commands as provided by the simulators while others are more complex hand-coded combinations of actions. The complete set of actions available is listed in Table 2 (opposite page). Differences in implementation between the two environments are noted where applicable. These hand-coded actions are referred to as “mid-level” skills throughout this thesis, since they are in most cases more complex than the basic actions provided by the soccer server but not as complex as “high-level” skills would generally be accepted to be. Examples of “high-level” skills are ball interception and moving to the closest point on the line segment between the ball and the goal being defended [175, 219].

3.4.1 Player Default Action

If a player is unable to determine an action to be taken based on the information known, the player may, if so configured, perform a hand-coded default *hunt* action - on the basis that the most likely cause for a player not being able to determine an action is that the ball is not

visible. The hand-coded hunt actions available as default actions are listed in Table 3 on page 84. In the RoboCupSoccer simulation environment the default action is hard-coded as *Hunt Action 3 – Random Turn*. The default action for the SimpleSoccer environment is configurable via the simulator configuration file, and is selected from the three hunt actions described.

Action Name	Action Description
Turn	As implemented by the simulator
Dash	For RoboCupSoccer: as implemented by the simulator. For SimpleSoccer: as implemented by the simulator, but is configurable at run time via the simulator configuration file to emulate the RoboCupSoccer implementation.
Kick	For RoboCupSoccer: as implemented by the simulator, except that the direction argument is forced to be the direction in which the player is facing (i.e. the player always kicks straight ahead). This was done to simplify the rule format – so that all commands required a single argument. For SimpleSoccer: as implemented by the simulator, but is configurable at run time via the simulator configuration file to emulate the RoboCupSoccer implementation.
RunTowardGoal	If the direction to the player’s goal is known, the player dashes once in that direction, otherwise no action is taken. This command requires one argument: the power to be applied for the dash.
RunTowardBall	If the direction to the ball is known, the player dashes once in that direction, otherwise no action is taken. This command requires one argument: the power to be applied for the dash.
GoToBall	If the direction to the ball is known, the player dashes towards the ball and continues to dash in that direction until the ball is within the kickable distance, otherwise no action is taken. This command requires one argument: the power to be applied for the dash.
KickTowardGoal	If the direction to the player’s goal is known, and the ball is within the kickable distance, the player kicks the ball once in the direction of its goal, otherwise no action is taken. This command requires one argument: the power to be applied for the kick.
DribbleTowardGoal	If the direction to the player’s goal is known, and the ball is within the kickable distance, the player kicks the ball once in the direction of its goal, then dashes once in the same direction. If the direction to the player’s goal is not known, or the ball is not within the kickable distance, no action is taken. This command requires one argument: the power to be applied for the dash

	and kick.
Dribble	If the ball is within the kickable distance, the player kicks the ball once in the direction it is facing, then dashes once in the same direction. If the ball is not within the kickable distance, no action is taken. This command requires one argument: the power to be applied for the dash and kick.
DoNothing	The player takes no action.

Table 2 Available Player Actions

Action Name	Action Description
Hunt Action 1 Goto Ball	if the ball is not visible then dash in a randomly chosen direction else if the player is not within the kickable distance of the ball then dash toward the ball else do nothing
Hunt Action 2 Locate Ball	if the ball is not visible then dash in a randomly chosen direction else do nothing
Hunt Action 3 Random Turn	turn 90° in a randomly chosen direction

Table 3 Player Default Actions

Nature adapts the organ to the function, and not the function to the organ.
- Aristotle, *De partib., animal., IV, xii, 694b; 13.*

Chapter 4

Evolving Goal-Scoring Behaviour

4.1 Introduction

If a soccer player is able to learn behaviours it should exhibit in response to stimuli, it may adapt to unpredictable, dynamic environments. Even though the overall objective a player is expected to achieve is able to be described, it is not always possible to precisely describe the behaviours a player should exhibit in achieving that objective. If a function can be described to evaluate the results of the player's behaviour against the desired outcome, that function can be used by some reinforcement learning algorithm to evolve the behaviours necessary to achieve the desired objective.

Fuzzy Sets are powerful tools for the representation of uncertain and vague data. Fuzzy inference systems make use of this by applying approximate reasoning techniques to make decisions based on such uncertain, vague data. However, a fuzzy inference system on its own is not usually self-adaptive and not able to modify its underlying rulebase to adapt to changing circumstances.

There has not been a great deal of success in the automatic generation of robot soccer players, and in fact hand-coded players, or players with hand-coded skills, generally outplay automatically generated players (see section 2.6 on page 56). Genetic algorithms are adaptive heuristic search algorithms premised on the evolutionary ideas of natural selection. By combining the adaptive learning capabilities of the genetic algorithm with the approximate reasoning capabilities of the fuzzy inference system, a hybrid system is produced, and the expectation is that this hybrid system will be capable of learning the behaviours a player needs

to exhibit in order to achieve a defined objective – in this case developing goal-scoring behaviour. While the combination of genetic algorithms and fuzzy inference systems have been studied in other areas (see section 2.3.5 on page 41), they have not generally been studied in an environment as complex, uncertain and dynamic as the robot soccer environment.

4.2 Chapter Goals

The specific research question examined in this chapter is:

How can a genetic algorithm be used to evolve a set of fuzzy rules to govern the behaviour of a simulated robot soccer player that produces consistent goal-scoring behaviour?

To address this question a simulated robot soccer player is developed and tested for both the RoboCupSoccer and SimpleSoccer environments. The player architecture, fuzzy inference system implementation, messy-coded genetic algorithm implementation and the methodology used to train the player and evolve the goal-scoring behaviour are all described in detail. Several experiments are performed in both environments to evolve players which exhibit goal-scoring behaviour, and the experimental setup and results are presented.

4.3 Player Architecture

The traditional decomposition for an intelligent control system is to break processing into a chain of information processing modules proceeding from sensing to action (Figure 22).

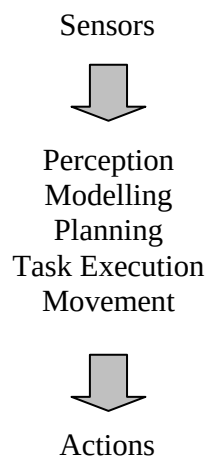


Figure 22 Traditional Control Architecture

The control architecture implemented in the work presented in this thesis is similar to the subsumption architecture described in [33]. This architecture implements a layering process where simple task achieving behaviours are added as required. Each layer is behaviour producing in its own right, although it may rely on the presence and operation of other layers. For example, in Figure 23 the *Movement* layer does not explicitly need to avoid obstacles: the *Avoid Objects* layer will take care of that. This approach creates players with reactive architectures and with no central locus of control [34].

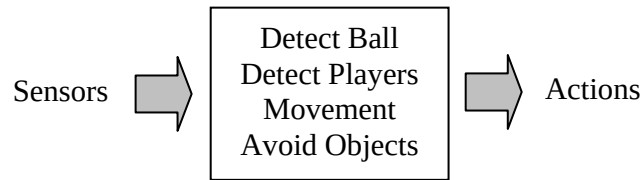


Figure 23 Soccer Player Layered Architecture

For the work presented in this thesis the behaviour producing layers are implemented as fuzzy if-then rules and governed by a fuzzy inference system comprised of the fuzzy rulebase, definitions of the membership functions of the fuzzy sets operated on by the rules in the rulebase, and a reasoning mechanism to perform the inference procedure. The fuzzy inference system is embedded in the player architecture, where it receives input from the soccer server and generates output necessary for the player to act Figure 24.

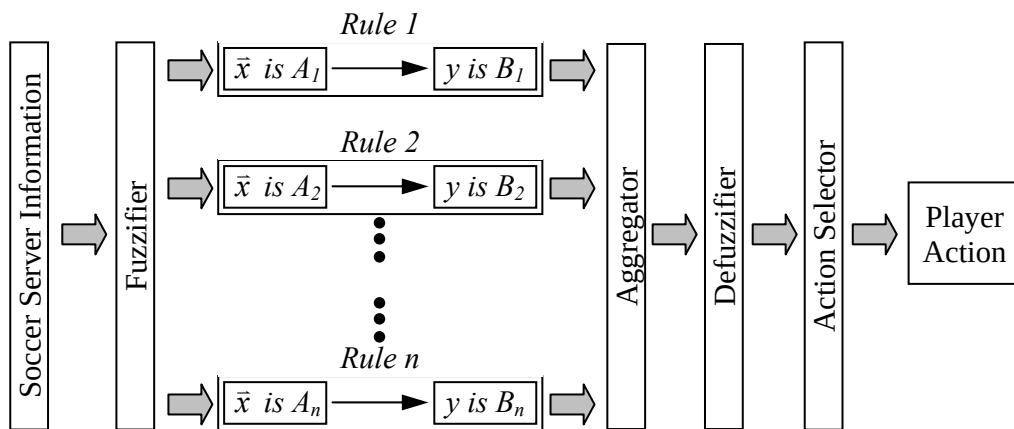


Figure 24 Player Architecture Detail

4.3.1 Soccer Server Information

The application by the inferencing mechanism of the fuzzy rulebase to external stimuli provided by the soccer server results in one or more fuzzy rules being executed and some resultant action being taken by the client. The external stimuli used as input to the fuzzy inference system are a subset of the visual information supplied by the soccer server: only sufficient information to situate the player and locate the ball is used. The environments studied in this work differ slightly with regard to the information supplied to the player:

- In the RoboCupSoccer environment the soccer server delivers regular *sense*, *visual* and *aural* messages to the players (see section 3.2 on page 75). The player implemented in this work uses only the *object name*, *distance* and *direction* information from the visual messages in order to determine its own position on the field and that of the ball. The player ignores any aural messages, and uses the information in the sense messages only to synchronise communication with the RoboCupSoccer server. Since the information supplied by the RoboCupSoccer server is not guaranteed to be complete or certain, the player uses its relative distance and direction from all fixed objects in its field of vision to estimate its position on the field. The player is then able to use the estimate of its position to estimate the direction and distance to the known, fixed location of its goal. The player is only aware of the location of the ball if it is in its field of vision, and only to the extent that the RoboCupSoccer server reports the relative direction and distance to the ball.
- In the SimpleSoccer environment the soccer server delivers only regular visual messages to the players (see section 3.3 on page 79): there are no aural or sense equivalents. Information supplied by the SimpleSoccer server is complete, in so far as the objects actually within the player's field of vision are concerned, and certain. Players in the SimpleSoccer environment are aware at all times of their exact location on the field, but are only aware of the location of the ball and the goal if they are in the player's field of vision. The SimpleSoccer server provides the *object name*, *distance* and *direction* information for objects in a player's field of vision. The only state information kept by a player in the SimpleSoccer environment is the co-ordinates of its location and the direction in which it is facing.

4.3.2 Fuzzification

Input variables for the fuzzy rules are fuzzy interpretations of the visual stimuli supplied to the player by the soccer server: the information supplied by the soccer server is fuzzified to represent the degree of membership of one of three fuzzy sets: *direction*, *distance* and *power*; and then given as input to the fuzzy inference system. Output variables are the fuzzy actions to be taken by the player. The universe of discourse of both input and output variables are covered by fuzzy sets (direction, distance and power), the parameters of which are predefined and fixed. Each input is fuzzified to have a degree of membership in the fuzzy sets appropriate to the input variable.

Both the RoboCupSoccer and the SimpleSoccer servers provide crisp values for the information they deliver to the players. These crisp values must be transformed into linguistic terms in order to be used as input to the fuzzy inference system. This is the fuzzification step: the process of transforming crisp values into degrees of membership for linguistic terms of fuzzy sets. The membership functions shown in Figure 25 on page 90 are used to associate crisp values with a degree of membership for linguistic terms. The parameters for these fuzzy sets were not learned by the evolutionary process, but were fixed empirically. The initial values were set having regard to RoboCupSoccer parameters and variables, and fine-tuned after minimal experimentation in the RoboCupSoccer environment.

4.3.3 Implication and Aggregation

The core section of the fuzzy inference system is the part which combines the facts obtained from the fuzzification with the rule base and conducts the fuzzy reasoning process: this is where the fuzzy inferencing is performed. The FIS model used in this work is a Mamdani FIS (see section 2.3.4.3 on page 38). The method implemented to apply the result of the antecedent evaluation to the membership function of the consequent is the correlation minimum, or *clipping* method, where the consequent membership function is truncated at the level of the antecedent truth (see section 2.3.4.2 on page 37). The aggregation method used is the min/max aggregation method described in section 2.3.4.3 on page 38. These methods were chosen because they are computationally less complex than other methods and generate an aggregated output surface that is relatively easy to defuzzify.

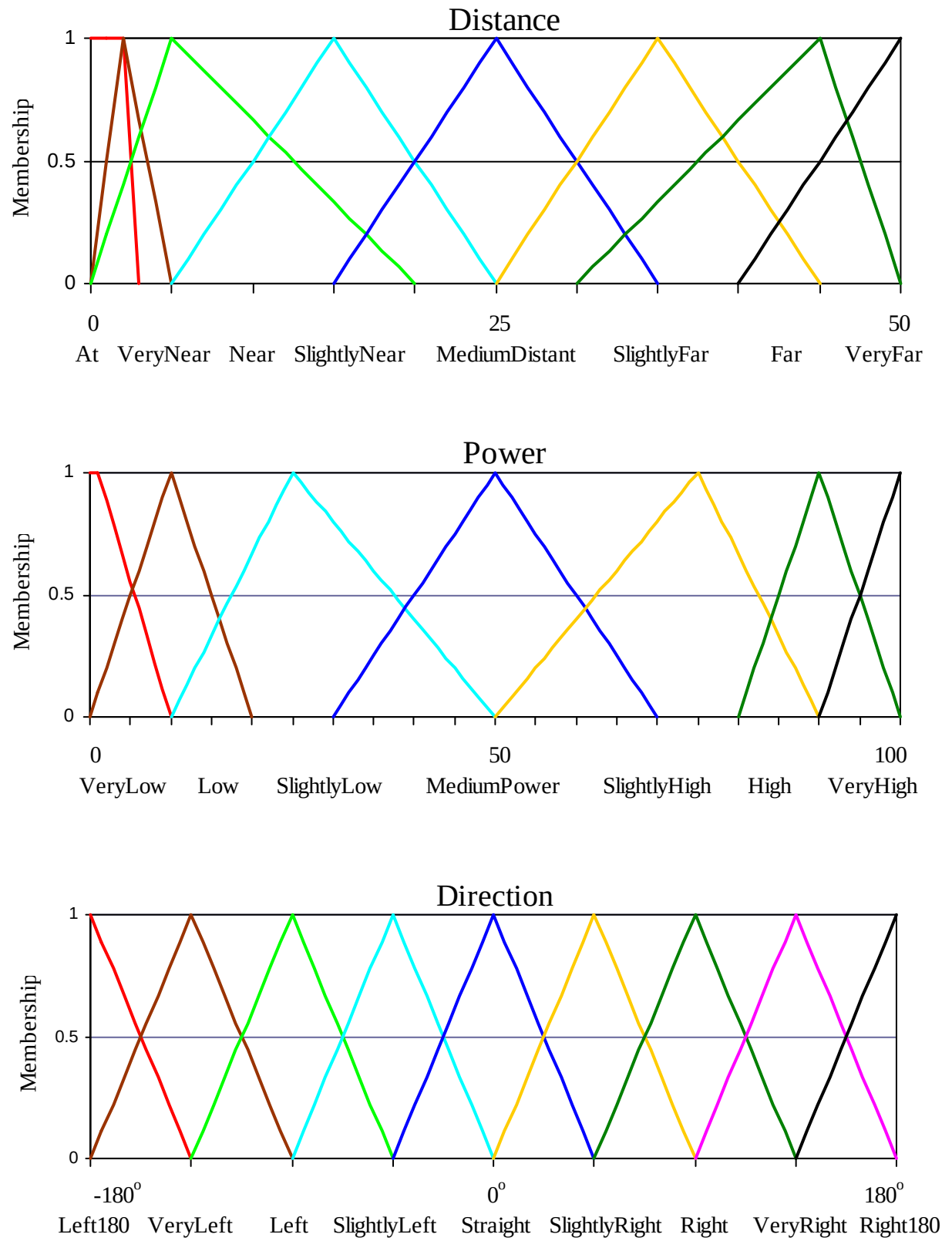


Figure 25 Distance, Power and Direction Fuzzy Sets

4.3.4 Defuzzification

The defuzzification method used is the *mean of maximum* method (see section 2.3.4.3 on page 38), also employed by Mamdani's fuzzy logic controllers. This technique takes the output distribution and finds its mean of maxima in order to compute a single crisp number. This is calculated as follows:

$$z = \sum_{i=1}^n \frac{z_i}{n}$$

where z is the mean of maximum, z_i is the point at which the membership function is maximum, and n is the number of times the output distribution reaches the maximum level. An example outcome of this computation is shown in Figure 26. This method of defuzzification was chosen because it is computationally less complex than other methods yet produces satisfactory results.

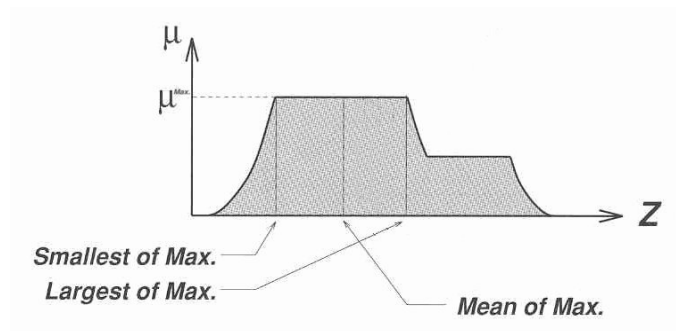


Figure 26 Mean of Maximum defuzzification method
(Adapted from [133])

4.3.5 Player Actions

A player will perform an action based on its skillset and in response to external stimuli; the specific response being determined in part by the fuzzy inference system. The action commands provided to the players by the RoboCupSoccer and SimpleSoccer simulation

environments are described in section 3.4 on page 82. For the experiments conducted for this chapter the SimpleSoccer simulator was, where appropriate, configured for RoboCupSoccer emulation mode.

4.3.6 Action Selection

The output of the fuzzy inference system is a number of $(action, value)$ pairs, corresponding to the number of fuzzy rules with unique consequents. The $(action, value)$ pairs define the action to be taken by the player, and the degree to which the action is to be taken. For example:

(KickTowardGoal, power)

(RunTowardBall, power)

(Turn, direction)

where *power* and *direction* are crisp values representing the defuzzified fuzzy set membership of the action to be taken.

Only one action is performed by the player in response to stimuli provided by the soccer server. Since several rules with different actions may fire, the action with the greatest level of support, as indicated by the value for truth of the antecedent, is selected.

4.4 Player Learning

This work investigates the use of an evolutionary technique in the form of a messy-coded genetic algorithm to efficiently construct the rulebase for a fuzzy inference system to solve a particular optimisation problem: goal-scoring behaviour for a robot soccer player. The flexibility provided by the messy-coded genetic algorithm is exploited in the definition and format of the genes on the chromosome, thus reducing the complexity of the rule encoding from the traditional genetic algorithm. With this method the individual player behaviours are defined by sets of fuzzy if-then rules evolved by a messy-coded genetic algorithm. Learning is achieved through testing and evaluation of the fuzzy rulebase generated by the genetic algorithm. The fitness function used to determine the fitness of an individual rulebase takes into account the performance of the player based upon the number of goals scored, or attempts made to move toward goal-scoring, during a game.

The genetic algorithm implemented in this work is a messy-coded genetic algorithm implemented using the Pittsburgh approach: each individual in the population is a complete ruleset (see section 2.3.5 on page 41 for ruleset definition).

4.5 Representation of the Chromosome

For these experiments, a chromosome is represented as a variable length vector of genes, and rule clauses are coded on the chromosome as genes. The encoding scheme implemented exploits the capability of messy-coded genetic algorithms to encode information of variable structure and length. It should be noted that while the encoding scheme implemented is a messy encoding, the algorithm implemented is the classic genetic algorithm: there are no primordial or juxtapositional phases implemented (see section 2.2.2 on page 17 for a discussion on messy genetic algorithms).

The basic element of the coding of the fuzzy rules is a tuple representing, in the case of a rule premise, a fuzzy clause and connector; and in the case of a rule consequent just the fuzzy consequent. The rule consequent gene is specially coded to distinguish it from premise genes, allowing multiple rules, or a ruleset, to be encoded onto a single chromosome.

For single-player trials, the only objects of interest to the player are the ball and the player's goal, and what is of interest is where those objects are in relation to the player. A premise is of the form:

(Object, Qualifier, {Distance | Direction}, Connector)

and is constructed from the following range of values:

Object: { BALL, GOAL }

Qualifier: { IS, IS NOT }

Distance: { AT, VERYNEAR, NEAR, SLIGHTLYNEAR, MEDIUMDISTANT, SLIGHTLYFAR, FAR, VERYFAR }

Direction: { LEFT180, VERYLEFT, LEFT, SLIGHTLYLEFT, STRAIGHT, SLIGHTLYRIGHT, RIGHT, VERYRIGHT, RIGHT180 }

Connector: { AND, OR }

Each rule consequent specifies and qualifies the action to be taken by the player as a consequent of that rule firing thus contributing to the set of (*action*, *value*) pairs output by the fuzzy inference system. A consequent is of the form:

(Action, {Direction | Null}, {Power | Null})

and is constructed from the following range of values (depending upon the skillset with which the player is endowed):

Action: { TURN, DASH, KICK, RUNTOWARDGOAL, RUNTOWARDBALL, GOTOBALL, KICKTOWARDGOAL, DRIBBLETOWARDGOAL, DRIBBLE, DONOTHING }

Direction: { LEFT180, VERYLEFT, LEFT, SLIGHTLYLEFT, STRAIGHT, SLIGHTLYRIGHT, RIGHT, VERYRIGHT, RIGHT180 }

Power: { VERYLOW, LOW, SLIGHTLYLOW, MEDIUMPOWER, SLIGHTLYHIGH, HIGH, VERYHIGH }

Fuzzy rules developed by the genetic algorithm are of the form:

if Ball is Near and Goal is Near then KickTowardGoal Low

if Ball is Far or Ball is SlightlyLeft then RunTowardBall High

In the example chromosome fragment shown in Figure 27 the shaded clause has been specially coded to signify that it is a consequent gene, and the fragment decodes to the following rule:

if Ball is Left and Ball is At or Goal is not Far then Dribble Low

In this case the clause connector *OR* in the clause immediately prior to the consequent clause is not required, so ignored.

(Ball, is Left, And)	(Ball, is At, Or)	(Goal, is not Far, Or)	(Dribble, Null, Low)
----------------------	-------------------	------------------------	----------------------

Figure 27 Messy-coded Genetic Algorithm Example Chromosome Fragment

Chromosomes are not fixed length: the length of each chromosome in the population varies with the length of individual rules and the number of rules on the chromosome. The number of clauses in a rule and the number of rules in a ruleset is only limited by the maximum size of a chromosome. The minimum size of a rule is two clauses (one premise and one consequent), and the minimum number of rules in a ruleset is one. Since the cut, splice and mutation operators implemented guarantee no out-of-bounds data in the resultant chromosomes, a rule is only considered invalid if it contains no premises. A complete ruleset is considered invalid only if it contains no valid rules.

An example complete chromosome and corresponding rules are shown in Figure 28 (with appropriate abbreviations).

(B,N,O)	(B,nF,A)	(G,N,A)	(RB,n,L)	(B,A,A)	(G,vN,O)	(KG,n,	(B,L,A)	(T,L,n)
---------	----------	---------	----------	---------	----------	--------	---------	---------

Premise	Consequent
---------	------------

Rule 1: if Ball is Near or Ball is not Far and Goal is Near then RunTowardBall Low

Rule 2: if Ball is At and Goal is VeryNear then KickTowardGoal MediumPower

Rule 3: if Ball is Left then Turn Left

Figure 28 Chromosome and corresponding rules

Some advantages of using a messy encoding in this case are:

- a ruleset is not limited to a fixed size
- a ruleset can be overspecified (i.e. clauses may be duplicated)
- a ruleset can be underspecified (i.e. not all genes are required to be represented)
- clauses may be arranged in any way

In contrast to classic genetic algorithms which use a fixed size chromosome and require “*don’t care*” values in order to generalise, no explicit *don’t care* values are, or need be, implemented for any attributes in this method. Since messy-coded genetic algorithms encode information of variable structure and length, not all attributes, particularly premise variables, need be present in any rule or indeed in the entire ruleset. A feature of the messy-coded genetic algorithm is that the format implies *don’t care* values for all attributes since any

premise may be omitted from any or all rules, so generalisation is an implicit feature of this method.

For the messy-coded genetic algorithm implemented in this work the selection operator is implemented in the same manner as for classic genetic algorithms. Roulette wheel selection was used in the RoboCupSoccer trials and the initial SimpleSoccer trials. Tests were conducted to compare several selection methods (see section 4.7.4.3 on page 113), and elitist selection was used in the remainder of the SimpleSoccer trials. Crossover is implemented by the *cut* and *splice* operators described in section 2.2.5.2 on page 27, and mutation is implemented as described in section 2.2.5.3 on page 28.

4.6 Experimental Evaluation

A series of experiments was performed in both the RoboCupSoccer and the SimpleSoccer simulation environments in order to test the viability of the fuzzy logic-based controller for the control of the player and the genetic algorithm to evolve the fuzzy ruleset. The following sections describe the trials performed, the parameter settings for each of the trials and other fundamental properties necessary for conducting the experiments.

An initial set of 20 trials was performed in the RoboCupSoccer environment in order to examine whether a genetic algorithm can be used to evolve a set of fuzzy rules to govern the behaviour of a simulated robot soccer player which produces consistent goal-scoring behaviour. This addresses part of the research question examined by this chapter.

As previously mentioned, the RoboCupSoccer environment was initially chosen as the problem domain for the examination research question addressed in this chapter, but because the RoboCupSoccer environment is a very complex real-time simulation environment it was found to be prohibitively expensive with regard to the time taken for the fitness evaluations for the evolutionary search. To overcome this problem the SimpleSoccer environment was developed so as to reduce the time taken for the trials. Following the RoboCupSoccer trials, a set of similar trials was performed in the SimpleSoccer environment to verify that the method performs similarly in the new environment.

Trials were conducted in the SimpleSoccer environment where the parameters controlling the operation of the genetic algorithm were varied in order to determine the parameters that

should be used for the messy-coded genetic algorithm in order to produce acceptable results, thereby addressing the remainder of the research question examined by this chapter.

4.6.1 Trials

For the results reported, a single trial consisted of a simulated game of soccer played with the only player on the field being the player under evaluation. The player was placed at a randomly selected position on its half of the field and oriented so that it was facing the end of the field to which it was kicking. For the RoboCupSoccer trials the ball was placed at the centre of the field, and for the SimpleSoccer trials the ball was placed at a randomly selected position along the centre line of the field.

4.6.2 Fitness Evaluation

The objective of the fitness function for the genetic algorithm is to reward the fitter individuals with a higher probability of producing offspring, with the expectation that combining the fittest individuals of one generation will produce even fitter individuals in later generations. All fitness functions implemented in this work indicate better fitness as a lower number, so representing the optimisation of fitness as a minimisation problem.

4.6.2.1 RoboCupSoccer Fitness Function

Since the objective of this work was to produce goal-scoring behaviour, the first fitness function implemented rewarded individuals for goal-scoring behaviour only, and was implemented as:

$$f = \begin{cases} 1.0 & , goals = 0 \\ \frac{1.0}{2.0 \times goals} & , goals > 0 \end{cases}$$

where *goals* is the number of goals scored by the player during the trial.

Equation 5 RoboCupSoccer Simple Goals-only Fitness Function

In early trials in the RoboCupSoccer environment the initial population of randomly generated individuals demonstrated no goal-scoring behaviour, so the fitness of each individual was the same across the entire population. This lack of variation in the fitness of the population resulted in the selection of individuals for reproduction being reduced to random choice. To overcome this problem a composite fitness function was implemented which effectively decomposes the difficult problem of evolving goal-scoring behaviour essentially from scratch - actually from the base level of skill and knowledge implicit in the primitives supplied by the environment – into two less difficult problems:

- evolve ball-kicking behaviour, and
- evolve goal-scoring behaviour from the now increased base level of skill and knowledge

In the RoboCupSoccer trials, individuals were rewarded for, in order of importance:

- the number of goals scored in a game
- the number of times the ball was kicked during a game

This combination was chosen to reward players primarily for goals scored, while players that do not score goals are rewarded for the number of times the ball is kicked on the assumption that a player which actually kicks the ball is more likely to produce offspring capable of scoring goals. The actual fitness function implemented in the RoboCupSoccer trials was:

$$f = \begin{cases} 1.0 & , kicks = 0 \\ 1.0 - \frac{kicks}{2.0 \times ticks} & , kicks > 0 \end{cases} , goals = 0$$

$$\frac{1.0}{2.0 \times goals} \quad , goals > 0$$

where

goals = the number of goals scored by the player during the trial

kicks = the number of times the player kicked the ball during the trial

ticks = the number of RoboCupSoccer server time steps of the trial

Equation 6 RoboCupSoccer Composite Fitness Function

4.6.2.2 SimpleSoccer Fitness Function

A similar composite fitness function was used in the trials in the SimpleSoccer environment, where individuals were rewarded for, in order of importance:

- the number of goals scored in a game
- minimising the distance of the ball from the goal

This combination was chosen to reward players primarily for goals scored, while players that do not score goals are rewarded on the basis of how close they are able to move the ball to the goal on the assumption that a player which kicks the ball close to the goal is more likely to produce offspring capable of scoring goals. This decomposes the original problem of evolving goal-scoring behaviour into the two less difficult problems:

- evolve ball-kicking behaviour that minimises the distance between the ball and goal, and
- evolve goal-scoring behaviour from the now increased base level of skill and knowledge

The actual fitness function implemented in the SimpleSoccer trials was:

$$f = \begin{cases} 1.0 & , kicks = 0 \\ 0.5 + \frac{dist}{2.0 \times fieldLen} & , kicks > 0 \\ \frac{1.0}{2.0 \times goals} & , goals > 0 \end{cases} , goals = 0$$

where

goals = the number of goals scored by the player during the trial

kicks = the number of times the player kicked the ball during the trial

dist = the minimum distance of the ball to the goal during the trial

fieldLen = the length of the field

Equation 7 SimpleSoccer Composite Fitness Function

The difference between the composite fitness function implemented in the RoboCupSoccer environment and the composite fitness function implemented in the SimpleSoccer environment is just an evolution of thinking – rewarding a player for kicking the ball often when no goal is kicked could reward a player that kicks the ball very often in the wrong direction more than a player that kicks the ball fewer times but in the right direction. The SimpleSoccer implementation of the composite fitness function rewards players more for kicking the ball closer to the goal irrespective of the number of times the ball was kicked. This is considered a better approach to encourage behaviour that leads to scoring goals.

4.6.2.3 Fitness Values

To facilitate the interpretation of fitness graphs and fitness values presented throughout this thesis, following is an explanation of the fitness values generated by the fitness functions used in this work. All fitness functions implemented in this work generate a real number R , where $0.0 < R \leq 1.0$, $R = 1.0$ indicates no ball movement and $R \approx 0.0$ indicates very good performance – smaller fitness values indicate better performance.

For ball movement in the RoboCupSoccer environment where a composite fitness function is implemented, fitness values are calculated in the range $x \leq R \leq y$, where $x = 0.5$ and $y = 1.0$. For ball movement in the SimpleSoccer environment where a composite fitness function is implemented, fitness values are calculated in the range $x \leq R \leq y$, where $x \approx 0.5$ and $y \approx 0.77$. This derivation of the values for x and y is explained in more detail in section 7.6.1 on page 178. Where a simple goals-only fitness function is implemented, ball movement alone is not rewarded: if no goals are scored the fitness function assigns $R = 1.0$. In both environments all fitness functions assign discrete values for goal-scoring, depending upon the number of goals scored. Table 4 summarises the fitness values returned by the various fitness functions.

In initial trials in the RoboCup environment players were evaluated over five separate games and then assigned the average fitness value of those games. Since each game in the Robocup environment is played in real time, this was a very time consuming method. The results of experiments where the player's fitness was calculated as the average of five games were compared with results where the player's fitness was assigned after a single game and

were found to be almost indistinguishable. Due to the considerable time savings gained by assigning fitness after a single game, this is the method used throughout this work. Since players evolved using the average fitness method are exposed to different starting conditions they may be more robust than those evolved using single-game fitness, but the effect is extremely small considering the number of different starting positions players could be evaluated against and the fact that the starting positions of the player and ball really only affect the first kick of the ball.

		Simple Goals-only Fitness Function	RoboCupSoccer Composite Fitness Function	SimpleSoccer Composite Fitness Function
No Goals Scored	No Ball Movement	1.0000	1.0000	1.0000
	Ball Movement	<i>n/a</i>	[0.5, 1.0]	[~0.5, ~0.77]
Goals Scored	1	0.5000	0.5000	0.5000
	2	0.2500	0.2500	0.2500
	3	0.1667	0.1667	0.1667
	4	0.1250	0.1250	0.1250
	5	0.1000	0.1000	0.1000
	6	0.0833	0.0833	0.0833
	7	0.0714	0.0714	0.0714
	8	0.0625	0.0625	0.0625
	9	0.0556	0.0556	0.0556
	10	0.0500	0.0500	0.0500
	...	...	...	...
	<i>n</i>	0.5/ <i>n</i>	0.5/ <i>n</i>	0.5/ <i>n</i>

Table 4 Fitness Assignment Summary

4.6.3 Control Parameters

The genetic algorithm parameters common to all 20 initial trials in both the RoboCupSoccer and SimpleSoccer environments are shown in Table 5.

Parameter	Value
Maximum Chromosome Length	64 genes
Population Size	200
Maximum Generations	25
Selection Method	Roulette Wheel
Crossover Method	Single Point
Crossover Probability	0.8
Mutation Rate	10%
Mutation Probability	0.35

Table 5 Genetic Algorithm Control Parameters

A game was terminated when:

- the target fitness of 0.05 was reached
- the ball was kicked out of play (RoboCupSoccer only)
- the elapsed time expired:
 - o 120 seconds real time for RoboCupSoccer
 - o 1000 ticks of simulator time for SimpleSoccer
- A period of no player movement or action expired
 - o 10 seconds real time for RoboCupSoccer
 - o 100 ticks of simulator time for SimpleSoccer

The target fitness of 0.05 reflects a score of 10 goals in the allotted playing time. This figure was chosen to allow the player a realistic amount of time to develop useful strategies yet terminate the search upon finding an acceptably good individual.

Two methods of terminating the evolutionary search were implemented. The first stops the search when a specified maximum number of generations have occurred; the second stops

the search when the best fitness in the current population becomes less than the specified target fitness. Both methods were active, with the first to be encountered terminating the search. Early stopping did not occur in any of the experiments reported in this chapter.

4.7 Results

The following sections describe the results for the experiments performed for both the RoboCupSoccer and the SimpleSoccer environments. Discussion and analysis of the results is also presented.

4.7.1 RoboCupSoccer Initial Trial Results

Figure 29 shows the average fitness of the population after each generation for each of the 20 trials for the RoboCupSoccer environment, showing that the performance of the population does improve steadily and, in some of the trials, plateaus towards a fitness of 0.5, or goal-scoring behaviour. Figure 30 on page 104 shows the best individual fitness from the population after each generation for each of 20 trials for the RoboCupSoccer environment, showing that good individuals are found after very few generations, in contrast to the gradual improvement in average fitness shown in Figure 29.

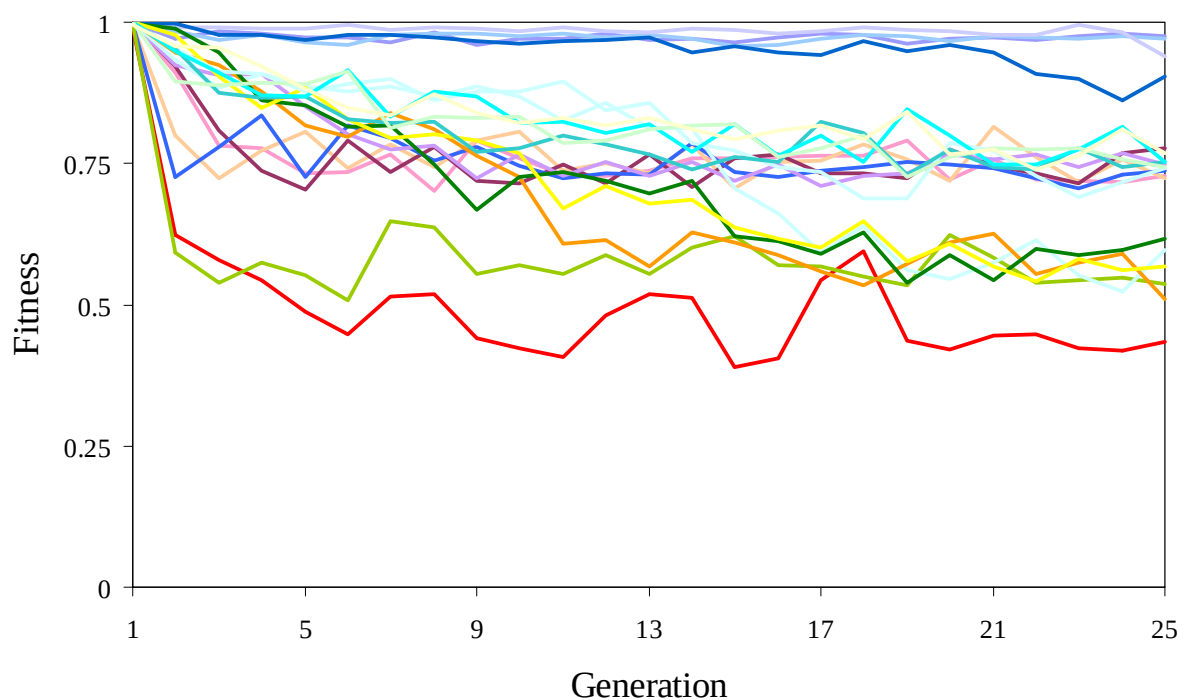


Figure 29 RoboCupSoccer: Average Fitness - Initial 20 Trials

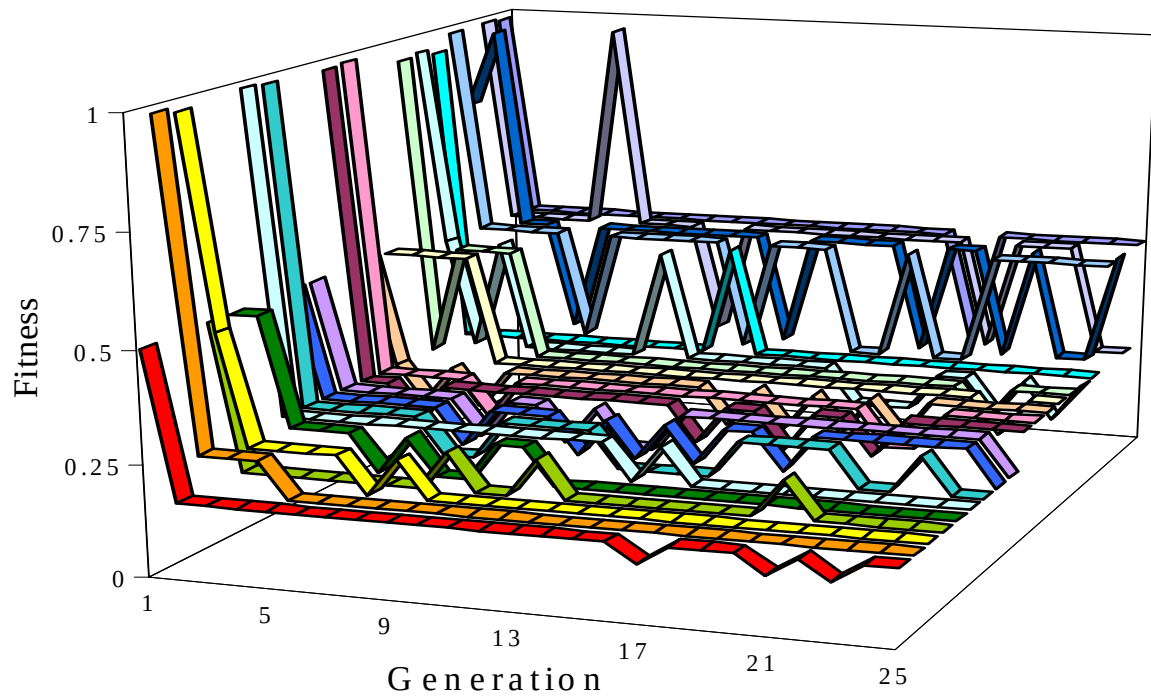


Figure 30 RoboCupSoccer: Best Fitness - Initial 20 Trials

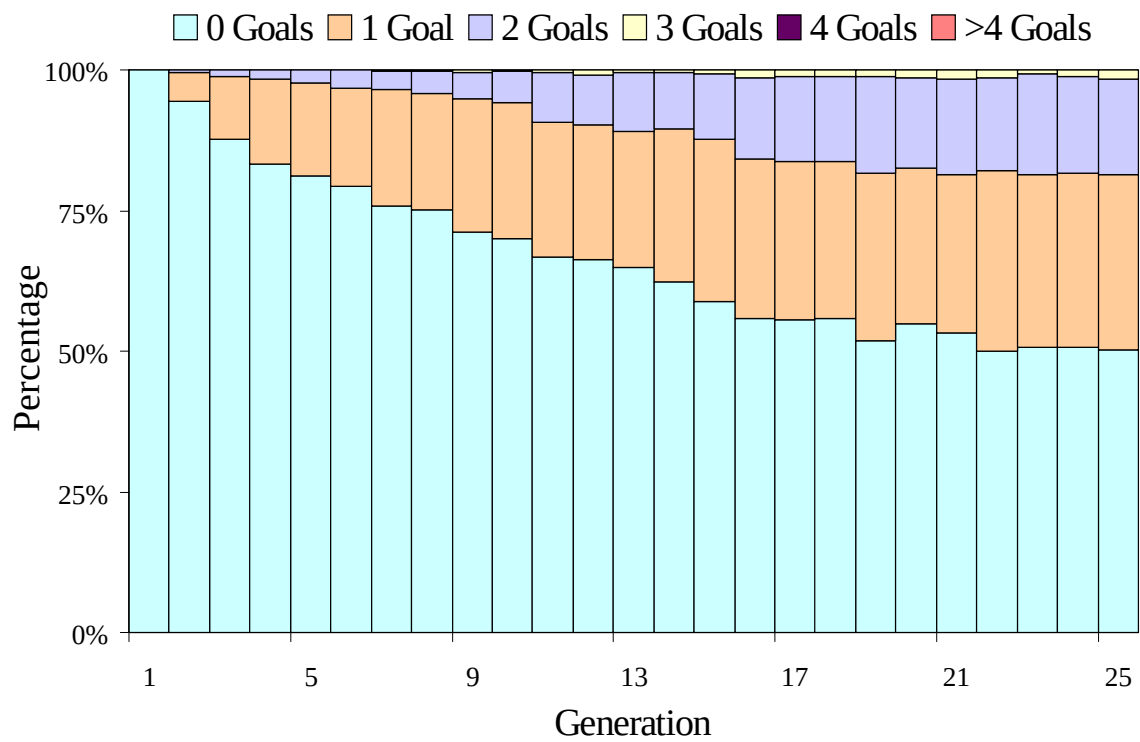


Figure 31 RoboCupSoccer: Frequency of Individuals Scoring Goals

Figure 31 is another visualisation of the progressive learning of the population from generation to generation, showing that not only do more players learn to kick goals over time, they learn to kick more goals more quickly. The histogram shows, for the initial 20 RoboCupSoccer trials, the average percentage of the population which scored 0, 1, 2, 3, 4 or more than 4 goals for each generation. The maximum number of goals scored by any individual was 3.

4.7.2 SimpleSoccer Initial Trial Results

Figure 32 shows the average fitness of the population after each generation for each of the 20 trials for the SimpleSoccer environment, and as for the RoboCupSoccer trials, this graph shows that the performance of the population does improve steadily and plateaus, but unlike the RoboCupSoccer trials the average performance of the population does not approach a fitness of 0.5, or goal-scoring behaviour. Figure 32 also shows that the average fitness curves for the SimpleSoccer trials are more tightly clustered than those of the RoboCupSoccer trials, and plateau towards a fitness of around 0.75 which, in the SimpleSoccer environment indicates ball-kicking behaviour rather than goal-scoring behaviour.

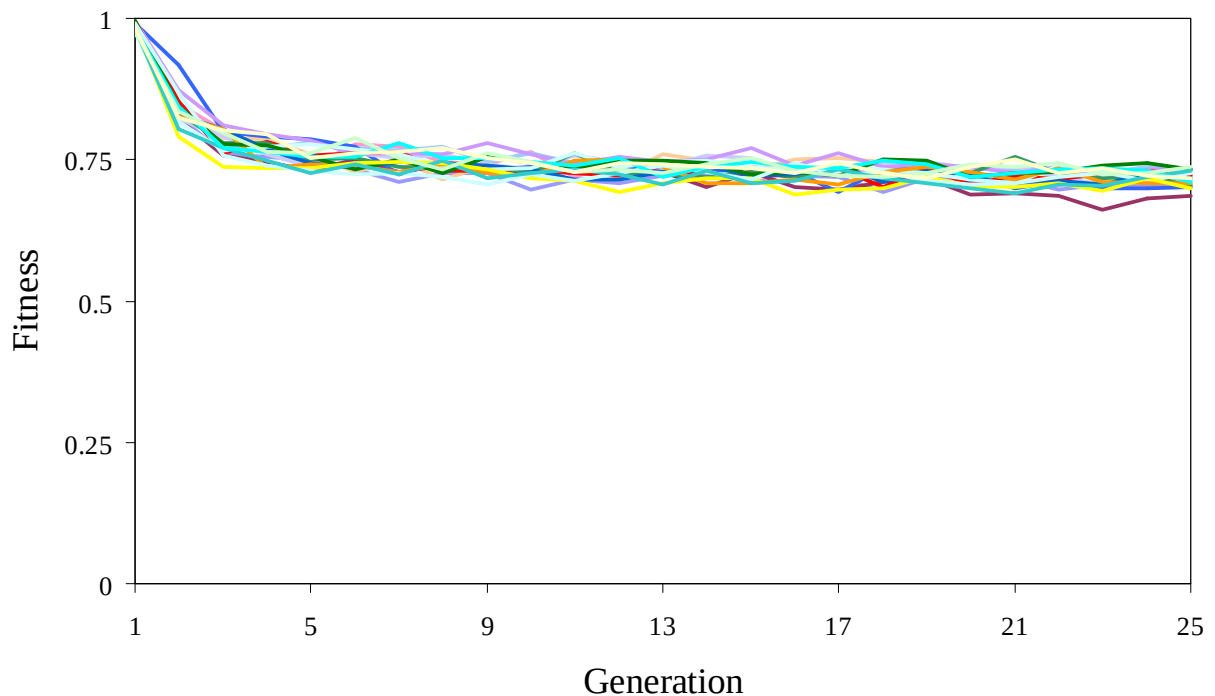


Figure 32 SimpleSoccer: Average Fitness – Initial 20 Trials

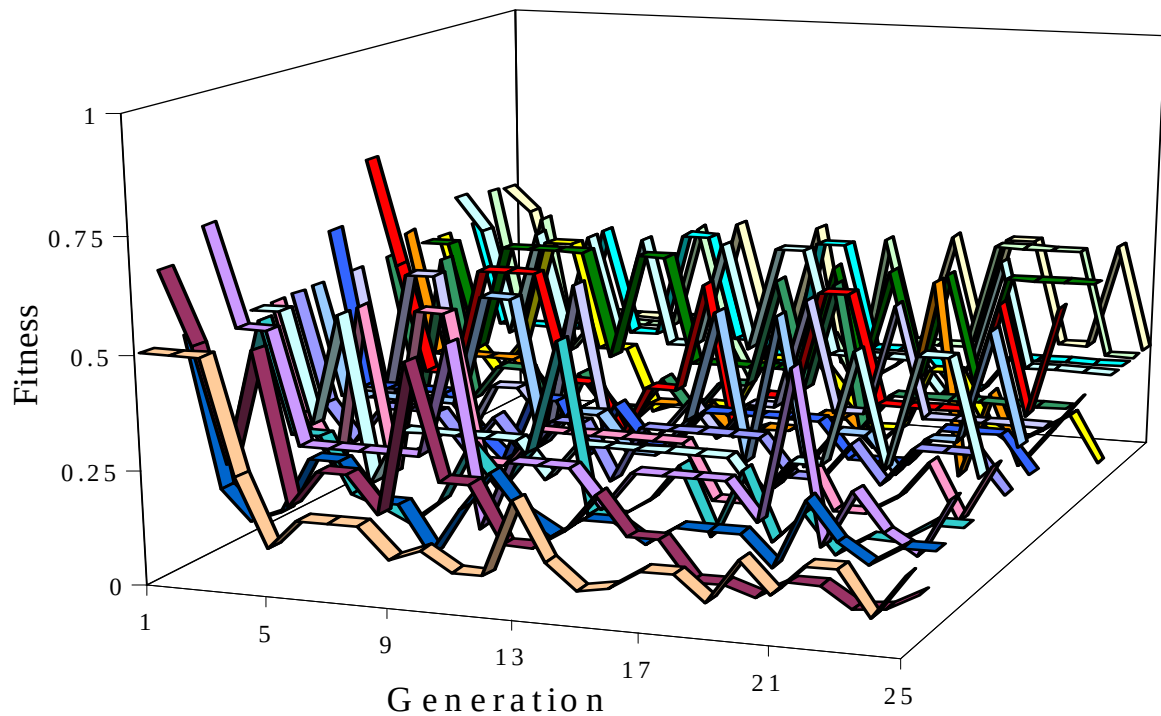


Figure 33 SimpleSoccer: Best Fitness - Initial 20 Trials

Figure 33 shows the best individual fitness from the population after each generation for each of 20 trials for the SimpleSoccer environment and, as for the RoboCupSoccer trials, this graph shows that good individuals are found after very few generations. It is evident from a comparison of Figure 30 (page 104) and Figure 33 that while good individuals are found quickly in both environments, the algorithm seems to be more stable in the RoboCupSoccer environment. The data shows that once a good individual is found in the RoboCupSoccer environment, good individuals are then more consistently found in future generations than in the SimpleSoccer environment.

Figure 34 shows, for the initial 20 SimpleSoccer trials, the average percentage of the population which scored 0, 1, 2, 3, 4 or more than 4 goals for each generation. The maximum number of goals scored by an individual was 10. The contrast with the equivalent graph for the RoboCupSoccer environment (Figure 31 on page 104) is striking since, although some individuals in the SimpleSoccer environment scored more goals than any individual in the RoboCupSoccer environment, the average goal-scoring behaviour of the population was less developed in the SimpleSoccer environment. This inconsistency is likely to be an indication that the combination of parameters used for the SimpleSoccer environment causes the genetic algorithm to converge more quickly than in the RoboCupSoccer environment, and a possible

explanation for the lower average performance of the population when compared to that of the RoboCupSoccer environment as seen in Figure 32 on page 105. Since these SimpleSoccer experiments were performed primarily as a comparison with the RoboCupSoccer experiments the genetic algorithm parameters were kept the same, but the soccer simulator implementations differ considerably and no tuning of simulator parameters to ensure similar performance was performed.

4.7.3 SimpleSoccer as a Model for RoboCupSoccer

While the difference in the results of the experiments in the RoboCupSoccer and SimpleSoccer environments indicate that SimpleSoccer is not an exact model of RoboCupSoccer (as indeed it was not intended to be), there is a broad similarity in the results which is sufficient to indicate that the SimpleSoccer environment is a good simplified model of the RoboCupSoccer environment. Because SimpleSoccer is considered a reasonable model for RoboCupSoccer, and to take advantage of the significantly reduced training times provided by the SimpleSoccer environment when compared to RoboCupSoccer, all results reported in the remainder of this chapter are for experiments conducted exclusively in the SimpleSoccer environment.

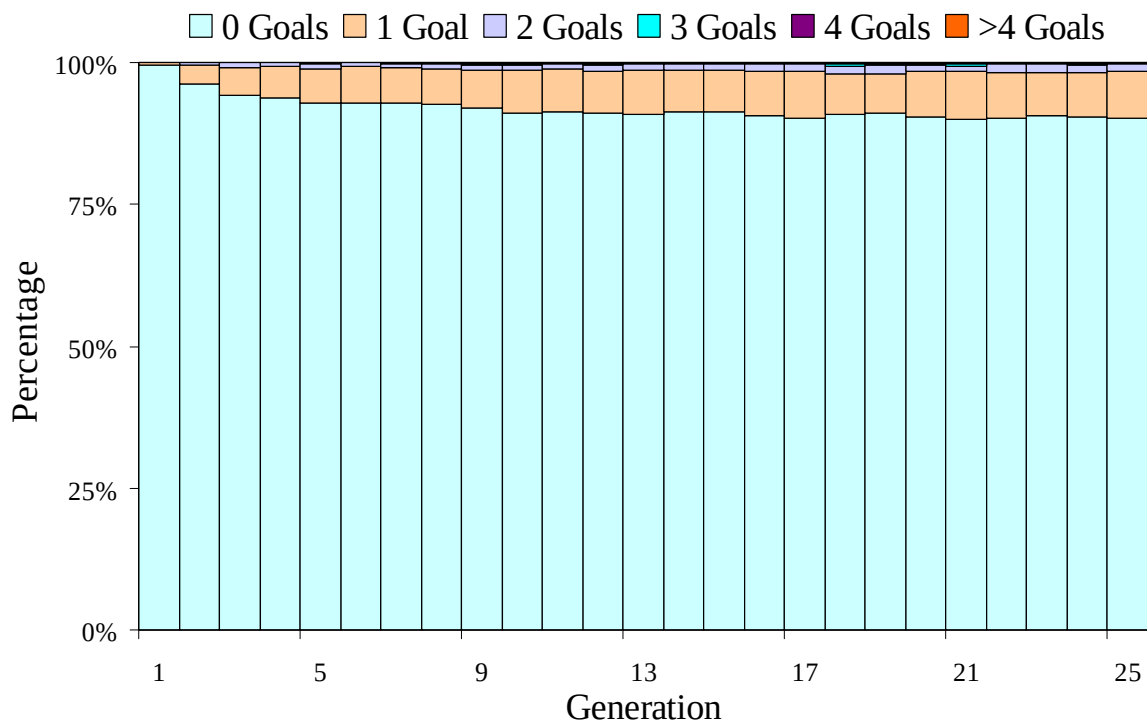


Figure 34 SimpleSoccer: Frequency of Individuals Scoring Goals

4.7.4 GA Parameter Determination

Several experiments were conducted in order to determine a set of genetic algorithm parameters conducive to producing acceptable results.

The following GA parameters were varied in these trials:

- Maximum Chromosome Length
- Population Size
- Selection Method
- Crossover Method
- Mutation Rate
- Maximum Generations

The values for the parameters shown in Table 5 on page 102 are used as control values, and in each of the trials presented in the following sections the value for a single GA parameter is varied. In the experiments conducted a series of 10 trials was performed for each different value of the parameter being varied, and in each case, with the exception of the experiment varying the maximum number of generations, the results presented are the averages of the 10 trials – each line on the graphs shown represents the average of the 10 trials. For the experiment varying the maximum number of generations, only 10 trials were conducted, and the results for each trial is reported individually – each line on the graph represents a single trial.

4.7.4.1 Maximum Chromosome Length

While the actual length of the chromosome, measured as the number of genes on the chromosome, may vary depending upon the location of the cut point during the cut-and-splice operation of crossover, the maximum length of the chromosome is fixed throughout the evolutionary process. In order to determine if the maximum length of the chromosome is a significant factor in determining the quality of the evolutionary search, and if so what value is a good value, a series of trials was performed with different maximums for the chromosome length. Figure 35 shows the average fitness of the population throughout the evolutionary process. It is evident from Figure 35 that while a maximum chromosome length of 100 offers

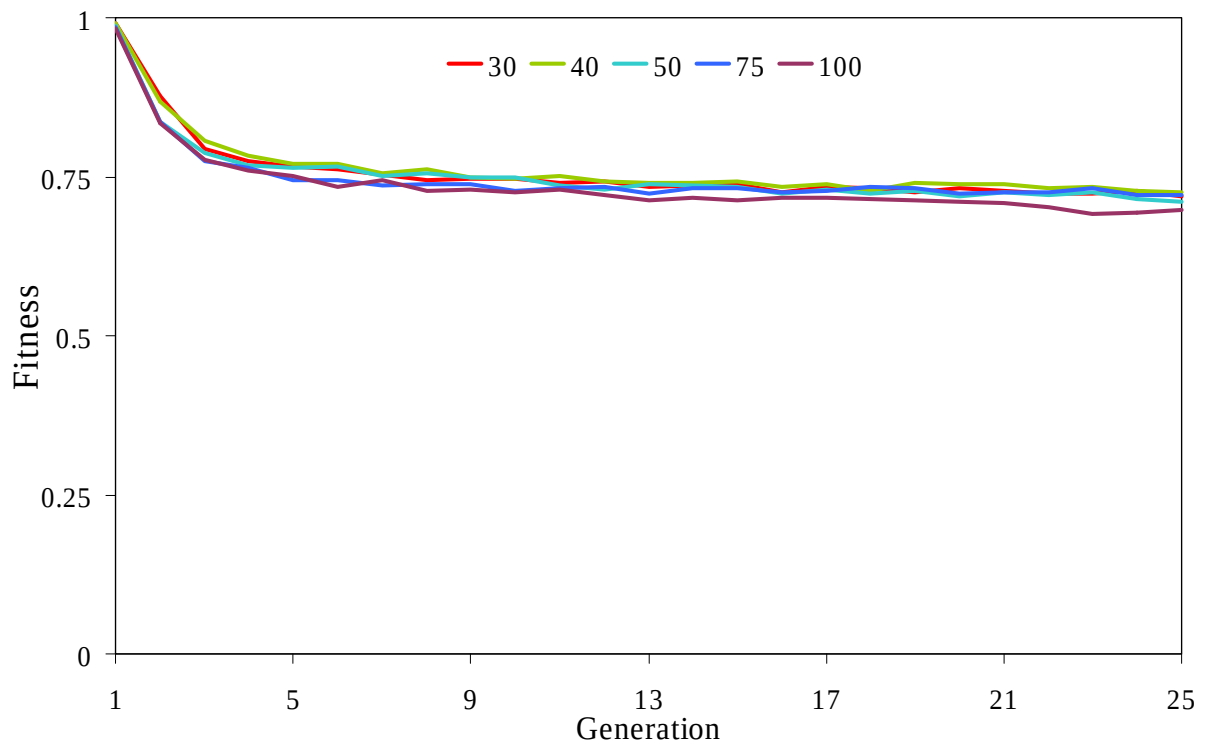


Figure 35 Average Fitness: Maximum Chromosome Length Variation

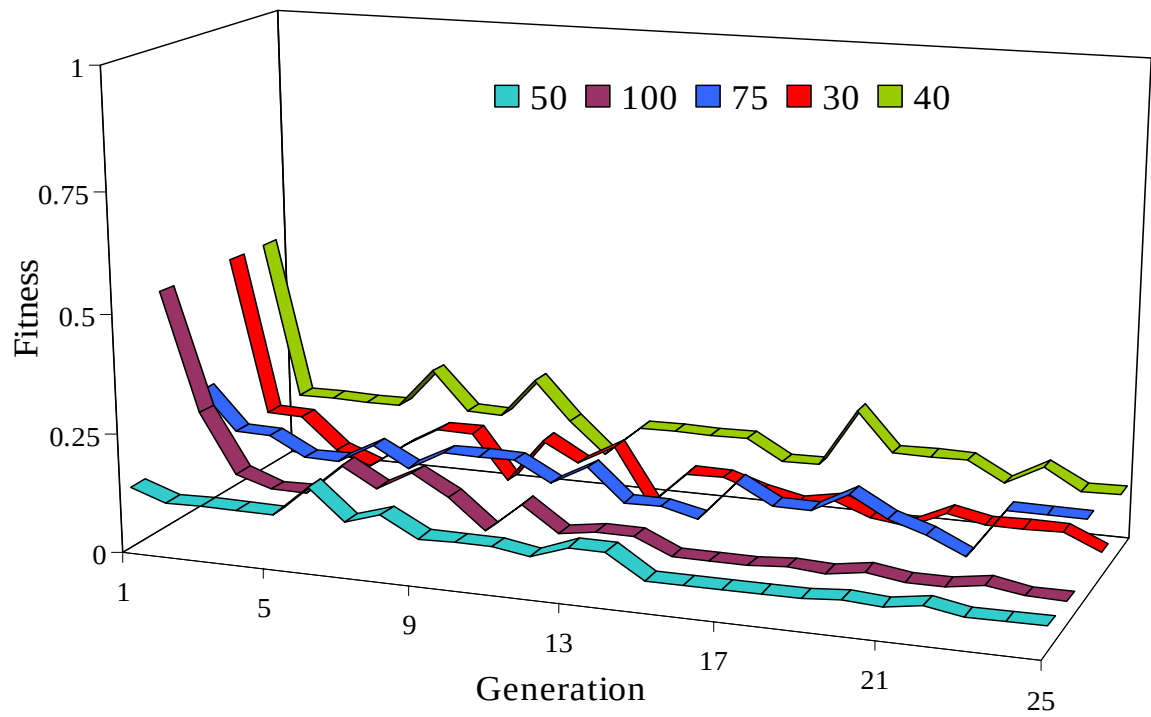


Figure 36 Best Fitness: Maximum Chromosome Length Variation

a very slight advantage, it is not significant. This is further substantiated by Figure 36 (page 109) which shows the best fitness in the population throughout the evolutionary process. The results shown indicate that while the method is not sensitive in any significant way to variations in the maximum chromosome length, a maximum chromosome length of somewhere between 50 and 100 genes, and most probably between 50 and 75 genes, produces less variation in the best fitness over the duration of the process.

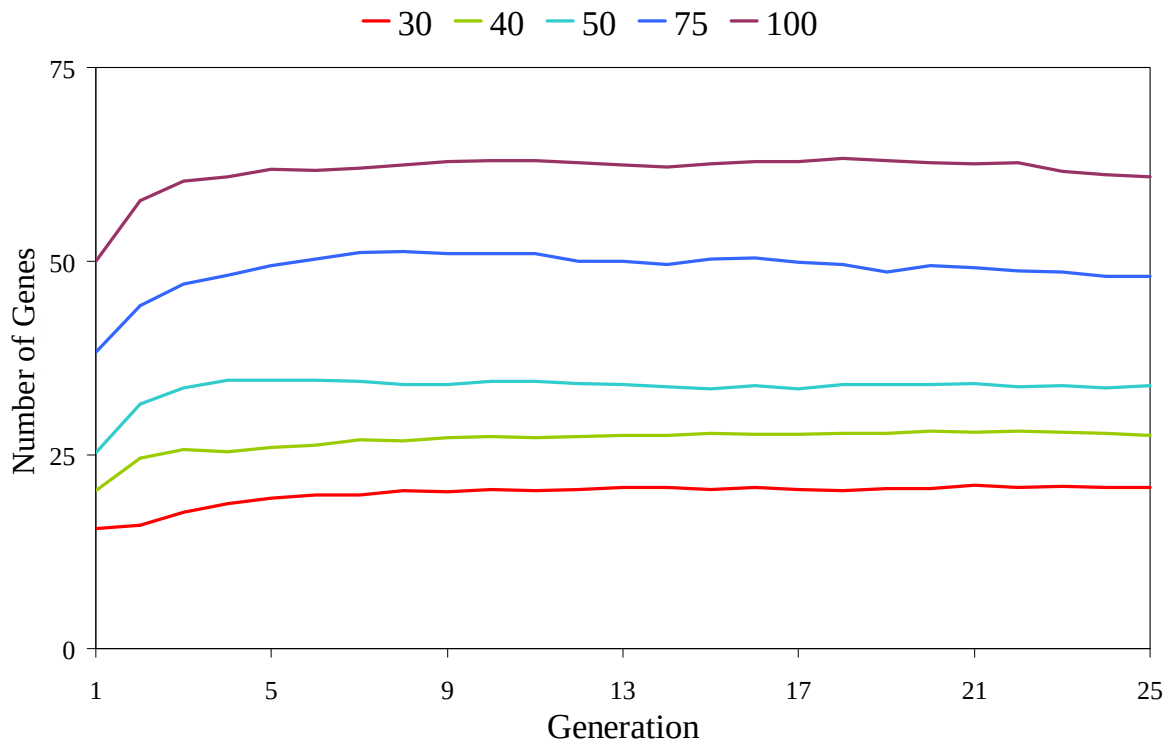


Figure 37 Average Chromosome Length: Maximum Chromosome Length Variation

Since the actual chromosome length may vary up to the maximum, the average chromosome length for each of the variations in maximum length was measured throughout the evolutionary process, as was the average number of valid rules per chromosome. These data are shown in Figure 37 and Figure 38 respectively. The chromosome lengths in the initial populations are initialised randomly between the minimum length of two genes (the smallest number of genes for a valid ruleset) and the maximum chromosome length, so the average chromosome lengths for the initial population shown in Figure 37 are as expected. All trials show the average chromosome length rising in the initial few generations, then settling to

around two-thirds of the maximum length. Given this, and since single-point crossover was used in these trials, with the cut point chosen randomly and chromosomes truncated at the maximum length after the cut-and-splice operation, the results indicate that chromosome length is unaffected by selection pressure. However, Figure 38 shows the average number of rules per chromosome rising for all of the trials. This would indicate that there is some selection pressure for more rules per chromosome or shorter rules, but since the chromosome length is bounded, so is the number of rules per chromosome. Though outside the scope of this thesis, some further trials to investigate whether the pressure is for more rules or shorter rules, and the optimum number and/or length of rules per chromosome, would be useful work to undertake in the future.

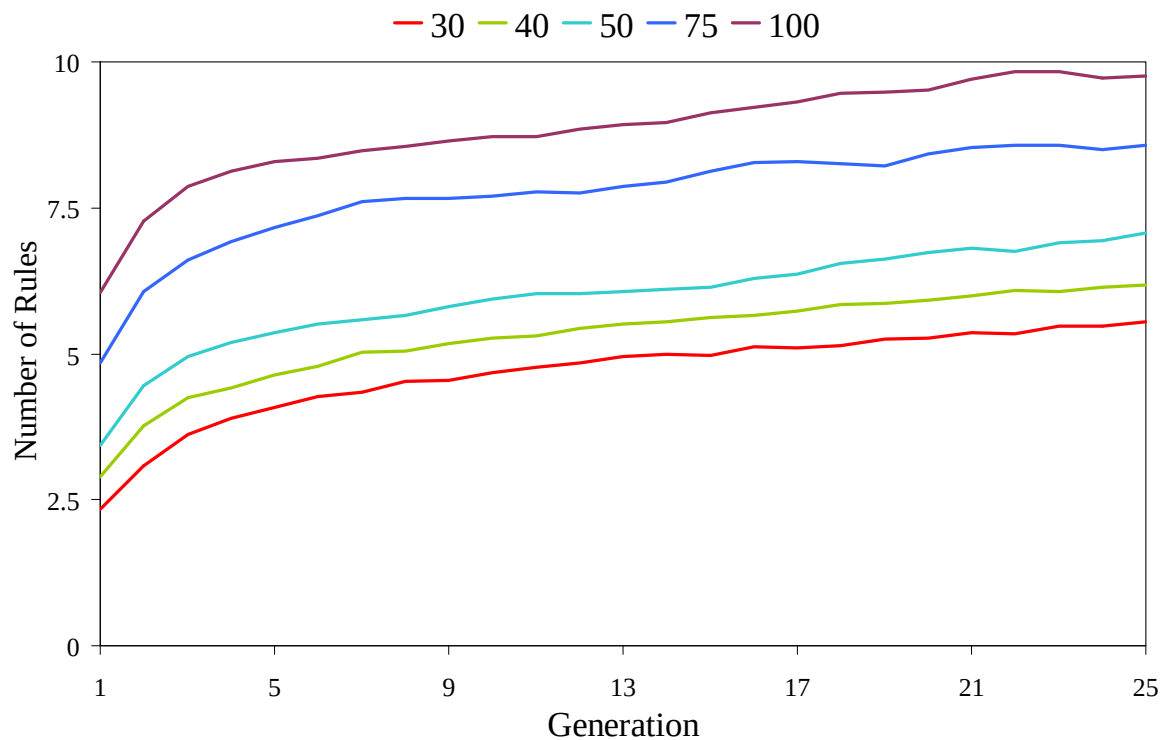


Figure 38 Average Valid Rules per Chromosome: Maximum Chromosome Length Variation

4.7.4.2 Population Size

Since the size of the population differs in this experiment, the number of generations was also varied according to the population size for each set of 10 trials to ensure the comparison between population sizes was for the same number of evaluations. Figure 39 shows the average fitness of the population over 10,000 evaluations, and Figure 40 shows the best fitness values for the same trials. It can be seen from the graphs that varying the population size has little effect on the population average fitness with only marginally better results for smaller population sizes, and a similarly small effect on individual best fitness, with larger populations producing slightly more stable results. Overall, the difference in performance between the population sizes tested is not significant, suggesting that it is the number of solutions evaluated, or the extent of the search, that is a significant factor affecting performance. This is consistent with the findings of other work in the area [176].

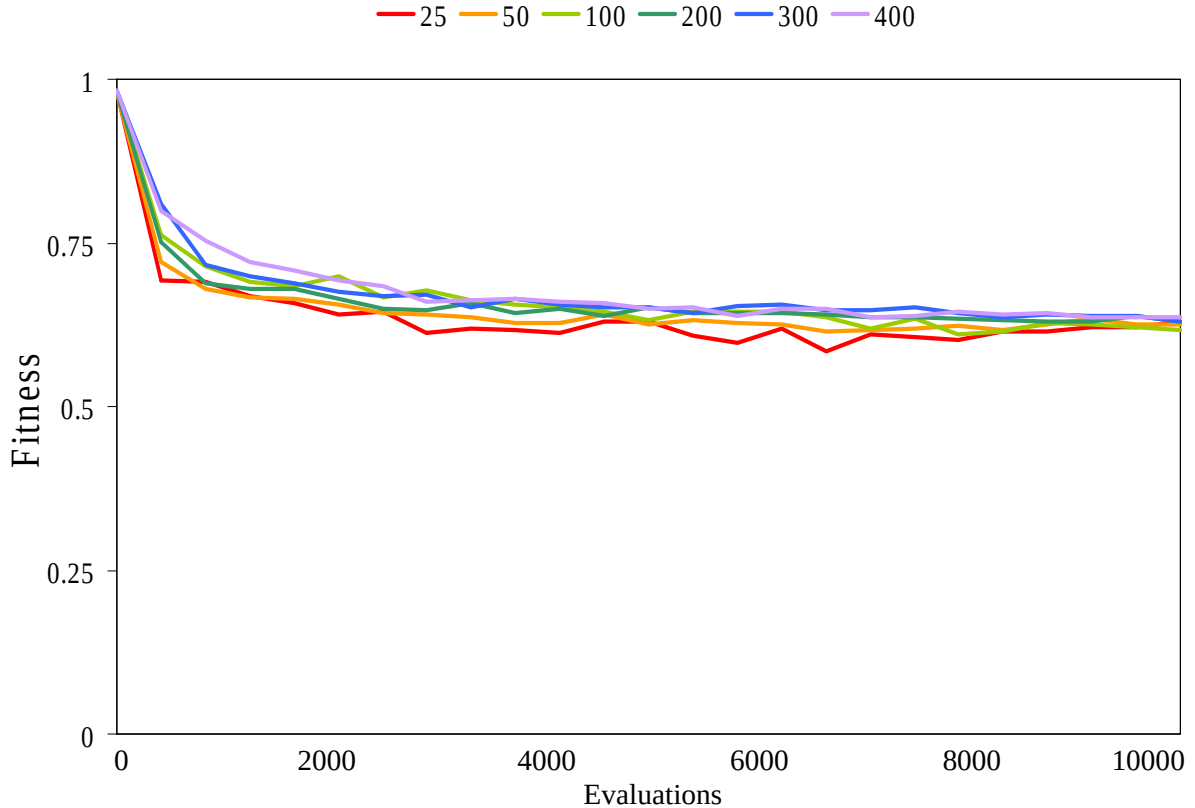


Figure 39 Average Fitness: Population Size Variation

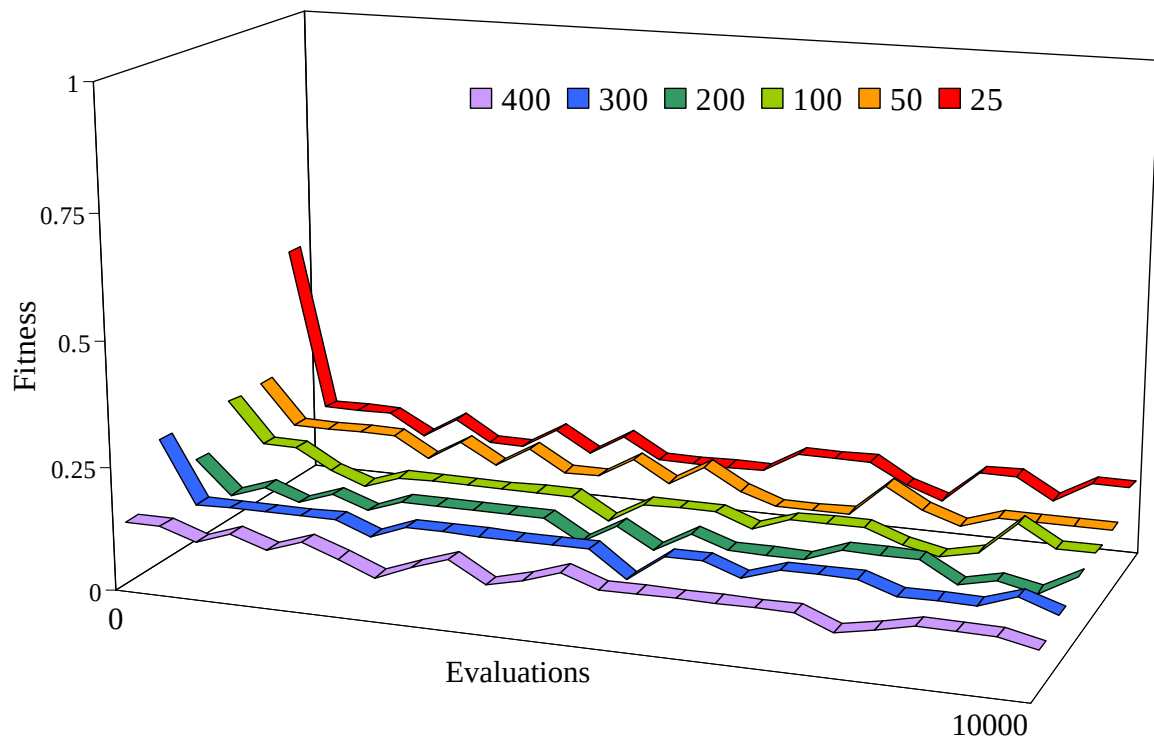


Figure 40 Best Fitness: Population Size Variation

4.7.4.3 Selection Methods

Trials were conducted to compare the performance of three methods of selection: roulette wheel selection, tournament selection with tournament size = 2 and tournament selector = 0.75, and elitist selection with 5% retention. The population average fitness for the 10 trials conducted for each method is shown in Figure 41 (page 114), and shows clearly that in terms of the population average fitness the method of selection is not a significant determinant. Figure 42 (page 114) shows the best fitness curves for these trials and shows that the elitist method produces similar results to the tournament method, with the roulette wheel method producing slightly less stable results. The good performance of the elitist method is probably due to the stochastic nature of the environment. Since the placement of the ball and the player is random, a player evaluated twice would likely be assigned different fitness values for each evaluation. The elitist method works well in this type of environment, allowing the better solutions to be evaluated several times thus allowing the reliability of the estimate of fitness to increase over time.

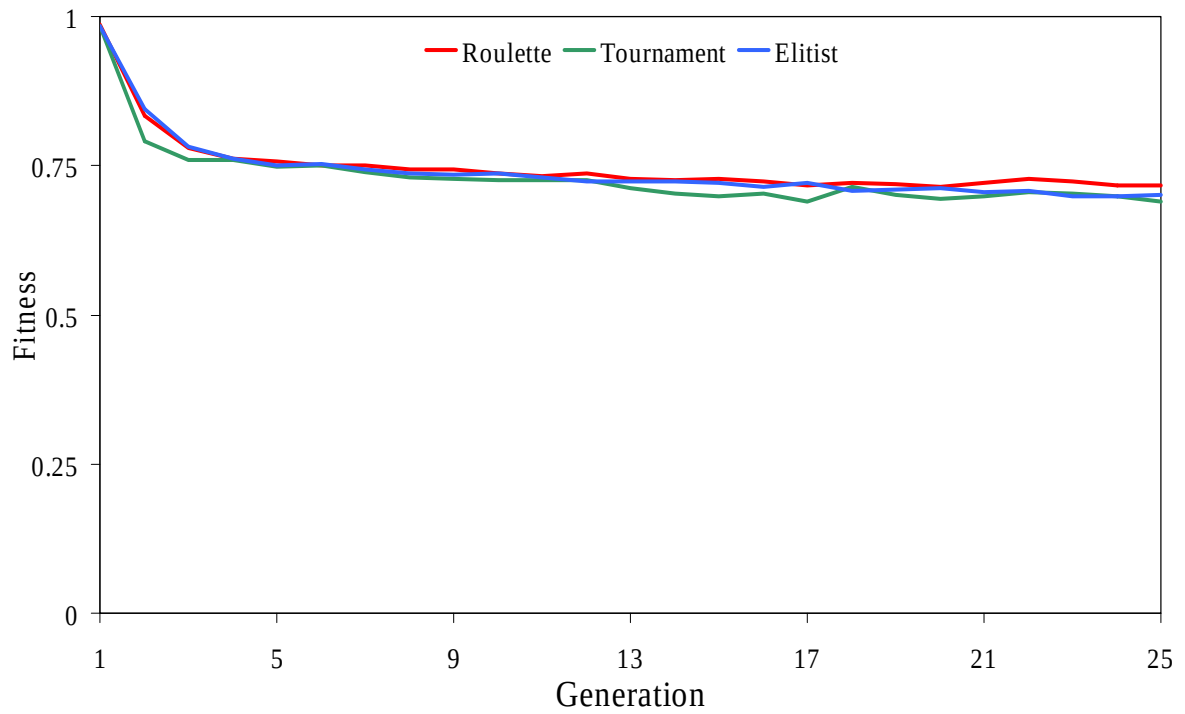


Figure 41 Average Fitness: Selection Method Variation

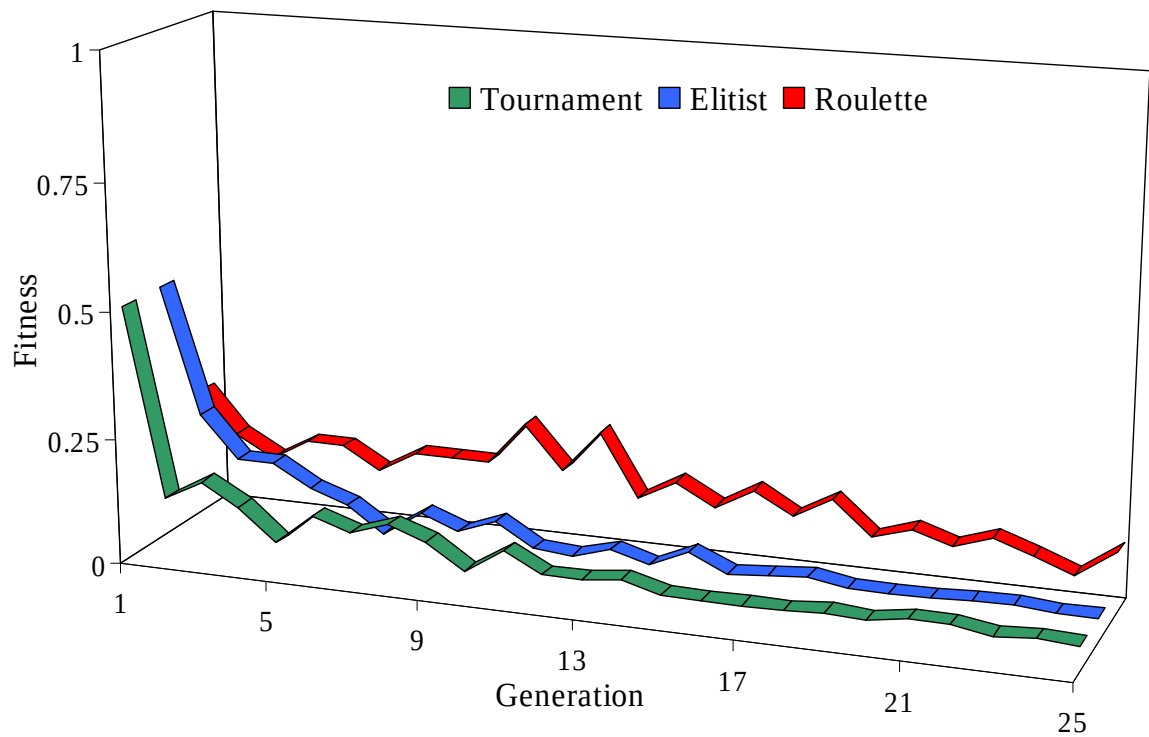


Figure 42 Best Fitness: Selection Method Variation

4.7.4.4 Crossover Methods

Since the chromosomes involved in crossover may be of different lengths, crossover methods that assume equal length chromosomes are not defined. The performance of two methods of crossover was compared: one-point and two-point. Figure 43 shows the population average fitness over the duration of the evolutionary process, and Figure 44 (page 116) shows the best fitness values for the same period. It can be seen from this data that there is no meaningful difference in performance between the two methods, either with respect to the population average fitness or the best fitness achieved. While two-point crossover is more disruptive than one-point crossover, it is not clear from this data if a much more disruptive crossover method, such as uniform crossover, would significantly affect the performance of the method. It is likely that the messy-coding of the genetic algorithm and the rules-based nature of the representation causes the method to be somewhat less sensitive to disruptive crossover.

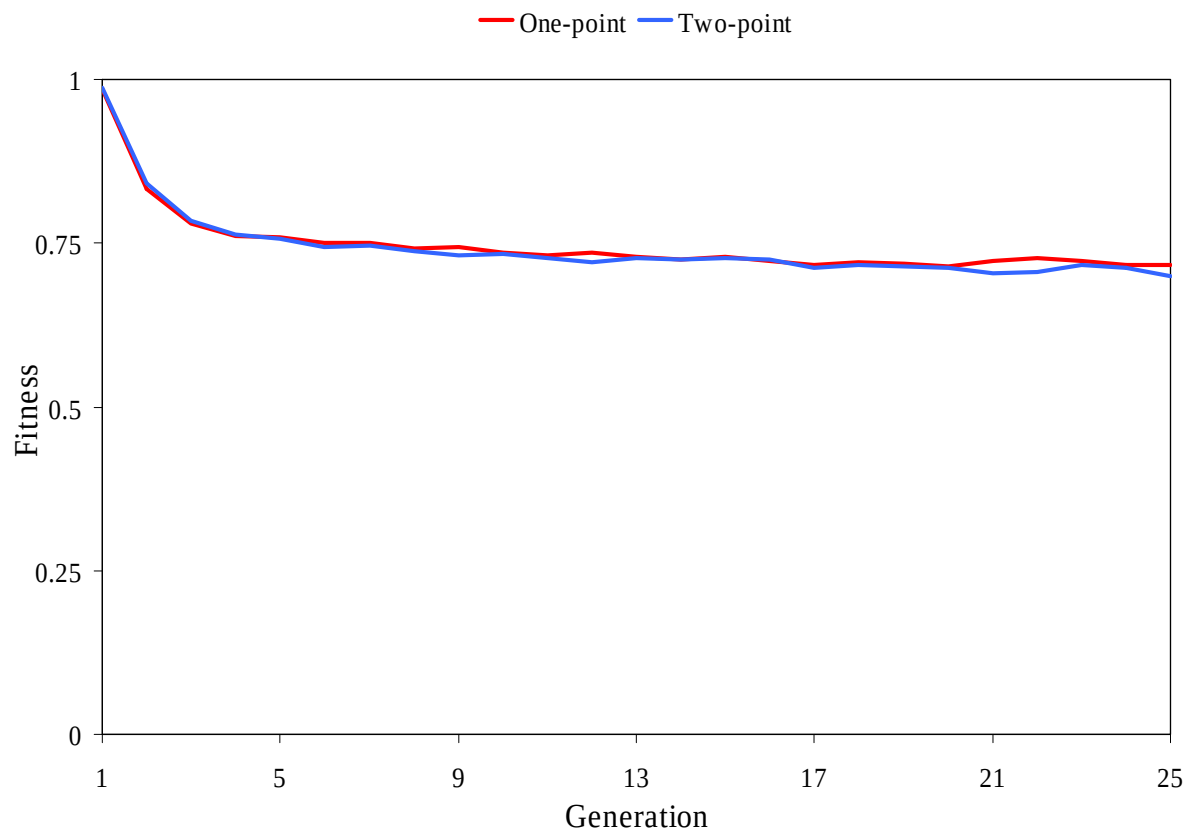


Figure 43 Average Fitness: Crossover Method Variation

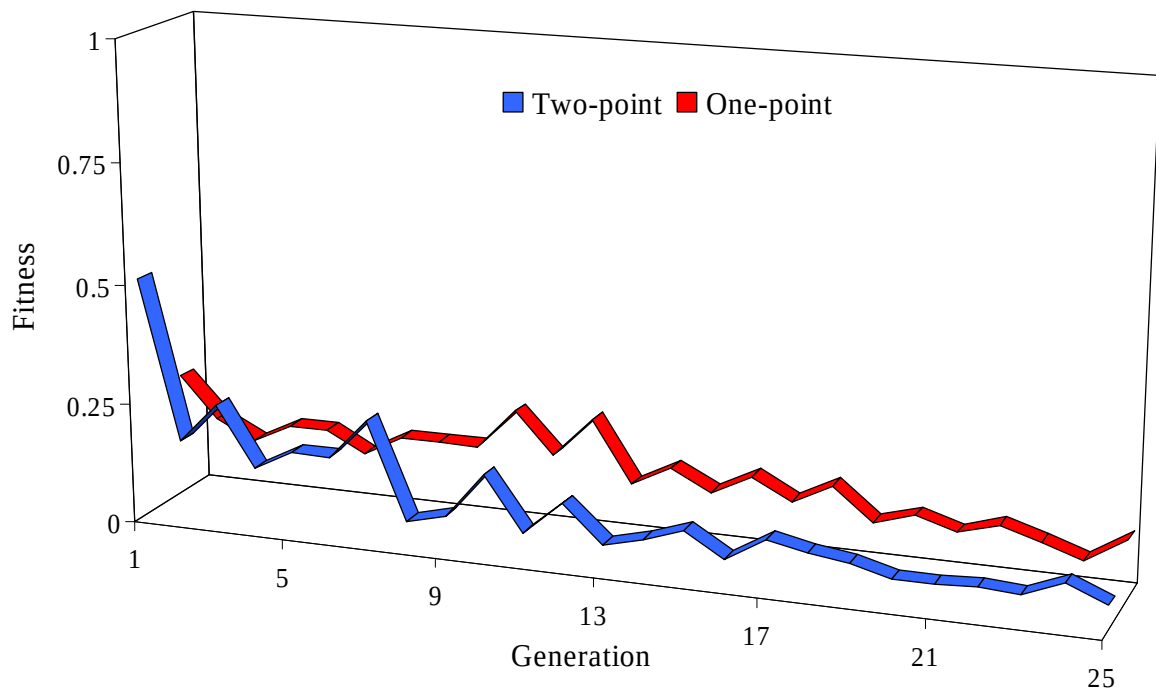


Figure 44 Best Fitness: Crossover Method Variation

4.7.4.5 Mutation Rate

To determine the effect of the rate of mutation on the evolutionary process, 10 trials for each of several mutation rates were performed, and the averages of those trials presented. Figure 45 shows the population average fitness for each mutation rate tested, and Figure 46 the best fitness for those mutation rates throughout the evolutionary search. While varying the mutation rate has only a marginal effect on the population average fitness and, for the most part the individual best fitness, a mutation rate of 15% does seem to improve the population average fitness slightly, and the individual best fitness more markedly. This suggests that a mutation rate of 15% is the best balance between maintaining sufficient diversity in the population to help drive the evolutionary process while minimising the disruption to the good building blocks being created throughout the process.

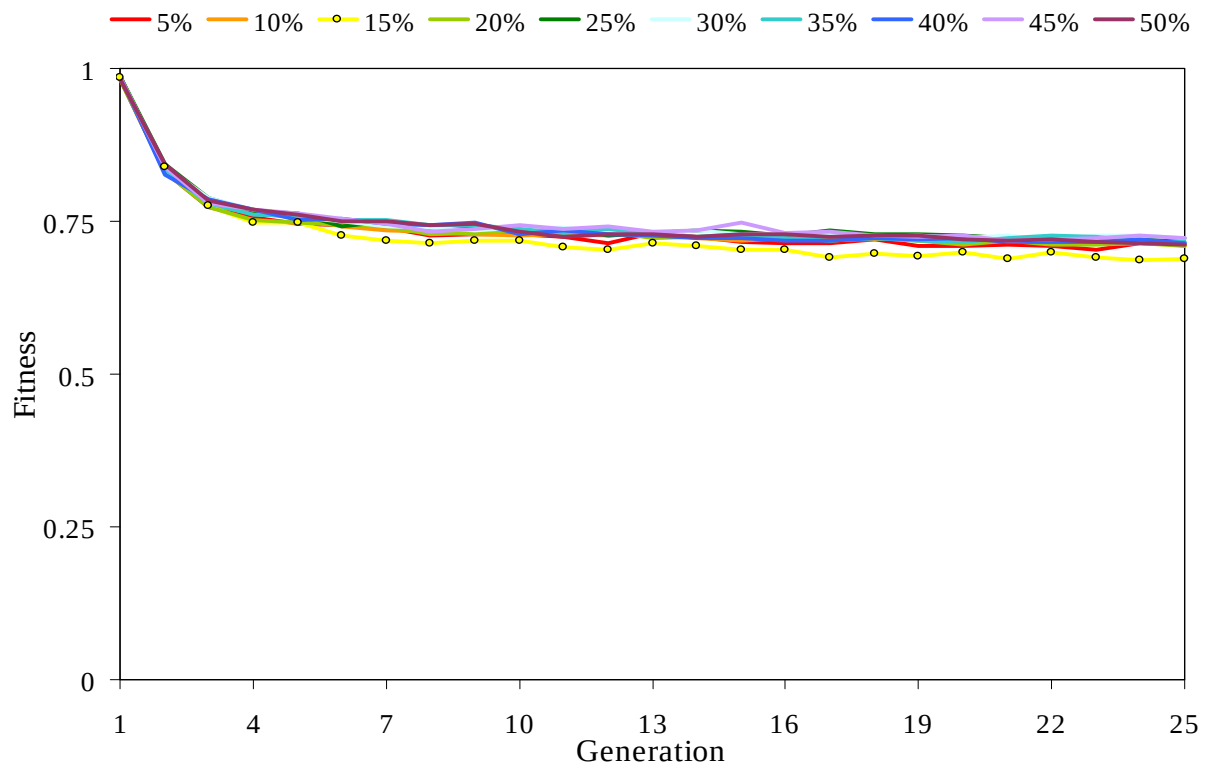


Figure 45 Average Fitness: Mutation Rate Variation

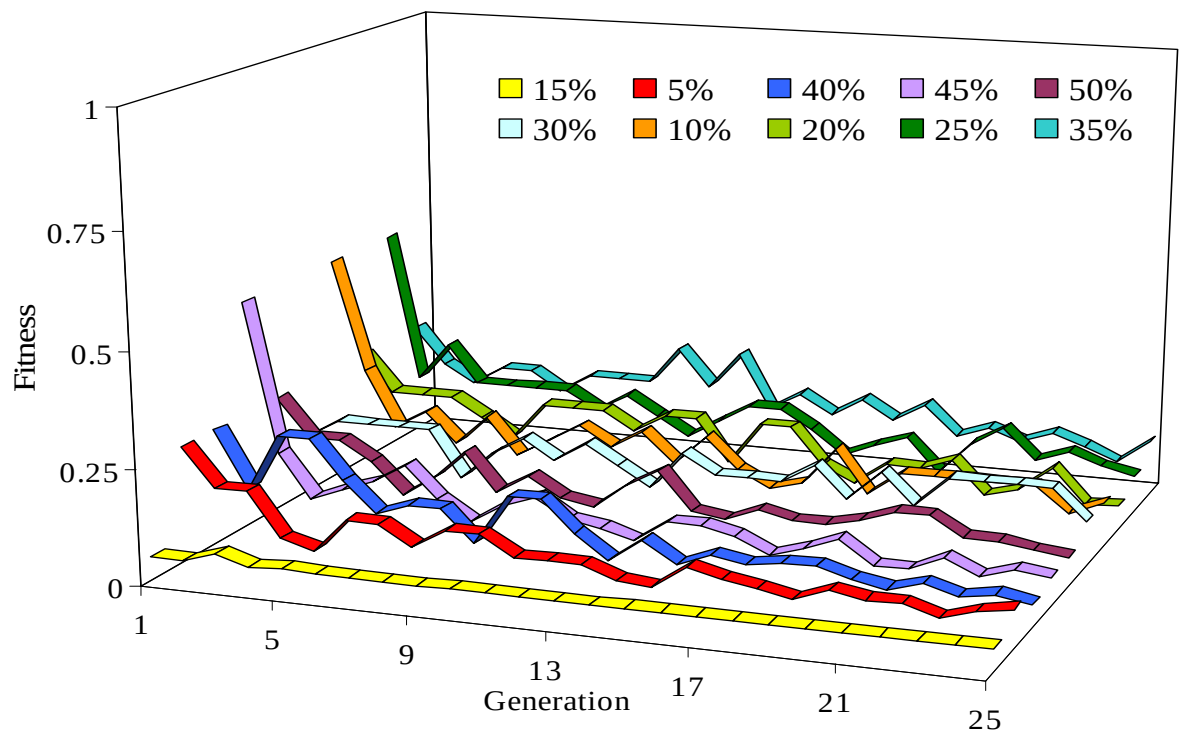
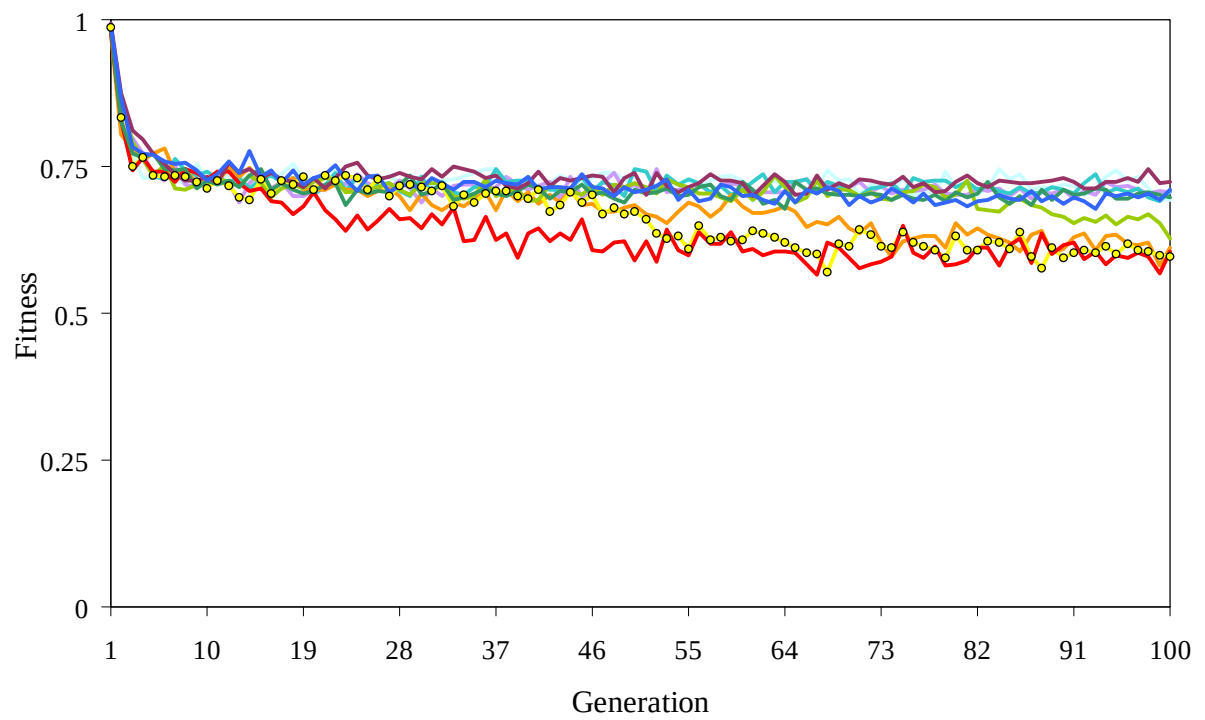
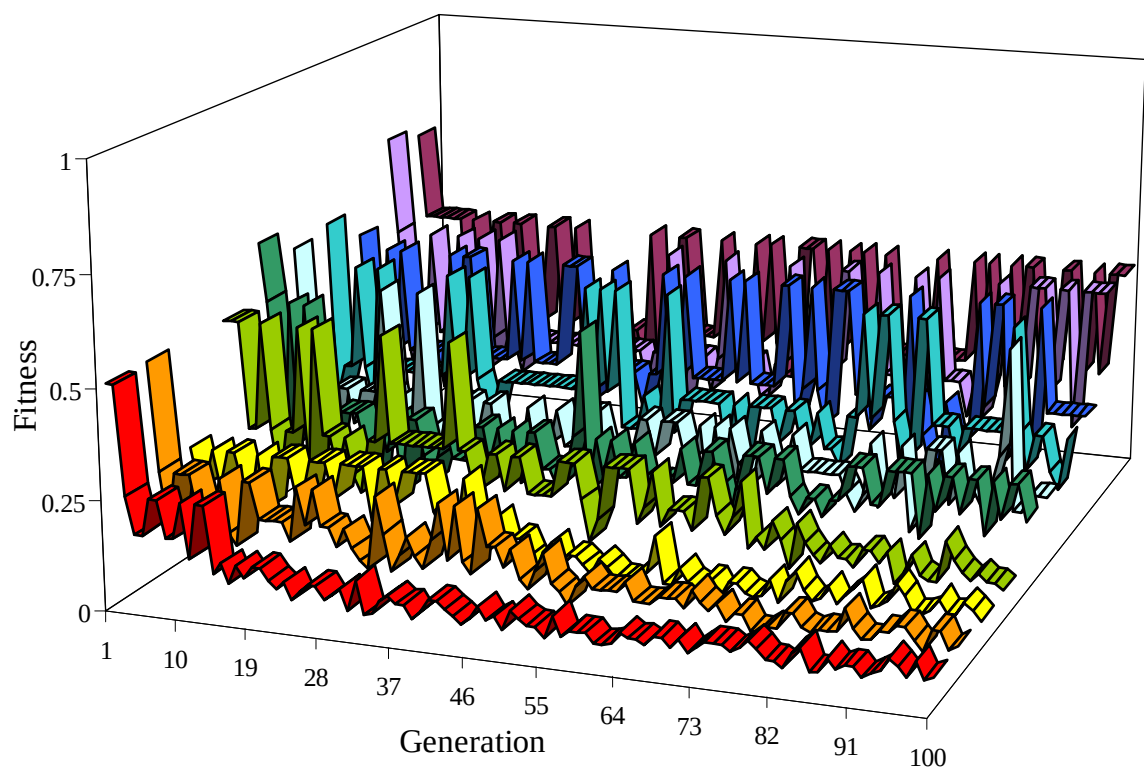


Figure 46 Best Fitness: Mutation Rate Variation

4.7.4.6 Maximum Generations

In order to determine the effect of allowing the evolutionary process to continue for an extended time, a series of 10 trials was conducted with each trial continuing the evolutionary process for 100 generations. Two graphs of the results of these trials are presented. Figure 47 shows the average fitness of the population for each of the 10 trials, and it can be seen that for more than half the trials the average fitness does not improve significantly after the tenth generation. Figure 48 shows the best individual fitness from the population after each generation for each of the trials, and presents a similar scenario to that of the average fitness values: the best fitness does not improve significantly in most of the trials after the first few generations, though for a small proportion of the trials some significantly fitter individuals are evolved. These graphs suggest that although for some instances continuing to allow the population to evolve for an extended period can produce an improved population average, and that in those instances the best performing individuals from the population are consistently better, there is no real advantage in extending the evolutionary process. In almost every case an individual from the first 10 to 15 generations achieved the equal best fitness seen over the 100 generations, so given that the objective is to find good goal-scoring behaviour there would seem to be no real advantage in extended evolution of the population.

This is similar to the result reported in [176], where Luke suggests that for some problems genetic programming encounters a critical point beyond which further evolution yields no improvement. Luke further suggests that performing many shorter experiments (i.e. fewer generations) is likely to produce better results than a single very long experiment (i.e. many generations).

**Figure 47** Average Fitness: 100 Generations**Figure 48** Best Fitness: 100 Generations

4.7.4.7 Gene Pools

The trials reported in the previous section provide an opportunity to study, in broad terms, the genetic makeup of the population being evolved. For each of the 100 generations, Figure 49 shows the average number of genes per chromosome, premises per chromosome, rules per chromosome, and valid rules per chromosome, with standard deviations for each. This graph shows that average chromosome length does not grow uncontrollably, and in fact plateaus at about 2/3 the maximum possible length. The average number of rules per chromosome, and hence the average number of consequents per chromosome, grows steadily throughout the evolutionary run. This agrees with the data presented in section 4.7.4.1 on page 108 for the maximum chromosome length variation trials. It is interesting to note that the number of rules per chromosome is still increasing after 100 generations. Since the minimum number of genes per rule is 2 the number of rules per chromosome is bounded by half the chromosome length, and although the graph shows the number of rules approaching the upper bound, it has not reached that figure after 100 generations. Though outside the scope of this thesis, some more experimentation to observe the effect of reaching the upper bound would be useful work to undertake in the future.

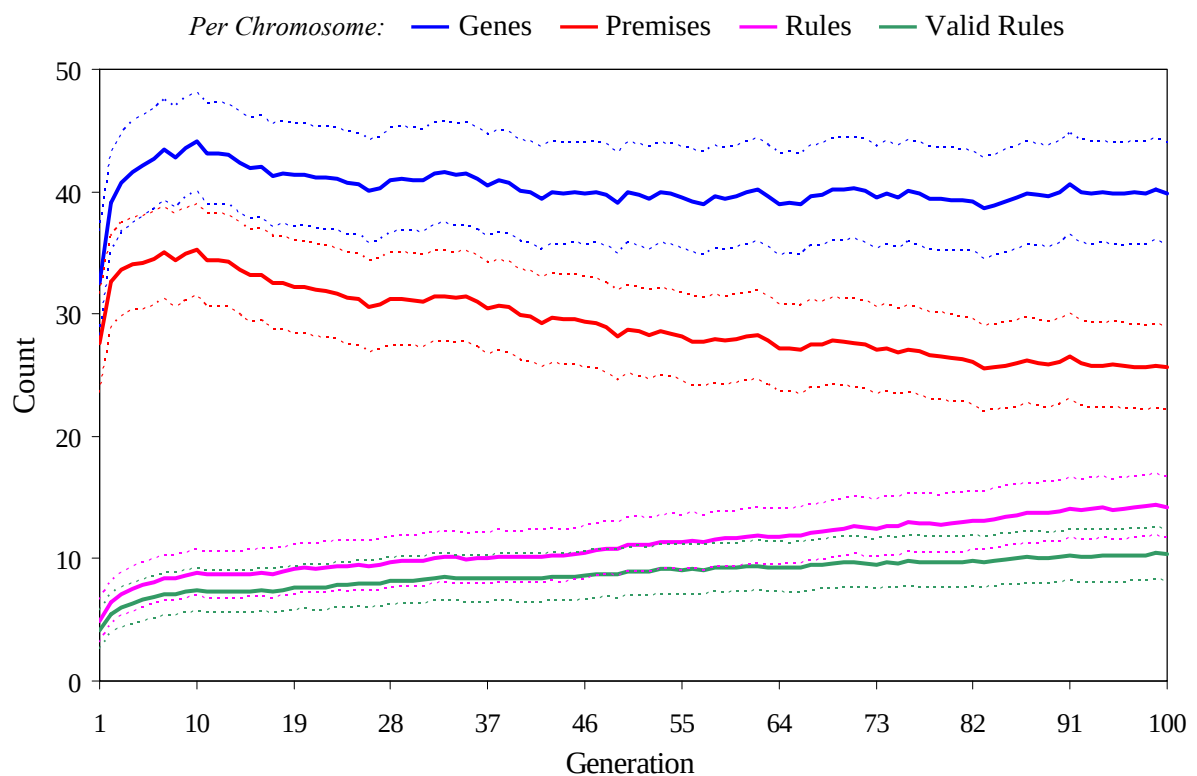


Figure 49 Population Composition Mean and Standard Deviation: 100 Generations

Figure 50 shows the population gene pool sizes throughout the evolutionary process. The graph shows raw numbers of unique premise genes, unique consequent genes, and the total gene pool available to each generation of individuals. It is evident from the graph that the pool of unique premises falls slowly, but steadily, from the first generation, while the pool of unique consequent genes stays reasonably constant for close to 40 generations after an initial decrease. This is not unexpected, and is an indication that some selection pressure is evident, but that the number of rules remains fairly constant.

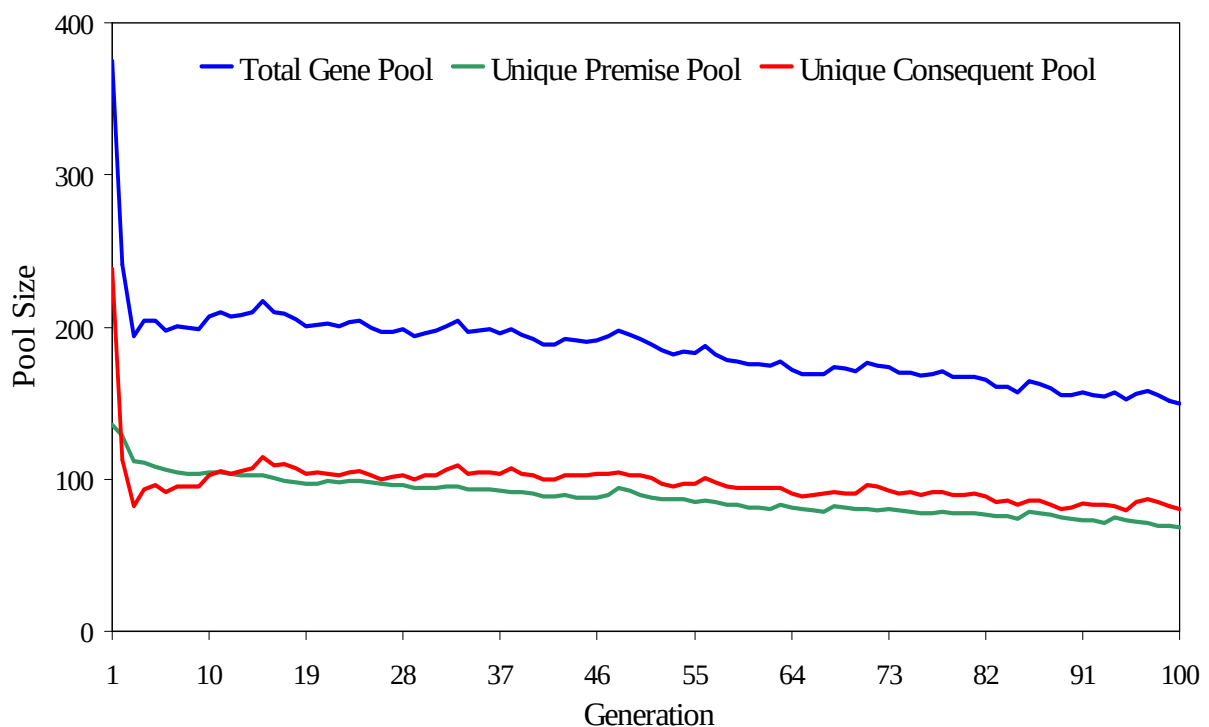


Figure 50 Gene Pools: 100 Generations

4.7.5 Performance Comparison – GA vs Random Search

In this section, in order to gauge the relative difficulty of the problem, the results obtained using the messy-coded GA search are compared to results obtained from random search.

The messy-coded GA results shown are the average of the 10 trials conducted for the “maximum generations” experiments described in section 4.7.4.6 on page 118. The random search technique was simply the random generation and evaluation of a “population” of 500 individuals repeated 40 times – to equal the number of evaluations completed for the messy-

coded GA trials. The “population” average fitness is shown in Figure 51, and the best individual fitness at intervals of 500 evaluations is shown in Figure 52. The average fitness curves are included only to illustrate that the genetic algorithm is able to consistently improve the quality of the population for the duration of the evolutionary process: random search would not be expected to perform in the same manner. The best fitness curves (Figure 52) show that although random search is able to find individuals that exhibit goal-scoring behaviour (i.e. $\text{fitness} \leq 0.5$), evolutionary search finds better individuals and finds them more consistently, indicating that evolutionary search is not only a more successful search technique than random search, it is more robust.

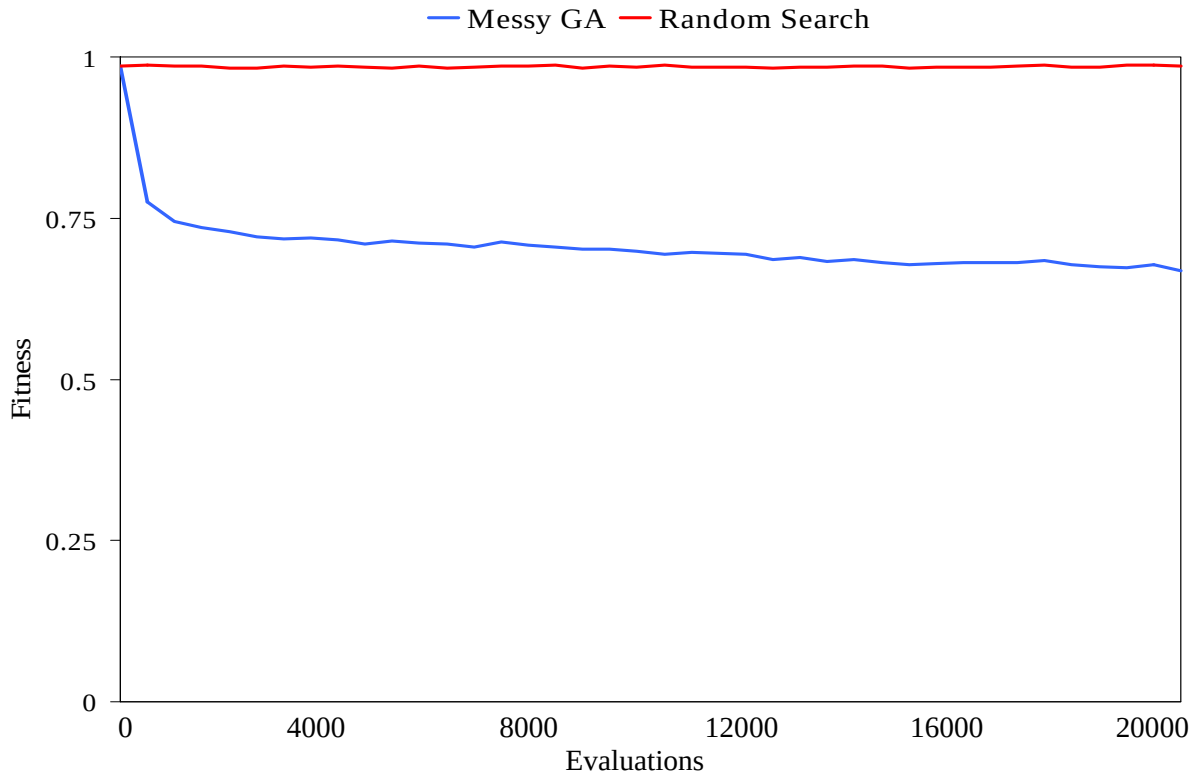


Figure 51 Average Fitness: Random Search Comparison

As noted, random search does successfully find individuals with reasonably good goal-scoring skills. This result is a little surprising at first glance, but on closer inspection of the problem an explanation does present itself – the problem of finding goal-scoring behaviour *in the solution space defined*, whether by evolutionary search or random search, is not as difficult as it first seems, and this is because the solution space defined is not the one envisaged

when first considering the problem. The solution space defined is one populated by players with mid-level, hand-coded skills available to them, as well as a “smart” default hunt action, which is a much richer solution space than the one usually envisaged when considering the problem of evolving goal-scoring behaviour “from scratch”. As evidenced by the results of the random search shown here, the density of “reasonably good” solutions in the solution space is sufficiently high that random search will occasionally, and with some consistency, find one of those “reasonably good” solutions.

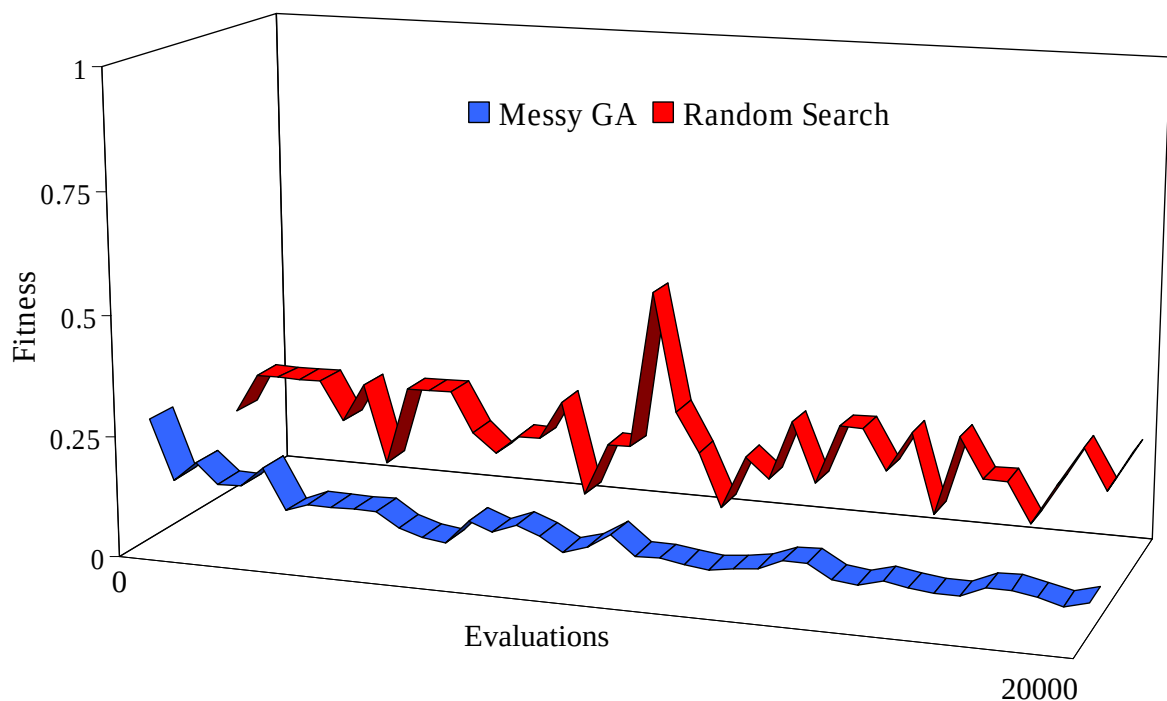


Figure 52 Best Fitness: Random Search Comparison

4.7.6 Rulesets Evolved

Examples of the best rulesets evolved in both the RoboCupSoccer and SimpleSoccer environments are presented in the following sections. These evolved players typically begin the game by hunting for the ball, then once the ball is located the players generally dribble the ball towards the goal in a reasonably direct route. Because the player developed is reactive and almost no state information is recorded (by the player), there are times when it loses sight of the ball or goal. These situations are characterised by the player momentarily hunting for the ball, or kicking the ball in the wrong direction.

4.7.6.1 RoboCupSoccer Ruleset

The rules evolved by the genetic algorithm for a typical best performing player from the initial 20 RoboCupSoccer trials are:

```

if    Goal is VeryNear or
      (Goal is At and Goal is VeryRight) or
      (Goal is Right and Ball is SlightlyLeft and Ball is Near and Ball is VeryLeft) or
      Goal is Right180 or
      Ball is Far or
      Goal is Near
then  Kick VerySoft

```

```

if    Goal is VeryNear or
      (Ball is VeryNear and Ball is SlightlyRight and Ball is Far)
then  GoToBall SlightlyHard

```

```

if    Goal is not VeryRight or
      Ball is VeryFar
then  DribbleTowardGoal Soft

```

```

if    Goal is MediumDistant or
      (Goal is not Left and Ball is not Right180) or
      Goal is SlightlyRight or
      Goal is VeryLeft or
      (Goal is VeryFar and Goal is not Left180) or
      (Ball is Left180 and Goal is SlightlyNear)
then  DribbleTowardGoal MediumPower

```

```

if    Goal is VeryNear or
      (Goal is Left180 and Ball is Near and Goal is Right)
then  Dribble Hard

```

The player defined by this ruleset achieved a fitness value of 0.1667 by kicking 3 goals in the allotted game time of 120 seconds. The best performing players from the trials were each tested in 100 games of 120 seconds in the RoboCupSoccer environment, with the player being placed in a different, randomly selected starting position for each game. The best performing players in these tests scored one or more goals in 60% of the games which, given the stochastic nature of the RoboCupSoccer environment, is a reasonable result.

4.7.6.2 SimpleSoccer Ruleset

The rules evolved by the genetic algorithm for a typical best performing player from the initial 20 SimpleSoccer trials are:

```
if    Ball is not Near
then  DribbleTowardGoal SlightlyHigh
```

```
if    (Goal is SlightlyNear and Goal is not Left and Ball is Near and Goal is Near and
      Goal is VeryNear and Goal is Left180 and Ball is not SlightlyFar) or
      Goal is VeryNear or
      Goal is Right or
      Ball is VeryNear or
      Ball is SlightlyLeft or
      Ball is SlightlyFar
then  Dribble Low
```

```
if    Ball is not SlightlyNear or
      Ball is not VeryLeft or
      Ball is MediumDistant or
      Ball is VeryRight or
      Goal is SlightlyLeft or
      Goal is SlightlyNear
then  GoToBall SlightlyLow
```

```
if    Ball is Far or
      Ball is Right or
      Ball is not VeryNear or
      (Ball is not Right180 and Goal is Near) or
      (Goal is VeryRight and Ball is Straight and Ball is VeryLeft and Ball is MediumDistant)
then  GoToBall VeryLow
```

```
if    Ball is SlightlyNear and Goal is VeryRight or
      Ball is VeryLeft or
      Ball is SlightlyLeft or
      Goal is not SlightlyNear or
      Goal is MediumDistant or
      Ball is Straight and Goal is VeryNear or
      Ball is Left and Ball is VeryRight or
      Ball is SlightlyLeft and Ball is Right
then  Dash VeryLow
```

```

if    (Ball is VeryLeft and Goal is Left180)
then  Turn SlightlyLeft

if    Ball is Right
then  DribbleTowardGoal VeryHigh

if    (Goal is Right180 and Ball is SlightlyRight)
then  DribbleTowardGoal VeryLow

if    (Ball is Straight and Ball is VeryRight) or
      Goal is MediumDistant
then  Dribble High

if    (Ball is not SlightlyFar and Goal is SlightlyFar) or
      Ball is At
then  RunTowardBall VeryHigh

```

The player defined by this ruleset achieved a fitness value of 0.0625 by kicking 8 goals in the allotted game time of 1000 ticks. The best performing players from the trials were each tested in 100 games of 1000 ticks in the SimpleSoccer environment, with the player being placed in a different, randomly selected starting position for each game. The best performing players in these tests scored one or more goals in 42% of the games. This is somewhat lower than for RoboCupSoccer, and consistent with the higher population average fitness values seen in the SimpleSoccer trials: behaviours evolved in the RoboCupSoccer environment generally seem to be more robust than those evolved in the SimpleSoccer environment. This is very likely due to rulesets evolved in the simpler, less stochastic nature of the SimpleSoccer environment being less able to cope with situations not previously encountered.

4.8 Summary and Discussion

This chapter of the thesis has provided an implementation and empirical analysis of the fuzzy logic-based robot soccer player and the messy-coded genetic algorithm training algorithm. The specific research question examined in this chapter was:

How can a genetic algorithm be used to evolve a set of fuzzy rules to govern the behaviour of a simulated robot soccer player that produces consistent goal-scoring behaviour?

In relation to this question, the outcomes are:

The research question was answered by presenting a description of the implementation of a messy-coded genetic algorithm which successfully evolves the ruleset for a fuzzy logic-based simulated robot soccer player. Several trials were performed to test the capacity of the method to produce goal-scoring behaviour. The results of the trials performed indicate that the player defined by the evolved fuzzy rules of the controller is capable of displaying consistent goal-scoring behaviour. This outcome indicates that for the problem of developing goal-scoring behaviour in a simulated robot soccer environment, when the initial population is endowed with a set of mid-level hand-coded skills, taking advantage of the flexible representation afforded by the messy-coded genetic algorithm and combining that with a fuzzy logic-based controller enables a fast and efficient search technique to be constructed.

Several experiments were performed to vary the genetic algorithm parameters being studied. The results of those tests indicate that within the range of the values tested, most parameters have little effect on the performance of the search. The *Maximum Chromosome Length* and *Selection Method* parameters had a marginal influence over the efficacy of the search, and although better performance was sometimes achieved after a long period of evolution, the *Maximum Generations* parameter is not considered to have a large effect on the performance of the algorithm after an upper bound of about 15 generations. Table 6 lists the default values for the genetic algorithm parameters for the experiments reported in this chapter, the empirically determined optimum values for those parameters, and the impact on performance of varying those parameter values.

Parameter	Default Parameter Value	Impact on Performance	Optimum Parameter
Maximum Chromosome Length	64	Slight	50 -75 genes
Population Size	200	None	Not relevant
Maximum Generations	25	Slight	15
Selection Method	Roulette Wheel	Slight	Elitist
Crossover Method	Single Point	None	Not relevant
Mutation Rate	10%	Slight	15%

Table 6 Empirically Determined Optimal GA Parameters

Everything should be made as simple as possible, but not simpler.

- Albert Einstein (attrib.)

Chapter 5

Accelerating Evolutionary Search for RoboCupSoccer

5.1 Introduction

The RoboCupSoccer simulated soccer league is a dynamic, complex and uncertain environment which presents many challenges to machine learning techniques, including evolutionary search methods. The asynchronous design of the RoboCupSoccer simulation environment can create long and unpredictable delays in the effects of actions, often causing onerous training times. A typical RoboCupSoccer training trial over 25 generations with a population of 200 runs for approximately 100 hours.

It has been suggested that the performance of a neighbourhood search algorithm can be affected by the quality of the initial solution, and that high quality starting solutions will yield better final solutions than will low quality, randomly chosen starting solutions (see section 2.2.3 on page 20). Furthermore, it has been noted that by providing the search algorithm with better starting solutions the performance of the algorithm is improved and search time may be decreased (see section 2.2.3 on page 20).

The objective in mind when developing the SimpleSoccer environment was to create an environment that retains much of the dynamics and some of the complexity of RoboCupSoccer, but provides more certainty and less delay, so increasing the viability of search and learning techniques for robot soccer. A goal was to make the SimpleSoccer environment complex and dynamic enough that while low-level tactics may differ due to the removal of systematic

uncertainty, high-level strategies directly applicable to RoboCupSoccer might be developed. The Ascii Soccer environment (see section 3.3 on page 79), while much simpler than the RoboCup Soccer environment, was not considered a sufficiently good model of the RoboCupSoccer environment to achieve the goals and objectives of the SimpleSoccer environment.

Having developed the SimpleSoccer environment, it is believed that an effective approach to evolving goal-scoring behaviour in a robot soccer environment is to use SimpleSoccer as an heuristic for developing high quality initial solutions for the genetic search in the more complex and time-consuming RoboCupSoccer environment.

5.2 Chapter Goals

This chapter of the thesis seeks to address the question of whether, for the robot soccer problem of evolving goal-scoring behaviour, learning in the SimpleSoccer environment is transferable to the more complex RoboCupSoccer environment, and whether the transfer of that learning improves either, or both, the quality and speed of the evolutionary search in the RoboCupSoccer environment. The specific research question examined in this chapter of the thesis is:

Will seeding the initial population for the genetic algorithm with rules evolved in the SimpleSoccer environment improve the quality and speed of the evolutionary process in the more complex RoboCupSoccer environment?

In order to answer the research question and test the conjecture that rulesets evolved in the SimpleSoccer environment can be used to improve the quality and speed of the evolutionary process for RoboCupSoccer, a series of experiments is conducted in the RoboCupSoccer environment for which the initial population is seeded with individuals previously evolved in the SimpleSoccer environment. The experiments are described in detail, the results presented, and the performance of the evolutionary search compared with that previously seen with randomly generated initial populations in the RoboCupSoccer environment. Conclusions are drawn as to the effect of seeding the initial population on the performance of the evolutionary search.

5.3 Experimental Overview

A series of experiments was conducted for RoboCupSoccer where the initial populations were seeded with different numbers of individuals previously evolved in the SimpleSoccer environment. Each experiment consisted of 10 trials, and the results reported for each trial are the averages of the 10 trials.

5.3.1 Control Parameters and Fitness Function

With the exception of the number of generations, the genetic algorithm parameters used in these experiments were, for the sake of consistency, kept the same as those used in the trials presented in Chapter 4, and are shown in Table 7, and the RoboCupSoccer parameters used are listed in Table 8. The number of generations was reduced in anticipation of the evolutionary search being shortened due to the seeding of the initial population. The fitness function used was the same as that used in the RoboCupSoccer trials presented in Chapter 4 (see Equation 6 on page 98).

Parameter	Value
Maximum Chromosome Length	64 genes
Population Size	200
Maximum Generations	5
Selection Method	Roulette Wheel
Crossover Method	Single Point
Crossover Probability	0.8
Mutation Rate	10%
Mutation Probability	0.35

Table 7 Genetic Algorithm Control Parameters: Seeded Trials

RoboCupSoccer Parameter	Value
Target Fitness	0.05 (10 goals)
Maximum Game Time	120 seconds
Terminate if No Action after	10 seconds

Table 8 RoboCupSoccer Control Parameters: Seeded Trials

5.3.2 Seed Rulesets

The individuals from the SimpleSoccer environment used to seed the RoboCupSoccer initial population were the best individuals from each of the 20 SimpleSoccer trials described in section 4.6.1 on page 97 and for which the results are presented in section 4.7.2 on page 105. For each experiment, a different number of individuals were placed in the initial population as seeds, and in all but the first experiment this involved placing multiple copies of the individuals in the initial population. In each case the remainder of the population was randomly generated. Table 9 shows the number of seeds used in each experiment.

Experiment	Number of Seeds in Initial Population	Number of Unique Seeds	Number of Copies of Individual Seeds
1	20	20	1
2	60	20	3
3	100	20	5
4	200	20	10

Table 9 Initial Population Composition: Seeded Trials

The results of the experiments are shown in Figure 53 (page 132) and Figure 54 (page 133), with the population average fitness for each of the 10 trials presented in Figure 53 and the individual best fitness shown in Figure 54. For convenience the average fitness for the five best trials using randomly generated initial populations is also shown in Figure 53 (data taken from Figure 29 on page 103). The results show clearly that, compared with a randomly generated initial population, the evolutionary search in the RoboCupSoccer environment is positively affected by seeding the initial population with rulesets evolved in the SimpleSoccer environment. The results obtained by seeding the initial population compare very favourably with the initial RoboCupSoccer results presented in Figure 29 and Figure 30 (page 104), with the population average fitness for the seeded trials being half that of the initial RoboCupSoccer trials and still trending down at the completion of the five generations. The individual best fitness values present a similar scenario, with the best fitness in each of the seeded trials not only below those of the initial RoboCupSoccer trials, but more stable, indicating that the results of the seeded trials are more robust.

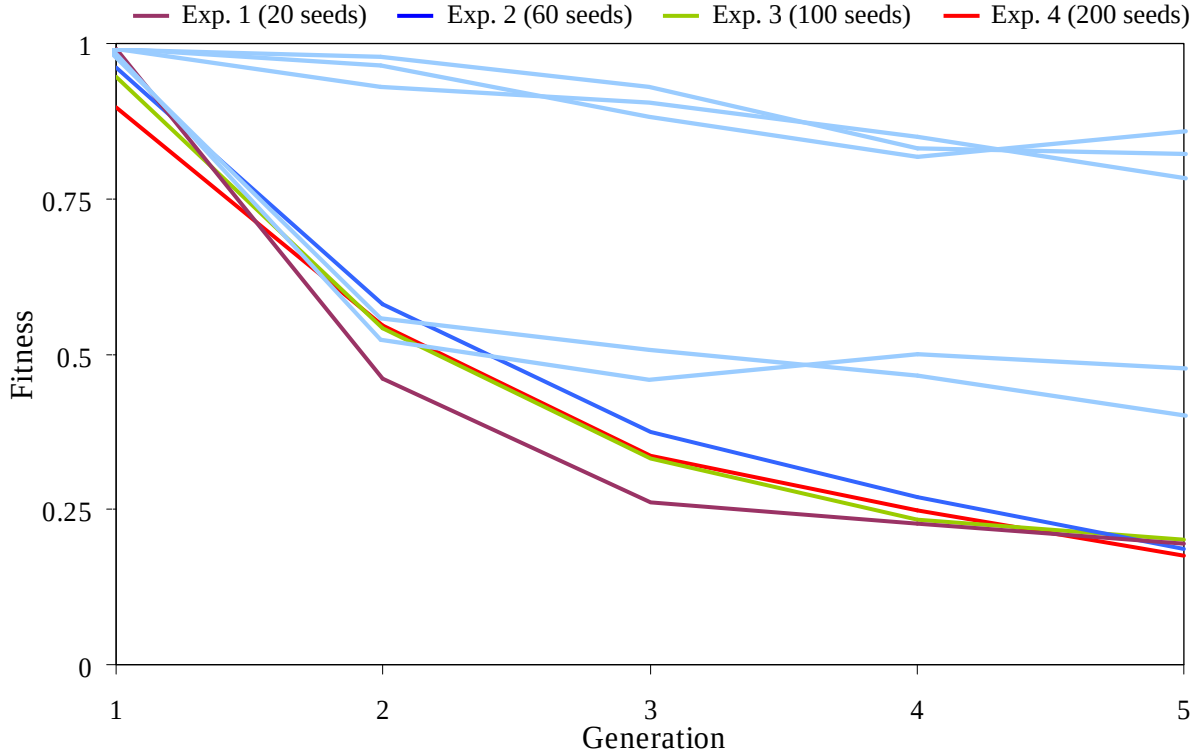


Figure 53 Average Fitness: Seeded Trials

The performance of the initial populations in each of the experiments is very similar, with the population with the smallest number of seeds (experiment 1) resulting in a slightly lower population average fitness for most of the period of the evolutionary search. Close inspection of Figure 53 reveals that experiment 4 finished with a lower population average fitness after the five generations, and in fact was trending lower than the other experiments. Of particular interest is that all experiments displayed a high population average fitness for the initial generations indicating, especially for the initial populations composed entirely of seeded rulesets (experiment 4), that the overall behaviour of the rulesets evolved in the SimpleSoccer environment is not directly transferable to RoboCupSoccer. However, the fact that the population average fitness falls very quickly for all experiments suggests that individual behaviours of individual rules is directly transferable and provides good material for evolutionary learning.

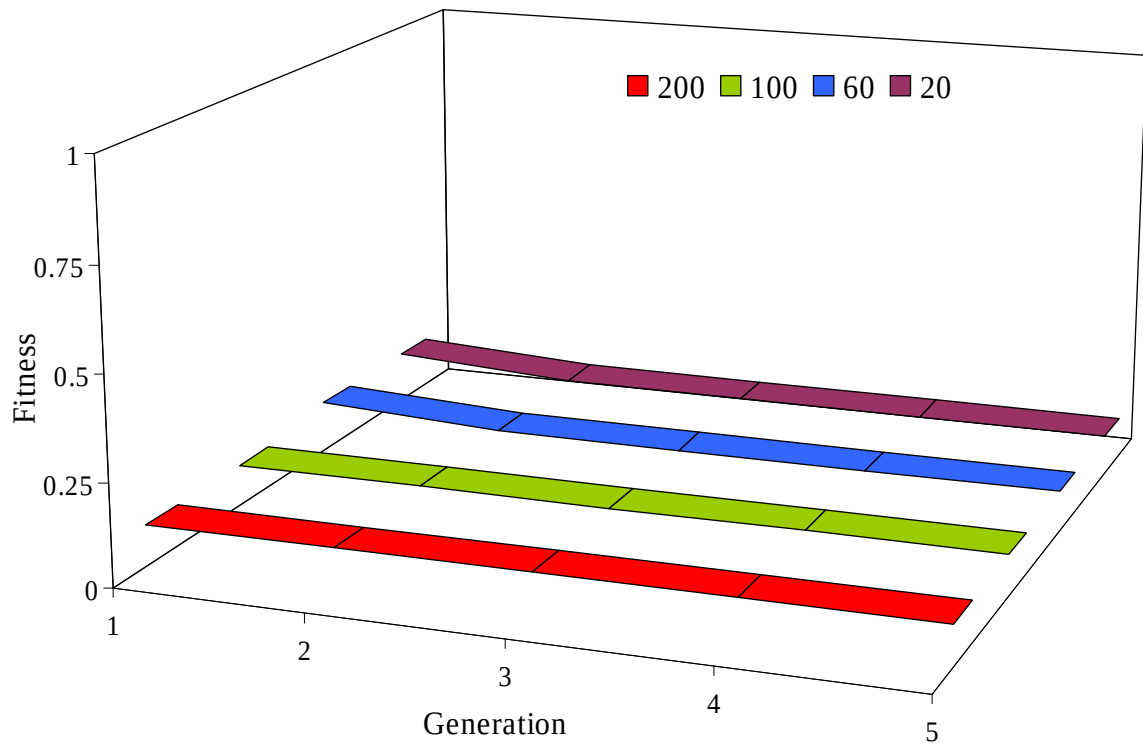


Figure 54 Best Fitness: Seeded Trials

Figure 55 on page 134 is a visualisation of the progressive learning of the population from generation to generation. The histogram shows, for experiment 4 of the seeded trials, the percentage of the population which scored 0, 1, 2, 3, 4 or more than 4 goals for each generation. The maximum number of goals scored by any individual was 5. When comparing this histogram with that of the initial RoboCupSoccer trials (Figure 31 on page 104) it is evident that the quality of the individuals in the population is very much higher in the seeded trials than the initial RoboCupSoccer trials, and that the improvement in goal-scoring ability is much more rapid.

The motivation for creating the SimpleSoccer environment was the prohibitive time to train players in the real-time RoboCupSoccer environment. Table 10 on page 135 shows the

average number of seconds of real time for a single fitness evaluation in each of the environments used to evolve players for robot soccer.

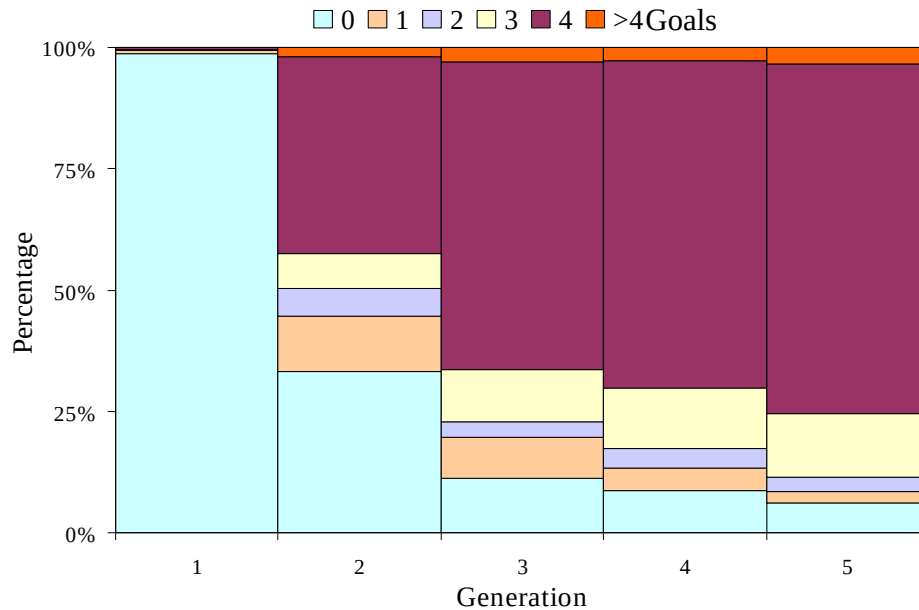


Figure 55 Seeded Trials: Frequency of Individuals Scoring Goals

The RoboCupSoccer simulator used throughout the work conducted for this thesis was version 7.10, compiled and executed on an HP 9000/777 workstation running HP-UX version 11.0. The SimpleSoccer simulator was originally developed on an HP9000/777 workstation running HP-UX version 11.0, and was later ported to an Intel Pentium-based PC running Windows XP. Evaluation times are quoted for each of those systems. Note that although the SimpleSoccer evaluation time is two orders of magnitude smaller on the PC, RoboCupSoccer evaluation times would not enjoy the same improvement if executed on the PC – the RoboCupSoccer evaluation times are constrained by the real-time nature of the simulator, and the training game times were 60 seconds. Any benefit from running the RoboCupSoccer simulations on faster hardware would be evident in the few seconds of overhead time only, and would not significantly reduce the evaluation time.

It is clear from the data shown in Table 10 that there is significant benefit in using the SimpleSoccer environment as an heuristic to generate high quality initial solutions for the RoboCupSoccer environment. For example, when run on an HP9000 workstation, a RoboCupSoccer experiment for a population of 100 individuals and consisting of 10 generations consumes approximately 19.6 hours for the fitness evaluations. The same

experiment on the same hardware in the SimpleSoccer environment consumes only 2.8 hours. The evolution of players displaying reasonable goal-scoring behaviour is achievable in the SimpleSoccer environment in a fraction of the time it would take in the RoboCupSoccer environment, and only a few generations are required in RoboCupSoccer to refine the behaviours evolved in the SimpleSoccer environment.

Simulator	Platform	Seconds/ Evaluation
RoboCupSoccer	HP 9000/777 workstation. 120MHz PA-7200 CPU, 256MB RAM, HP-UX 11.0 Operating System	70.65
SimpleSoccer	HP 9000/777 workstation. 120MHz PA-7200 CPU, 256MB RAM, HP-UX 11.0 Operating System	10.20
SimpleSoccer	Compaq PC. 1.6GHz Pentium M CPU, 512MB RAM, Windows XP Operating System	0.112

Table 10 Evaluation Times

5.4 Summary and Discussion

This chapter of the thesis has described a series of experiments, conducted in the RoboCupSoccer environment, for which the initial population was seeded with individuals previously evolved in the SimpleSoccer environment. The specific research question examined in this chapter was:

Will seeding the initial population for the genetic algorithm with rules evolved in the SimpleSoccer environment improve the quality and speed of the evolutionary process in the more complex RoboCupSoccer environment?

With regard to the research question being addressed, the experiments performed and the data presented indicate clearly that rules evolved in the SimpleSoccer environment can be used to seed the initial population of solutions and improve both the quality and the speed of the evolutionary process in the more complex RoboCupSoccer environment. The results demonstrate that the average quality of the final population of solutions evolved from an initial population seeded with rulesets evolved in the SimpleSoccer environment is superior to those evolved exclusively in the RoboCupSoccer environment. Moreover, the individual best fitness values achieved by players evolved from a seeded population are superior to those evolved

from a randomly generated initial population. That is, players evolved from a seeded population kick more goals more often than players evolved from randomly generated solutions, providing a positive outcome in relation to the research question.

Nothing in the body is made in order that we may use it.

What happens to exist is the cause of its use.

- Lucretius, De nat. rerum, IV, 833; cf. 822-56.

Chapter 6

Varying Player Skills and the Fitness Function

6.1 Introduction

Previous work in the area of evolving goal-scoring behaviour has resulted in varying degrees of success and a general consensus that this is a “difficult” problem (see section 2.6.3 on page 59). This is somewhat at odds with the results reported in Chapter 4 of this thesis, but the diversity of results could be explained in part by a difference in the level of soccer-playing or goal-scoring ability inherent in the players evolved in the work reported in the literature.

This chapter describes a series of experiments, conducted in the SimpleSoccer environment, for which a number of control parameters and player attributes were varied. These experiments were conducted in order to study the effect that varying the control parameters and player skills has on the progress and outcome of the evolutionary search. The experiments are described in detail, the results presented, and the performance of the evolutionary search in each of the experiments compared. Conclusions are drawn as to the effect of varying the control parameters and player attributes on the performance of the evolutionary search.

Evolutionary algorithms have been, and continue to be, applied with mixed success to a wide range of search and optimisation problems. An ongoing debate about what constitutes a hard problem for evolutionary algorithms, and how such hard problems might be identified, continues without clear resolution. The work presented in this chapter seeks to address one aspect of this problem. Rather than attempt to identify and categorise hard problems, this work

shows how difficult problems are, often necessarily and sometimes inadvertently, made easier for the evolutionary algorithm to solve. The simplifying of problems for evolutionary algorithms is most often achieved in one of two ways: either by ensuring the initial population of solutions is close to the desired outcome thereby reducing the amount of evolution necessary, or by using a composite fitness functions to decompose the problem into several smaller, more easily solved problems. Often the problem parameters are changed as it is found the evolutionary algorithm is unable to solve the problem – an initial set of skills or primitives is replaced with a more intelligent set, which is replaced with an even more intelligent set until the evolutionary algorithm is able to solve the problem.

The experiments reported in this chapter were conducted in the SimpleSoccer environment and are divided into two broad sections: *varying player skills* and *varying the fitness function*.

6.2 Chapter Goals

The specific research question examined in this chapter of the thesis is:

How does varying the

- *skills available to the players, and/or*
- *the implementation of the fitness function*

affect the result of the evolutionary search?

To examine the extent to which the quality of the initial population (as defined in section 1.5.2 on page 8) and the skills available to players affects the efficacy of the evolutionary search, and thus addressing the first part of the research question, a series of experiments was conducted for which the skills available to the players being evolved was varied, and the performance of the evolutionary search compared.

The second part of the research question was addressed by conducting a number of experiments in which different fitness functions were implemented and comparing the results. These experiments help to examine the issue of using a composite fitness function to decompose a very difficult problem into smaller, less difficult problems.

6.3 Available Player Skills

The skills available to players being evolved are those listed in section 3.4 on page 82, and different combinations of those skills are given to players in the initial populations for each of the trials reported.

In order to encourage the creation of mid- and high-level skills by giving the players as much flexibility as possible, in the experiments conducted for this chapter the SimpleSoccer simulator was configured for native mode operation. This means that instead of the output of the fuzzy inference system being a number of *(action, value)* pairs, the output will be a number of *(action, value, value)* triples, corresponding to the number of fuzzy rules with unique consequents. The added *value* argument is to allow for the SimpleSoccer native mode commands that require both *direction* and *power* arguments. For commands that require only two arguments (e.g. *KickTowardBall*, *RunTowardBall*, *Turn*), the second *value* argument is not required and so set to *NULL*. The *(action, value, value)* triples define the action to be taken by the player, and the degree to which the action is to be taken. For example:

(Kick, direction, power)
(Dash, direction, power)
(KickTowardBall, NULL, power)
(RunTowardBall, NULL, power)
(Turn, NULL, direction)

where *power* and *direction* are crisp values representing the defuzzified fuzzy set membership of the action to be taken.

Only one action is performed by the player in response to stimuli provided by the soccer server. Since several rules with different actions may fire, the action with the greatest level of support, as indicated by the value for truth of the antecedent, is selected.

In addition, two new directions were added to the set of possible directions available to players listed in section 4.5 on page 93. The additional directions are *TowardBall* and *TowardGoal* and, as their names suggest, they provide the direction, relative to the player's facing direction, to the ball and player's goal respectively. The two new directions are available only for rule consequents – they are not defined for rule premises – and are only relevant to the *Turn*, *Kick* and *Dash* consequent actions.

The problem was made more difficult than that examined in Chapter 3 by limiting the player's field of vision (see section 3.3 on page 79 for a description of a player's visual capabilities). For the experiments reported in this chapter the player's view angle was reduced from 135° to 90° , and the view length from 15 cells to 5 cells. This means that for these experiments the player is less likely to be able to see the ball or goal at any particular time, and more likely to lose sight of the ball after kicking. By limiting the player's field of vision the *Hunt* skill becomes more important than it was previously.

6.4 Observations from Previous Trials

An interesting observation from the results of the RoboCupSoccer and SimpleSoccer trials presented in Chapter 3 is that for the typical best performing player from each of the environments, the consequent actions appearing most in the rulesets are *DribbleTowardGoal*, *GoToBall* and *Dribble*. In the ruleset evolved in the RoboCupSoccer environment shown in section 4.7.6.1 on page 124, of the five consequent actions, two are *DribbleTowardGoal* with the remaining three being one each of *GoToBall*, *Dribble* and *Kick*. Similarly, in the ruleset evolved in the SimpleSoccer environment shown in section 4.7.6.2 on page 125, of the ten consequent actions, three are *DribbleTowardGoal*, two are *GoToBall*, two *Dribble*, and one each of *RunTowardBall*, *Dash* and *Turn*.

In the following sections, the results of several experiments conducted to gauge the extent to which varying the skills available to players affects the performance of the search algorithm will be reported. Determining which skills have the greatest effect is an important part of those results.

6.5 Varying Player Skills

Four sets of experiments were conducted to determine the effect on the performance of the evolutionary search of varying the skills available to the player. So that the effect of the player's default action could also be measured separately, each set of experiments was conducted with a different default action defined for the player.

The default actions tested are:

- Hunt Action 1 – *Goto Ball*
- Hunt Action 2 – *Locate Ball*
- Hunt Action 3 – *Random Turn*
- No default action

The hunt actions are as described in section 3.4.1 on page 82.

GA Parameter	Value
Maximum Chromosome Length	64 genes
Population Size	500
Maximum Generations	500
Selection Method	Elitist
Elitist Retention Percentage	2.5
Crossover Method	Single Point
Crossover Probability	0.95
Mutation Rate	10%
Mutation Probability	0.15

Table 11 Genetic Algorithm Control Parameters: SkillSet Variation

SimpleSoccer Parameter	Value
Target Fitness	None
Maximum Game Time	1000 ticks
Terminate if No Action after	100 ticks

Table 12 SimpleSoccer Control Parameters: Skillset Variation

The genetic algorithm parameters used in these experiments are listed in Table 11. The SimpleSoccer parameters used in these experiments are listed in Table 12. Note that early stopping was not implemented for these experiments. The fitness function used in these experiments is the function shown in Equation 7 on page 99 (see section 4.6.2.3 on page 100 for a discussion on the relationship between the fitness value and goals scored).

Default Action	Skills Additional to Base Skills	Experiment													
		1	2	3	4	5	6	7	8	9	10	11	12	13	14
Hunt Action 1	RunTowardGoal		X	X	X	X	X	X	X						
	RunTowardBall		X	X	X	X	X	X		X					
	GoToBall			X	X	X	X	X			X				
	KickTowardGoal				X	X	X	X				X			
	DribbleTowardGoal					X	X	X					X		
	Dribble						X	X						X	
	Added Directions							X							X
Hunt Action 2	RunTowardGoal		X	X	X	X	X	X	X						
	RunTowardBall		X	X	X	X	X	X		X					
	GoToBall			X	X	X	X	X			X				
	KickTowardGoal				X	X	X	X				X			
	DribbleTowardGoal					X	X	X					X		
	Dribble						X	X						X	
	Added Directions							X							X
Hunt Action 3	RunTowardGoal		X	X	X	X	X	X	X						
	RunTowardBall		X	X	X	X	X	X		X					
	GoToBall			X	X	X	X	X			X				
	KickTowardGoal				X	X	X	X				X			
	DribbleTowardGoal					X	X	X					X		
	Dribble						X	X						X	
	Added Directions							X							X
No Default Action*	RunTowardGoal		X	X	X	X	X	X	X						
	RunTowardBall		X	X	X	X	X	X		X					
	GoToBall			X	X	X	X	X			X				
	KickTowardGoal				X	X	X	X				X			
	DribbleTowardGoal					X	X	X					X		
	Dribble						X	X						X	
	Added Directions							X							X

* added *BALL is/is not VISIBLE* premise construct and *HUNT* consequent action (see section 6.5.4 on page 152)

Table 13 Player Skills

Each series of experiments was performed with a varied set of skills available to the player. A base set of skills was available to the player in all trials, consisting of the skills *Turn*, *Kick*, *Dash* and *DoNothing*. Additional skills were as detailed in Table 13. For example, in experiment 1 the players were given the base level of skills only (turn, kick and dash), and for experiment 2 the *RunTowardGoal* and *RunTowardBall* skills were added to the base skills, then for experiment 3 the *RunTowardGoal*, *RunTowardBall* and *GoToBall* skills were added to the base skills, etc. For each of the 14 experiments listed in Table 13, a series of 10 trials was conducted, and the results presented are the averages of those 10 trials. Note that it is not the aim of these experiments to find the best set of skills for goal-scoring behaviour (though that may be a reasonable aim at some later time), but to determine the effect of changing the skills available to the player.

To improve the clarity of the graphs presented, the results are presented in two graphs: experiments 1 through 7 in one graph and experiments 8 through 14 in another. The graph legends are ordered by performance left to right – the best performing trials are listed first on the graph legends. The results of these experiments are presented in detail in the following sections. Any deviations from the control parameters and player skills documented are noted with the results.

6.5.1 Default Action: *Hunt Action 1 – Goto Ball*

The first series of experiments was performed in an environment where the default action for the player was defined as *Hunt Action 1 – Goto Ball*. Each skillset shown in Table 13 (experiments 1 -14) was evaluated with the default action *Hunt Action 1*. The average fitness of the population after each of the 500 generations is shown in Figure 56 (experiments 1 – 7, page 144) and Figure 57 (experiments 8 – 14, page 144). The best fitness achieved by any member of the population during each of the 500 generations is shown in Figure 58 (experiments 1 – 7, page 145) and Figure 59 (experiments 8 – 14, page 146).

The data shows that, for this series of experiments, all players display goal-scoring behaviour to some degree. While the player without any hand-coded mid-level skills is able to occasionally score goals, the results indicate that being endowed with one or more of those hand-coded skills enables players to more consistently display goal-scoring behaviour.

The results indicate that *KickTowardGoal* and *DribbleTowardGoal* are the most useful hand-coded skills with regard to goal-scoring behaviour, and upon reflection this is somewhat intuitively obvious.

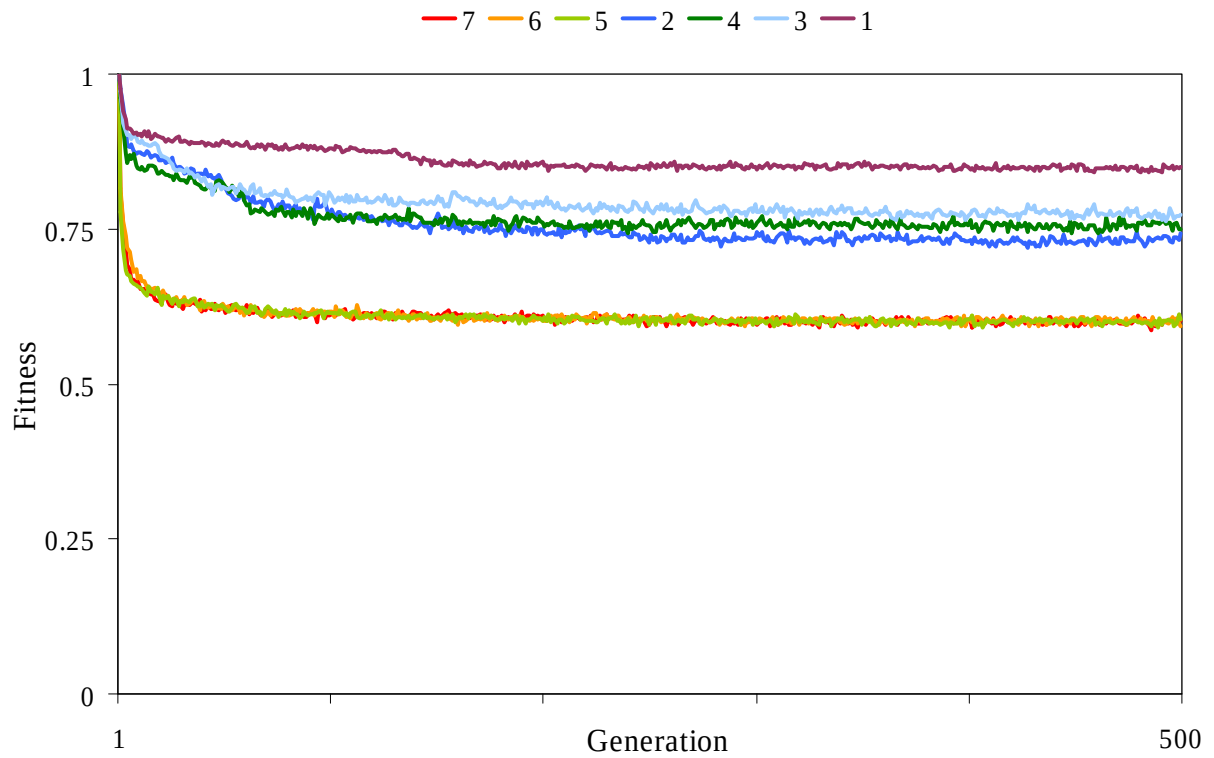


Figure 56 Average Fitness: Hunt Action 1 – *Goto Ball*. Experiments 1 - 7

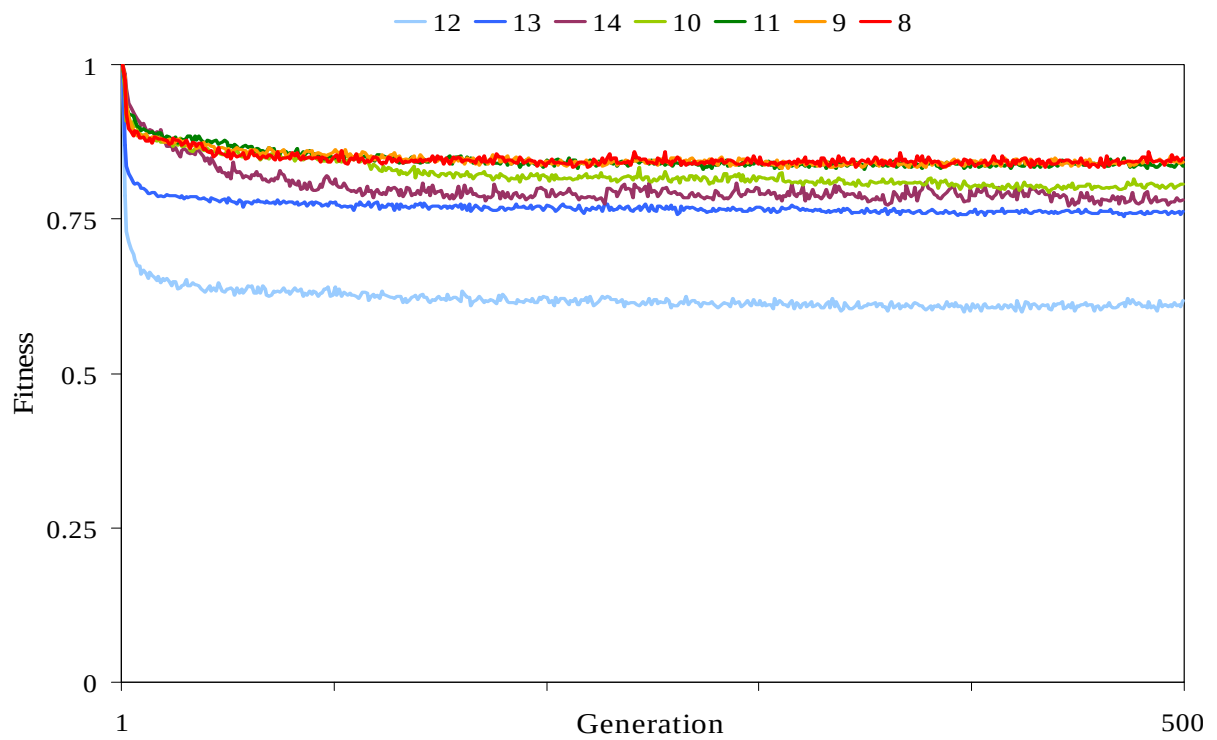


Figure 57 Average Fitness: Hunt Action 1 – *Goto Ball*. Experiments 8 – 14

Though it is a naïve strategy, regardless of the situation, and especially when there are no other players on the field, if the player kicks the ball toward the goal there is a good chance it will actually score a goal. In the case of *DribbleTowardGoal*, the fact that the player follows the ball after kicking it toward the goal allows it to possibly repeat the kick in the direction of the goal, thus increasing the chance of scoring a goal. This is evidenced by the superior (average) performance of *DribbleTowardGoal* over *KickTowardGoal* (lines 12 and 11 respectively in Figure 57 and Figure 59 on page 146).

RunTowardBall and *RunTowardGoal*, when available to the player together, are an effective combination of skills for goal-scoring behaviour, but on their own seem not to be effective at all. *GotoBall* on its own is only marginally more effective than *RunTowardBall* and *RunTowardGoal* on their own. (lines 10, 9 and 8 respectively in Figure 57 and Figure 59). The results for this series of experiments agree with observations regarding the selection of consequent actions made in section 6.4 on page 140 – the consequent actions most selected in those trials appear to be the most effective for goal-scoring behaviour when the default action for the player is *Hunt Action 1 – Goto Ball*.

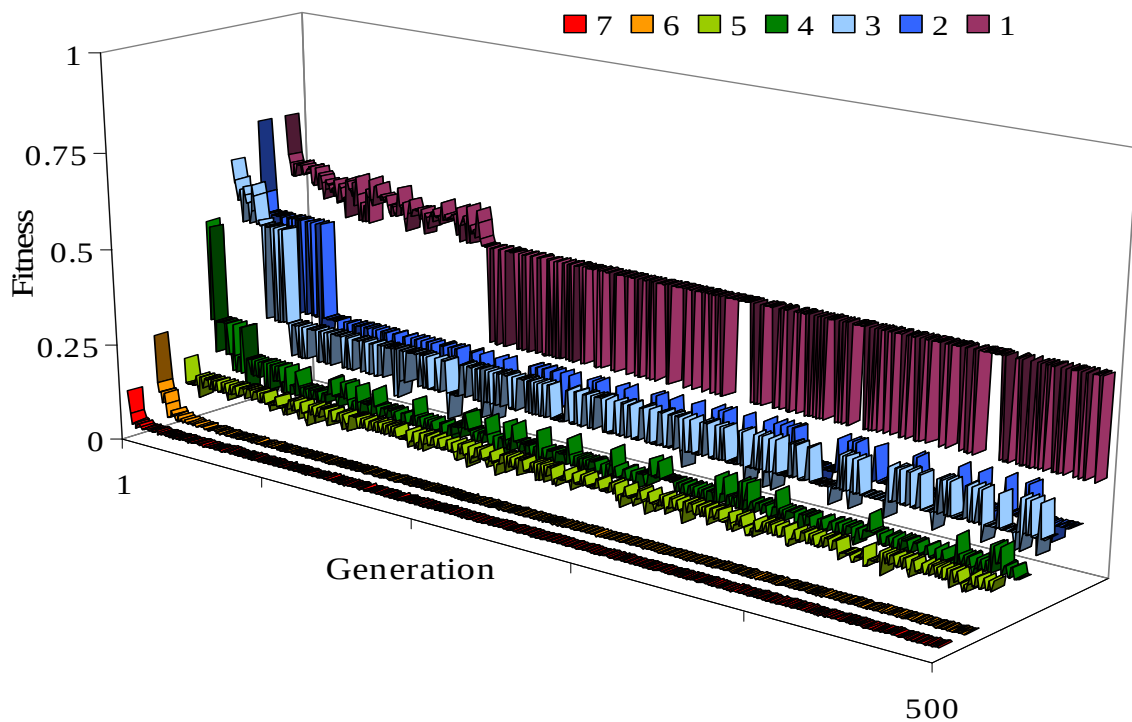


Figure 58 Best Fitness: Hunt Action 1 – *Goto Ball*. Experiments 1 - 7

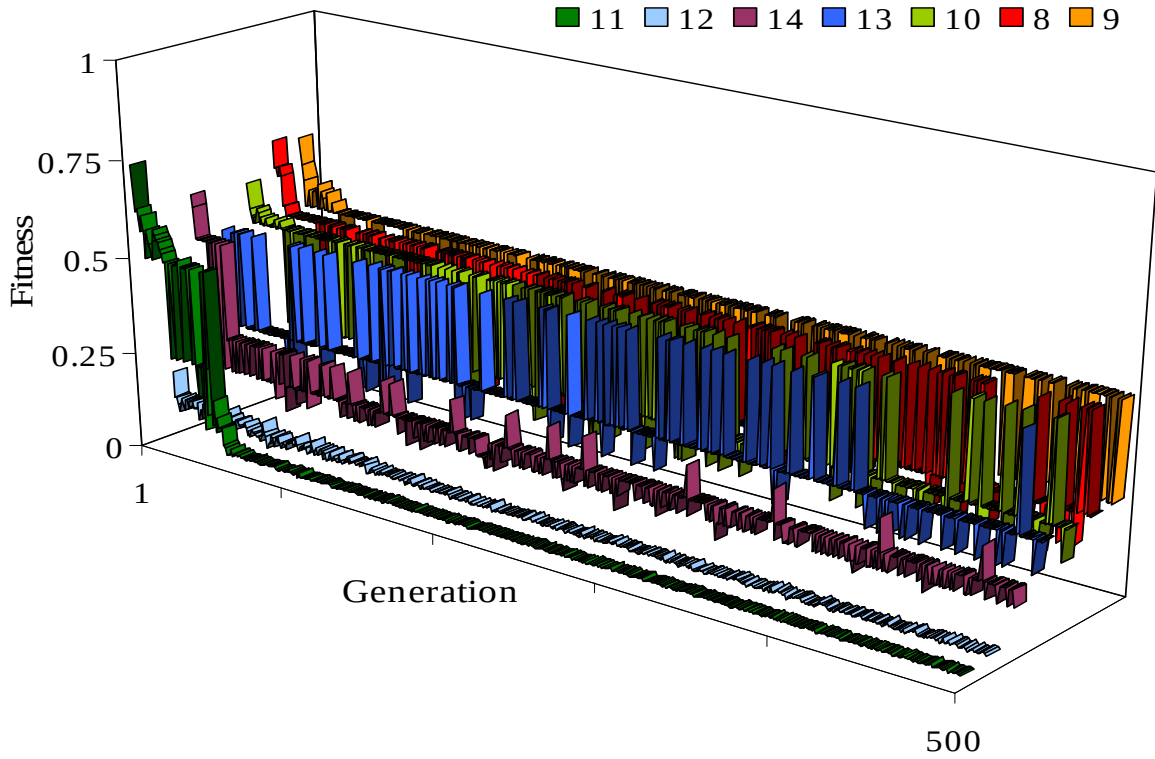


Figure 59 Best Fitness: Hunt Action 1 – *Goto Ball*. Experiments 8 – 14

6.5.2 Default Action: *Hunt Action 2 – Locate Ball*

The second series of experiments was performed in an environment in which the default action for the player was defined as *Hunt Action 2 – Locate Ball*. Each skillset shown in Table 13 (page 142, experiments 1 - 14) was evaluated with the default action *Hunt Action 2*. The average fitness of the population after each of the 500 generations is shown in Figure 60 (experiments 1 - 7) and Figure 61 (experiments 8 - 14). The best fitness achieved by any member of the population during each of the 500 generations is shown in Figure 62 (experiments 1 – 7, page 148) and Figure 63 (experiments 8 – 14, page 149).

The results for this experiment are very similar to the results for the *Hunt Action 2 – Goto Ball* results presented in the previous section – not unexpected given the reasonable similarity in player capability. The difference in player capability is only in that for the default action, in this experiment the player stops hunting when the ball is sighted rather than continuing the hunt action until the ball is within kicking distance.

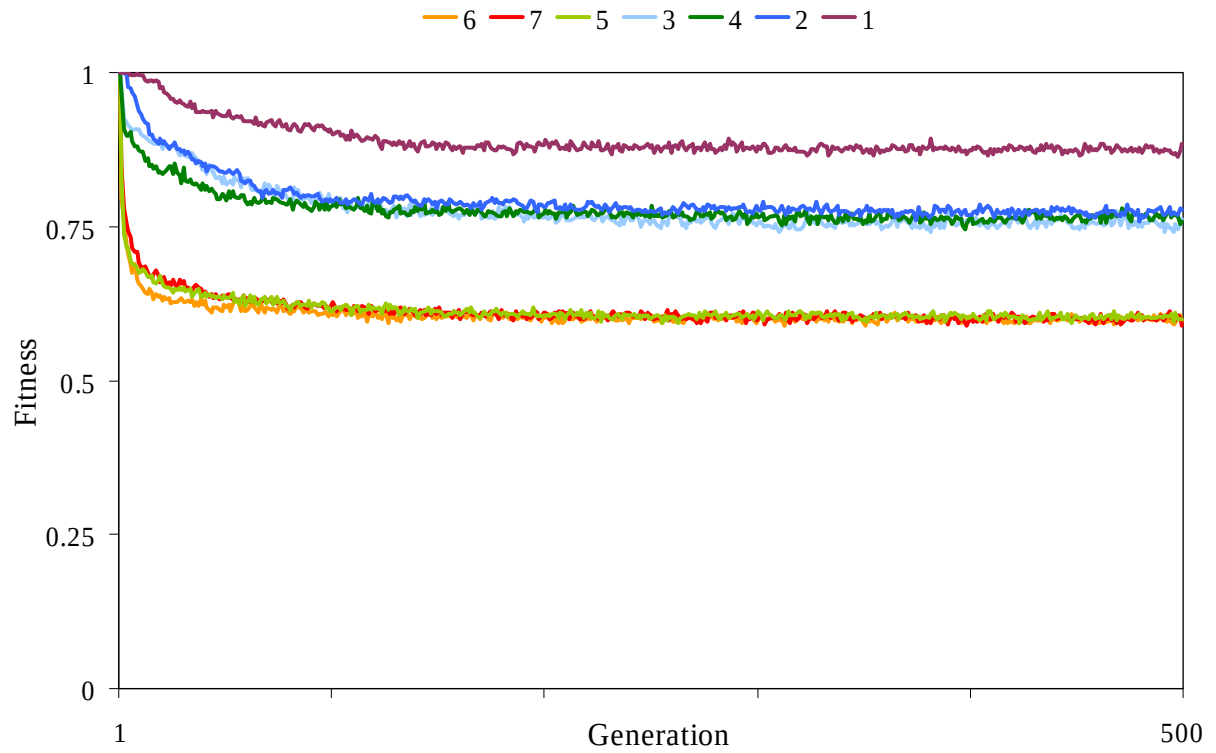


Figure 60 Average Fitness: Hunt Action 2 – *Locate Ball*. Experiments 1 - 7

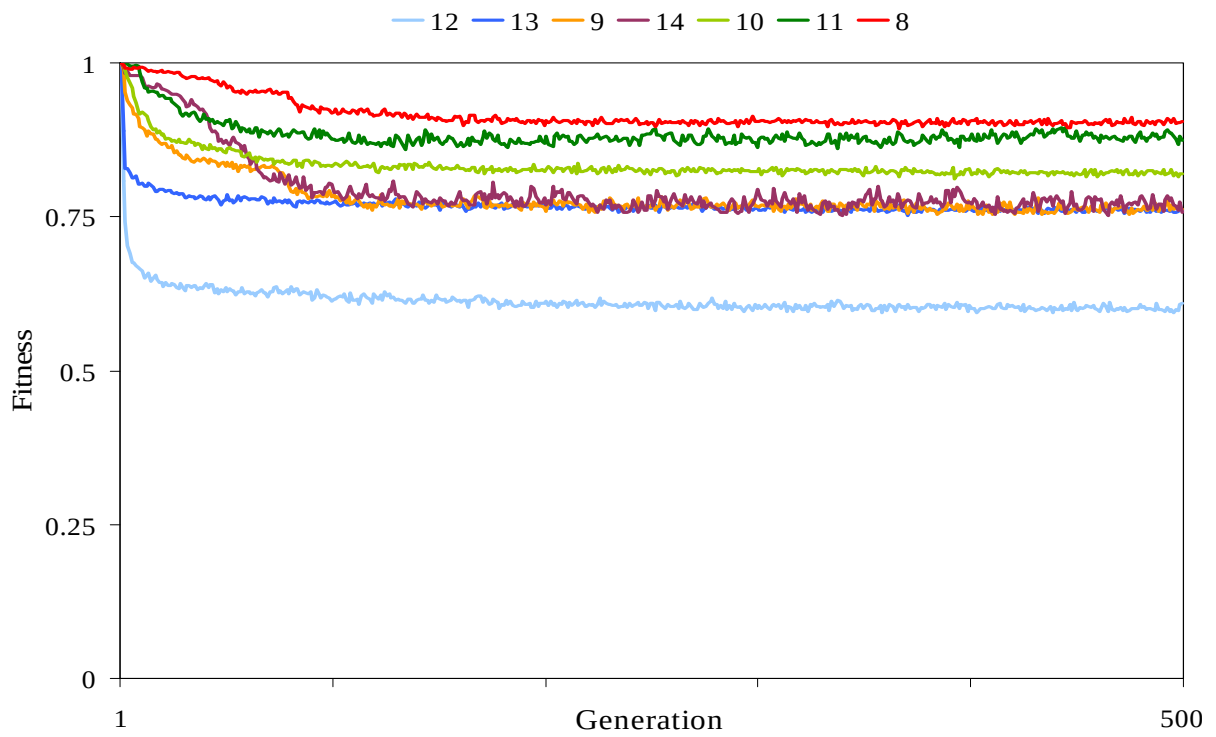


Figure 61 Average Fitness: Hunt Action 2 – *Locate Ball*. Experiments 8 - 14

An interesting difference is the change in effectiveness of the *RunTowardBall* skill (line 9 in Figure 61 on page 147 and Figure 63). The essential difference between the two experiments is that the player in this experiment has had a “go to ball” or “run to ball” action removed from its default action, and the increase in effectiveness of the *RunTowardBall* skill would seem to be in compensation for that loss in the default action, and an indication that higher-level skills are being constructed from lower-level skills. Again, this is somewhat intuitive - if the player is able to locate the ball, being able to run towards it is a useful skill for goal-scoring behaviour.

Other differences of note are that the *KickTowardGoal* skill (line 11 in Figure 61 and Figure 63) is slightly less effective, though still useful, in this experiment, and the *Added Directions* (line 14) have become more effective. This could be another indication that higher-level skills are being constructed from lower-level skills, with the *TowardBall* and *TowardGoal* directions being used more effectively to construct higher-level skills. Inspection of the rules generated indicates that the added directions *TowardBall* and *TowardGoal* figure consistently.

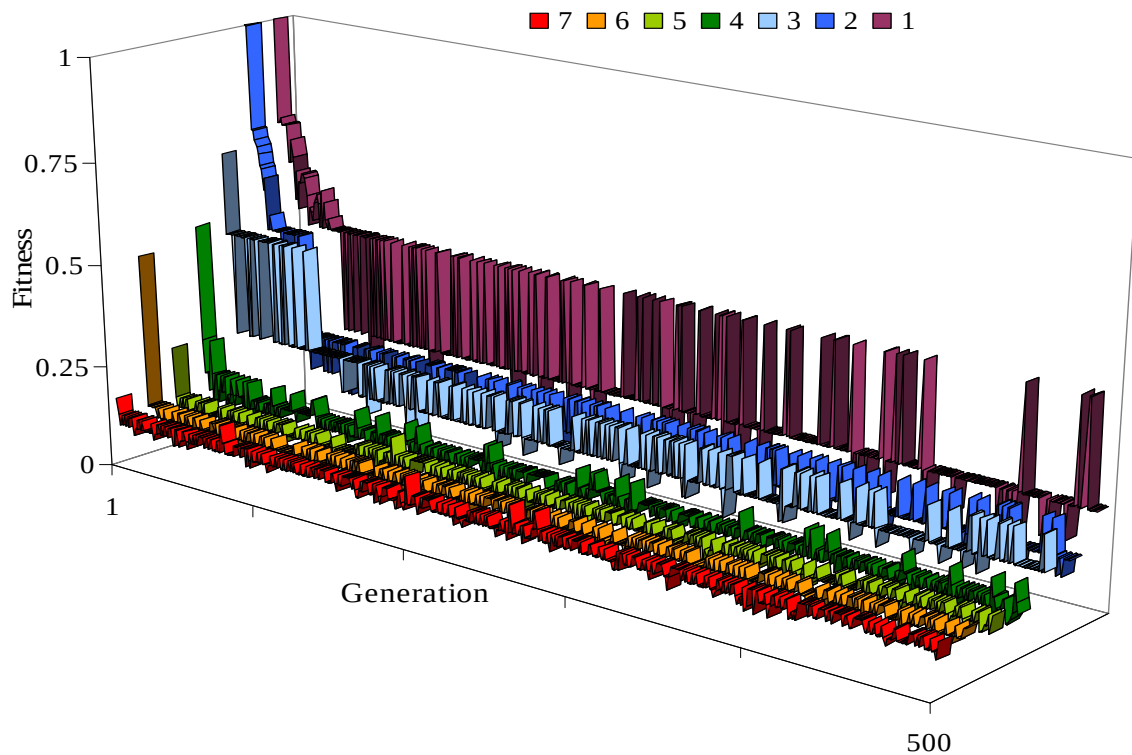


Figure 62 Best Fitness: Hunt Action 2 – *Locate Ball*. Experiments 1 - 7

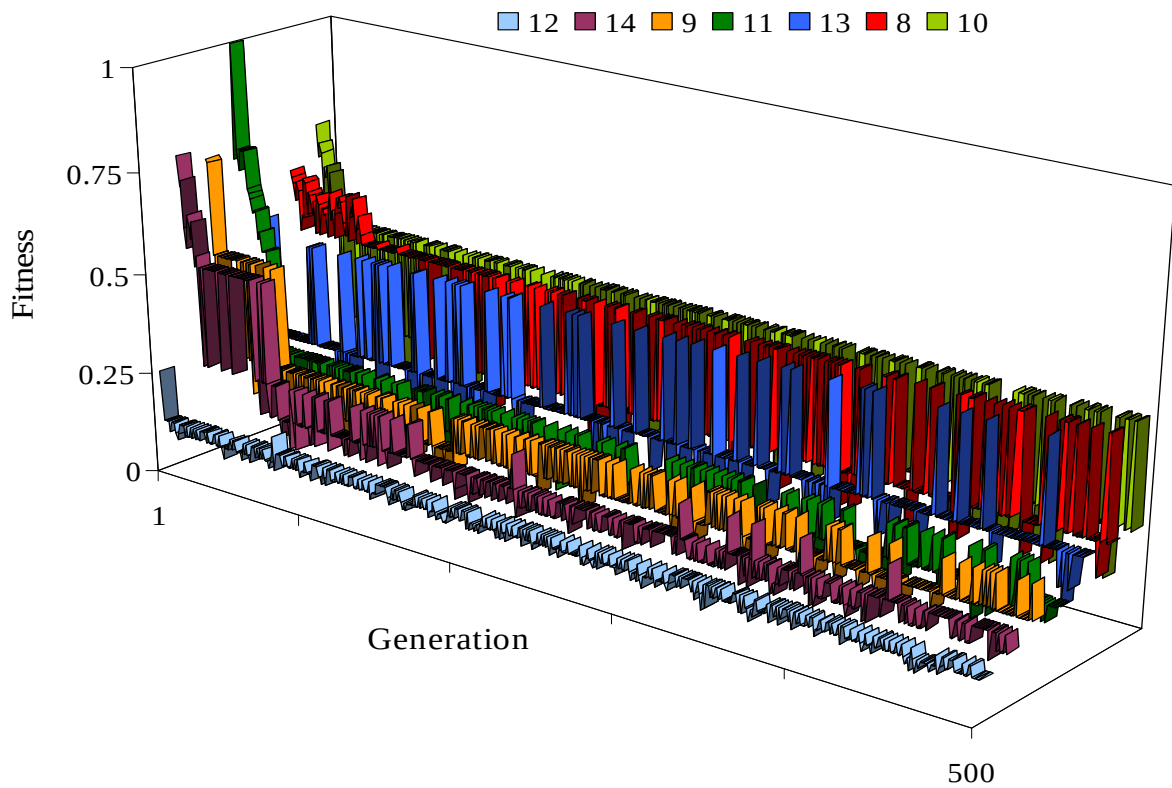


Figure 63 Best Fitness: Hunt Action 2 – *Locate Ball*. Experiments 8 – 14

6.5.3 Default Action: *Hunt Action 3 – Random Turn*

The third series of experiments was performed in an environment where the default action for the player was defined as *Hunt Action 3 – Random Turn*. Each skillset shown in Table 13 (page 142, experiments 1 -14) was evaluated with the default action *Hunt Action 3*. The average fitness of the population after each of the 500 generations is shown in Figure 64 (experiments 1 – 7, page 150) and Figure 65 (experiments 8 – 14, page 151). The best fitness achieved by any member of the population during each of the 500 generations is shown in Figure 66 (experiments 1 – 7, page 151) and Figure 67 (experiments 8 – 14, page 152).

The data shows that, as for the previous experiments, while the players without the hand-coded mid-level skills are able to occasionally score goals, some combination of the hand-coded skills is required to more consistently display goal-scoring behaviour. The population average fitness is poor even for the players with various combinations of the hand-coded skills, suggesting that even though players displaying goal-scoring behaviour were located during

each generation the frequency of occurrence of those players must have been reasonably low, thus negatively affecting the average fitness of the population.

The data illustrates clearly that, for this series of experiments, players with more mid-level skills available to them outperform those with fewer of those skills available. This is an expected result. The population average fitness curves for players with single mid-level skills shown in Figure 65 indicate that, for this series of experiments, multiple different mid-level skills are required in order to produce consistent goal-scoring behaviour. Figure 67 on page 152 suggests that, for this series of experiments, the individual skills which produce better occasional goal-scoring behaviour are, as noted for the previous experiments, *DribbleTowardGoal* and *KickTowardGoal* (lines 12 and 11 respectively in Figure 65 and Figure 67).

An interesting phenomenon is that the hand-coded *RunTowardGoal* and *RunTowardBall* skills (lines 8 and 9 respectively in Figure 65 and Figure 67) produce better occasional goal-scoring behaviour than the *GoToBall* skill (line 10). This is also seen in the *Hunt Action 2 – Locate Ball* series of experiments. *RunTowardBall* could produce behaviour similar to *GoToBall* so it is possible that in some experiments rules evolved with the *RunTowardBall* consequent which mimic the *GoToBall* skill. Similarly, *RunTowardGoal* could produce behaviour similar to *DribbleTowardGoal* which was identified as a useful skill for goal-scoring behaviour.

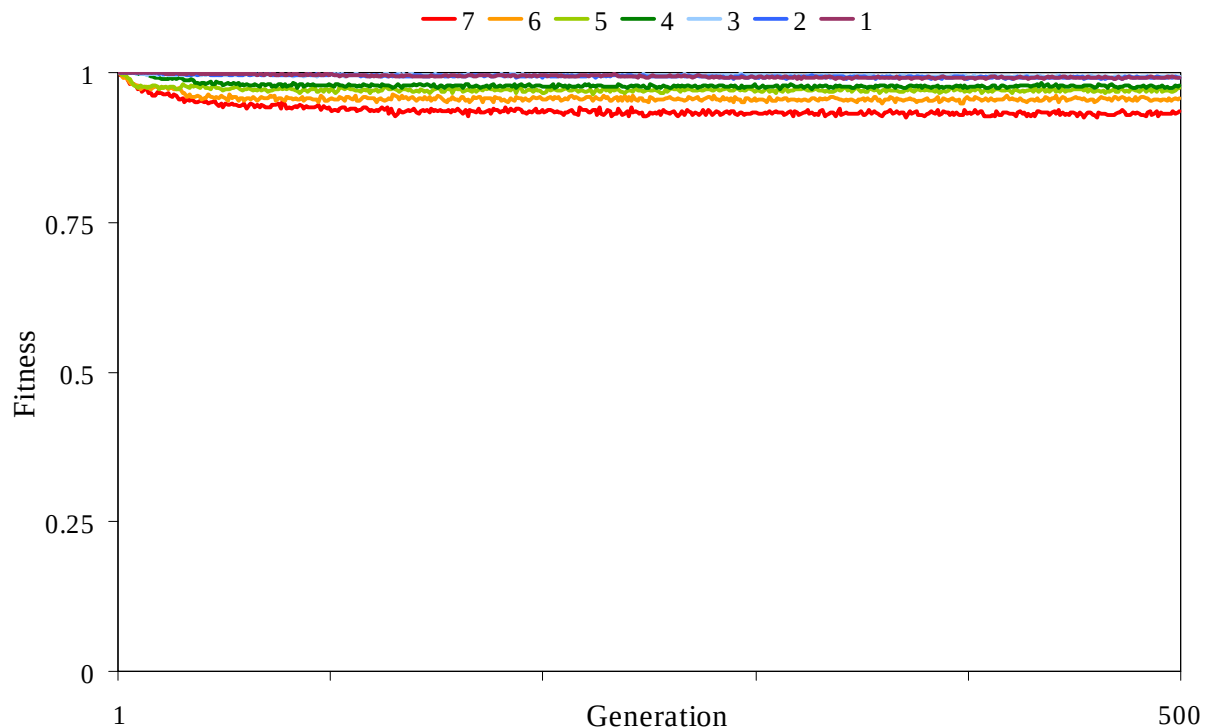


Figure 64 Average Fitness: Hunt Action 3 – *Random Turn*. Experiments 1 - 7

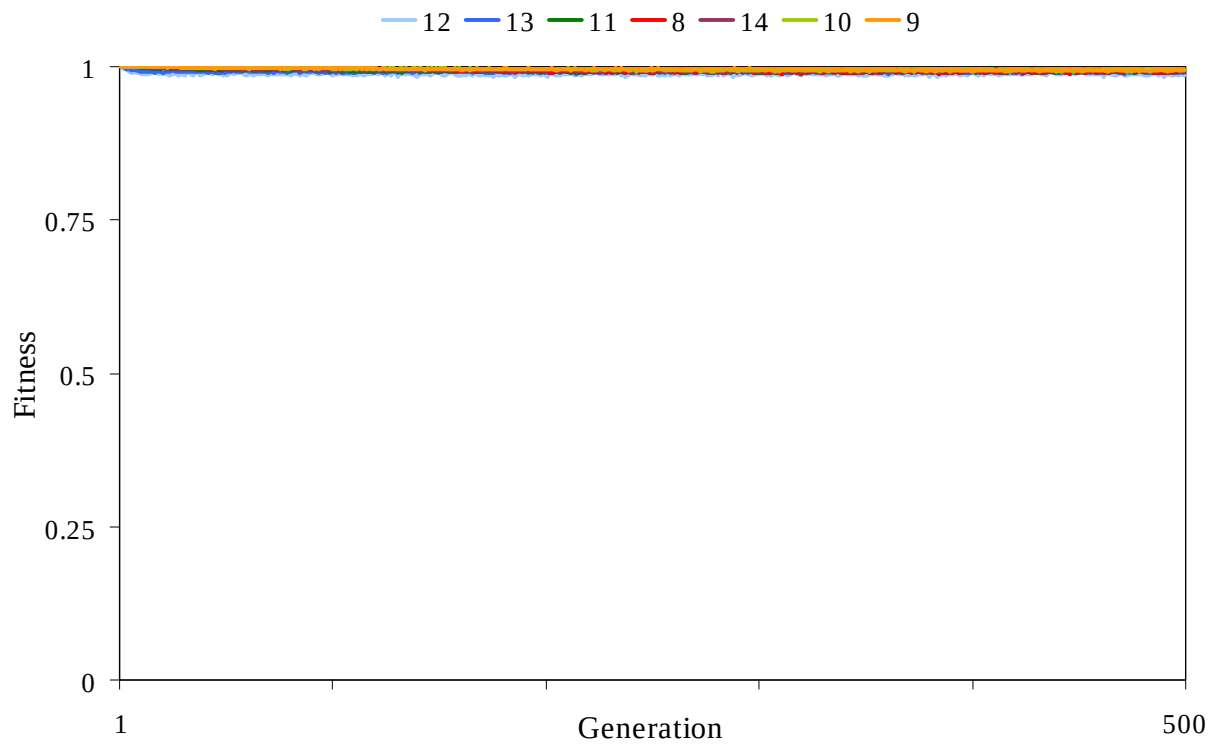


Figure 65 Average Fitness: Hunt Action 3 – *Random Turn*. Experiments 8 - 14

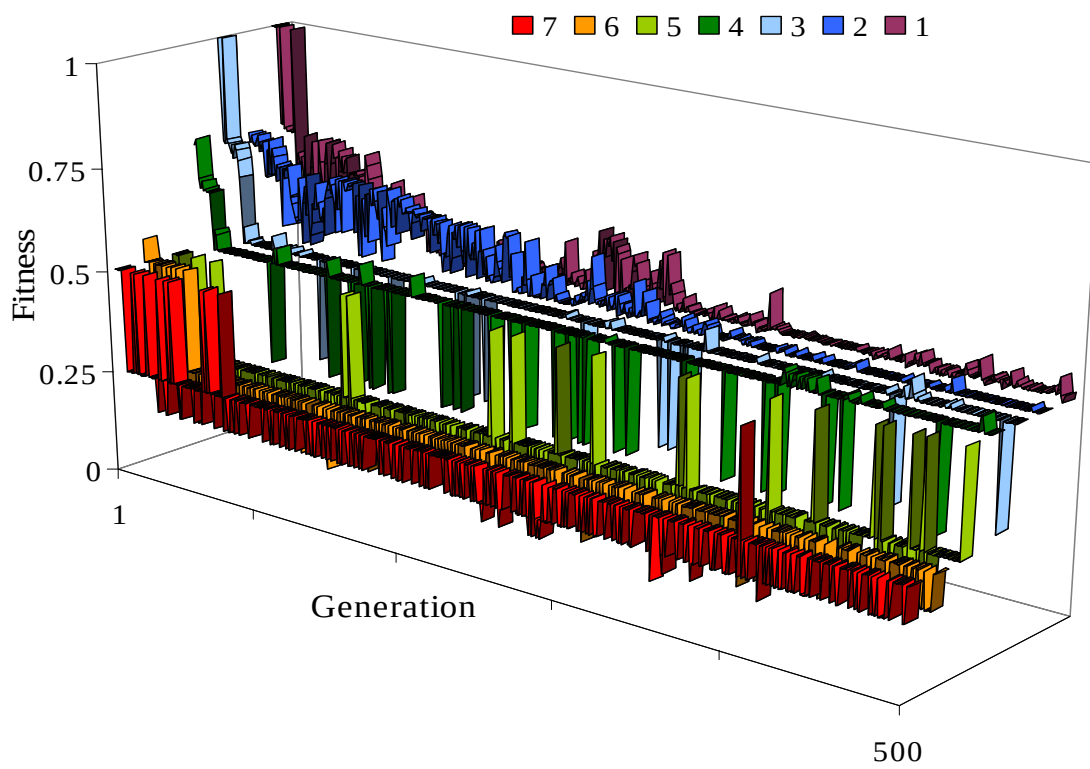


Figure 66 Best Fitness: Hunt Action 3 – *Random Turn*. Experiments 1 - 7

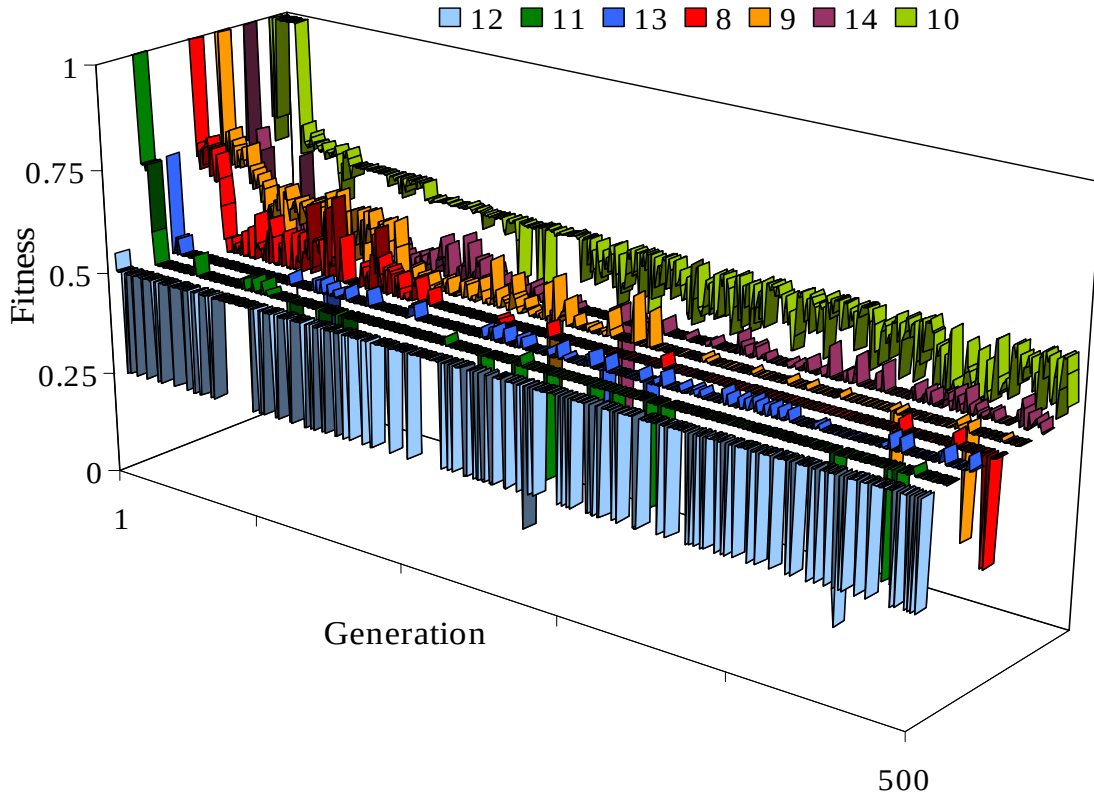


Figure 67 Best Fitness: Hunt Action 3 – *Random Turn*. Experiments 8 – 14

6.5.4 No Default Action

The final series of experiments was performed in an environment in which there was no default action defined for the player: if no rule fired then the player did not act during that game cycle. Each skillset shown in Table 13 (page 142, experiments 1 -14) was evaluated with no default action.

For this series of experiments only, a new premise variable was introduced for use with the premise term BALL. The new variable allows premises of the form “BALL is|is not VISIBLE” to be constructed by the player. Also for this series of experiments only, the player was given the ability to use the *Hunt* action as a deliberate consequent action rather than as a default action – for example, “IF ... THEN HUNT”. The hunt action implemented was *Hunt Action 2* as described in section 3.4.1 on page 82. The results presented in the previous sections seem to indicate that in some circumstances the evolutionary algorithm is able to compensate for the lack of a mid-level skill by constructing a similar skill from low-level behaviours, so this change was made to allow the evolutionary algorithm to construct the equivalent of the (removed) default hunt action. The new premise variable and consequent action were added to

provide the evolutionary algorithm the means to construct a mid-level hunt skill. For example, as compensation for the lack of a default hunt action, the algorithm has the opportunity to evolve the rule “*IF BALL is not VISIBLE THEN HUNT*”.

Two sets of trials were performed so that the impact of the *VISIBLE* premise term and the ability to choose to use the *Hunt* action as a deliberate consequent action could be measured. The first set of trials allowed the player only the base set of actions plus the additional groups of actions listed in Table 13. The second set of trials allowed the player the base set of actions plus the additional groups of actions listed in Table 13, but further allowed the player to use the *Hunt* action as a deliberate consequent action, and to utilise the new *VISIBLE* premise variable.

6.5.4.1 Set 1: No Hunt Consequent

For this set of experiments, the average fitness of the population after each of the 500 generations is shown in Figure 68 (experiments 1 – 7, page 154) and Figure 69 (experiments 8 – 14, page 154). The best fitness achieved by any member of the population during each of the 500 generations is shown in Figure 70 (experiments 1 – 7, page 155) and Figure 71 (experiments 8 – 14, page 155).

For all experiments the population average fitness is high, even for the players with various combinations of the hand-coded skills, suggesting that even though players displaying goal-scoring behaviour were located during each generation, either the frequency of occurrence of those players, or the number of goals scored, is low. Inspection of the results for the individual best fitness values tends to indicate that the number of players exhibiting goal-scoring behaviour in each generation is low.

It is clear from Figure 70 and Figure 71 that the most useful skill in this environment is the *DribbleTowardGoal* skill (line 12). Again, though it is a naïve strategy it is clearly one which works. Any combination of skills which includes the *DribbleTowardGoal* skill has performed reasonably well, whereas any combination of skills not including the *DribbleTowardGoal* skill has performed somewhat less well. The *KickTowardGoal* skill (line 11) seems not to have been as effective in this set of experiments as previously seen. This is probably because if the player loses sight of the ball after executing *KickTowardGoal* it is less able to locate it than previously - a problem less likely to occur when executing *DribbleTowardGoal* because in that case the player immediately dashes in the direction of the kick, so is less likely to lose sight of the ball. It is interesting to note that even in this configuration, with no default action and no hunt consequent allowed, and even for players with just the base set of skills, goal-scoring behaviour evolved – albeit less often than in previous experiments.

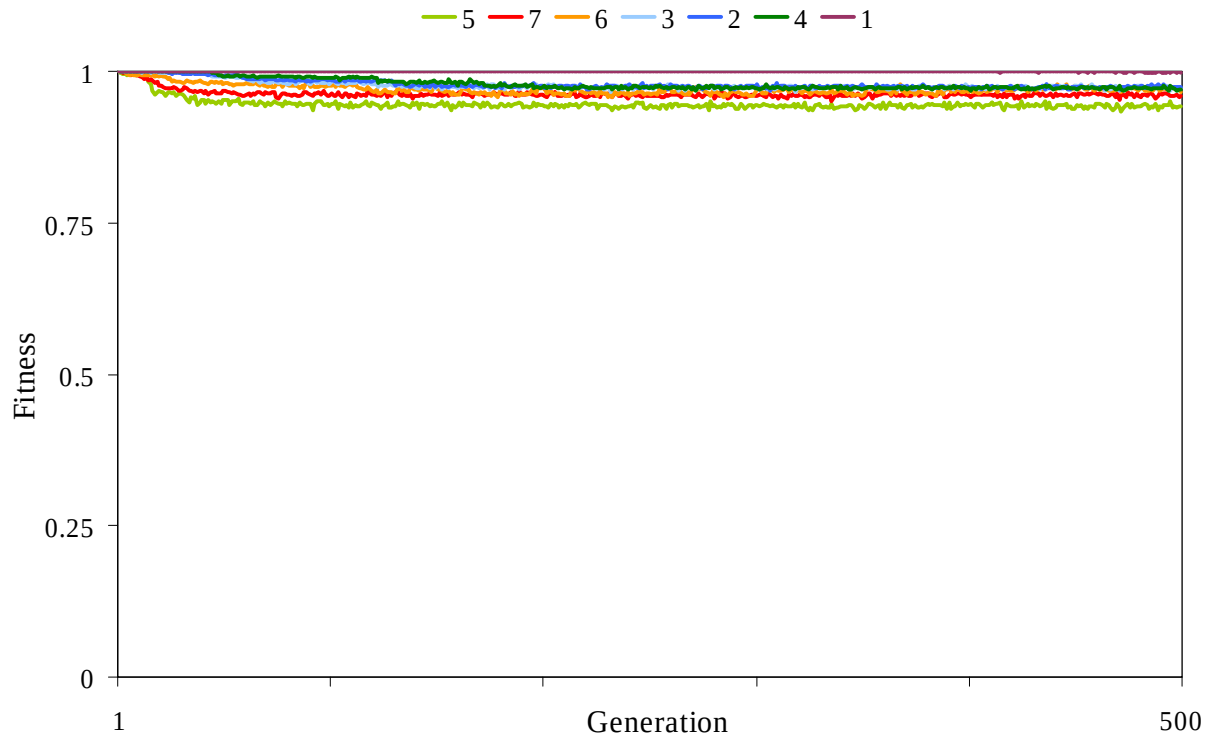


Figure 68 Average Fitness: No Default Action – *No Hunt Consequent*. Experiments 1 - 7

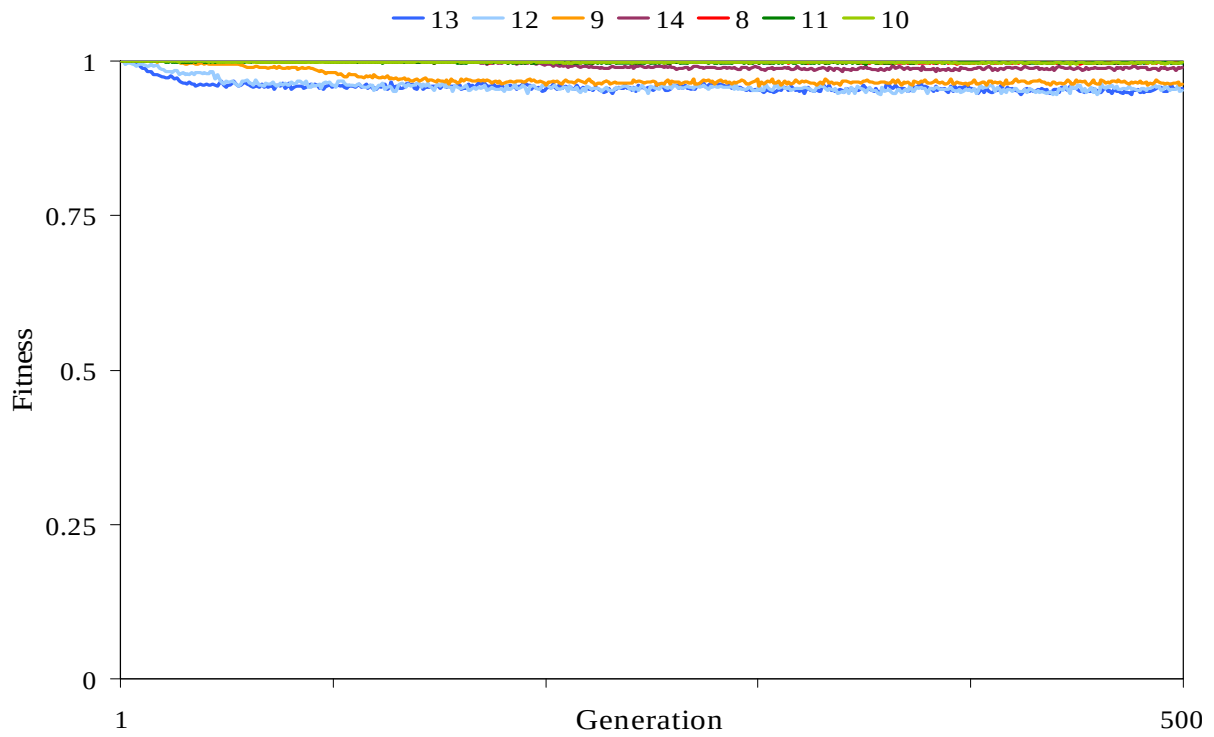


Figure 69 Average Fitness: No Default Action – *No Hunt Consequent*. Experiments 8 - 14

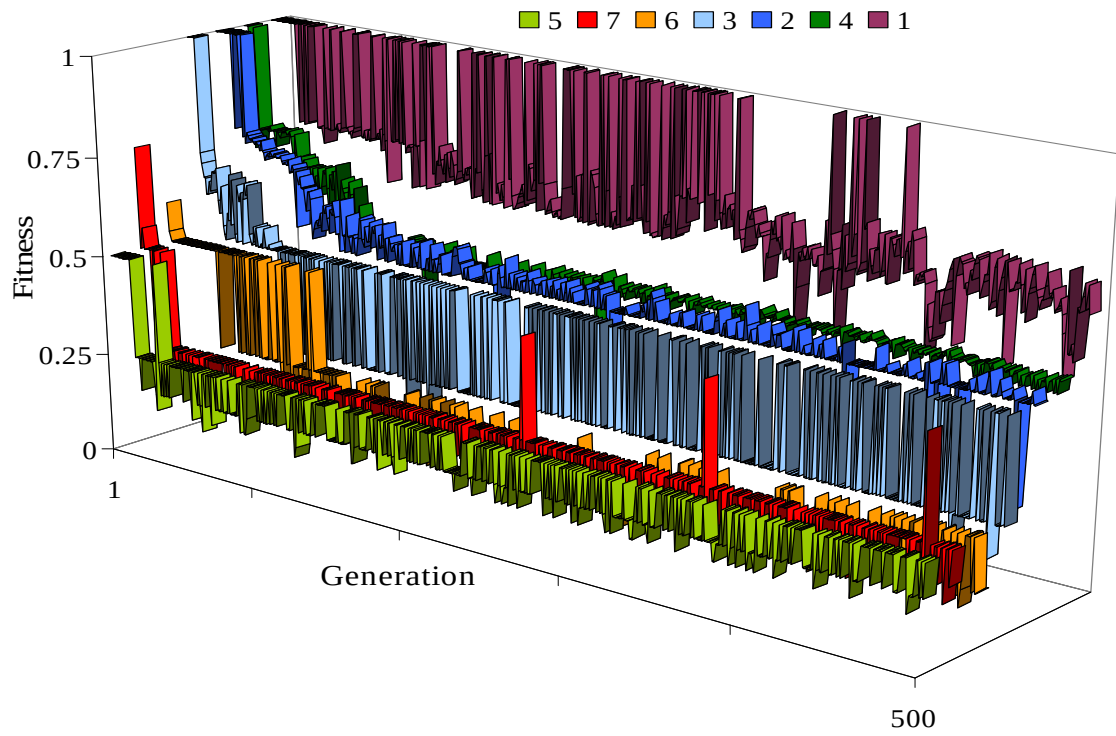


Figure 70 Best Fitness: No Default Action – *No Hunt Consequent*. Experiments 1 - 7

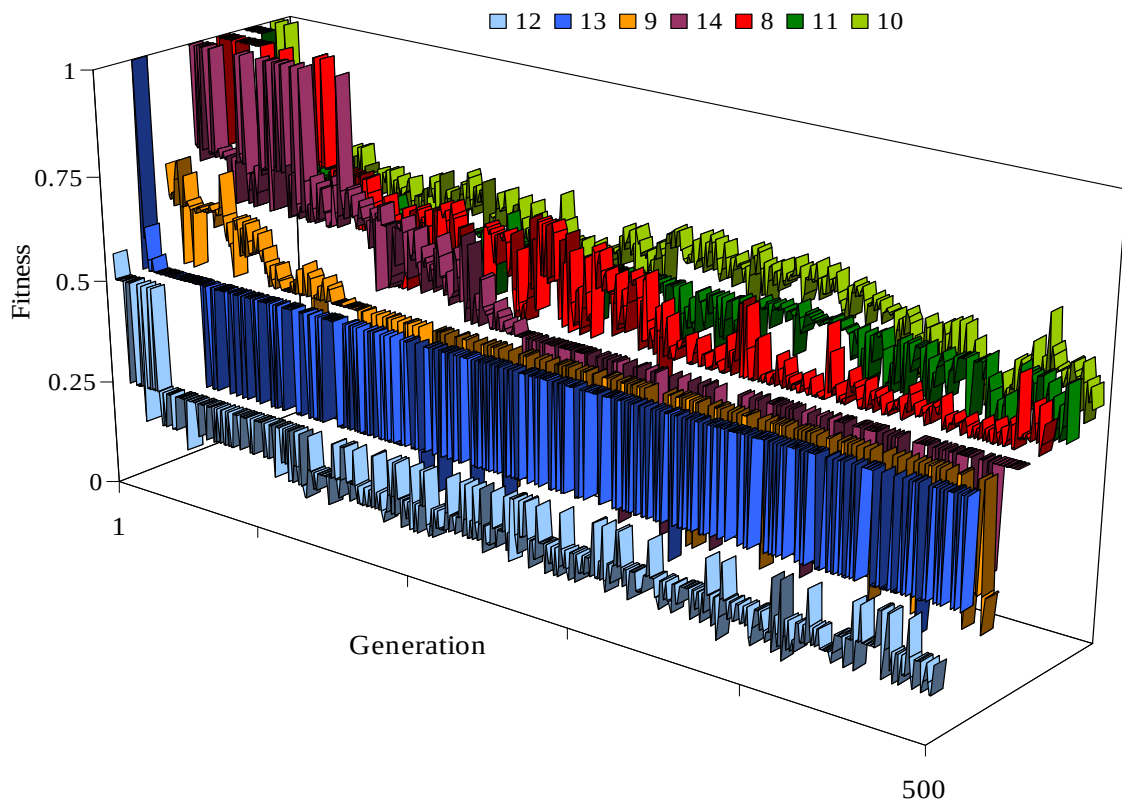


Figure 71 Best Fitness: No Default Action – *No Hunt Consequent*. Experiments 8 - 14

6.5.4.2 Set 2: *Hunt Consequent Allowed*

For this set of experiments, the average fitness of the population after each of the 500 generations is shown in Figure 72 (experiments 1 - 7) and Figure 73 (experiments 8 - 14). The best fitness achieved by any member of the population during each of the 500 generations is shown in Figure 74 (experiments 1 – 7, page 158) and Figure 75 (experiments 8 – 14, page 158).

Again, for all experiments the population average fitness is high, even for the players with various combinations of the hand-codes skills, though slightly better than the population average fitness values seen in the “no hunt consequent” experiments. As for the “no hunt consequent” experiments, players displaying goal-scoring behaviour were located during each generation, but the number of players exhibiting goal-scoring behaviour in each generation is low.

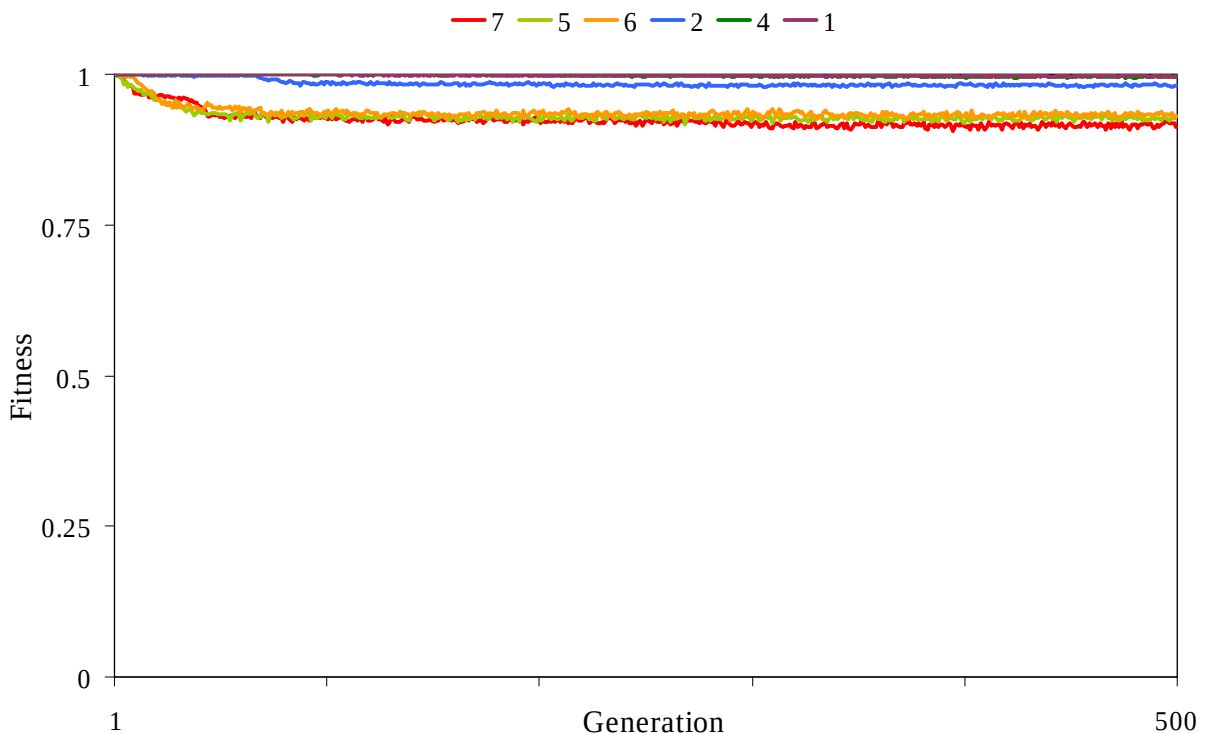


Figure 72 Average Fitness: No Default Action – *Hunt Consequent Allowed*. Experiments 1 - 7

It is clear from Figure 74 and Figure 75 that, as for previous experiments, the most useful skill in this environment is the *DribbleTowardGoal* skill (line 12). Any combination of skills which includes the *DribbleTowardGoal* skill has again performed reasonably well, with combinations of skills not including the *DribbleTowardGoal* performing less well. As for the “no hunt consequent” experiments, and likely for the same reasons, the *KickTowardGoal* skill (line 11) seems not to have been as effective in this set of experiments as previously seen.

Interestingly, from Figure 74 it is evident that the skillset for experiment 3 - base skills plus *RunTowardBall*, *RunTowardGoal* and *GoToBall* (lines 9, 8 and 10 respectively) - did not perform as well as in the “no hunt consequent” experiments. This is possibly because in this set of experiments the availability of the Hunt consequent lessens the impact of, or need for, the *RunTowardBall* and *GoToBall* skills.

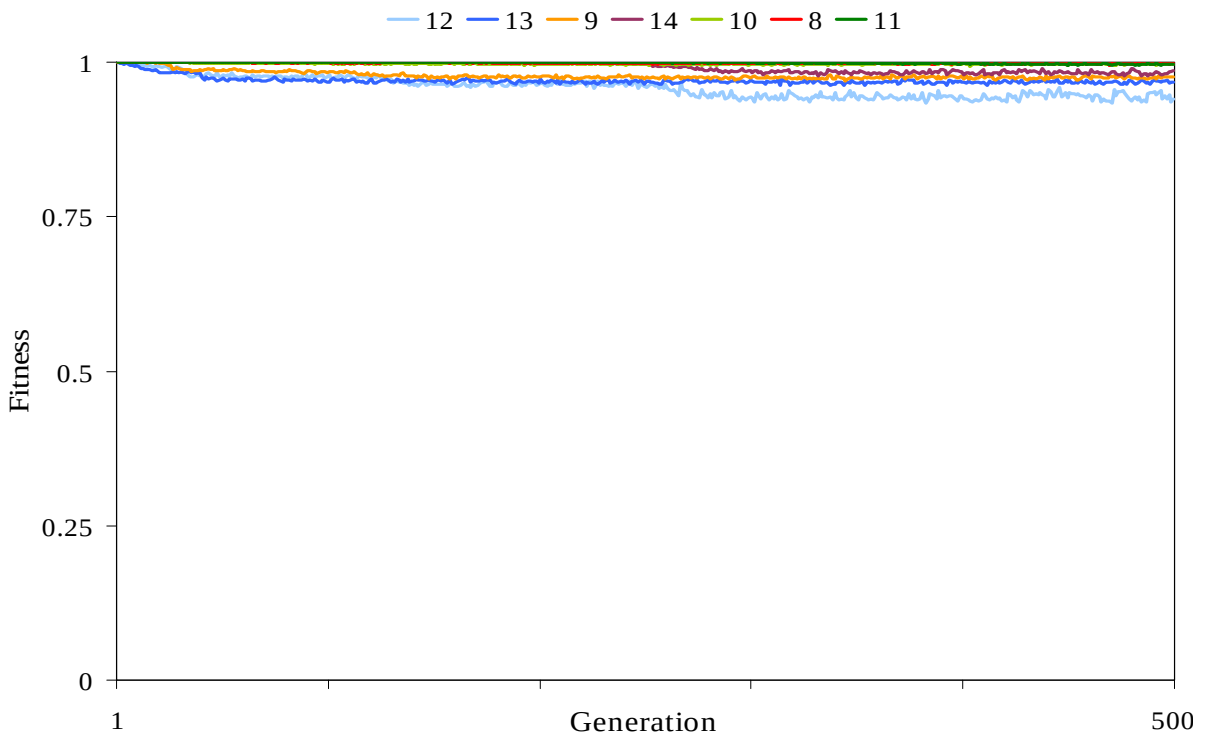


Figure 73 Average Fitness: No Default Action – *Hunt Consequent Allowed*. Experiments 8 - 14

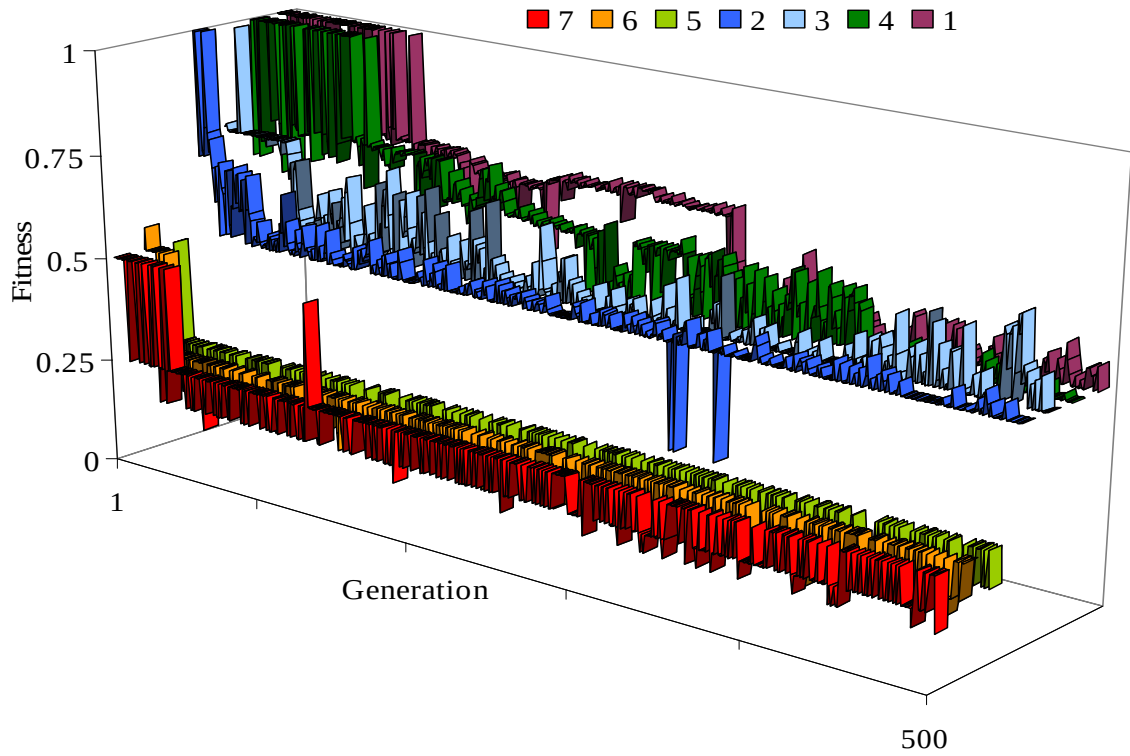


Figure 74 Best Fitness: No Default Action – *Hunt Consequent Allowed*. Experiments 1 - 7

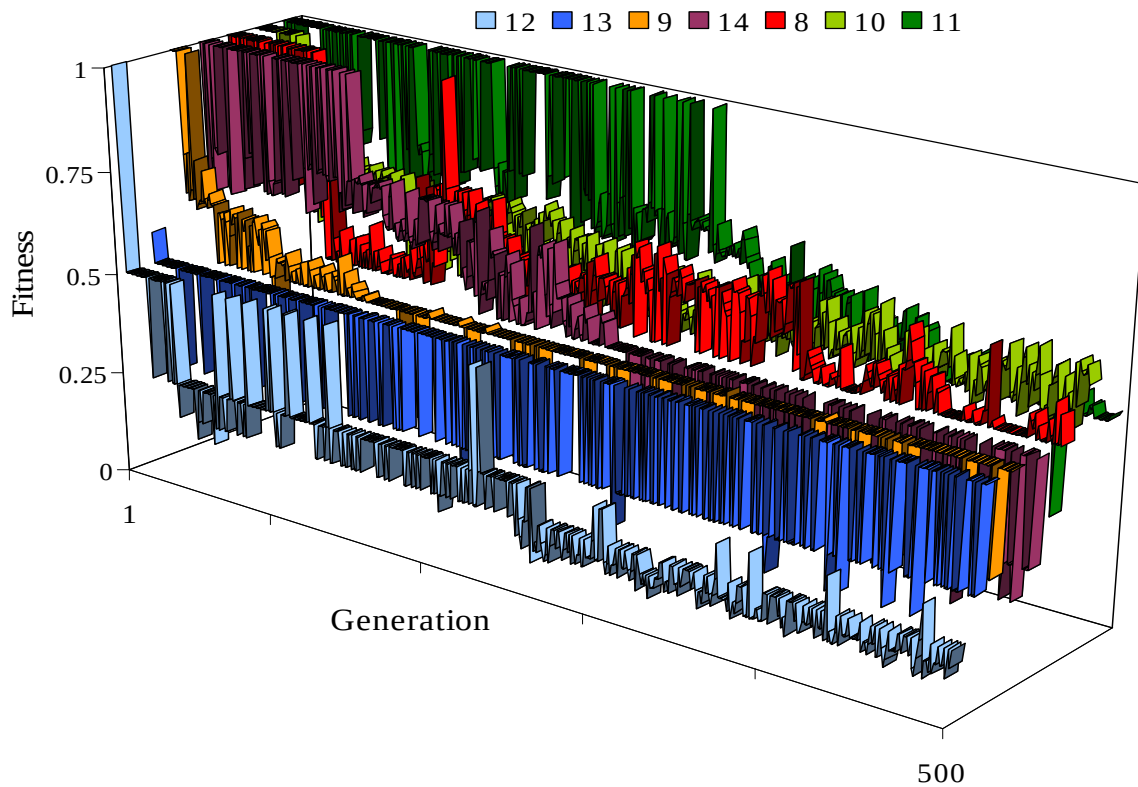


Figure 75 Best Fitness: No Default Action – *Hunt Consequent Allowed*. Experiments 8 - 14

6.6 Varying the Fitness Function

Four sets of experiments were conducted to determine the effect on the performance of the evolutionary search of varying the fitness function used to evaluate the player. A base set of skills was available to the player in all trials, consisting of the skills *Turn*, *Kick*, *Dash* and *DoNothing*. The first two series of experiments were conducted with only the base set of skills available to the players, and the player's default action was varied. The next two series of experiments were conducted with the base set of skills plus all the hand-coded skills described in section 4.3.5 on page 91 available to the player, as well as the two new directions described in section 6.3 on page 139. As for the first two series of experiments, the player's default action was varied. The experiment parameters are listed in Table 14. The player's field of vision was limited as described in section 6.3 on page 139.

Experiment	Player Skills	Hunt Consequent	Default Action
1	Base Skills	No	None
2	Base Skills	Yes	None
3	All Skills	No	None
4	All Skills	No	Hunt Action 1

*Base Skills = turn, kick, dash & no added directions; All Skills = Base + all hand-coded skills & added directions

Table 14 Goals-only Fitness Experiments: Player Skills

The genetic algorithm parameters used in these experiments are the same as those listed in Table 11 on page 141. The SimpleSoccer parameters used in these experiments are the same as those listed in Table 12 on page 141. Rather than being implemented as a composite function rewarding different behaviours, the fitness function used in these experiments rewarded individuals for goal-scoring behaviour only, and was implemented as shown in Equation 5 on page 97. The results of the experiments are presented in Figure 76, Figure 77 and Figure 78 on pages 160 and 161, and are compared with the equivalent experiments already conducted

using the composite fitness function (the results of which were presented in the preceding sections). The comparison is presented in section 6.7.2 on page 165.

Figure 76 shows the population average fitness for each of the experiments, and it is obvious from this graph that the population average fitness is high for all experiments. For experiments 1 and 2, the population average fitness remained at 1.0 for the entire evolutionary process. The population average fitness remained above 0.99 for experiment 3, and above 0.89 for experiment 4. This indicates the average member of the population in all four experiments did not develop consistent goal-scoring behaviour. Since the fitness function used rewards goal-scoring only, it is not possible to determine from these results alone whether members of the population that did not kick goals displayed ball-kicking behaviour. Through observation of the experiments it was ascertained that some players that did not kick goals did indeed kick the ball, but this behaviour was not consistent and, since it was not rewarded, not encouraged by the evolutionary process.

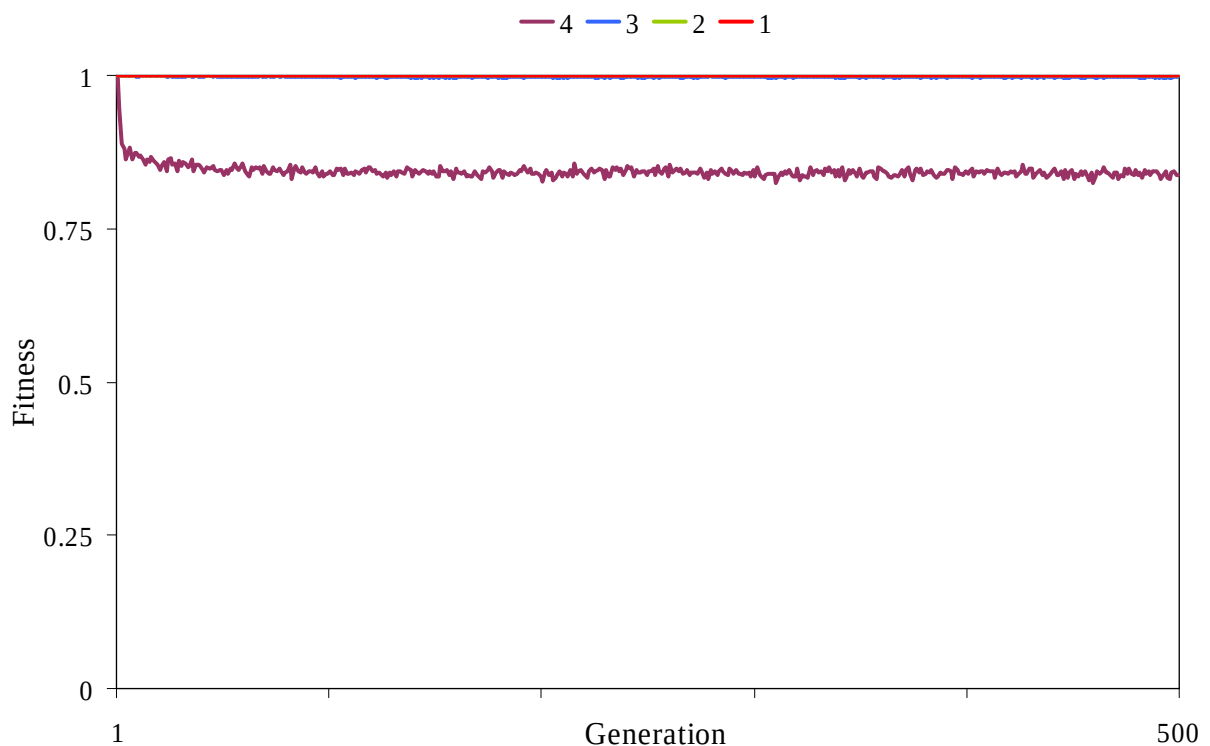


Figure 76 Average Fitness: Goals-only Fitness Function

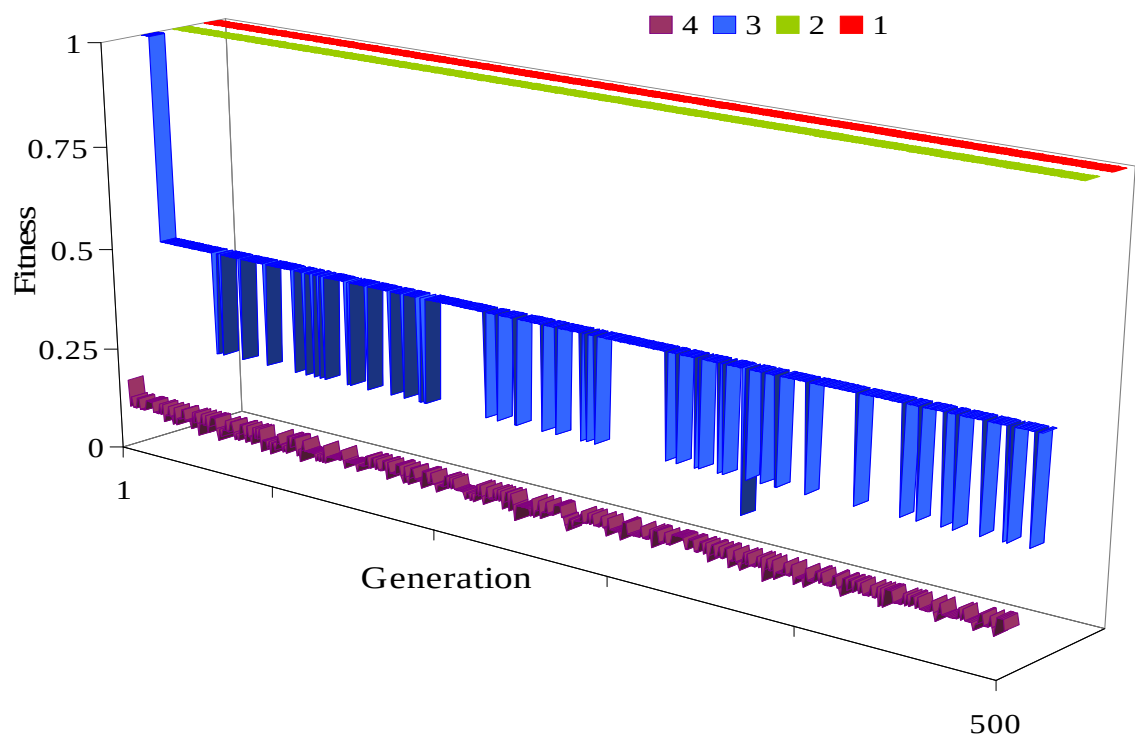


Figure 77 Best Fitness: Goals-only Fitness Function

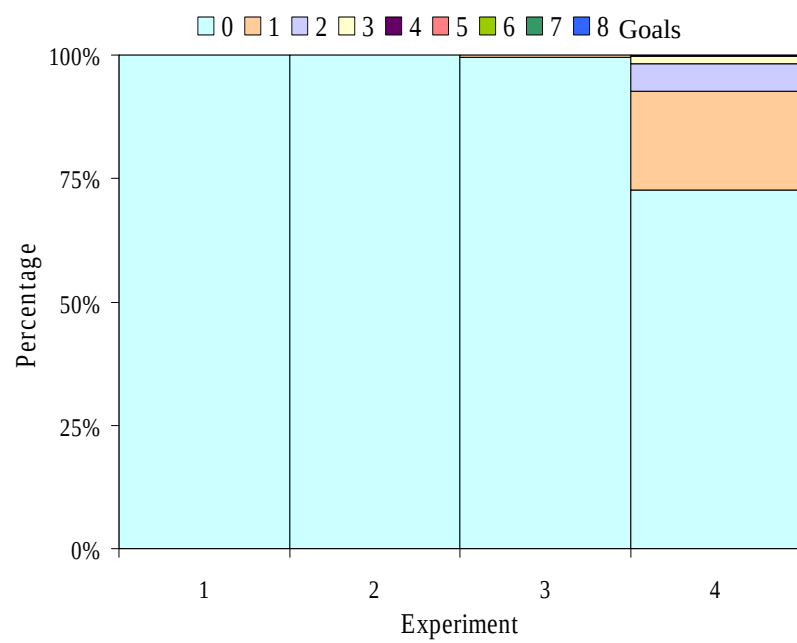


Figure 78 Goals-only Fitness: Frequency of Individuals Scoring Goals

The individual best fitness values are shown in Figure 77, and it can be seen from this graph that players in two of the experiments did in fact score goals. Although the population average fitness indicates that the average player developed no consistent goal-scoring behaviour, some individual players did score multiple goals. This can be seen in the goal-scoring frequency data presented in Figure 78 and Table 15. Table 15 lists, for each of the experiments, the number of valid chromosomes (players) evaluated throughout the evolutionary process and the number of players that scored 0 to 8 and >8 goals.

Experiment	Valid Evaluations	Number of Players Scoring n Goals									
		0	1	2	3	4	5	6	7	8	>8
1	2,498,864	2,498,864	0	0	0	0	0	0	0	0	0
2	2,499,651	2,499,651	0	0	0	0	0	0	0	0	0
3	2,499,754	2,487,171	12,484	99	2	0	0	0	0	0	0
4	2,498,562	1,815,847	498,726	142,311	33,909	6,553	1,081	134	0	1	0

Table 15 Goals-only Fitness: Number of Individuals Scoring Goals

6.7 Summary and Discussion

This chapter of the thesis has described a series of experiments, conducted in the SimpleSoccer environment, for which a number of control parameters and player attributes were varied. The experiments, and the comparisons presented in the following sections, show clearly the effect on the performance of the evolutionary search of varying the range of skills available to the player, as well as the effect of varying the fitness function.

The specific research question investigated by the work presented in this chapter is:

How does varying the

- *skills available to the players, and/or*
- *the implementation of the fitness function*

affect the result of the evolutionary search?

The outcomes in relation to that research question are detailed in the following sections.

6.7.1 Varying Player Skills

Experiments were conducted using a composite fitness function and with various combinations of hand-coded, mid-level skills available to the player being evolved. A tabulation of the results is given in Table 16 which shows, for each of the experiments conducted:

- the skills of the players in addition to the base skills *turn*, *kick* and *dash*
- the population average fitness at the end of the final generation
- the best fitness achieved by any player throughout the evolutionary process
- the highest number of goals scored by any player in a single game (related to best fitness)

The results of these experiments demonstrate that some of the mid-level skills defined are more effective goal-scoring skills than others, and that certain combinations of skills produce more consistent goal-scoring behaviour than others. This is not unexpected, and in fact the notion that players with a higher level of innate skills will perform better, and therefore be more easily located by any search algorithm, is an intuitive one.

It is clear that the range of skills available to the player being evolved is a very significant factor in determining the performance of the evolutionary search. Varying the skills available to the players changes the quality of the initial population (in terms of ball-kicking and goal-scoring ability), and so moves the players in the initial population closer to or further from the desired result.

The experiments performed also demonstrated that as the default action is made less intelligent the performance of the search decreases, indicating that an innate ability to locate the ball is very important to the outcome of the search.

Varying the skills available to the player, either by limiting the range of mid-level skills available as consequent actions or by varying the effectiveness of the player's default action effectively changes the difficulty of the problem for the evolutionary algorithm. The results of the experiments performed show that ability of the genetic algorithm to find good solutions is limited by the skills available to the players being evolved - the genetic algorithm needs the players to have a certain level of initial skill in order for the algorithm to reliably evolve players that exhibit consistent goal-scoring behaviour. While the base level skills (i.e. *turn*, *kick* and *dash*) are sufficient for players to exhibit occasional goal-scoring behaviour, and the evolutionary algorithm occasionally compensates for a lack of a mid-level hand-coded skill by

Skills Additional to Base Skills	Experiment													
	1	2	3	4	5	6	7	8	9	10	11	12	13	14
RunTowardGoal		X	X	X	X	X	X	X						
RunTowardBall		X	X	X	X	X	X		X					
GoToBall			X	X	X	X	X			X				
KickTowardGoal				X	X	X	X				X			
DribbleTowardGoal					X	X	X					X		
Dribble						X	X						X	
Added Directions							X							X
	Default Action = Hunt Action 1													
Average Fitness	0.849	0.742	0.773	0.750	0.603	0.406	0.375	0.849	0.841	0.806	0.764	0.613	0.761	0.781
Best Fitness	0.25	0.1	0.1	0.071	0.071	0.026	0.026	0.167	0.167	0.167	0.028	0.033	0.167	0.083
Most Goals Scored	2	5	5	7	7	19	19	3	3	3	18	15	3	6
	Default Action = Hunt Action 2													
Average Fitness	0.883	0.778	0.762	0.757	0.600	0.595	0.589	0.904	0.761	0.821	0.875	0.609	0.760	0.758
Best Fitness	0.125	0.125	0.1	0.071	0.071	0.071	0.071	0.167	0.1	0.125	0.083	0.063	0.125	0.1
Most Goals Scored	4	4	5	7	7	7	7	3	5	4	6	8	4	5
	Default Action = Hunt Action 3													
Average Fitness	0.992	0.991	0.993	0.978	0.974	0.958	0.935	0.991	0.995	0.994	0.991	0.986	0.992	0.993
Best Fitness	0.5	0.5	0.25	0.25	0.1	0.1	0.1	0.25	0.25	0.5	0.25	0.167	0.25	0.25
Most Goals Scored	1	1	2	2	5	5	5	2	2	1	2	3	2	2
	No Default Action - Hunt Consequent Allowed													
Average Fitness	0.997	0.979	0.997	0.997	0.929	0.932	0.911	0.998	0.976	0.996	0.997	0.941	0.968	0.987
Best Fitness	0.5	0.25	0.5	0.5	0.125	0.1	0.125	0.5	0.25	0.5	0.25	0.071	0.167	0.125
Most Goals Scored	1	2	1	1	4	5	4	1	2	1	2	7	3	4
	No Default Action – No Hunt Consequent													
Average Fitness	0.998	0.975	0.972	0.970	0.942	0.966	0.959	0.998	0.963	0.998	0.997	0.953	0.955	0.991
Best Fitness	0.5	0.25	0.167	0.25	0.1	0.063	0.125	0.25	0.167	0.5	0.5	0.071	0.167	0.125
Most Goals Scored	1	2	3	2	5	8	4	2	3	1	1	7	3	4

Table 16 Composite Fitness Function; Varying Player Skills

using one or more of the other skills available, the method generally does not seem to consistently and reliably construct reusable mid- and high-level skills from the base primitives. This suggests that the evolutionary algorithm as implemented for this work only compensates for a lack of skill when it needs to – if goal-scoring behaviour can be achieved by using the skills available the method generally does not construct any new, higher-level skills. Similarly, the experiments with no default action would indicate that the hunt action available as a consequent is not used if goals can be scored without it. Upon reflection, this is no more or less than should be expected – the algorithm will do only what is required to solve the problem it is given and no more. The corollary is that evolutionary algorithms may be just as impacted by too much help as too little – not enough quality in the initial population and the algorithm has nothing to work with, but too much quality in the initial population and the algorithm does not need to evolve different (better) ways to find a solution.

Specifically with regard to the problem of evolving goal-scoring behaviour in a robot soccer environment, the quality of the initial population, and therefore the amount of evolution required to solve the problem, is governed largely by the level of skill apparent in the initial population – both by way of the default action and the hand-coded mid-level skills available as consequent actions. The player needs sufficient innate skill to enable it to be a potential solution, otherwise any search would fail, but the question of what level of innate skill is reasonable remains unanswered – not enough and the problem is not solvable; too much and the problem is essentially solved before the algorithm is applied.

6.7.2 Varying the Fitness Function

Experiments were conducted using a simple goals-only fitness function, and with both the base level of skills and the base level plus all hand-coded, mid-level skills available to the player being evolved. A tabulation of the results is given in Table 17 which shows, for each of the experiments conducted:

- the skills of the players, including the player's default action
- the population average fitness at the end of the final generation
- the best fitness achieved by any player throughout the evolutionary process
- the highest number of goals scored by any player in a single game (related to best fitness)
- the average fitness, best fitness and goals scored for the equivalent experiments using the composite fitness function (from Table 16).

Exp	Player Skills	Hunt Consequent	Default Action	Goals-only Fitness			Composite Fitness		
				Average Fitness	Best Fitness	Most Goals Scored	Average Fitness	Best Fitness	Most Goals Scored
1	Base Skills	No	None	1.0	1.0	0	0.998	0.5	1
2	Base Skills	Yes	None	1.0	1.0	0	0.997	0.5	1
3	All Skills	No	None	0.998	0.167	3	0.959	0.125	4
4	All Skills	No	Hunt Action 1	0.837	0.063	8	0.375	0.026	19

*Base Skills = turn, kick, dash & no added directions; All Skills = Base + all hand-coded skills & added directions

Table 17 Simple Goals-only and Composite Fitness Functions; Varying Player Skills

The results of these experiments demonstrate very clearly that removing all the hints previously given to the genetic algorithm, in the form of default actions, hand-coded skills and a composite fitness function, severely limits the ability of the algorithm to find solutions to the problem.

These results, when compared to the results for the experiments conducted with the composite fitness function, indicate that the composite fitness function is a useful means of guiding the evolutionary search. Using a simple goals-only fitness function rather than a composite fitness function makes the problem much harder for an evolutionary algorithm, and this is likely due to the gradient information of the fitness landscape that guides the search being much more sparse and difficult to find. The number of players found that score goals is evidence that there are solutions in the search space. With a goals-only fitness function the only gradient information in the fitness landscape is given by players scoring different numbers of goals, whereas with a composite fitness function that gradient information is augmented by the different fitness values achieved by players managing to move the ball close to the goal. The problem the evolutionary algorithm faces with a goals-only fitness function is that it does not see the “extra” gradient information provided by a composite fitness function – gradient information that could lead the algorithm to the more sparsely distributed peaks on the fitness landscape defined by goal scorers.

A player with no innate knowledge or skills does not “know” that it needs to find the ball in order to score goals. Such a player does not know that kicking the ball is a requirement for scoring goals. Until the player stumbles across some behaviour that is rewarded by the fitness function it knows nothing – hence the advantage of the composite fitness function and rewarding for incremental behaviours.

A composite fitness function decomposes a large problem into a number of smaller problems and essentially provides a recipe for solving those smaller problems. The mere implementation of a composite fitness function indicates that someone has applied their own intelligence to the problem to decide how it might be broken into smaller problems and what those smaller problems will be – to the extent that an estimation is made of the size of the problem able to be solved by the particular algorithm being used and what behaviours need to be rewarded in order to solve those incremental problems.

The regular implementation of composite fitness function reported in the literatures seems to be an acknowledgement that evolutionary algorithms are unable to solve very difficult problems as monolithic problems – that very difficult problems can only be solved by evolutionary algorithms if the problem is first decomposed into a number of smaller problems.

6.8 Conclusions

The results presented in this chapter indicate that both the level of skills with which players are endowed and the fitness function used to evaluate players can have a great impact on the performance of the evolutionary search. The analysis of the experiments performed tends to indicate that unless at least one of those ingredients is present in a form advantageous to the search algorithm - either a high level of skills or a composite fitness function - the evolutionary algorithm is unable to find a solution to the problem. This is corroborated by the analyses of the previous work of Luke, Riedmiller et al., and Chellapilla and Fogel presented in section 2.7 on page 62.

Tripping hither, tripping thither, nobody knows why or whither.

“Iolanthe”, Gilbert and Sullivan, 1882.

Chapter 7

Analysis of the Search Spaces and Fitness Landscapes

7.1 Introduction

This chapter of the thesis presents an analysis of the search spaces and some features and statistics of the fitness landscapes associated with the trials from Chapter 6 conducted in the SimpleSoccer environment. The objective of the analysis is to better understand why the robot soccer problem is difficult for evolutionary algorithms, and how changing the player skills and fitness evaluation changes the fitness landscape, hence the difficulty of the problem.

7.2 Chapter Goals

As described in Chapter 6, previous work in the area of evolving goal-scoring behaviour has resulted in varying degrees of success and a general consensus that this is a “difficult” problem (see section 2.6.3 on page 59), and this is somewhat at odds with the results reported in Chapter 4 of this thesis. In Chapter 6 it was shown that the amount of help given to the evolutionary algorithm by way of initial player skills and the fitness function has a significant effect on the performance of the algorithm, and that injecting a copious amount of human expert knowledge can transform a difficult problem into a much less difficult one. This chapter examines the underlying reasons for the level of difficulty of the problem and how that is affected by initial player skills and varying the fitness functions.

The specific research question examined in this chapter of the thesis is:

Why is evolving goal-scoring behaviour from scratch in a simulated robot soccer environment a difficult problem for evolutionary algorithms?

The examination of this question will be achieved through an analysis of the search spaces and fitness landscapes associated with robot soccer, or more specifically the search spaces and fitness landscapes associated with the problem of evolving goal-scoring behaviour using a fuzzy rule-based representation for the soccer player, and a messy-coded genetic algorithm to conduct the evolutionary search.

While this research question is examined in the context of the SimpleSoccer environment, a fuzzy rules-based representation and the messy-coded genetic algorithm, it is believed that due to the nature of the results and the underlying cause of the difficulty of the search, the results are representation and simulator independent, and apply equally well to the general evolutionary algorithm as they do to the specific algorithm used in this work. This will become more apparent as the search space and fitness landscape analyses are presented.

7.3 The Robot Soccer Search Space

For the simulated soccer problem being studied in this work, multiple search spaces are explored depending upon the structure of the players being evolved (in this case, the “skills” available to the players). The search algorithm employed, a genetic algorithm, uses a population of potential solutions in the form of chromosomes, each representing a complete set of rules governing the behaviour of an individual player, to conduct a partial search of the search space. In this work, the behaviour of a simulated soccer player is defined by a set of fuzzy if-then rules, and it is this set of rules which is being evolved, so the search space is the collection of all possible rulesets.

As described in section 4.5 on page 93, a complete set of rules is an array with each array element being either a premise or a consequent. A ruleset may therefore contain multiple rules with the boundaries of each rule being defined by the placement of rule consequents. For example, the array of premises and consequents shown in Figure 79 is a complete ruleset of five rules, four valid and one invalid, consisting of eight premises and five consequents.

p01	p02	c01	p03	c02	c03	p04	p05	p06	c04	p07	p08	c05
-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----

Figure 79 Example Ruleset Array: p## is a premise and c## a consequent

Following is an explanation of how the size of the various search spaces is calculated. For convenience some of the information regarding the representation of the chromosome presented in section 4.5 on page 93 is reproduced here.

For the experiments in which the *VISIBLE* variable was not used, a premise is of the form:

(Object, Qualifier, {Distance | Direction}, Connector)

and for the experiments in which the *VISIBLE* variable was used, a premise is of the form:

(BALL, Qualifier, {Distance | Direction | *VISIBLE*}, Connector)

or

(GOAL, Qualifier, {Distance | Direction}, Connector)

A premise is constructed from the following range of possible values:

Object: { BALL, GOAL }

Qualifier: { IS, IS NOT }

Distance: { AT, VERYNEAR, NEAR, SLIGHTLYNEAR, MEDIUMDISTANT, SLIGHTLYFAR, FAR, VERYFAR }

Direction: { LEFT180, VERYLEFT, LEFT, SLIGHTLYLEFT, STRAIGHT, SLIGHTLYRIGHT, RIGHT, VERYRIGHT, RIGHT180 }

Connector: { AND, OR }

For the experiments in which the *VISIBLE* variable was not used, the number of possible premises is given by:

$$\begin{aligned} \#Object \times \#Qualifier \times (\#Distance + \#Direction) \times \#Connectors &= 2 \times 2 \times (8 + 9) \times 2 \\ &= 136 \end{aligned}$$

and similarly, for the experiments in which the *VISIBLE* variable was used, the number of possible premises is:

$$(2 \times (8 + 9 + 1) \times 2) + (2 \times (8 + 9) \times 2) = 140$$

A rule consequent is of the form:

(Action, {Direction | Null}, {Power | Null})

and, with the inclusion of the *HUNT* action is constructed from the following range of possible values:

Action:	{ TURN, DASH, KICK, RUNTOWARDGOAL, RUNTOWARDBALL, GOTOBALL, KICKTOWARDGOAL, DRIBBLETOWARDGOAL, DRIBBLE, DONOTHING, HUNT }
Direction:	{ LEFT180, VERYLEFT, LEFT, SLIGHTLYLEFT, STRAIGHT, SLIGHTLYRIGHT, RIGHT, VERYRIGHT, RIGHT180 }
Added Directions:	{ TOWARDBALL, TOWARDGOAL }
Power:	{ VERYLOW, LOW, SLIGHTLYLOW, MEDIUMPOWER, SLIGHTLYHIGH, HIGH, VERYHIGH }

The total number of possible consequents is the sum of the number of possible consequents for each action, taking into consideration the utilisation or not of the *Added Directions*. The values for each action are shown in Table 18.

A ruleset is initially constructed by randomly initialising each array element as either a valid premise or a valid consequent, with the probability an element will be initialised as a premise 0.95, and the probability an element will be initialised as a consequent 0.05. The last element of the ruleset so constructed is forced to be a (randomly selected) consequent.

For an array of L elements, corresponding to a chromosome of L genes, the *maximum* possible number of rulesets is:

$$\begin{aligned} \#Consequents \times ((\#Premises + \#Consequents)^{(L-1)}) &= 209 \times ((140 + 209)^{(L-1)}) \\ &= 209 \times 349^{(L-1)} \end{aligned}$$

since it is possible for any element, except the last, to be initialised as a premise or a consequent.

CONSEQUENT ACTION	Direction Argument Count		Power Argument Count	Number of Possible Consequents	
	No Added Directions	Added Directions		No Added Directions	Added Directions
Turn	9	11	<i>n/a</i>	9	11
Dash	9	11	7	63	77
Kick	9	11	7	63	77
RunTowardGoal	<i>n/a</i>	<i>n/a</i>	7	7	7
RunTowardBall	<i>n/a</i>	<i>n/a</i>	7	7	7
GoToBall	<i>n/a</i>	<i>n/a</i>	7	7	7
KickTowardGoal	<i>n/a</i>	<i>n/a</i>	7	7	7
DribbleTowardGoal	<i>n/a</i>	<i>n/a</i>	7	7	7
Dribble	<i>n/a</i>	<i>n/a</i>	7	7	7
DoNothing	<i>n/a</i>	<i>n/a</i>	<i>n/a</i>	1	1
Hunt	<i>n/a</i>	<i>n/a</i>	<i>n/a</i>	1	1

Table 18 Number of Consequents per Action

The value of L used in this work is 64, so the number of rulesets in the largest search space for any experiment conducted for this work (corresponding to the base set of skills plus all additional hand-coded skills, *Added Directions*, and the use of the *VISIBLE* premise variable and the *Hunt* consequent action) is:

$$\begin{aligned}
 209 \times ((140 + 209)^{(L-1)}) &= 209 \times 349^{(L-1)} \\
 &= 209 \times 349^{63} \\
 &\cong 3.3 \times 10^{162}
 \end{aligned}$$

The smallest search space for any experiment conducted for this work is the search space for just the base set of skills (*Turn*, *Dash* and *Kick*), no *Added Directions*, and no use of the *VISIBLE* premise variable or the *Hunt* consequent action, and the number of rulesets in this search space is:

$$\begin{aligned}
 135 \times ((136 + 135)^{(L-1)}) &= 135 \times 271^{(L-1)} \\
 &= 135 \times 271^{63} \\
 &\cong 2.6 \times 10^{155}
 \end{aligned}$$

A valid ruleset must contain at least one rule which has at least one premise and a single consequent. Some rulesets may contain invalid rules (the example ruleset array shown in Figure 79 on page 169 contains four valid rules and one invalid rule), but only rulesets containing all consequents (no premises) will not be valid rulesets. For the smallest and largest search spaces used in this work, the number of possible invalid rulesets is 135^{64} and 209^{64} respectively, so the percentage of the search space which is invalid is

$$\frac{100 \times 135^{64}}{(135 \times 271)^{63}} = 100 \times \left(\frac{135}{271} \right)^{63} \cong 8.6 \times 10^{-18} \quad \text{for the smallest search space, and}$$

$$\frac{100 \times 209^{64}}{(209 \times 349)^{63}} = 100 \times \left(\frac{209}{349} \right)^{63} \cong 9.4 \times 10^{-13} \quad \text{for the largest.}$$

It is clear that the search space for this representation of the soccer problem is extremely large and consists almost entirely of valid rulesets.

Some rulesets will be functionally equivalent even if they are not identical strings, but are still valid, searchable members of the search space. Whether the efficiency of the search could be improved by identifying that a functionally equivalent ruleset had previously been evaluated, thus obviating the need for re-evaluation, is both debatable and separate from the issue being addressed in this chapter.

7.4 Which Fitness Landscape?

A review of relevant literature (see section 2.4 on page 46) indicates there are several possible definitions of, and representations for, fitness landscapes, and choosing the definition and representation which best describes the combination of the problem being studied and the algorithm being used to study it is extremely important. A fitness landscape is most often defined by three basic attributes:

- a search space
- a relation that defines which points are neighbours in the search space
- a fitness function that assigns a fitness value to each point in the search space

A significant problem with this definition of fitness landscapes is the specification of the neighbourhood relation. The neighbourhood relation and its specification is extremely important because any discussion of landscapes invariably involves the terms “peaks” and “valleys”, and no peak or valley can exist without the notion of neighbourhood – a peak is only a peak because it is higher than its neighbours.

Often the neighbourhood relation is defined in simple terms, such as the hamming distance for bit strings, or by defining all single bit mutations of a bit string as neighbours. This potentially ignores an important ingredient in the evolutionary processes: evolutionary algorithms are usually governed by some combination of several operators. A definition of the neighbourhood relation in terms of the actual mix of genetic operators (for example, selection, mutation and crossover) being used by the algorithm would seem to be more relevant and useful to an analysis of the performance of the algorithm. Since the search is for individuals of good fitness, the individuals which comprise the search space and their associated fitness values should ideally be arranged as a fitness landscape which has individuals of better fitness clustered together so that the landscape contains peaks and valleys representing the fitness extremes. The issue is how to represent the fitness landscape to achieve that: *what attributes of an individual should determine the locality of that individual?*

If the neighbourhood relation is not defined correctly in terms of describing the actions of the algorithm being used to perform the search, then the fitness landscape is not the landscape searched by the search algorithm, and so not necessarily indicative of the difficulty of the search. On a physical landscape points are neighbours, or next to each other, because (for example) a person walking on that landscape can traverse the distance between those points in a single step. The corollary is that if a searcher walking on a landscape is not able to traverse the distance between two points in a single step, then those points are not next to each other and so, by definition, are not neighbours on the landscape defined by that searcher and the associated granularity of search (a single step). Simply plotting attributes of the members of the search space against fitness – for example chromosome length in the case of a genetic algorithm, or program length for genetic programming – while possibly useful for visualising the fitness distribution of the search space, may not be sufficient to describe the fitness landscape traversed by the search algorithm, since there may be no reason to expect that the

search algorithm could traverse the distance between neighbouring points on that landscape with a single step.

A fitness landscape is a metaphor and an aid to the visualisation of the operation of a search algorithm, but for anything other than the actual landscape traversed by the search algorithm the metaphor is flawed and the visualisation is misleading. For example, the analogy of the search algorithm to a short-sighted explorer wandering over a (possibly rough) terrain [164] is only valid for the actual fitness landscape searched by the algorithm, and anything else gives a misleading view of the complexity, or ruggedness, of the landscape. For the explorer analogy to be useful, the neighbourhood relation must reflect the notion that an individual's neighbour on the fitness landscape is any individual in the search space which can be reached in a single step of the explorer. In the case of an evolutionary algorithm, the explorer is the algorithm, and the single step is the combination of whichever genetic operators are implemented by that algorithm: selection, mutation and crossover. It should also be remembered that during the search the explorer may not be able to see all its theoretical neighbours. This is because, for an evolutionary algorithm using crossover, the individuals reachable in a single step from one parent depend upon the other individuals selected for mating, and while an individual can theoretically mate with any other individual in the search space, in reality the individuals available for selection are restricted by the size of the population. So to continue the analogy of the short-sighted explorer, not only is the explorer short-sighted, but also lacks peripheral vision.

While it should be remembered that the 3-dimensional fitness landscape is just a metaphor, since in most cases the surface being traversed by the searcher will be multi-dimensional and so complex that there will be no landscape that could actually be visualised, the analogy of the searcher on a 3-dimensional landscape is a useful aid to imagining how a search algorithm might use the available attributes of the search space.

It is important to understand that the (metaphoric) fitness landscape is defined by a number of attributes, which include the search algorithm, and not solely by the specification of the problem. It is not valid to visualise a fitness landscape without reference to a search algorithm, and then decide what algorithm will best search the landscape – the operation of the search algorithm defines the neighbourhood relation, and so the shape, of the landscape.

Previous work with fitness landscapes recognises the need to refer to the search algorithm when characterising the landscape [124, 140, 216, 262]. Jones and others [124, 137] further suggest that each operator employed by a search algorithm (e.g. selection, mutation and crossover for a genetic algorithm) should be viewed as operating on its own landscape. The notion that search operators define and act on separate landscapes may be useful for studying the effect of individual operators, but the combined effect of each move on each landscape needs to be considered. If the algorithm employs multiple operators, then the output of the algorithm is some combination of those operators, so it is not reasonable to consider a move on the “mutation landscape” without considering how that then affects a subsequent move on the “crossover landscape”.

A further problem with the “one operator, one landscape” approach is determining what constitutes an operator. Jones [137] has, in the case of evolutionary algorithms, suggested that selection, mutation and crossover should each define and operate on separate landscapes, but none of those operators are necessarily atomic. Is an operator considered to define its own landscape simply because it has a well-known label? The breakdown of the evolutionary operation into the three operators of selection, mutation and crossover seems somewhat arbitrary.

It is clear from the analysis of the literature that the neighbourhood relation for fitness landscapes is not well understood and even less well defined, making it imperative for any real analysis of the fitness landscape associated with a particular problem to be preceded by a clear definition of the neighbourhood relation. The definition of the neighbourhood relation and fitness landscape for the robot soccer problem addressed in this thesis is presented in the following section.

7.5 Fitness Landscape Definition

For this work the fitness landscape is considered to be defined by the overall operation of the genetic algorithm. Consider an observer watching a genetic algorithm searcher perform a *random walk* on a fitness landscape (where the selection of individuals is random rather than based on fitness), and assume that although the observer is able to discern the granularity of the

search (the genetic algorithm's single steps), the means by which the GA determines where each step takes it is hidden from the observer. The random walk is conducted as follows:

- An individual i_0 is randomly selected from the search space
- For each step s , $s = 1 \dots \text{maxsteps}$
 - i_{s-1} undergoes mutation with probability P_{mutation}
 - Another individual i , $i \neq i_{s-1}$, is randomly selected from the search space
 - Crossover is performed between i and i_{s-1} with probability $P_{\text{crossover}}$, resulting in two new individuals i'_1 and i'_2 , both of which are neighbours (a single step from) of i_{s-1}
 - Set $i_s = i'_1$ and step to i_s

The observer sees the searcher walking randomly over the landscape and considers points on the landscape one step apart to be neighbours. The definition of the neighbourhood relation is of no consequence to, and is not required by, the observer since the searcher is defining neighbouring points by performing the walk. If the random walk performed by the genetic algorithm searcher was sufficiently long, and the “altitude” (fitness) at each step recorded for the observer, the entire fitness landscape would be determined by observation. The landscape so determined would be the precise fitness landscape defined by the search algorithm.

This “black box” view of the genetic algorithm operation and consequential determination of the neighbourhood relation and fitness landscape satisfies the requirement that the landscape neighbourhood relation be defined by the search algorithm, and is the definition used for the robot soccer problem addressed by this thesis.

7.6 Search Space and Fitness Landscape Analysis

In light of the experiments reported in Chapter 6, a natural question to ask is “*How does changing the skills available to the soccer player affect, or change, the fitness landscape?*” The results presented in Chapter 6 clearly indicate that different initial skills mean that players achieve different fitness values, so it would seem to be somewhat intuitive that changing the skills available to the players in some way alters the fitness landscape over which the search is conducted. In fact the fitness landscape is altered by changing the skills available to players, but it changes because the underlying search space is different for each set of skills: the chromosomes are interpreted differently for each skillset, and in fact may take on different values for each skillset depending upon the range of skills available in the skillset, so the

individuals (players) that comprise the search space are different for each skillset. Since changing the skills available to the players defines a new search space it also defines a new fitness landscape. The same is not true of changing the function used for the fitness evaluation. Different fitness functions alter the fitness landscape, not the underlying search space, because the individuals that comprise the search space remain the same. Furthermore, since only the means by which the individuals are evaluated is changed, the change to the fitness landscape is not in the neighbourhood relation but only in the fitness values of the individuals. For this reason, changing the fitness function has the potential to significantly change the fitness landscape, and so affect the effectiveness of the search process. Some fitness functions could make “mountains out of molehills” – teasing gradient information out of otherwise flat landscapes. Sometimes this is valid, but sometimes the definition of the fitness function effectively solves the problem for the search algorithm.

In the following sections the results of some of the experiments described in Chapter 6 will be analysed in order to examine how the fitness landscape, and the resulting search over it, is affected by the varying the skills of individual members of the population, as well as the method by which those members are evaluated.

7.6.1 Search Space and Fitness Landscape Representation

Since the search space defined by this work is prohibitively large, it is impossible to enumerate and show the entire space graphically. In order to gain some idea of what a fitness landscape might look like for the robot soccer problem, a subset of the search space is investigated and presented graphically. As a visual aid to understanding the evolutionary process, the individuals evaluated during the trials conducted for Chapter 6 of this thesis are plotted on a series of fitness graphs. These graphs help visualise the evolution of the fitness of the population over 500 generations without the need for a visual indication of a neighbourhood relationship. Each graph represents the fitness of the individuals comprising the population at the completion of a specific generation, so the series of graphs for a particular trial show the progress of the population over the length of the trial. Note that since each graph is a snapshot of the population at a specific generation it is possible that a particular instance which occurred during a different generation may not be shown.

The fitness graph is represented as a scatter graph and reflects the nature of the fitness function. The distance of an individual from the centre point of the fitness graph depends upon the fitness of the individual. The co-ordinates of an individual on the graph are calculated thus:

$$x = f \sin \theta$$

$$y = f \cos \theta$$

where

f is the fitness of the individual, and $0.0 \leq f \leq 1.0$

θ is a randomly assigned angle, and $0^\circ \leq \theta \leq 360^\circ$

The graph is:

- discrete for no ball movement,
- for the composite fitness function, continuous for ball movement but no goals scored
- discrete for goals scored

An example fitness graph is shown in Figure 80 on the following page, in which the fitness values of all individuals in the population at a given time have been plotted. It can be seen from the example that individuals with the same fitness values form discrete concentric rings about the centre point, and that there is a range of fitness values between 0.5 and ~0.77 representing fitness values assigned by the composite fitness function, with values in that range forming a cloud of points. The areas between the “fitness rings” indicate fitness values not defined by the fitness function. Note that not all fitness rings in Figure 80 are labelled.

As previously described, two fitness functions were used for the SimpleSoccer trials described in Chapter 6: a simple fitness function and a composite fitness function (see section 4.6.2 on page 97). The simple fitness function (Equation 5 on page 97) is just a measure of the number of goals scored by a player. The composite fitness function (Equation 7 on page 99) is a measure of the number of goals scored by a player, or if the player scored no goals then how close the player was able to move the ball to the goal. Both fitness functions indicate a better fitness as a lower number, so representing the optimisation of fitness as a minimisation problem. All SimpleSoccer fitness functions implemented in this work generate a real number R , where $0.0 < R \leq 1.0$, $R = 1.0$ indicates no ball movement and $R \approx 0.0$ indicates very good performance.

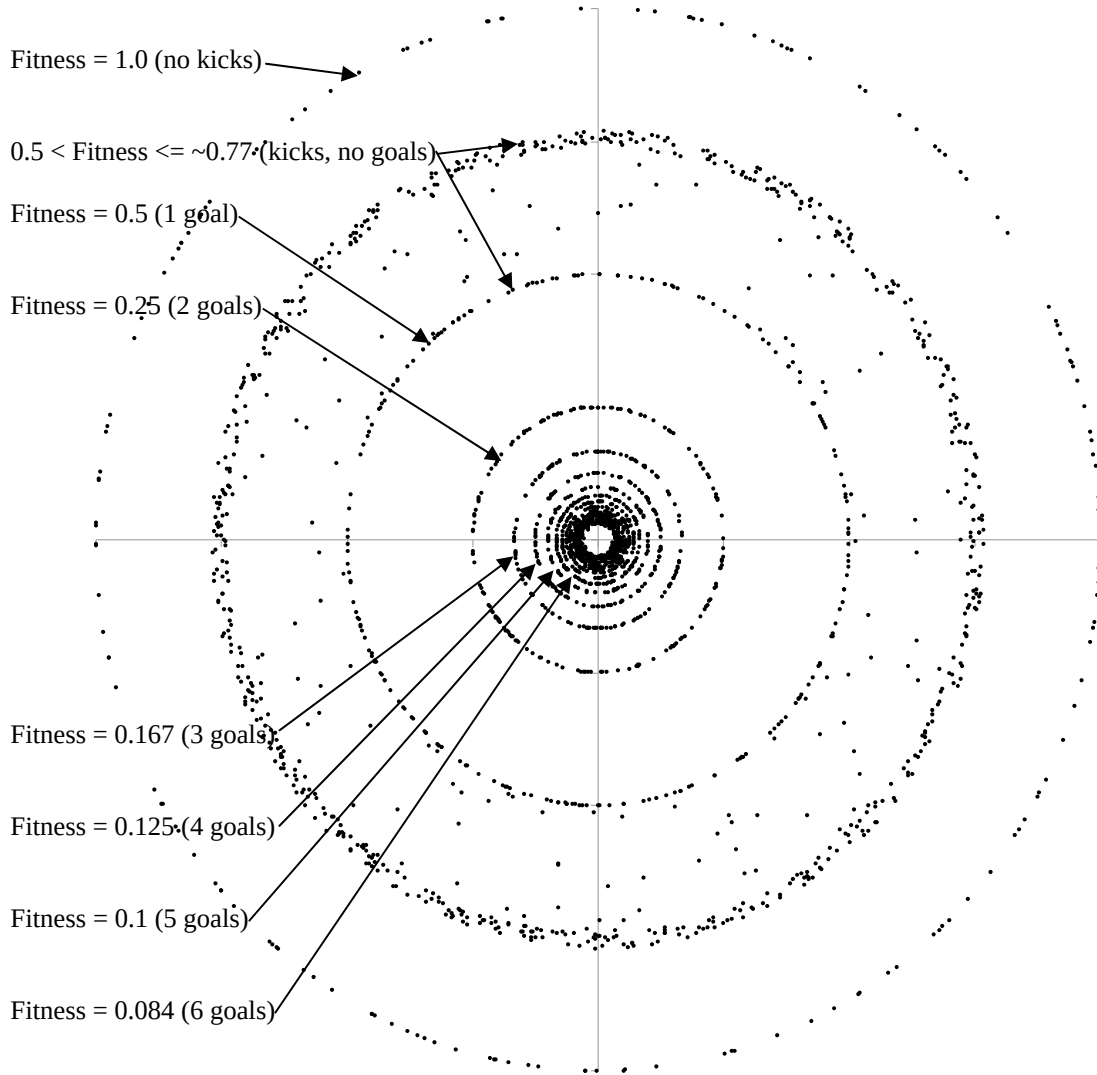


Figure 80 Example fitness graph

For ball movement, composite fitness functions assign values in the range $x \leq R \leq y$, where x and y are functions of, respectively, the smallest and greatest distance of the ball from the goal at any time without the ball being in the goal or out of the playing area. Since the ball always starts a game somewhere along the centre line of the playing area, and that the playing area size used was 31×61 , x and y are given by

$$x = 0.5 + \frac{\sqrt{(0^2 + 1^2)}}{2\sqrt{(16^2 + 62^2)}} \approx 0.508$$

$$y = 0.5 + \frac{\sqrt{(15^2 + 31^2)}}{2\sqrt{(16^2 + 62^2)}} \approx 0.769$$

Simple and composite fitness functions will assign discrete values for no ball movement (1.0) and goal-scoring ($\frac{1}{2 \times goals}$).

7.6.2 Fitness Landscapes Measures

As described in Chapter 2, the methods used in this thesis for measuring the structure of the fitness landscapes are Weinberger's *autocorrelation* method and Vassilev's *information content* method. These methods are described in detail in section 2.4.1 on page 50.

7.6.3 Isotropy

The assumption of statistical isotropy (see section 2.4.2 on page 54) is especially significant with the enormous search space of the robot soccer problem. As a means of determining whether the assumption is valid, several tests, each consisting of 20 random walks of 100,000 steps with different starting positions, were performed on the fitness landscapes of several of the experiments reported in Chapter 6. The autocorrelation, correlation length, and the three information measures were compared for each of the 20 walks in each test, and were found to be consistent with the landscapes being statistically isotropic. In addition, 10 random walks of 1,000,000 steps with different starting positions were performed on the fitness landscape of a selected experiment from Chapter 6. The autocorrelation, correlation length, and the three information measures for those walks were compared to 10 walks of 100,000 steps on the same landscape and were also found to be consistent with the landscapes being statistically isotropic.

While these tests were not exhaustive, and statistical isotropy cannot be guaranteed, the results of the tests do allow a degree of confidence that the fitness landscapes being analysed are statistically isotropic and therefore the application of the measures described is valid.

7.6.4 Analysis

Results of a selected number of the experiments described in Chapter 6 are analysed in this chapter. The experiments analysed are listed in Table 19.

Five random walks, as described in section 7.5 on page 176, were conducted for each experiment, each walk starting at a randomly selected point on the fitness landscape and continuing for a duration of 100,000 steps. The statistics gathered during the walks are analysed. The analysis is presented in three sections:

- the effect of the varying the skills given to players
- the effect of the hunt action
- the effect of a composite vs. simple fitness function is examined

Experiment	Fitness Function	Player Skills	Default Action	Hunt Consequent
1	Composite	All Skills*	Hunt Action 1	No
2	Composite	Base Skills*	Hunt Action 1	No
3	Composite	All Skills	Hunt Action 2	No
4	Composite	Base Skills	Hunt Action 2	No
5	Composite	All Skills	Hunt Action 3	No
6	Composite	Base Skills	Hunt Action 3	No
7	Composite	All Skills	None	Yes
8	Composite	Base Skills	None	Yes
9	Composite	All Skills	None	No
10	Composite	Base Skills	None	No
11	Goals-only	All Skills	Hunt Action 1	No
12	Goals-only	All Skills	None	No
13	Goals-only	Base Skills	None	Yes
14	Goals-only	Base Skills	None	No

*Base Skills = turn, kick, dash & no added directions; All Skills = Base + all hand-coded skills & added directions

Table 19 Experiments Analysed

For each experiment the following information is presented for analysis and comparison:

- *Data from the evolutionary search:*
 - Fitness scatter graphs showing the status of the population with regard to fitness at generations 1, 100, 200, 300, 400 and 500.
 - Line graphs showing the population average fitness and individual best fitness for generations 1 to 500.
 - A bar chart showing the cumulative fitness distribution for all individuals evaluated during the 500 generations, showing the percentage of all individuals evaluated which failed to kick the ball (denoted by “nm” on the graph x-axis), moved the ball closer to the goal (graduated), kicked one goal, two goals, three goals etc.
- *Data from the random walk over the fitness landscape:*
 - A correlogram showing the autocorrelation data for time lags 1 to 100 for the five random walks and the two-standard error bounds.
 - A graph showing the information content data for $0 \leq \mathcal{E} \leq 1.0$ for the five walks.
 - A table showing
 - the mean fitness and fitness variance of the points visited in the random walks.
 - the fitness distribution for all individuals evaluated during the random walks.
 - the average autocorrelation for the first time lag (i.e. for immediate neighbours on the random walk).
 - the average correlation length.
 - the average information content $H(\mathcal{E})$ and average partial information content $M(\mathcal{E})$ for selected $\mathcal{E}$.
 - the average information stability $\mathcal{E}^*$.

7.6.4.1 Varying Player Skills

In this section, the effect of varying the skills available to the players is examined. Experiments 1 to 10 from Table 19 were considered and compared in pairs – each pair of experiments differing only in the player skills. Since the results and comparisons of the different pairs of experiments are similar, only the comparison of experiments 1 and 2 is discussed in detail. Graphs and tables for experiments not discussed in detail in the following sections of this chapter are presented in Appendix .

Exp. 1: Composite Fitness; All Skills & Directions; Default Action = Hunt Action 1

Experiment 1 is the problem for which the search algorithm has been given the most help in the form of initial player skills, added directions, the default action, and a composite fitness function to guide the search. It is evident from the fitness scatter graphs for experiment 1 (Figure 81) that as early as 100 generations into the evolutionary process a good proportion of the population has evolved sufficient ball-kicking skills to at least kick the ball closer to the goal than where it started. The scatter graphs further indicate that most of the learning occurs in the first 100 generations, and that beyond that the fitness distribution of the population remains reasonably static. The data shown on the graph of population average fitness (Figure 81) tends to indicate that the population as a whole ceases to improve after 30 to 40 generations though, as evidenced by the graph of best fitness values, individuals of good fitness continue to be found beyond that point. The percentage of the population exhibiting ball-kicking or goal-scoring behaviour is reasonably high, as shown by the frequency distribution (Figure 81).

The difference in the number of generations required to evolve reasonable solutions in the experiments reported in this chapter and similar experiments reported in Chapter 4 can be explained by the difference in the players' field of vision in the two series of experiments. For the experiments reported in Chapter 4 the players' field of vision was much larger than for the experiments reported in this chapter. The extended field of vision allows players to more easily maintain or regain sight of the ball, resulting in much less time spent hunting for the ball thus more time for kicking goals.

From the data presented in Table 20 on page 187 it is apparent that although the mean fitness of the 100,000 individuals evaluated during the random walk is close to 1.0 (indicating no ball movement) and the variance small, the random walk did find individuals which exhibited goal-scoring behaviour, and in fact found more than 100 individuals which scored multiple goals.

The autocorrelation data shown in Figure 82 on page 186 and the correlation length given in Table 20 indicate that the fitness landscape for this problem (as described by the random walk) offers a reasonable amount of gradient information that the search algorithm can use to guide the search. With an autocorrelation of ~ 0.32 for points on the random walk a single step apart and a fairly steep descent for points further apart, the correlation between next and near neighbours

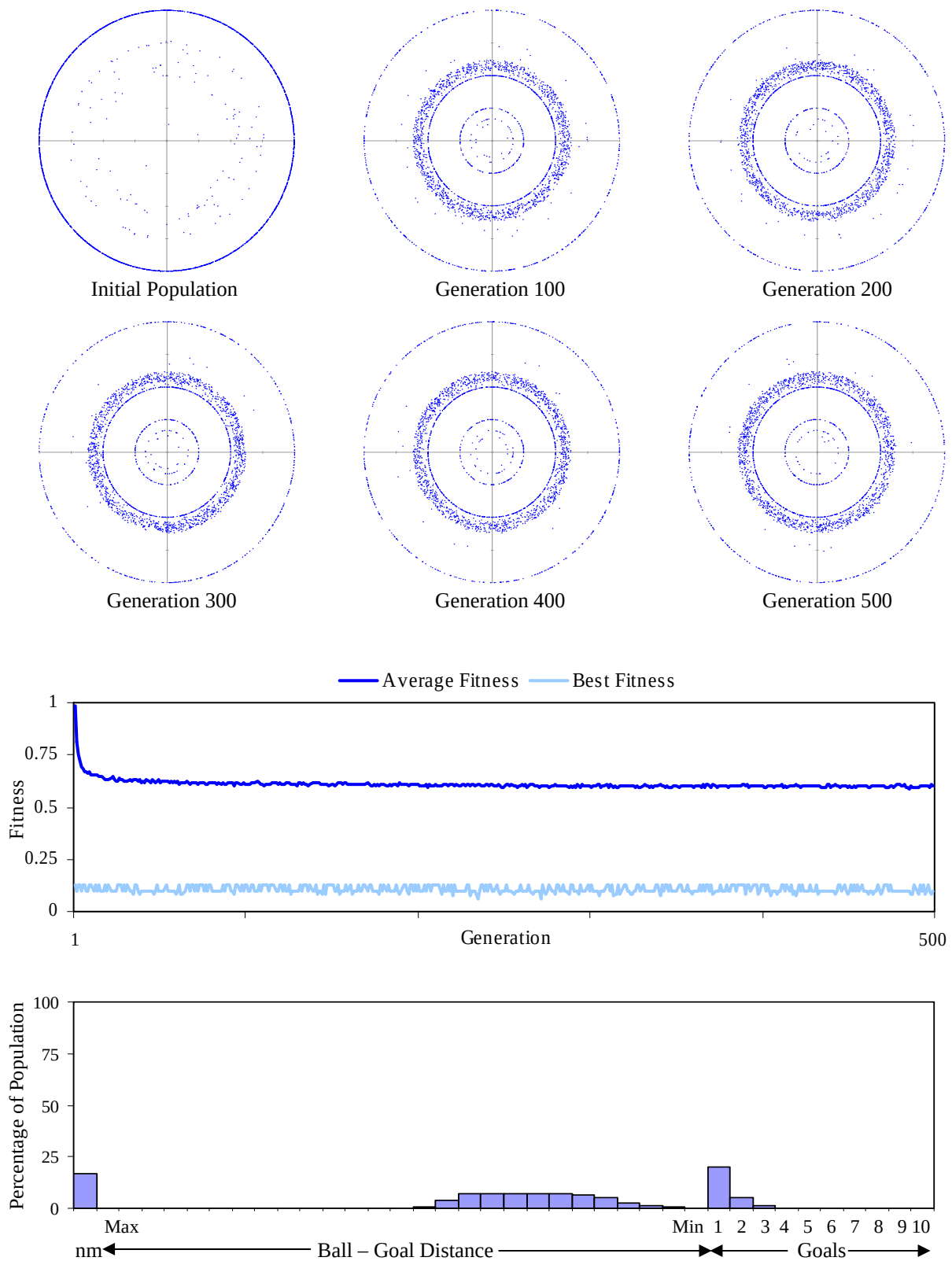


Figure 81 Composite Fitness; All Skills & Directions; Default Action = Hunt Action 1

on this fitness landscape is not so high that a search algorithm is led unerringly to a solution, but with a good correlation and a long correlation length the problem, in this form, should be readily solved by a search algorithm able to take advantage of the landscape features.

The information content graph shown in Figure 82 supports the autocorrelation data for this experiment. Information stability is quite high at 0.885, indicating a high difference in fitness among neighbouring points, so pointing to some good gradient information being present in the landscape. $H(0.0)$ is not particularly large, indicating that the diversity of shapes on the landscape is not high. Similarly $M(0.0)$ is relatively small, indicating that the degree of modality of the landscape is low.

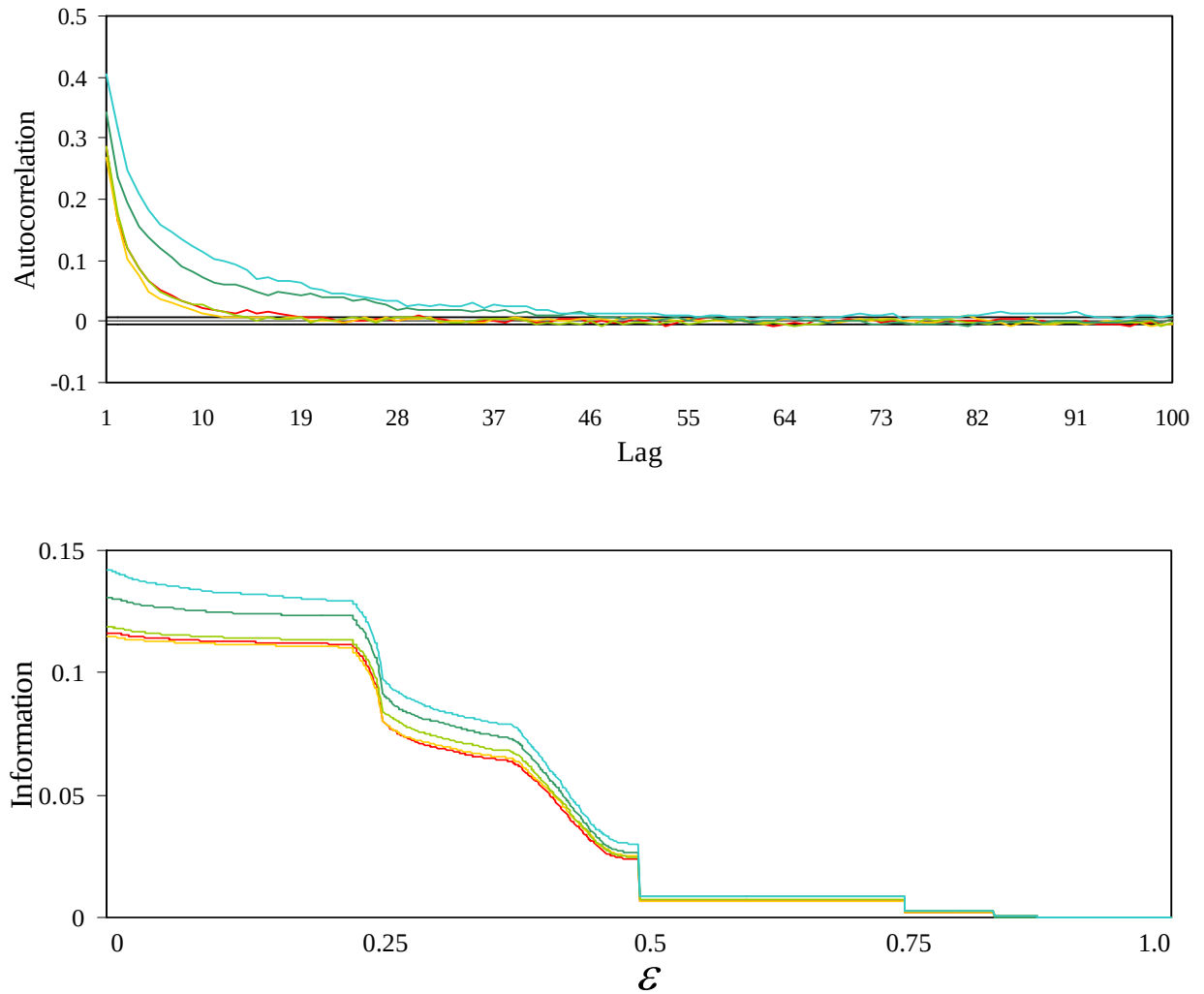


Figure 82 Composite Fitness; All Skills & Directions; Default Action = Hunt Action 1

Random Walk Statistics (Average over 5 walks)								
Mean Fitness	0.98283							
Fitness Variance	0.00669							
Individual Fitness	No Movement	Ball Movement	Goals					
			1	2	3	4	5	>5
Number of Individuals	95974	3462	434	99	26	4	1	0
Autocorrelation $r(1)$	0.31716							
Correlation Length	41							
$\mathcal{E}$	Information Content $H(\mathcal{E})$		Partial Information Content $M(\mathcal{E})$					
0.0	0.12447		0.03266					
0.2	0.09770		0.03020					
0.4	0.06179		0.01269					
0.6	0.00754		0.00098					
0.8	0.00229		0.00025					
0.885	0.00000		0.00000					

Table 20 Composite Fitness; All Skills & Directions; Default Action = Hunt Action 1

Exp. 2: Composite Fitness; Base Skills Only; Default Action = Hunt Action 1

The difference between this experiment and experiment 1 is that instead of the players being endowed with all possible skills and added directions, the players in this experiment have only the base skills of *turn*, *kick* and *dash*, and no access to the added directions. The search algorithm still has the aid of the default action and a composite fitness function to guide the search.

From the fitness scatter graphs for this experiment (Figure 83, page 188) it is apparent that by generation 200 the evolutionary search has converged and that the great majority of the population has evolved no more than sufficient ball-kicking skills to kick the ball closer to the goal than where it started. The data shown on the graph of population average fitness (Figure 83) indicates that improvement of the population stops at about generation 150, and

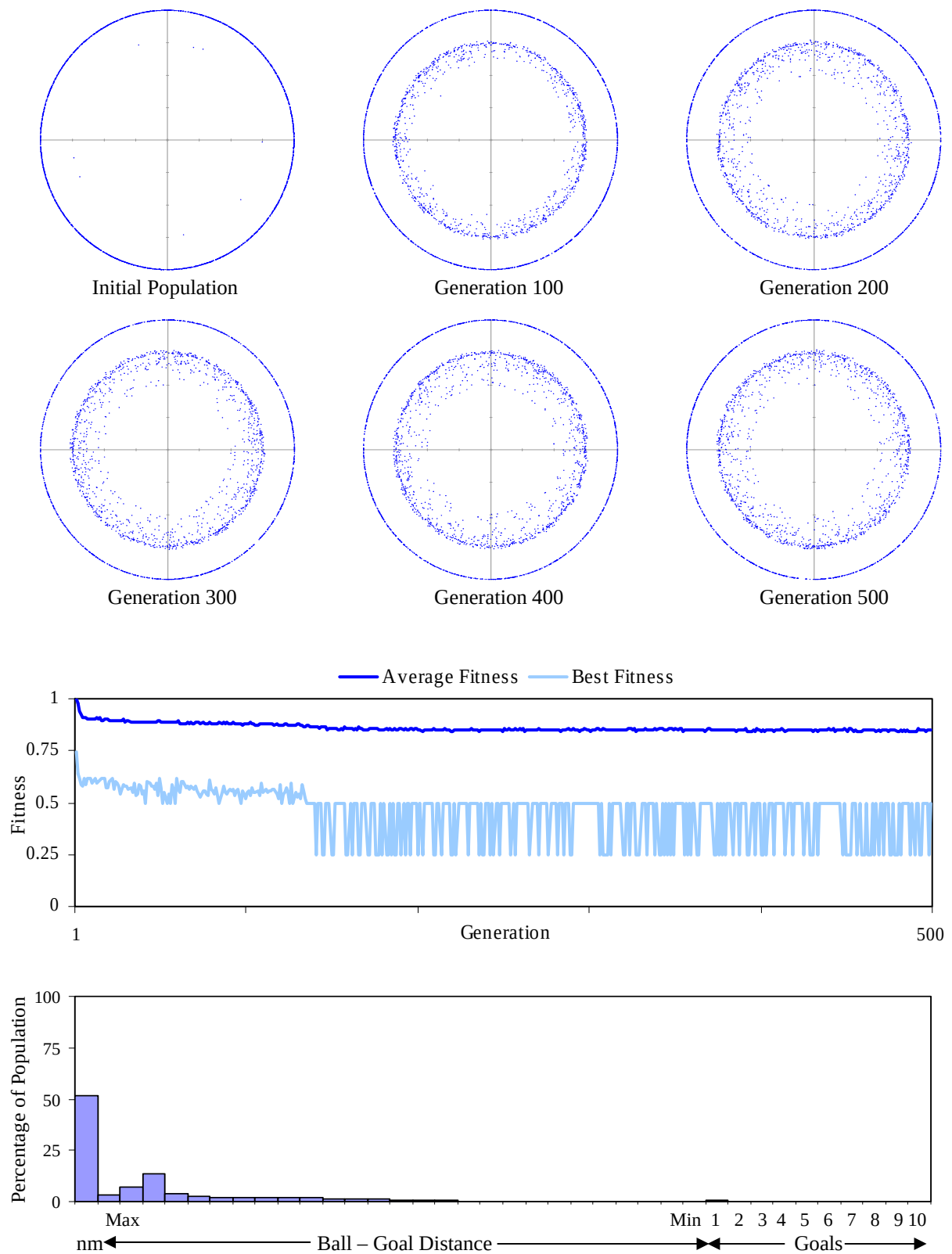


Figure 83 Composite Fitness; Base Skills Only; Default Action = Hunt Action 1

although the graph of best fitness values indicates that individuals exhibiting goal-scoring behaviour continue to be found, terminating the search after generation 150 would not have adversely affected the result. Figure 83 shows that the percentage of the population exhibiting goal-scoring behaviour is extremely small, with a very large proportion of the population not kicking the ball at all.

The results presented in the graphs in Figure 83 show clearly the effect of removing from the players a range of mid-level hand-coded skills. These results raise the question of what effect removing those skills has on the structure of the fitness landscape and how that affects the evolutionary search.

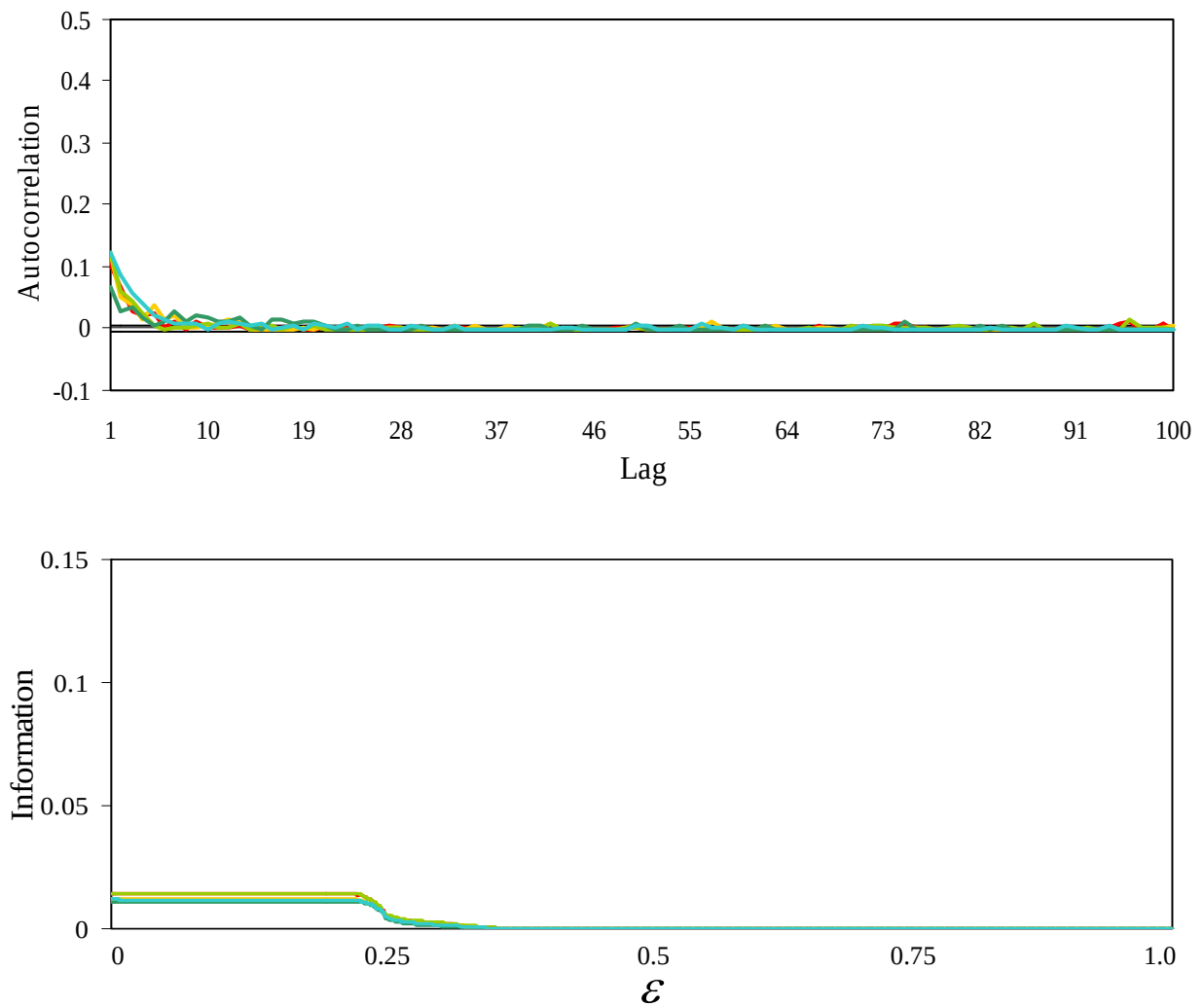


Figure 84 Composite Fitness; Base Skills Only; Default Action = Hunt Action 1

The autocorrelation data shown in Figure 84 and the correlation length given in Table 21 indicate that the fitness landscape for this problem offers only a limited amount of useful gradient information that the search algorithm can use to guide the search. With an autocorrelation of ~ 0.1 for points on the random walk a single step apart and a falling to zero for points just a few steps further apart, the correlation between next and near neighbours on this fitness landscape indicates that the structure of the fitness landscape is close to random and not as conducive to search as was the fitness landscape of experiment 1, thus increasing the difficulty of the problem.

Similar to experiment 1, the data presented in Table 21 shows the mean fitness of the 100,000 individuals evaluated during the random walk close to 1.0 (indicating no ball movement) and the variance very small, but in this case no individuals exhibiting goal-scoring behaviour were found during the random walk.

Random Walk Statistics (Average over 5 walks)								
Mean Fitness	0.99947							
Fitness Variance	0.00014							
Individual Fitness	No Movement	Ball Movement	Goals					
			1	2	3	4	5	>5
Number of Individuals	99776	224	0	0	0	0	0	0
Autocorrelation $r(1)$	0.10724							
Correlation Length	9							
$\mathcal{E}$	Information Content $H(\mathcal{E})$		Partial Information Content $M(\mathcal{E})$					
0.0	0.01268		0.00180					
0.1	0.01258		0.00178					
0.2	0.01256		0.00178					
0.3	0.00202		0.00022					
0.371	0.00000		0.00000					

Table 21 Composite Fitness; Base Skills Only; Default Action = Hunt Action 1

Again, similar to experiment 1 the information content graph, shown for this experiment in Figure 84, supports the autocorrelation data for this experiment. Information stability is relatively low at 0.371, indicating a low difference in fitness among neighbouring points. With the autocorrelation data indicating a near random landscape, and information stability

indicating a low fitness variation among neighbouring points, there is almost no useful gradient information in the landscape to guide the search. $H(0.0)$ is very small, indicating that the diversity of shapes on the landscape is very low. Similarly $M(0.0)$ is extremely small, indicating that the landscape lacks any real degree of modality. Both values further indicate the lack of useful landscape data to guide the search.

The data presented all indicates that the removal of a set of mid-level, hand-coded skills has changed the relative difficulty of the problem, and that this is a result of the structure and features of the fitness landscape being altered by the problem representation – what was a landscape reasonably rich in features that helped guide the search has become a relatively barren landscape lacking in information useful for search.

7.6.4.2 Varying the Hunt Action

The effect of the hunt action is examined in this section. Experiments 1, 3, 5, 7 and 9 from Table 19 (page 182) were considered and compared, but since the results are similar for all experiments only experiments 1, 3 and 5 are discussed in detail. The analysis of experiment 1 was presented in section 7.6.4.1 on page 183. The analyses of experiments 3 and 5 follow.

Exp. 3: Composite Fitness; All Skills & Directions; Default Action = Hunt Action 2

Both the form and the results of experiment 3 are very similar to experiment 1. The difference between experiment 3 and experiment 1 is that the default hunt action of experiment 3 stops when the ball has been located: it does not proceed to the ball as it did for experiment 1.

The fitness scatter graphs for experiment 3 (Figure 85, page 192) show that, as for experiment 1, most of the evolution was over by generation 100, and that by that stage a good proportion of the population has evolved sufficient ball-kicking skills to at least kick the ball closer to the goal than where it started. The data shown on the graph of population average fitness (Figure 85) indicates that the population as a whole ceases to improve after 30 to 40 generations. The percentage of the population exhibiting ball-kicking or goal-scoring behaviour is reasonably high, as shown by the frequency distribution (Figure 85). From the data presented in Table 22 it is apparent that, as for experiment 1, although the mean fitness of the 100,000 individuals evaluated during the random walk is close to 1.0 (indicating no ball movement) and the variance small, the random walk did find individuals which exhibited goal-scoring behaviour, and in fact found more than 50 individuals that scored multiple goals. The number of goal-scorers found was significantly less than for experiment 1.

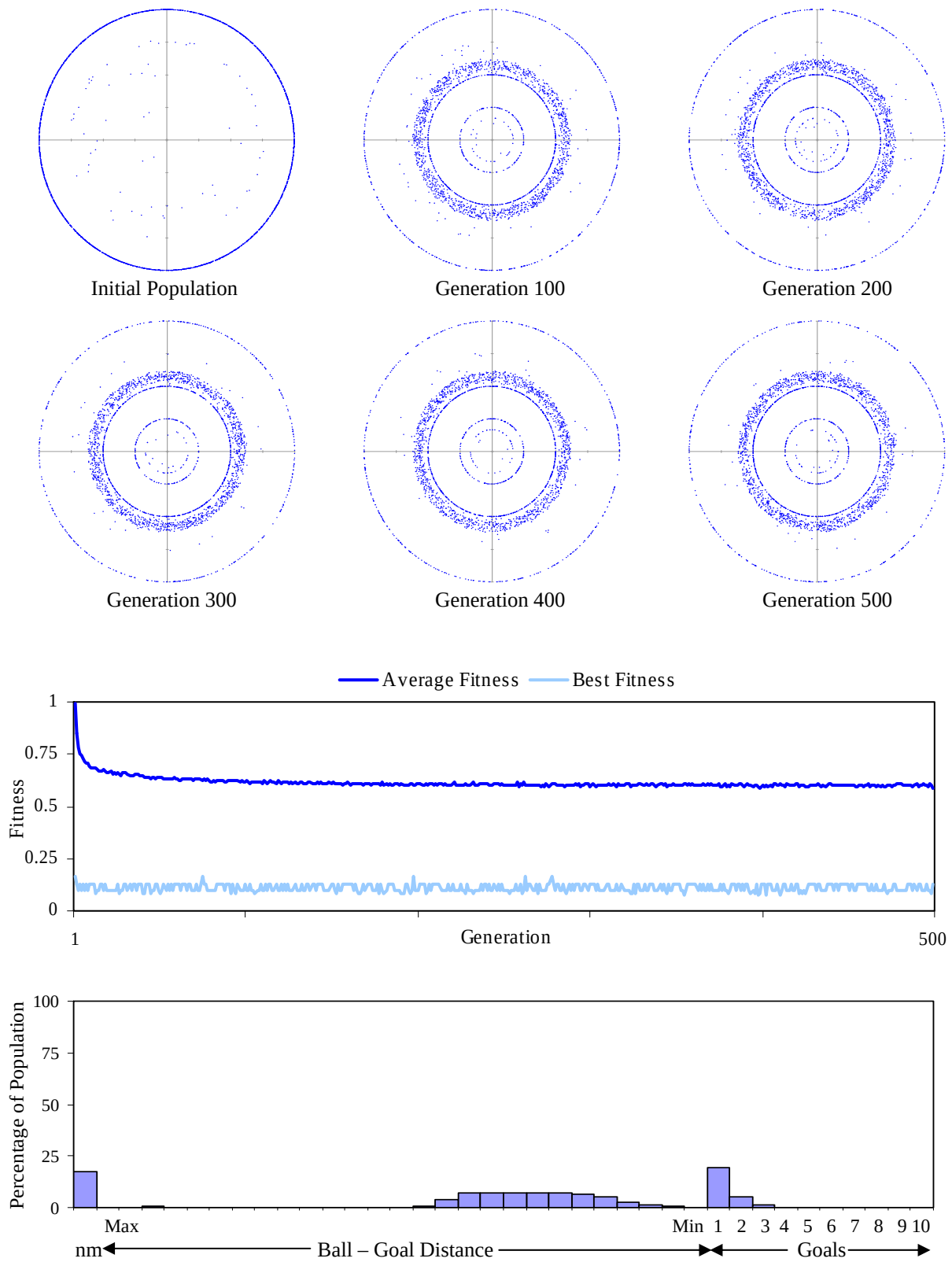


Figure 85 Composite Fitness; All Skills & Directions; Default Action = Hunt Action 2

The autocorrelation data shown in Figure 86 and the correlation length given in Table 22 (page 194) indicate that the fitness landscape for this problem (as described by the random walk) offers some gradient information that the search algorithm can use to guide the search, but somewhat less than for experiment 1. With an autocorrelation of ~ 0.27 for points on the random walk a single step apart and a fairly steep descent for points further apart, the correlation between next and near neighbours on this fitness landscape is not high. That, coupled with a correlation length significantly less than for experiment 1, suggests that the fitness landscape for experiment 3 contains less information useful to guide the search than does the fitness landscape for experiment 1.

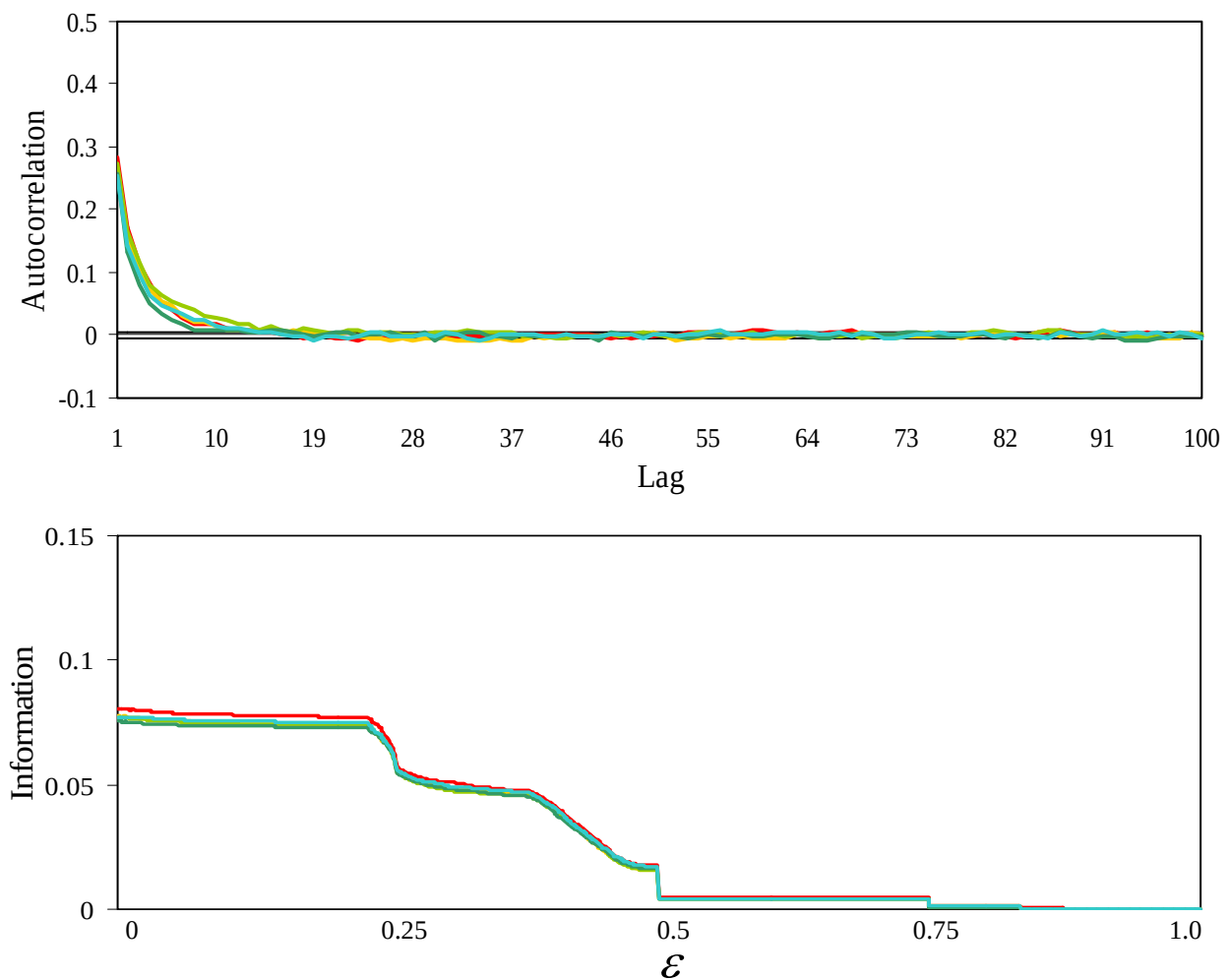


Figure 86 Composite Fitness; All Skills & Directions; Default Action = Hunt Action 2

The information content graph shown in Figure 86 supports the autocorrelation data for this experiment. Information stability is quite high at 0.881, indicating a high difference in fitness among neighbouring points, so pointing to the possibility of some good gradient information being present in the landscape. However, $H(0.0)$ is reasonably small, indicating that the diversity of shapes on the landscape is low. Similarly $M(0.0)$ is relatively small, indicating that the degree of modality of the landscape is low. These factors, when taken together, tend to indicate that the ruggedness and randomness of the fitness landscape is increasing, thus inhibiting the search process.

Random Walk Statistics (Average over 5 walks)								
Mean Fitness	0.99154							
Fitness Variance	0.00338							
Individual Fitness	No Movement	Ball Movement	Goals					
			1	2	3	4	5	>5
Number of Individuals	97712	1948	272	55	10	2	1	0
Autocorrelation $r(1)$	0.26680							
Correlation Length	15							
$\mathcal{E}$	Information Content $H(\mathcal{E})$		Partial Information Content $M(\mathcal{E})$					
0.0	0.07748		0.01705					
0.2	0.07473		0.01625					
0.4	0.04139		0.00760					
0.6	0.00414		0.00049					
0.8	0.00097		0.00009					
0.881	0.00000		0.00000					

Table 22 Composite Fitness; All Skills & Directions; Default Action = Hunt Action 2

Exp. 5: Composite Fitness; All Skills & Directions; Default Action = Hunt Action 3

For experiment 5 the default hunt action is further limited – in this case the default action is just to turn 90° in a randomly chosen direction. The results show that this limiting of the default hunt action continues to negatively affect the search.

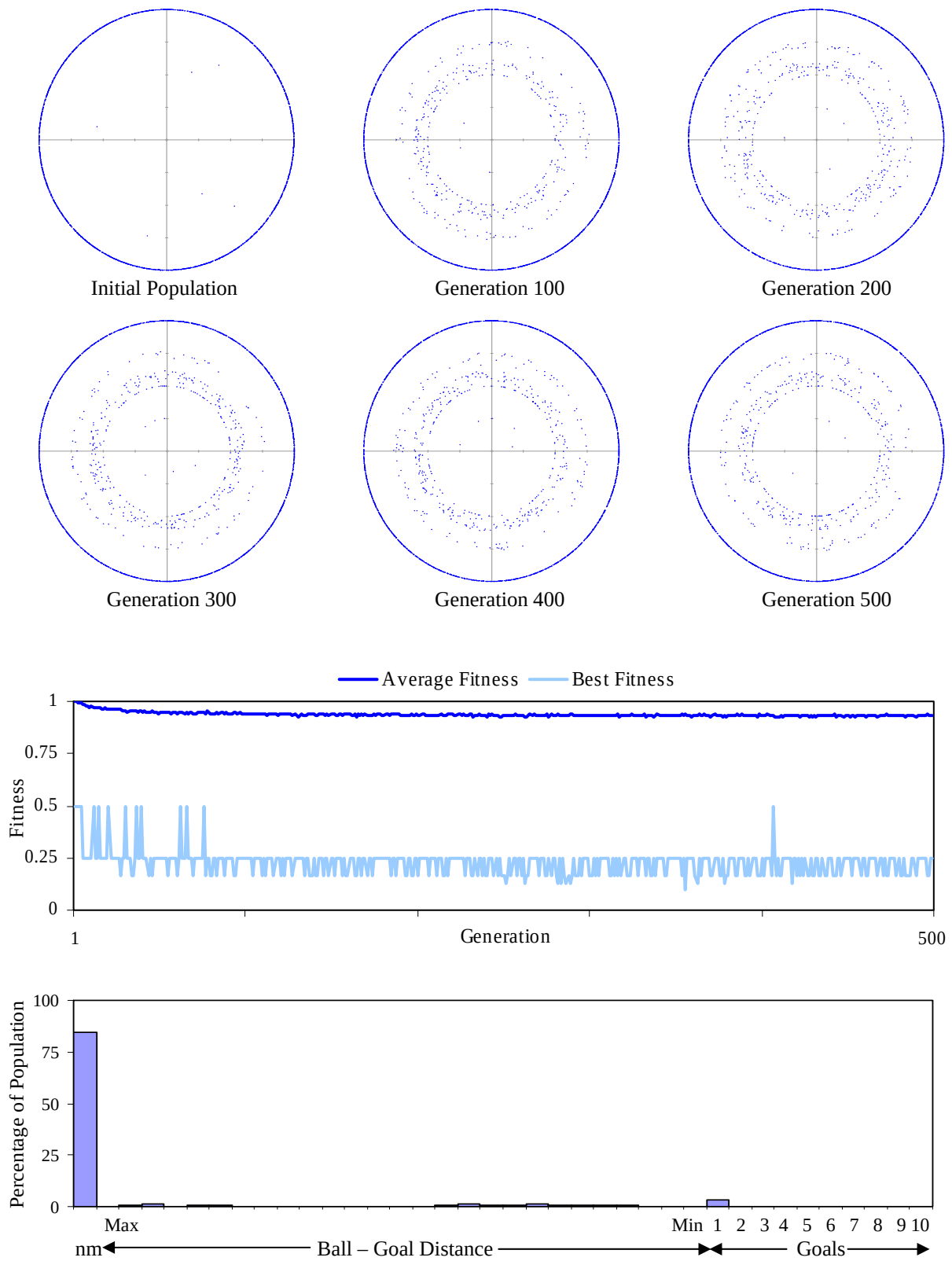


Figure 87 Composite Fitness; All Skills & Directions; Default Action = Hunt Action 3

The fitness scatter graphs (Figure 87) show that fewer individuals have achieved ball-kicking or goal-scoring behaviour, and as in previous experiments the evolution of the population tends to stop prior to generation 100. The percentage of the population exhibiting ball-kicking or goal-scoring behaviour is low, as shown by the frequency distribution (Figure 87). From the data presented in Table 23 it is evident that the random walk found very few individuals which exhibited goal-scoring behaviour – significantly fewer than in previous experiments.

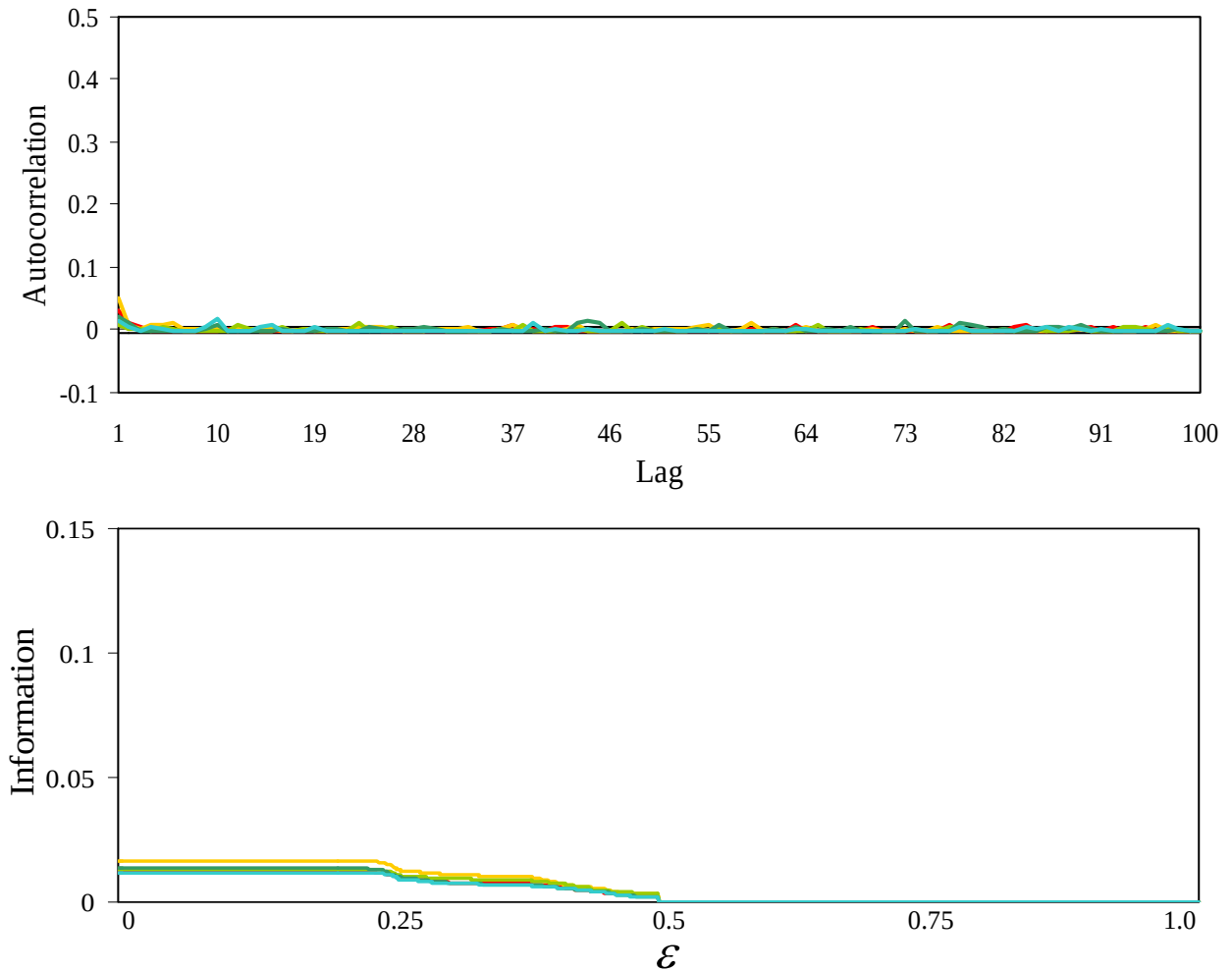


Figure 88 Composite Fitness; All Skills & Directions; Default Action = Hunt Action 3

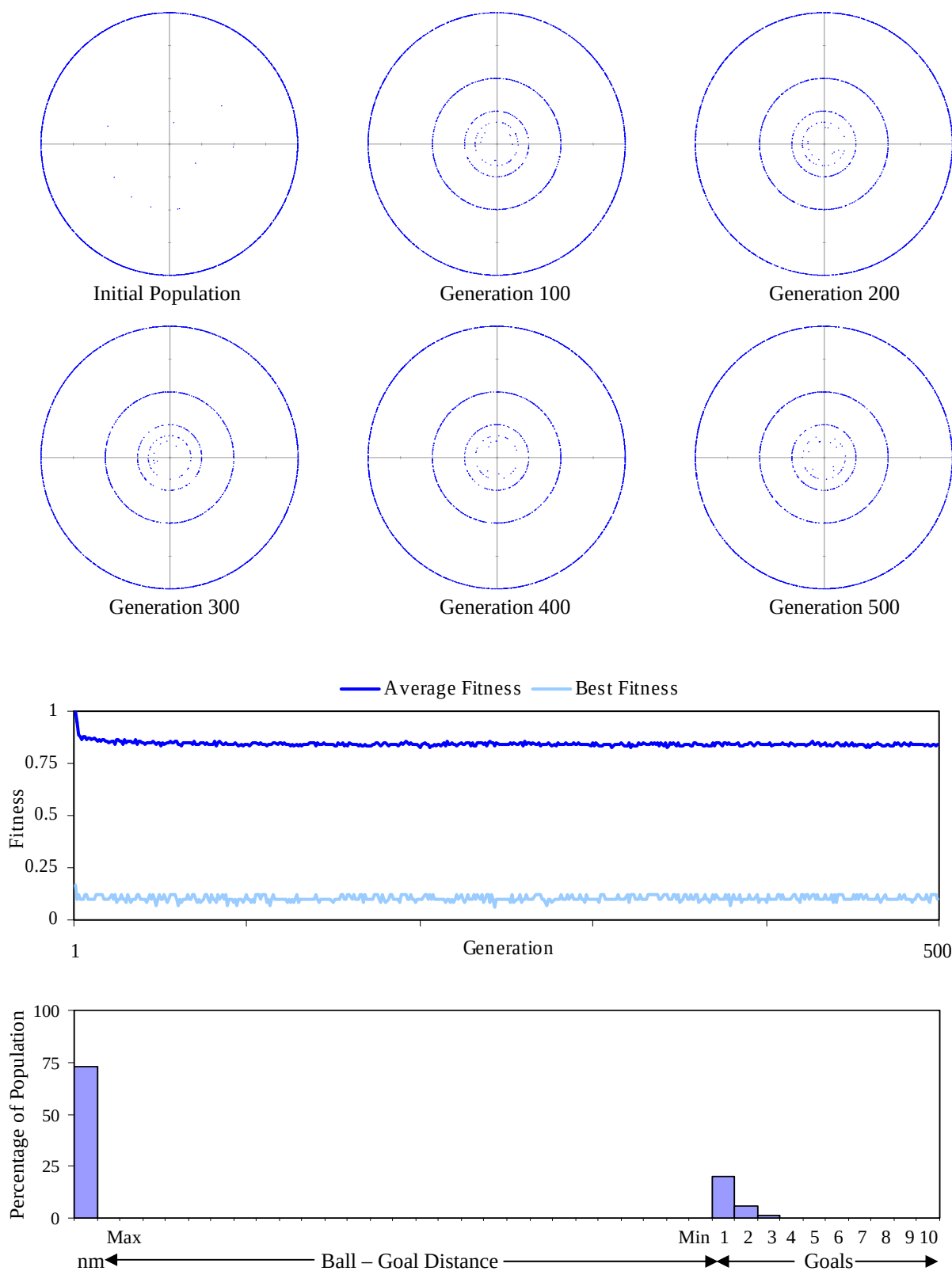
The autocorrelation and information content data shown in Figure 88 and in Table 23 paint a similar picture – from this data it is clear that the fitness landscape lacks much of the gradient information seen in the earlier experiments. This is entirely due to the removal of any sort of intelligence from the default hunt action.

Random Walk Statistics (Average over 5 walks)								
Mean Fitness	0.99927							
Fitness Variance	0.00029							
Individual Fitness	No Movement	Ball Movement	Goals					
			1	2	3	4	5	>5
Number of Individuals	99801	168	31	1	0	0	0	0
Autocorrelation $r(1)$	0.02546							
Correlation Length	3							
$\mathcal{E}$	Information Content $H(\mathcal{E})$		Partial Information Content $M(\mathcal{E})$					
0.0	0.01344		0.00193					
0.1	0.01344		0.00193					
0.2	0.01344		0.00193					
0.3	0.00899		0.00120					
0.4	0.00703		0.00090					
0.5	0.00000		0.00000					

Table 23 Composite Fitness; All Skills & Directions; Default Action = Hunt Action 3

7.6.4.3 Varying the Fitness Function

The effect of the fitness function is examined in this section. The composite fitness function experiments 1, 8, 9, and 10 from Table 19 (page 182) were compared with their goals-only counterparts (experiments 11, 13, 12 and 14 respectively), but since the results are similar for all experiments only experiments 1 and 11, and 9 and 12 are discussed in detail. The analysis of experiment 1 was presented in section 7.6.4.1 on page 183. The analyses of the remaining experiments follow.

Exp. 11: Goals-only Fitness; All Skills & Directions; Default Action = Hunt Action 1**Figure 89** Goals-only Fitness; All Skills & Directions; Default Action = Hunt Action 1

The data for experiment 11 are very similar to those for experiment 1, with the major difference being the autocorrelation and information content data (Figure 90). The fitness scatter graphs (Figure 89) are almost identical for goal-scoring behaviour, with the only real difference being that for experiment 1 individuals that kicked the ball closer to the goal were rewarded and show in the “ball movement” cloud of points. The population average fitness is, as expected, higher for experiment 11, but the best fitness and fitness frequency graphs are almost identical (for goal-scoring behaviour in the case of the fitness frequency graph). The autocorrelation and information content graphs (Figure 90) indicate that the fitness landscape for experiment 11 has somewhat less gradient information useful for search, but still sufficient to facilitate a successful search. This is another indication that when the players are given the full complement of hand-coded skills the difficulty of the problem is reduced significantly.

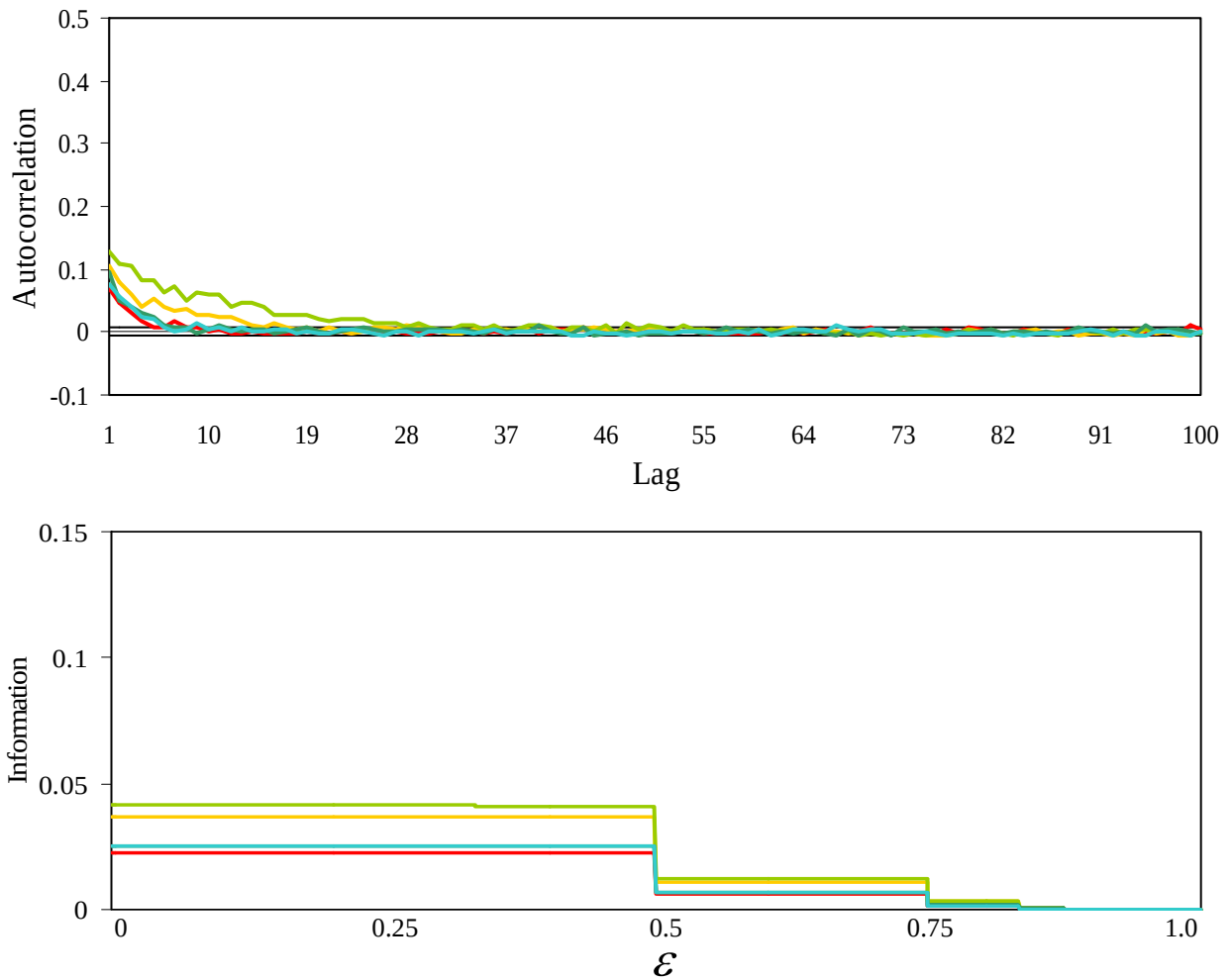


Figure 90 Goals-only Fitness; All Skills & Directions; Default Action = Hunt Action 1

Random Walk Statistics (Average over 5 walks)								
Mean Fitness	0.99672							
Fitness Variance	0.00189							
Individual Fitness	No Movement	Ball Movement	Goals					
			1	2	3	4	5	>5
Number of Individuals	99417	n/a	456	104	19	3	1	0
Autocorrelation $r(1)$	0.09497							
Correlation Length	20							
$\mathcal{E}$	Information Content $H(\mathcal{E})$		Partial Information Content $M(\mathcal{E})$					
0.0	0.03028		0.00520					
0.2	0.03028		0.00520					
0.4	0.03020		0.00519					
0.6	0.00868		0.00116					
0.8	0.00229		0.00025					
0.882	0.00000		0.00000					

Table 24 Goals-only Fitness; All Skills & Directions; Default Action = Hunt Action 1

Exp. 9: Composite Fitness; All Skills & Directions; No Default Action; No Hunt Consequent

The players evolved in experiment 9 have been endowed with all skills and added directions but have not been afforded the benefit of any default action which, when compared to the results of experiment 1, is clearly a significant loss and has a big impact on the result of the search. Even with the aid of a composite fitness function to guide the search the algorithm was not able to consistently find players that exhibited goal-scoring behaviour.

From the fitness scatter graphs for this experiment (Figure 91) it is apparent that by generation 100 the evolutionary search has converged and that most of the population has evolved no more than sufficient ball-kicking skills to kick the ball closer to the goal than where it started. The data shown on the graph of population average fitness (Figure 91) indicates that improvement of the population stops at about generation 40, and although the graph of best fitness values indicates that individuals exhibiting goal-scoring behaviour continue to be found throughout the remainder of the evolutionary process, terminating the search after generation 40

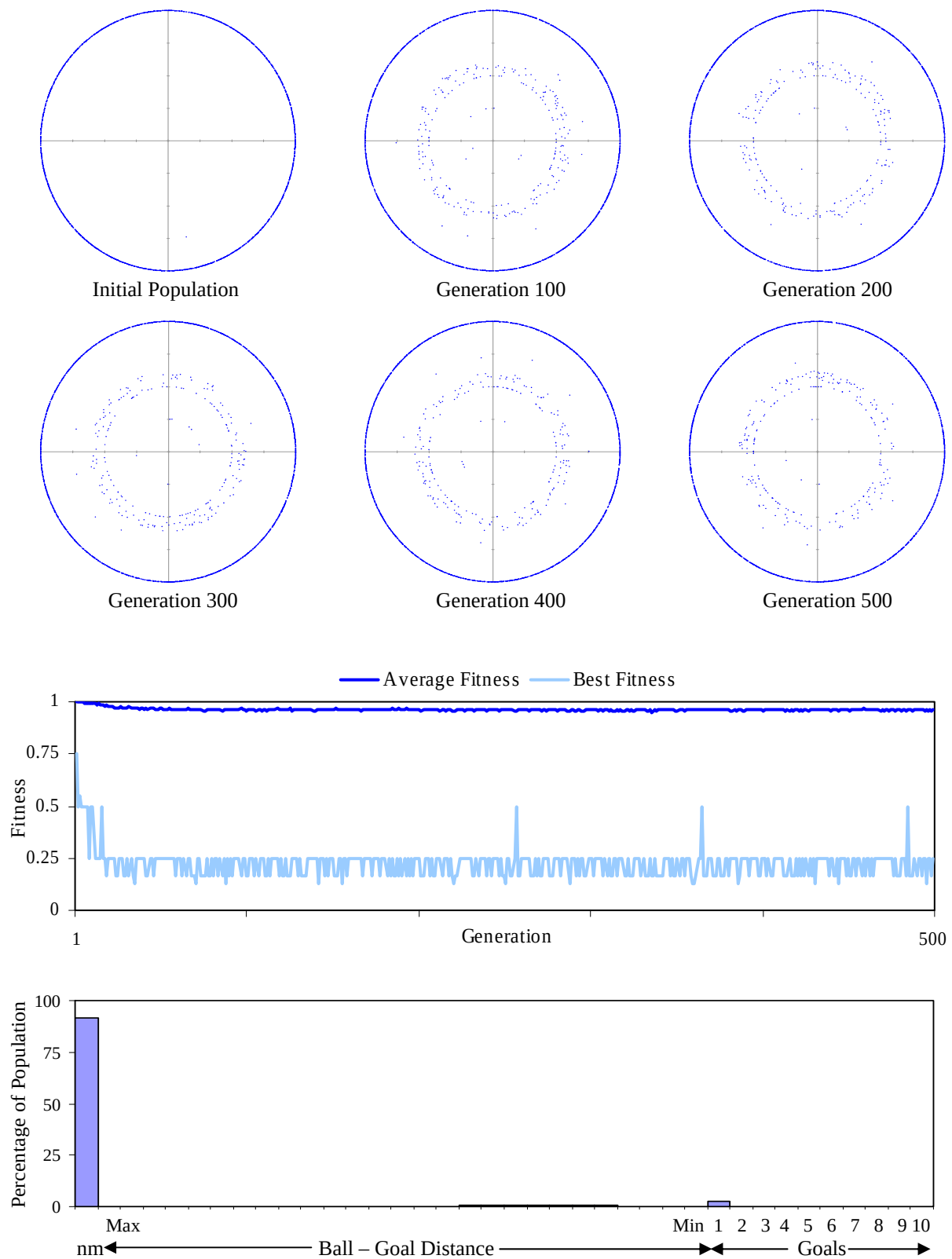


Figure 91 Composite Fitness; All Skills & Directions; No Default Action; No Hunt Consequent

would not have adversely affected the result. Figure 91 shows that the percentage of the population exhibiting goal-scoring behaviour is extremely small, with a very large percentage of the population not kicking the ball.

The comparison of the results for this experiment with the results for experiment 1 is a stark indication of the importance of the default hunt action to the success of the search. The autocorrelation data shown in Figure 92 and the correlation length given in Table 25 indicate that the fitness landscape for this problem offers a very limited amount of useful gradient information that the search algorithm can use to guide the search. With an autocorrelation of ~ 0.015 for points on the random walk a single step apart and a falling to zero for points just a few steps further apart, the correlation between next and near neighbours on this fitness landscape indicates that the structure of the fitness landscape is close to random and not very conducive to search.

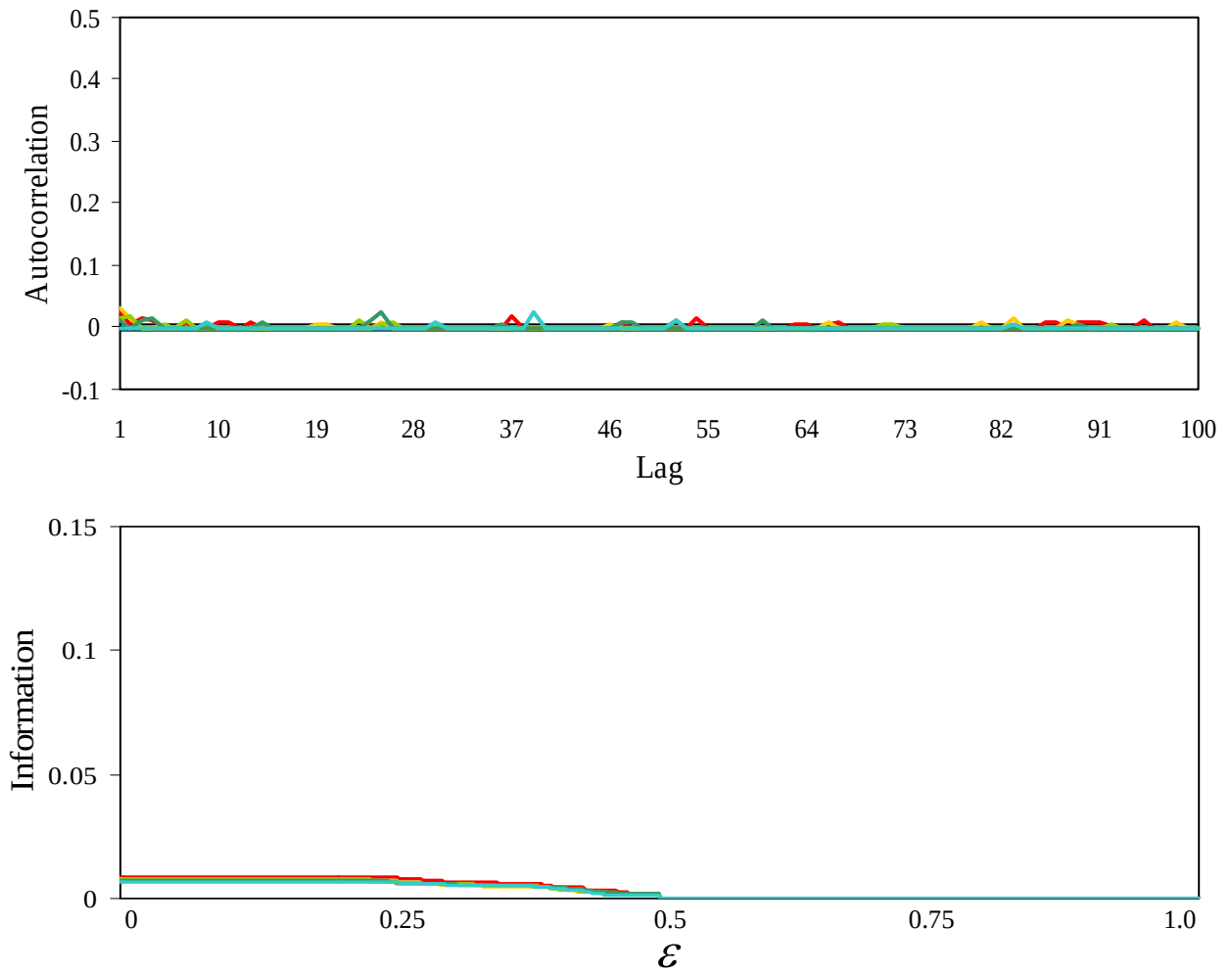


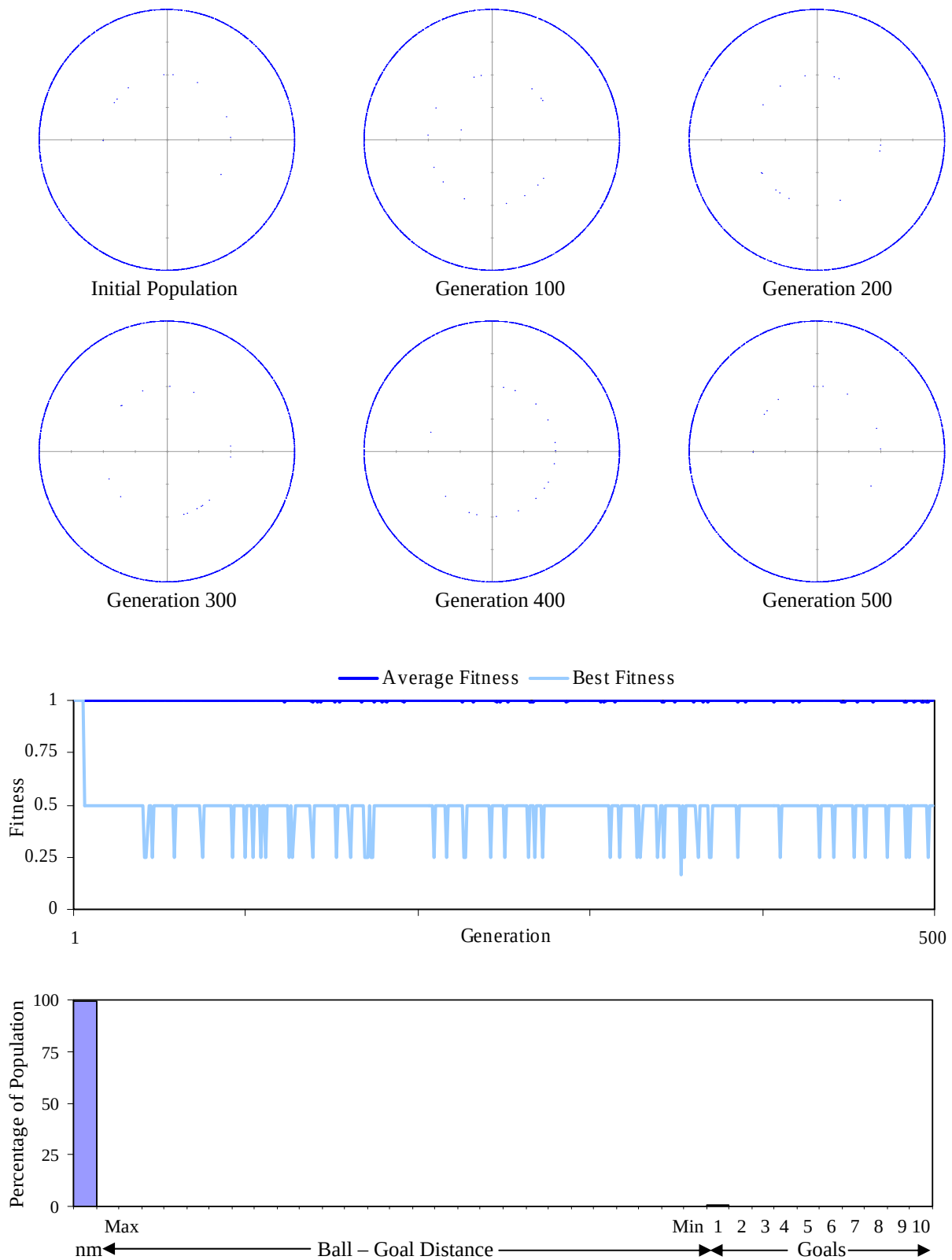
Figure 92 Composite Fitness; All Skills & Directions; No Default Action; No Hunt Consequent

The data presented in Table 25 shows the mean fitness of the 100,000 individuals evaluated during the random walk close to 1.0 (indicating no ball movement) and the variance very small. A small number of individuals exhibiting goal-scoring behaviour were found during the random walk which is some indication that the problem is difficult, but not impossible.

The information content graph in Figure 92 supports the autocorrelation data for this experiment. Information stability is medium-high at 0.6, but with the low information content values this is not very significant. With the autocorrelation data indicating a near random landscape, and the low information content values, there is very little useful gradient information in the landscape to guide the search.

Random Walk Statistics (Average over 5 walks)								
Mean Fitness	0.99961							
Fitness Variance	0.00016							
Individual Fitness	No Movement	Ball Movement	Goals					
			1	2	3	4	5	>5
Number of Individuals	99901	79	19	1	0	0	0	0
Autocorrelation $r(1)$	0.01629							
Correlation Length	3							
$\mathcal{E}$	Information Content $H(\mathcal{E})$		Partial Information Content $M(\mathcal{E})$					
0.0	0.00752		0.00097					
0.1	0.00752		0.00097					
0.2	0.00752		0.00097					
0.3	0.00591		0.00073					
0.4	0.00440		0.00052					
0.6	0.00000		0.00000					

Table 25 Composite Fitness; All Skills & Directions; No Default Action; No Hunt Consequent

Exp. 12: Goals-only Fitness; All Skills & Directions; No Default Action; No Hunt Consequent**Figure 93** Goals-only Fitness; All Skills & Directions; No Default Action; No Hunt Consequent

The players evolved in experiment 12 have also been endowed with all skills and added directions, and have not been afforded the benefit of any default action. As for experiment 9, the results of this experiment, when compared to the results of experiment 1, show that the default action is a significant loss and has a big impact on the result of the search. The evolution of the players in this experiment was not aided by a composite fitness function, and this is reflected in the autocorrelation and information content data presented in Figure 94 and Table 26 (page 206) which show clearly that the fitness landscape features offer no benefit to the search process. As for experiment 9, a small number of individuals exhibiting goal-scoring behaviour were found during the random walk, again indicating that the problem is difficult but not impossible.

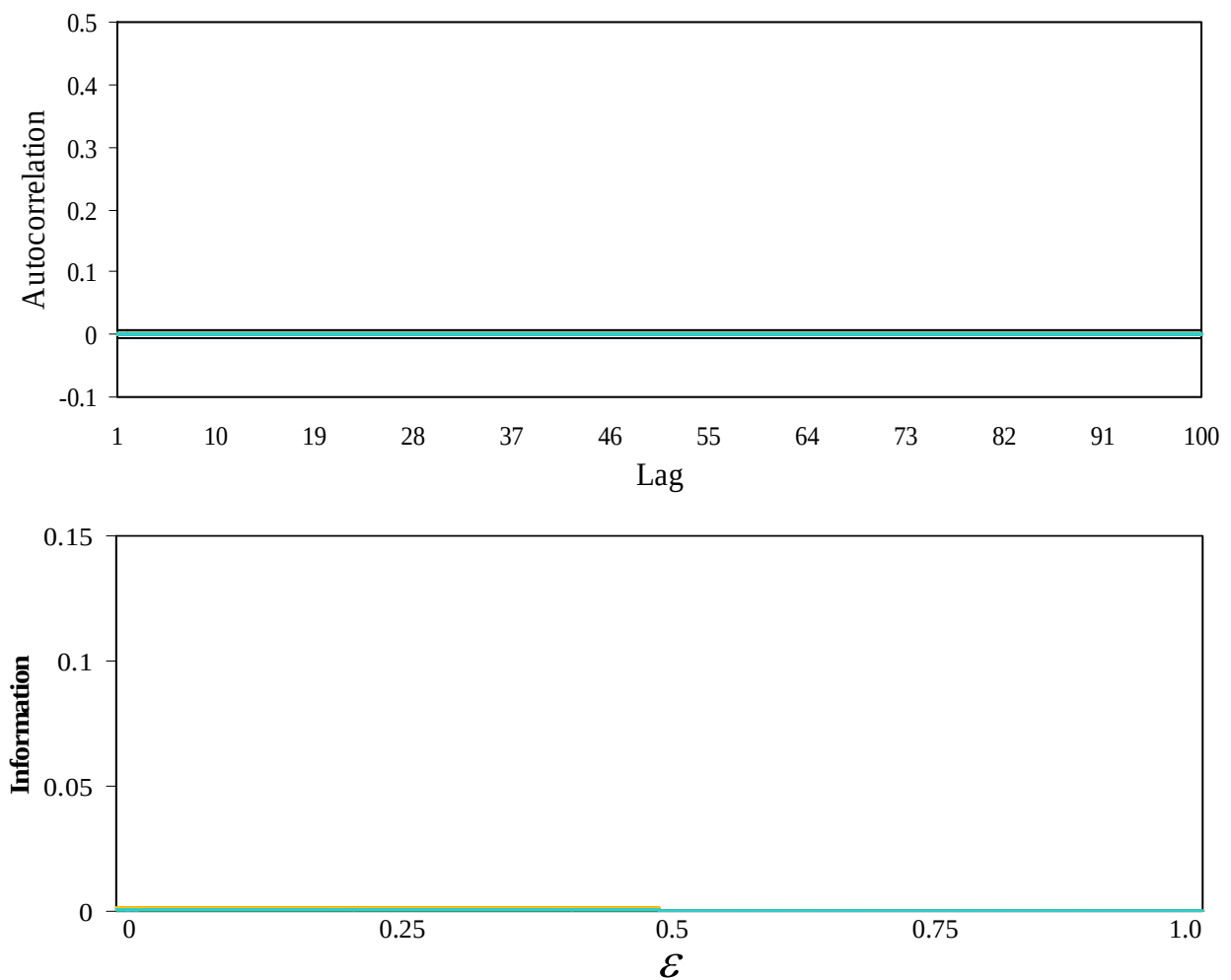


Figure 94 Goals-only Fitness; All Skills & Directions; No Default Action; No Hunt Consequent

From the fitness scatter graphs and the graph of population average fitness for this experiment (Figure 93, page 204) it is apparent that there is virtually no change in the overall quality of the population throughout the evolutionary process, with very few players exhibiting goal-scoring behaviour and the population average fitness showing no improvement. Although individuals exhibiting goal-scoring behaviour are found, and the graph of best fitness values indicates that they continue to be found throughout the evolutionary process, the frequency with which they are found is significantly less than for experiment 9. The goal-scoring ability of individuals found is also less than that of the individuals found in experiment 9. Figure 93 shows that the percentage of the population exhibiting goal-scoring behaviour is even smaller than that of experiment 9. These results indicate that removing the benefit of a composite fitness function to guide the search has a significant impact on the result of the search.

Random Walk Statistics (Average over 5 walks)								
Mean Fitness	0.99995							
Fitness Variance	0.00002							
Individual Fitness	No Movement	Ball Movement	Goals					
			1	2	3	4	5	>5
Number of Individuals	99990	n/a	9	1	0	0	0	0
Autocorrelation $r(1)$	0.00009							
Correlation Length	0							
$\mathcal{E}$	Information Content $H(\mathcal{E})$		Partial Information Content $M(\mathcal{E})$					
0.0	0.00082		0.00008					
0.1	0.00082		0.00008					
0.2	0.00082		0.00008					
0.3	0.00082		0.00008					
0.4	0.00082		0.00008					
0.551	0.00000		0.00000					

Table 26 Goals-only Fitness; All Skills & Directions; No Default Action; No Hunt Consequent

7.7 Summary and Discussion

This chapter of the thesis has presented analyses of the search spaces and some features and statistics of the fitness landscapes associated with the trials, conducted in the SimpleSoccer environment, from Chapter 6. The analyses have provided insight to the problem and the basis for a better understanding of why the robot soccer problem is difficult for evolutionary algorithms. The analyses show how changing the player skills, the default action of the player, and the fitness evaluation changes the fitness landscape, hence the difficulty of the problem.

The specific research question examined in this chapter of the thesis was:

Why is evolving goal-scoring behaviour from scratch in a simulated robot soccer environment a difficult problem for evolutionary algorithms?

Varying player skills changes the search space of the problem – a different set of players is being searched for each skillset. It is therefore not surprising that changing the innate skills of the players has a significant effect on the outcome of the search – searching the set of players which have essentially been pre-coded to exhibit good soccer-playing skills is almost guaranteed to find more players exhibiting better goal-scoring behaviour than searching the set of players which have no pre-coded soccer-playing skills. This is akin to searching for good goal-scoring behaviour in an assembled group of premier league soccer players, and then searching an assembled group of randomly selected people off the street – every now and then a person from the street will score a goal, but the premier league players will do it more often and more consistently.

The default player action was shown to be a very significant determinant of performance for the robot soccer problem. This was also seen in the results presented in [175]. For the work conducted for this thesis the default action was almost always invoked when the ball could not be seen. The importance of the default action tends to suggest that while some basic skills such as kicking, dribbling, perhaps even passing, can be somewhat readily learned, learning subtle tactics and strategies is difficult.

Similarly, it was shown through experiments that compared the effectiveness of a simple goals-only fitness function against that of a composite fitness function that a composite fitness function can better guide the search. A well-designed composite fitness function can

effectively act as a recipe for solving a problem. A great deal of human expertise and expert knowledge can be injected into a composite fitness function, and in effect the problem can virtually be solved by the fitness function before the algorithm begins the search.

Figure 95 shows the frequency of a range of fitness values against the experiments listed in Table 19 on page 182. In Figure 95 the frequency is plotted as the percentage of the total population which was assigned the fitness value.

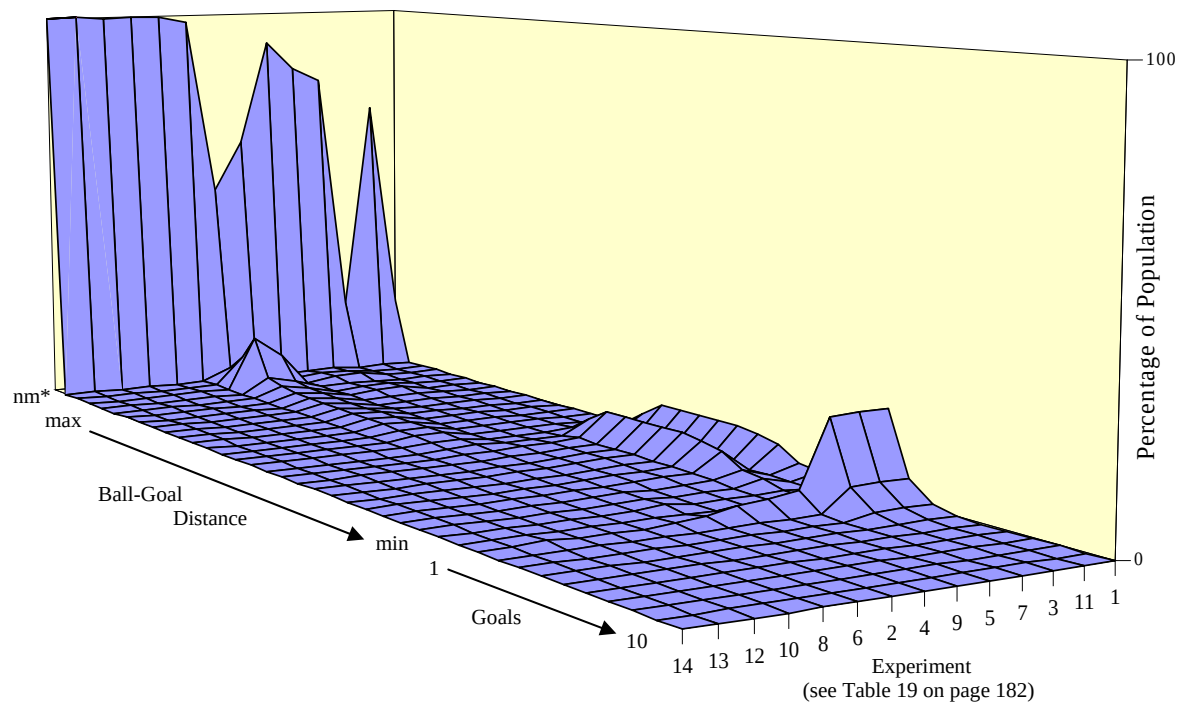


Figure 95 Fitness Frequency

Figure 95 shows that as the amount of expert knowledge injected increases (by way of innate player skills, default player action and composite fitness function), the frequency of better performing players found by the search algorithm increases. The search space and fitness landscape analyses suggest two possible reasons for this, both related to the same underlying cause. The first is that the number of individuals in the solution space is bigger – that is, the percentage of the search space which represents the solution space increases. This is demonstrated by the search space and fitness landscape analyses of the experiments for which

the players' skills were varied. The second reason is that the solutions are easier to find – that is, the number of actual solutions in the search space may not necessarily increase, but the fitness landscape traversed by the search algorithm is changed so that solutions are found more easily by the search algorithm. This is demonstrated by the search space and fitness landscape analyses of the experiments for which the fitness function was varied.

The analyses show that the search space changes as the skill level changes, and it changes by becoming more saturated with solutions as the skill level increases – or more precisely as the amount of expert knowledge injected into the algorithm increases. This suggests that searching in some search spaces is going to be more successful simply because some search spaces contain more solutions. Changing the skill level may not actually change the fitness landscape *per se*, but it does change the search space (because the set of individuals that comprises the search space is different for each set of innate skills), and different search spaces will almost certainly generate different fitness landscapes.

The analyses also further corroborate the conjecture that if at least one of the “advantageous ingredients” is present - either a high level of skills or a composite fitness function - the evolutionary algorithm is much more likely to find a solution to the problem. This conjecture is supported in part by the result reported by Luke in [175] and discussed in section 2.7.3 on page 65. In the work reported by Luke, the fitness function was initially a composite function based on a number of factors, such as the number of goals, time in possession of the ball, average position of the ball, number of successful passes, etc., but it was found in early tests that the entire population converged to very poor solutions. The fitness function was changed to simple goal difference alone and the population avoided such convergence. It was discovered that initial game scores were very high and quite variable, and that this suited the simple fitness function more than the composite function. Luke concludes that this was in part due to vectors to the ball and to the goal being fundamental parts of the function set, allowing teams to perform well offensively – so a rich skill set obviates the need for a composite fitness function.

These results suggest that in order to evolve competent robot soccer players either a rich set of mid- to high-level soccer playing skills needs to be provided, or a composite fitness function used that effectively guides the evolutionary process to developing the skills required to be a competent soccer player.

Evolution does not work like an engineer. It works like a tinkerer.

François Jacob, Science, 1977.

Chapter 8

Conclusions and Future Work

8.1 General Conclusions

This thesis has presented a method for evolving goal-scoring behaviour in the very difficult problem domain of simulated robot soccer. The method was shown to produce competent goal-scoring behaviour in very reasonable timeframes when applied in a manner consistent with bringing the problem within the reach of the capabilities of the evolutionary algorithm.

An approach for accelerating the evolutionary process in the robot soccer environment was also presented. This approach makes use of prior knowledge, in the form of behaviours evolved in a simpler environment, as a starting point for the evolutionary process. The success of the approach suggests wider uses – for example, the approach could be used in an iterative and layered manner to solve very difficult problems incrementally.

The thesis has also presented a method for systematic analysis of the problem search spaces and fitness landscapes generated by evolutionary algorithms, and a study of the performance of a genetic algorithm when applied to the problem of evolving goal-scoring behaviour in a simulated robot soccer environment. The performance of the genetic algorithm was measured for problems of varying degrees of difficulty in the chosen environment as a means of addressing several research questions. Specific outcomes and conclusions with respect to the research questions are given in the following sections, followed by a conjecture regarding the inherent ability of evolutionary algorithms to solve difficult, monolithic problems, and finally some further work suggested by this thesis and a brief, final summary.

8.2 Research Question 1

The first research question addressed by this thesis examined the problem of evolving goal-scoring behaviour for robot soccer and was addressed in two parts:

- a) How can a genetic algorithm be used to evolve a set of fuzzy rules to govern the behaviour of a simulated robot soccer player that produces consistent goal-scoring behaviour?*
- b) Will seeding the initial population for the genetic algorithm with rules evolved in the SimpleSoccer environment improve the quality and speed of the evolutionary process in the more complex RoboCupSoccer environment?*

To address the first part of the question a fuzzy rule-based representation was developed for players in the simulated robot soccer environments, and a messy-coded genetic algorithm used to evolve the rules that govern the behaviour of the players. Depending on the level of skill of the initial population and the fitness function used, the method consistently evolved, in very few generations, players that displayed very good goal-scoring behaviour. This demonstrates that the method can be used to successfully train goal-scoring behaviour for a reactive simulated soccer player when either the quality of the initial population or the fitness function implemented brings the problem within reach of the evolutionary algorithm. The good performance of the method with a small population and relatively few generations is likely to be due to

- the initial skillset of the population in the form of the hand-coded skills with which the players were endowed, and
- the reduced complexity of the rule encoding afforded by the flexibility of the messy-coded genetic algorithm.

The representation and inferencing techniques used bring more members of the search space into the “reasonably good” solution space than other representations and techniques, and were chosen because the problem does not require precise, exact solutions.

Several tests were performed to determine the effect of varying the genetic algorithm parameters on the performance of the method and analyses of the results presented. The method was found to be only slightly sensitive to some parameters (such as the mutation rate

and selection method), and not sensitive at all to others (the population size and crossover method). Where applicable recommendations were made for the most effective parameter values (see Table 6 on page 127).

The simulated robot soccer environment, particularly the RoboCupSoccer environment, is a complex, dynamic and uncertain environment, and the results presented indicate that the method can define reactive behaviours for complex situations where the environment changes quickly and there is a lot of uncertainty – the method is considered sufficiently general to be applied to other complex, dynamic and uncertain environments.

Since this method produces human-readable rules which govern the behaviour of the players, it is possible to gain some understanding of the knowledge, often novel, that the player has learned through the evolutionary process. This is considered an advantage over many existing methods of learning behaviours where often the learned behaviour is not easily understood or extracted.

The second part of the question was addressed by developing the SimpleSoccer robot soccer environment and using that to model the RoboCupSoccer environment. Unlike other simple game simulators, such as Ascii Soccer, the SimpleSoccer environment retains the dynamics and many of the characteristics of RoboCupSoccer. A method of accelerating the evolutionary search in the RoboCupSoccer environment by seeding the initial population with individuals that exhibit behaviours previously learned in the SimpleSoccer environment was examined. It was demonstrated that seeding the initial population produces better goal-scoring behaviour more quickly than if population seeding is not used. Indeed, it was determined that the behaviours learned in the simple robot soccer environment of SimpleSoccer were directly transferable to the more complex RoboCupSoccer environment, and that they produced good goal-scoring behaviour after very little, or no evolutionary refinement. Evolutionary search in the SimpleSoccer environment was shown to be seven times faster than in the RoboCupSoccer environment, reducing a typical 25 generation run with a population of 100 from 49 hours in the RoboCupSoccer environment to 7 hours in the SimpleSoccer environment, so the advantage of the SimpleSoccer environment is significant. This is an important result because it allows, in the very difficult and time-consuming domain of simulated robot soccer, much more evolutionary learning than would otherwise be possible. The ease with which the behaviours were transferred from SimpleSoccer to RoboCupSoccer is in some part due to the fact that the player behaviours were implemented using fuzzy rules and fuzzy inferencing.

8.3 Research Question 2

The effect of injecting human expertise and expert knowledge into the evolutionary algorithm was examined by the second research question, also addressed in two parts:

How does varying

a) the skills available to the players, and/or

b) the implementation of the fitness function

affect the result of the evolutionary search?

To determine how varying the skill level of the initial population affects the result of the evolutionary search, several experiments were conducted with initial populations endowed with various combinations of mid-level hand-coded skills. The results obtained and the analysis presented show that more skilled starting populations yield better final populations, indicating that the quality of the initial population is an important determinant in the quality of the final population. Further experiments were conducted in which the player's default action was varied. The results of these experiments showed that smarter default actions result in better goal-scoring behaviour and in higher quality final populations.

Several experiments were also conducted to compare the effectiveness of a composite fitness function with that of a simple fitness function. The results of these experiments suggest that for a difficult problem that an evolutionary algorithm would otherwise not be able to solve, a composite fitness function which rewards incremental improvements in behaviour can effectively decompose the problem into a connected series of problems small enough for the evolutionary algorithm to solve.

The combination of the results of the trials conducted with varying qualities of initial populations and player default actions, and the results investigating the impact of different fitness functions suggests that for some problems evolutionary algorithms, without the aid of an improved quality initial population, a smarter default action, or a composite fitness function to move the starting point closer to the finish point, perform not much better than random search. The corollary is that for some very difficult problems which are solved by evolutionary algorithms, much of the intelligence in the method is actually in the expertise of the researcher designing the algorithm - the way the problem is first decomposed, what innate skills are given to the algorithm, and very often governed by the selection of the fitness function.

Often researchers will adjust evolutionary algorithm parameters such as the population size, mutation rate, number of generations etc. in order to elicit a “better” solution, but such manipulation is really just playing at the margins – very often the big differentiator in the performance of an evolutionary algorithm is the human expertise and expert knowledge injected into the algorithm.

8.3.1 Research Question 3

The underlying cause of the difficulty of evolving goal-scoring behaviour for robot soccer was addressed by research question 3:

Why is evolving goal-scoring behaviour from scratch in a simulated robot soccer environment a difficult problem for evolutionary algorithms?

The results of the experiments performed to address research question two were analysed in some detail, with search space and fitness landscape statistics generated and analysed.

The analyses indicate that when human expertise and expert knowledge is injected into the algorithm via hand-coded innate skills, smart default actions and a composite fitness function to guide the search, the problem is solvable - that is, the genetic algorithm is able to evolve individuals which display goal-scoring behaviour to the extent that they are able to consistently score multiple goals in the tests conducted. The fitness landscape analyses further indicate that as human expertise and expert knowledge is removed from the algorithm by restricting the hand-coded innate skills and smart default actions available to the players, or by using a simple goals-only fitness function, at some threshold the problem becomes intractable for the evolutionary algorithm. The injection of human expertise and expert knowledge acts like a magnifying glass to the searcher - as more expertise and knowledge is injected the fitness landscape features conducive to search are magnified, and as the expertise and knowledge is removed those landscape features become less discernable.

This suggests that while for some problems there may be gradient information in the fitness landscape, and the problem may not actually be a classic needle-in-a-haystack problem, as the density of solutions in the search space becomes very low the “mountain ranges” in the landscape begin to become isolated from each other and the landscape begins to appear as flat plain, sparsely populated by individual peaks – so the problem begins to resemble a needle-in-

a-haystack problem. For such problems evolutionary algorithms are not able to locate the sparsely distributed gradient information any way other than by randomly sampling the search space, so perform little or no better than random search.

This is one of the underlying causes of the difficulty of the robot soccer problem for evolutionary algorithms. This also leads to the conjecture, presented in more detail in the following section, that the capacity of evolutionary algorithms to solve some very difficult problems is limited.

The analyses of the fitness landscapes give some general directions as to the composition of the landscapes most conducive to evolutionary search for the robot soccer problem – without further work to quantify these values (discussed in section 8.5 below) only generalities can be taken from the data and analyses. A mid-range autocorrelation and information content for near neighbours is a good starting point, with a mid to high value for information stability to ensure that while there is some correlation between neighbours, there is still sufficient difference to ensure some useful gradient information to guide the search. A double-digit correlation length to ensure that the gradient data is sufficiently widespread is better than a short correlation length

8.4 An Evolutionary Quantum?

That evolutionary algorithms cannot effectively solve needle-in-a-haystack problems is a hint that a less well recognised limitation of this class of optimisation and search techniques exists: the amount of learning possible, or the size of the evolutionary step that can be taken, by an evolutionary algorithm is restricted by the nature of the algorithm and the available feedback. We only need to look to natural evolution to see that this is so. The natural evolutionary process is very slow because the size of the evolutionary step is small and limited – if there was no limit then almost by definition the size of the evolutionary step would have evolved to be larger and so speed the process.

This thesis proffers the conjecture that evolutionary algorithms, as simplified models of natural evolution, are not suited to solving very difficult, monolithic problems, and this is because the size of the evolutionary step - the “evolutionary quantum” - required to find solutions for those problems exceeds the capacity of the algorithm to elicit useful feedback from the search space: if we try to push the evolutionary quantum past a problem-dependent threshold the problem becomes a needle-in-a-haystack problem. It is further suggested that it is

sometimes more illusion than fact that evolutionary algorithms are able to solve some very difficult problems, and that the illusion is sustained by those problems being made less difficult by artificially guiding the search. This is most often achieved in one of two ways: by placing the starting point closer to the expected end point, or by breaking the problem into a number of smaller problems and solving the smaller problems separately. The starting point is made closer to the end point by changing the quality, or innate level of knowledge or skill, of the initial population – if the members of the initial population are hand-coded to be better solutions (than random) before evolution begins, then the evolutionary process has a much better chance of success. Alternatively, and sometimes additionally, the task is made easier for the evolutionary algorithm by decomposing a very difficult problem into smaller, more easily solved problems and having the evolutionary algorithm solve the smaller problems separately. This is done, intentionally or not, by introducing a composite fitness function which indirectly achieves the same, or similar, decomposition by rewarding the different, incremental behaviours required to solve the smaller problems. This contention is supported by the analyses of previous work in section 2.7 on page 62, particularly that of Luke [175] and ChellaPilla and Fogel’s work with Blondie24 [40-44, 76, 77].

This is not to suggest that evolutionary algorithms offer no benefits – they are clearly capable of solving many optimisation and search problems more effectively and efficiently than some other techniques. Evolutionary algorithms are also very good at finding novel solutions to less difficult problems that would probably not be found by other methods. What is suggested by this thesis is that in some circumstances there is an inherent limit to the usefulness of evolutionary algorithms as techniques for solving very difficult monolithic problems – problems for which the fitness function or reward does not directly map to a solution. The work presented demonstrates this possibility for a particular type of evolutionary algorithm - a genetic algorithm - in the difficult problem domain of simulated robot soccer.

8.5 Future Work

The work presented in thesis developed and demonstrated a method capable of producing competent goal-scoring behaviours in the simulated robot soccer environment. The method uses a messy-coded genetic algorithm to evolve fuzzy rules for a fuzzy inferencing system. A method for systematic analysis of the search spaces and fitness landscapes generated by the search algorithm was also presented, as were the results of analyses of those search spaces and

fitness landscapes as a study of the difficulty of the problem. This suggests two broad areas in which to pursue future work related to this thesis: robot soccer, and landscape analysis and problem difficulty. These are discussed separately in the following sections, but first the point should be made that neither the method for evolving soccer players, nor the landscape analysis method should be considered specific to robot soccer, and further work to evaluate both in other areas requiring reactive behaviour in complex, dynamic environments would be useful.

8.5.1 Robot Soccer

Future work in the area of robot soccer can be further divided into two streams: evolving competent players, and modelling the environment.

8.5.1.1 Evolving Competent Players

There are many avenues for further work in the area of evolving more competent robot soccer players. Some of these are:

- The work was conducted with a single player. An obvious avenue for further research is to extend the method to a multi-player environment. Players in a multi-player environment need to be more flexible and resilient in the way they react to changes in their environment, especially when opposition players are present. Developing teamwork and co-operative behaviour in a multi-player environment is a challenging area for further research. Action planning also becomes more complicated and important when multiple players are involved – both teammates and opponents, so presents an opportunity for further research.
- Goal-scoring is just one of the skills necessary for competent soccer playing behaviour. Further work could be undertaken to extend the method to evolve other skills, both offensive and defensive. Given the nature of the player architecture, players with a mix of general offensive and defensive skills could be evolved, as well as players with specialised skills in one of those areas (for example, the goal keeper). This could also involve evolving players in a multi-player environment so that, for example, specialised defensive players would develop their skills in an environment where opposition players are making attacking moves.

- Since the method demonstrated produces human-readable rules, an interesting line of enquiry would be the possible post-processing and optimisation of the rules evolved. This would be a useful means of combining the often novel solutions found by evolutionary algorithms with the experience and knowledge of human experts.

The means by which the players are being evolved could also be improved, and so the following avenues of further work are suggested:

- Though some experimentation with the values of the genetic algorithm parameters was performed and some broad recommendations for optimal values were made, some further experimentation in this area would be useful, particularly, as suggested earlier in the thesis, with the length of the chromosome and the number of rules, and the size of the rules, on the chromosome.
- In this work the number and shape of the fuzzy sets for the fuzzy inferencing system was fixed, and only the fuzzy rules were evolved. This suggests two avenues for further work: the first is manual experimentation with different fuzzy sets and membership functions; the second is evolving the fuzzy set parameters (i.e. the number of sets and their membership functions) along with, or in parallel to, the fuzzy rules.
- Often an “arms race” in an adversarial environment is a good way to develop new skills more quickly, suggesting that perhaps coevolution [10, 114] might be a useful strategy to evolve players. Because of the competition introduced, a coevolutionary approach could be a better path to evolving players that are not only competent with regard to the basic soccer skills, but that also learn good soccer strategies.
- Experimenting with different fitness functions with either of two objectives in mind:
 - to evolve more competent players
 - to evolve players more quickly

In the current implementation fitness evaluation relies on the players actually playing some amount of soccer - how much play that needs to take place in order to get a good estimate of fitness is not really known. The investigation of a dynamic fitness evaluation function which evolves over time would be a useful, as would considering coevolution to evolve the fitness function, as suggested by de Jong and Pollack [64, 65].

- The Mamdani fuzzy inferencing model allows a more intuitive output than the Sugeno (or TSK) model, but the Sugeno model may be more suited to the real-time control environment of robot soccer. Further investigation of different fuzzy inferencing models, not limited to Mamdani and Sugeno models, is warranted.

8.5.1.2 Modelling the Environment

The RoboCupSoccer environment is, as discussed throughout this thesis, a very complex and time-intensive environment in which to train players, and this was the motivation for developing the SimpleSoccer environment as a model for RoboCupSoccer. Modelling the RoboCupSoccer environment with a simpler environment that allows players to be developed that will perform well when transferred to RoboCupSoccer is a promising avenue for further work. How much simpler can the model be before it is no longer viable? Is there a correlation between simplicity of the model and the amount of further evolution required in the RoboCupSoccer environment – how complex does the model need to be in order that players can be transferred directly without further evolution?

This avenue of inquiry has benefits outside the field of robot soccer – modelling any environment in an effort to reduce evaluation time can only help evolutionary algorithms perform better in complex environments.

8.5.2 Landscape Analysis and Problem Difficulty

This work presented a method of systematically investigating search spaces and fitness landscapes by applying and comparing several existing landscape measures. There is no general agreement within the evolutionary computation community as to the definition of the fitness landscape traversed by an evolutionary algorithm. A definition of the fitness landscape described by the combination of the genetic algorithm and the problem being investigated was proposed, but further work needs to be done to develop a consistent definition of the fitness landscape, or landscapes, described by the operation of an evolutionary algorithm. With a consistent landscape definition, more work can be done to develop measures that will aid researchers in tuning algorithms and search methods based on landscape analysis. Can the evolutionary quantum be tracked down and quantified in the fitness landscape analysis?

The fitness landscape definition used for this work is based on the assumption that the combination of the genetic operators implemented defines the neighbourhood relation of the

landscape, and that the fitness function defines the height of the landscape at each point. Given this definition the shape, ruggedness, and indeed the searchability of the landscape, will be affected by changes to any of the genetic operators and the fitness function. This suggests that it would be advantageous to experiment with various combinations of those parameters in order to determine if a search might be successful or what combination would improve the chance of a successful search, prior to launching into an exhaustive search. A useful avenue of further work is to develop a framework for this prior analysis so that a description of the means by which the analysis should be conducted can be determined (for example, the number and size of random walks conducted over the landscape in order to calculate the landscape measures), as well as quantifying the desired values for the landscape measures.

A proposition proffered by this work is that with a difficult problem such as robot soccer, an evolutionary algorithm will only find a reasonable solution if one of:

- a rich skill set (placing the initial population closer to the desired solution)
- a composite fitness function (providing a solution recipe)

is present - if both of those components are absent the problem becomes very difficult for evolutionary algorithms. Further work to ascertain the best balance between the two components would be useful, as would work to determine if there is a limit in the level of initial skills at which the problem becomes intractable. This is related to the conjecture proffered by this work that an evolutionary quantum exists and that some problems become intractable for evolutionary algorithms at some (probably problem-dependent) threshold. An important and promising direction for future work resulting from this thesis is in the verification and quantification of the evolutionary quantum and where the threshold lies for various problem types. Quantifying these values would be an invaluable guide to researchers and practitioners in the field of evolutionary algorithms.

Bibliography

1. Abbott, R., B. Parviz, and C. Sun. *Genetic Programming Reconsidered* in *Proceedings of the 2004 International Conference on Machine Learning: Models, Technologies and Applications*. 2004. Las Vegas NV.
2. Affenzeller, M. and S. Wagner, *Reconsidering the Selection Concept of Genetic Algorithms from a Population Genetics Inspired Point of View*, in *Cybernetics and Systems*. 2004, Austrian Society for Cybernetic Studies. p. 701-706.
3. Affenzeller, M. and S. Wagner, *Offspring Selection: A New Self-Adaptive Selection Scheme for Genetic Algorithms*, in *Adaptive and Natural Computing Algorithms, Springer Computer Science*. 2005, Springer. p. 218-221.
4. Ahuja, R.K., J.B. Orlin, and A. Tiwari, *A Greedy Genetic Algorithm for the Quadratic Assignment Problem*, in *Technical Report*. 1995, Sloan School of Management, Massachusetts Institute of Technology: Cambridge, MA.
5. Alba, E., C. Cotta, and J.M. Troya., *Evolutionary Design of Fuzzy Logic Controllers Using Strongly-typed GP*. *Mathware & Soft Computing*,, 1999. 6(1): p. 109-124.
6. Altenberg, L., *The Schema Theorem and Price's Theorem*, in *Foundations of Genetic Algorithms 3*, D. Whitley and M. Vose, Editors. 1995, Darrell Whitley and Michael Vose: San Francisco CA. p. 23-49.
7. Andre, D. and A. Teller. *Evolving Team Darwin United* in *M. Asada and H. Kitano eds, RoboCup-98: Robot Soccer World Cup II. Lecture Notes in Computer Science*. 1999: Springer-Verlag.
8. Angeline, P.J. and J.B. Pollack. *Competitive Environments Evolve Better Solutions for Complex Tasks* in *Proceedings of the Fifth International Conference on Genetic Algorithms*, p. 264-270. 1993: Morgan Kaufmann.
9. Aronsson, J., *Genetic Programming of Multi-agent System in the RoboCup Domain*, in *Masters Thesis, Department of Computer Science*. 2003, Lund Institute of Technology: Lund, Sweden.
10. Axelrod, R., *The Evolution of Strategies in the Iterated Prisoner's Dilemma*, in *Genetic Algorithms and Simulated Annealing*, L. Davis, Editor. 1987, Pitman Publishing: London. p. 32-41.

11. Bäck, T. *Selective Pressure in Evolutionary Algorithms: A Characterization of Selection Mechanisms* in *Proceedings of the First IEEE Conference on Evolutionary Computation. IEEE World Congress on Computational Intelligence*. 1994. Orlando FLA: IEEE.
12. Bäck, T. and M. Schutz. *Intelligent Mutation Rate Control in Canonical Genetic Algorithms* in *Proceedings of the International Symposium on Methodologies for Intelligent Systems*. 1996.
13. Bajurnow, A. and V. Ciesielski. *Layered Learning for Evolving Goal Scoring Behaviour in Soccer Players* in G. Greenwood, ed., *Proceedings of the 2004 Congress on Evolutionary Computation, Vol. 2*, p. 1828-1835. 2004: IEEE.
14. Baker, J.E. *Adaptive Selection Methods for Genetic Algorithms* in J.J. Grefenstette, ed., *Proceedings of the First International Conference on Genetic Algorithms and Their Applications*. 1985: Erlbaum.
15. Baker, J.E. *Reducing Bias and Inefficiency in the Selection Algorithm* in J.J. Grefenstette, ed., *Proceedings of the Second International Conference on Genetic Algorithms*. 1987: Erlbaum.
16. Balch, T., *The Ascii Robot Soccer Home Page*. 1995: <http://www-2.cs.cmu.edu/~trb/soccer/>.
17. Balch, T., *Teambots Domain*. 2005, <http://www.teambots.org>.
18. Baray, C., *Evolution of Coordination in Reactive Multi-Agent Systems*, in *PhD Thesis, Computer Science Department*. 1999, Indiana University: Bloomington IN.
19. Barberá, H.M., M.Z. Izquierdo, and A.G. Skarmeta. *Fuzzy Behaviours for Mobile Robots Control* in *Proceedings of the 2001 UK Workshop on Computational Intelligence*. 2001. Edinburgh, United Kingdom.
20. Barberá, H.M. and A.G. Skarmeta, *A Framework for Defining and Learning Fuzzy Behaviours for Autonomous Mobile Robots*. *International Journal of Intelligent Systems*, 2002. 17(1): p. 1-20.
21. Bentley, J.L., *Experiments on Geometric Traveling Salesman Heuristics*, in *Computing Science Technical Report 151*. 1990, AT&T Bell Laboratories: Holmdel, NJ.
22. Bentley, P., *Evolving Fuzzy Detectives: An Investigation into the Evolution of Fuzzy Rules*, in *Soft Computing in Industrial Applications*, Suzuki, et al., Editors. 2000, Springer Verlag: London.
23. Bethke, A.D., *Genetic Algorithms as Function Optimizers*, in *PhD Thesis*. 1980, University of Michigan: Ann Arbor.
24. Blickle, T. and L. Thiele, *A Comparison of Selection Schemes used in Genetic Algorithms*. *Evolutionary Computation*, 1997. 4(4): p. 361-394.
25. Bonarini, A., *Evolutionary Learning of Fuzzy Rules: Competition and Cooperation*, in *Fuzzy Modelling: Paradigms and Practice*, W. Pedrycz, Editor. 1996, Kluwer Academic Press: Norwell MA. p. 265-284.

26. Bonarini, A. *An Introduction to Learning Fuzzy Classifier Systems* in P.L. Lanzi, W. Stolzmann and S.W. Wilson, eds, *Learning Classifier Systems - from Foundations to Applications. Lecture Notes in Artificial Intelligence*, p. 83-106. 2000: Springer Verlag, Berlin.
27. Borel, E., *Probability and Certainty. Translated from the original, Probabilite et Certitude, Presses Universitaires de France, Paris (1950)*. 1963: Dover.
28. Borenstein, Y. and R. Poli. *Information Landscapes* in *Proceedings of the 2005 Conference on Genetic and Evolutionary Computation*, p. 1551-1522. 2005. Washington DC: ACM Press.
29. Borenstein, Y. and R. Poli. *Information Landscapes and Problem Hardness* in *Proceedings of the 2005 Conference on Genetic and Evolutionary Computation*, p. 1425-1431. 2005. Washington DC: The ACM Press.
30. Borenstein, Y. and R. Poli. *Information Landscapes and the Analysis of Search Algorithms* in *Proceedings of the 2005 Conference on Genetic and Evolutionary Computation*, p. 1287-1294. 2005. Washington DC: The ACM Press.
31. Bramlette, M.F. *Initialization, Mutation, and Selection Methods in Genetic Algorithms for Function Optimization* in R.K. Belew and L.B. Booker, eds, *Proceedings of the Fourth International Conference on Genetic Algorithms*. 1991. La Jolla: Morgan Kaufmann.
32. Brindle, A., *Genetic Algorithms for Function Optimization*, in *PhD Thesis*. 1981, University of Alberta.
33. Brooks, R., *Robust Layered Control System for a Mobile Robot*, in *A.I. Memo 864*. 1985, Massachusetts Institute of Technology, Artificial Intelligence Laboratory.
34. Brooks, R., *Intelligence Without Representation*. *Artificial Intelligence*, 1991. 47: p. 139-159.
35. Brussey, J.P., *Learning Behaviours for Robot Soccer*, in *PhD Thesis, School of Computer Science and Information Technology*. 2002, RMIT University: Melbourne, Australia.
36. Carse, B., T.C. Fogarty, and A. Munro., *Evolving Fuzzy Rule Based Controllers Using Genetic Algorithms*. *Fuzzy Sets and Systems*, 1996. 80: p. 273-294.
37. Caruana, R.A. and J.D. Schaffer. *Representation and Hidden Bias: Gray vs. Binary Coding for Genetic Algorithms* in *Proceedings of the Fifth International Conference on Machine Learning*. 1988: Morgan Kaufmann.
38. Castillo, C., M. Lurgi, and I. Martinez. *Chimps: An Evolutionary Reinforcement Learning Approach for Soccer Agents* in *Proceedings of the 2003 IEEE International Conference on Systems, Man and Cybernetics, Vol. 1*, p. 60-65. 2003.
39. Chaitin, G.J., *Information, Randomness & Incompleteness*. 1987, Singapore: World Scientific.
40. Chellapilla, K. and D.B. Fogel, *Co-Evolving Checkers Playing Programs using only Win, Lose or Draw*. *AeroSense99, Symposium on Applications and Science of Computational Intelligence II*, 1999. 3722: p. 303-312.

41. Chellapilla, K. and D.B. Fogel, *Evolving Neural Networks to Play Checkers Without Relying on Expert Knowledge*. IEEE Transactions on Neural Networks, 1999. 10(6): p. 1382-1391.
42. Chellapilla, K. and D.B. Fogel. *Anaconda Defeats Hoyle 6-0: A Case Study Competing an Evolved Checkers Program against Commercially Available Software in Proceedings of the 2000 Congress on Evolutionary Computation*, p. 857-863. 2000. La Jolla CA.
43. Chellapilla, K. and D.B. Fogel, *Evolving a Neural Network to Play Checkers without Human Expertise*, in *Computational Intelligence in Games*, N. Baba and L.C. Jain, Editors. 2001, Springer-Verlag. p. 39-56.
44. Chellapilla, K. and D.B. Fogel, *Evolving an Expert Checkers Playing Program Without Using Human Expertise*. IEEE Transactions on Evolutionary Computation, 2001. 5(4): p. 422-428.
45. Chen, M., et al., *Users Manual, RoboCup Soccer Server*. 2002: <http://www.robocup.org/>.
46. Chung, S. and R.A. Perez. *The Schema Theorem Considered Insufficient in Proceedings Sixth International Conference on Tools with Artificial Intelligence*, p. 748-751. 1995. Los Alamitos CA: IEEE press.
47. Ciesielski, V. and S.Y. Lai. *Developing a Dribble-and-Score Behaviour for Robot Soccer using Neuro Evolution in Proceedings of the Fifth Australia-Japan Joint Workshop on Intelligent and Evolutionary Systems*, p. 70-78. 2001. Dunedin, New Zealand.
48. Ciesielski, V., D. Mawhinney, and P. Wilson. *Genetic Programming for Robot Soccer in Proceedings of the RoboCup 2001 Symposium. Lecture Notes in Artificial Intelligence*, p. 319-324. 2002.
49. Ciesielski, V. and P. Wilson. *Developing a Team of Soccer Playing Robots by Genetic Programming in Proceedings of the Third Australia-Japan Joint Workshop on Intelligent and Evolutionary Systems*, p. 101-108. 1999. Canberra, Australia.
50. Cliff, D., I. Harvey, and P. Husbands, *Explorations in Evolutionary Robotics*. Adaptive Behavior, 1993. 2(1): p. 71-104.
51. Cordón, O. and F. Herrera, *A Three-stage Evolutionary Process for Learning Descriptive and Approximate Fuzzy Logic Controller Knowledge Bases From Examples*. International Journal of Approximate Reasoning, 1997. 17(4): p. 369-407.
52. Cordón, O., et al., *Ten Years of Genetic Fuzzy Systems: Current Framework and New Trends*. Fuzzy Sets and Systems, 2004. 141(1): p. 5-31.
53. Cordón, O., et al., *Genetic Fuzzy Systems: Evolutionary Tuning and Learning of Fuzzy Knowledge Bases*. 2001: World Scientific.
54. Cox, E., *Fuzzy Fundamentals*. IEEE Spectrum, 1992: p. 58-61.
55. Cox, E., *The Fuzzy Systems Handbook*. 1994: Academic Press.
56. Cui, Z.-H. and J.-C. Zeng. *Schema Theorem of Real-coded Non-linear Genetic Algorithm in Proceedings of the 2002 International Conference Machine Learning and Cybernetics, Vol. 3, p. 1429-1431*. 2002: IEEE.

57. Culberson, J.C., *Mutation-Crossover Isomorphisms and the Construction of Discriminating Functions*. Evolutionary Computation, 1995. 2(3): p. 279-311.
58. Czarnecki, C., R. John, and S. Bennett. *The Application of Fuzzy Logic To Real Time Multiple Robot Collision Avoidance in Proceedings of the First International ICSC Symposium on Fuzzy Logic*, p. C116-C121. 1995. Zurich, Switzerland.
59. Darwin, C., *On the Origin of Species by Natural Selection, or Preservation of Favoured Races in the Struggle for Life*. 1859, London: Murray.
60. Darwin, C., *The Origin of Species by Means of Natural Selection; or, The Preservation of Favoured Races in the Struggle for Life*, ed. J.W. Burrow. 1968, Baltimore: Penguin Books.
61. Darwin, C., *On the Origin of Species by Natural Selection, or Preservation of Favoured Races in the Struggle for Life*. Modern Library ed. 1998, New York NY: The Modern Library.
62. Davis, L. *Adapting Operator Probabilities in Genetic Algorithms* in J.D. Schaffer, ed., *Proceedings of the Third International Conference on Genetic Algorithms*. 1989: Morgan Kaufmann.
63. De Gregory, J., et al. *Robotics and Intelligent Machines: A DOE Critical Technology Roadmap* in *Proceedings of the 2001 Waste Management Symposium*. 2001. Tucson AZ.
64. de Jong, E.D. and J.B. Pollack. *Learning the Ideal Evaluation Function* in *Proceedings of the 2003 Genetic and Evolutionary Computation Conference*, p.277-288. 2003.
65. de Jong, E.D. and J.B. Pollack, *Ideal Evaluation from Coevolution*. Evolutionary Computation, 2004. 12(2): p. 159-192.
66. De Jong, K.A., *An Analysis of the Behavior of a Class of Genetic Adaptive Systems*, in *PhD Thesis*. 1975, University of Michigan: Ann Arbor.
67. De Jong, K.A., M.A. Potter, and W.M. Spears. *Using Problem Generators to Explore the Effects of Epistasis* in T. Bäck, ed., *Proceedings of the Seventh International Conference on Genetic Algorithms*. 1997. East Lansing MI: Morgan Kaufmann.
68. De Jong, K.A. and J. Sarma, *Generation Gaps Revisited*, in *Foundations of Genetic Algorithms 2*, L.D. Whitley, Editor. 1993, Morgan Kaufmann.
69. de la Maza, M. and B. Tidor, *Boltzmann Weighted Selection Improves Performance of Genetic Algorithms*, in *A.I. Memo 1345*. 1991, Massachusetts Institute of Technology, Artificial Intelligence Laboratory.
70. de la Maza, M. and B. Tidor. *An Analysis of Selection Procedures with Particular Attention Paid to Proportional and Boltzmann Selection* in S. Forrest, ed., *Proceedings of the Fifth International Conference on Genetic Algorithms*. 1993: Morgan Kaufmann.
71. Di Pietro, A., L. While, and L. Barone. *Learning in RohoCup Keepaway Using Evolutionary Algorithms* in Langdon et al., eds, *Proceedings of the Genetic and Evolutionary Computation Conference*, p. 1065-1072. 2002. New York, NY: Morgan Kaufmann.

72. Dukkupati, A., M.N. Murty, and S. Bhatnagar. *Cauchy Annealing Schedule: An Annealing Schedule for Boltzmann Selection Scheme in Evolutionary Algorithms* in *Proceedings of the 2004 Congress on Evolutionary Computation*, p. 55-62. 2004: IEEE Press, Picataway, NJ.
73. Eravasar, M., *A Comparison of Genetic Algorithm Parameterization on Synthetic Optimization Problems*, in *Masters Thesis, AFIT/ENS/99M-05, Department of Operational Sciences*. 1999, Air Force Institute of Technology: Wright-Patterson AFB, OH.
74. Eshelman, L. and J. Schawer, *Real-coded Genetic Algorithms and Interval Schemata*, in *Foundations of Genetic Algorithms 2*. 1993, Morgan Kaufmann. p. 187-202.
75. Floreano, D. and F. Mondada. *Automatic Creation of an Autonomous Agent: Genetic Evolution of a Neural-Network Driven Robot* in D. Cliff, P. Husbands, J. Meyer, and S.W. Wilson, eds, *From Animals to Animats 3: Proceedings of the Third International Conference on Simulation of Adaptive Behavior*. 1994: The MIT Press.
76. Fogel, D.B., *Blondie24: Playing at the Edge of AI*. 2002, San Francisco CA: Morgan Kaufmann.
77. Fogel, D.B., *Beyond Samuel: Evolving a Nearly Expert Checkers Player*, in *Evolutionary Computing: Theory and Applications*, A. Ghosh and S. Tsutsui, Editors. 2003, Natural Computing Series. Springer-Verlag: New York NY. p. 989-1004.
78. Fogel, D.B. and A. Ghozeil, *Schema Processing under Proportional Selection in the Presence of Random Effects*,. *IEEE Transactions on Evolutionary Computation*, 1997. 1(4): p. 290-293.
79. Fogel, D.B. and A. Ghozeil, *The Schema Theorem and the Misallocation of Trials in the Presence of Stochastic Effects*, in *Evolutionary Programming VII: Proceedings of the Seventh Ann. Conf. on Evolutionary Programming*, V.W. Porto, et al., Editors. 1998, Springer: Berlin. p. 313-321.
80. Fogel, D.B. and A. Ghozeil, *Schema Processing, Proportional Selection, and the Misallocation of Trials in Genetic Algorithms*. Information Sciences, 1999.
81. Fogel, L.J., A.J. Owens, and M.J. Walsh, *Artificial Intelligence Through Simulated Evolution*. 1966, John Wiley.
82. Forrest, S., *Scaling Fitness in the Genetic Algorithm*, in *Documentation for PRISONERS DILEMMA and NORMS Programs That Use the Genetic Algorithm. Technical Report*. 1985, University of Michigan.
83. Frauenfelder, H., et al., eds. *Landscape Concepts in Physics and Biology*. Special Issue of Physica D. Vol. 107, p. 2-4. 1997, Elsevier Science: Amsterdam, Holland.
84. Gandibleux, X., H. Morita, and N. Katoh. *The Supported Solutions Used as a Genetic Information in a Population Heuristic* in *Proceedings of the First International Conference on Evolutionary Multi-Criterion Optimization. Lecture Notes in Computer Science*, vol. 1993, p. 429-442. 2001. Zurich, Switzerland.
85. Geyer-Schulz, A., *Fuzzy Rule-Based Expert Systems and Genetic Machine Learning*. 1995: Physica-Verlag, Heidelberg.

86. Goldberg, D., *Genetic Algorithms in Search, Optimization and Machine Learning*. 1989: Addison-Wesley.
87. Goldberg, D., *A Note on Boltzmann Tournament Selection for Genetic Algorithms and Population-Oriented Simulated Annealing*. Complex Systems 4, 1990.
88. Goldberg, D. and K. Deb, *A Comparative Analysis of Selection Schemes used in Genetic Algorithms*, in *Foundations of Genetic Algorithms*, G. Rawlins, Editor. 1991, Morgan Kaufmann.
89. Goldberg, D., B. Korb, and K. Deb, *Messy Genetic Algorithms: Motivation, Analysis, and First Results*. Complex Systems 3, 1989.
90. Goldberg, D.E., K. Deb, and B. Korb. *Don't Worry, Be Messy* in R.K. Belew and L.B. Booker, eds, *Proceedings of the Fourth International Conference on Genetic Algorithms*, p. 24-30. 1991. La Jolla CA: Morgan Kaufmann.
91. Gomez, F., *Robust Non-Linear Control through Neuroevolution*, in PhD Thesis. Technical Report AI-TR-03-303. 2003, University Of Texas: Austin TX.
92. Gómez, J. *A Framework for Evolving Fuzzy Rule Classifiers* in *Proceedings of the 2004 Congress on Evolutionary Computation*, Vol. 2, p. 1727-1734. 2004. Portland OR.
93. Gomi, T., ed. *Evolutionary Robotics I: From Intelligent Robots to Artificial Life*. 1997, AAI Books: Ontario, Canada.
94. Gomi, T., ed. *Evolutionary Robotics II: From Intelligent Robots to Artificial Life*. 1998, AAI Books: Ontario, Canada.
95. Gomi, T., ed. *Evolutionary Robotics III: From Intelligent Robots to Artificial Life*. 2000, AAI Books: Ontario, Canada.
96. González, A. and F. Herrera, *Multi-Stage Genetic Fuzzy Systems Based on the Iterative Rule Learning Approach*. Mathware & Soft Computing,, 1997. 4(3): p. 233-249.
97. González, A. and R. Pérez, *SLAVE: A Genetic Learning System Based on an Iterative Approach*. IEEE Transactions on Fuzzy Systems, 1999. 7(2): p. 176-191.
98. Grefenstette, J.J. *GENESIS: A System for Using Genetic Search Procedures* in *Proceedings of the 1984 Conference on Intelligent Systems and Machines*, p. 161-165. 1984.
99. Grefenstette, J.J., *Optimization of Control Parameters for Genetic Algorithms*. Transactions on Systems, Man and Cybernetics, 1986. SMC-16,1: p. 22-128.
100. Grefenstette, J.J. and J.E. Baker. *How Genetic Algorithms Work: A Critical Look at Implicit Parallelism* in J.D. Schaffer, ed., *Proceedings of the Third International Conference on Genetic Algorithms*, p. 20-27. 1989. San Mateo CA: Morgan Kaufmann.
101. Gustafson, S.M., *Layered Learning in Genetic Programming for a Cooperative Robot Soccer Problem*, in Masters Thesis, Department of Computing and Information Science, College of Engineering. 2000, Kansas State University: Manhattan, KS.
102. Gustafson, S.M. and W.H. Hsu. *Layered Learning in Genetic Programming for a Co-operative Robot Soccer Problem* in *Proceedings of the Fourth European Conference on Genetic Programming*. 2001. Lake Como, Italy: Springer.

103. Haijun, Y., L. Minqiang, and K. Jisong. *Exact Schema Theorem based on the Space of Schema in Proceedings of the 2003 IEEE International Conference on Systems, Man and Cybernetics*, p. 349-354. 2003: IEEE.
104. Hall, J.M., *A Novel, Real-Valued Genetic Algorithm for Optimizing Radar Absorbing Materials*. NASA/CR-2004-212669. 2004, Lockheed Martin Space Operations: Hampton, VA.
105. Hancock, P.J.B. *An Empirical Comparison of Selection Methods in Evolutionary Algorithms* in T.E. Fogarty, ed., *Evolutionary Computing, AISB Workshop. Lecture Notes in Computer Science*, p. 80-94. 1994: Springer-Verlag, Berlin.
106. Harley, E., *Blondie24, Playing at the Edge of AI (Book Review)*. IEEE Computational Intelligence Bulletin, 2002. 1(1): p. 25-27.
107. Harvey, I. and P. Husbands. *Evolutionary Robotics in Proceedings of IEE Colloquium on Genetic Algorithms for Control Systems Engineering*. 1992: IEE Publications.
108. Harvey, I., P. Husbands, and D. Cliff. *Issues in Evolutionary Robotics* in J.-A. Meyer, H. Roitblat and S. Wilson, eds, *From Animals to Animats 2: Proceedings of the Second International Conference on Simulation of Adaptive Behavior*, p. 364-373. 1993: The MIT Press.
109. Haubelt, C., J. Gamenik, and J. Teich. *Initial Population Construction for Convergence Improvement of MOEAs in Evolutionary Multi-Criterion Optimization. Lecture Notes in Computer Science*, vol. 3410. 2005: Springer-Verlag, Berlin.
110. Heckendorn, R.B., S. Rana, and D.L. Whitley, *Test Function Generators as Embedded Landscapes*, in *Foundations of Genetic Algorithms 5*, W. Banzhaf and C. Reeves, Editors. 1999, Morgan Kaufmann: San Francisco CA.
111. Herrera, F. and M. Lozano, eds. *Real Coded Genetic Algorithms: Operators, Models and Foundations*. Soft Computing. Vol. 9, Number 4. 2005.
112. Herrera, F., M. Lozano, and J.L. Verdegay., *Tuning Fuzzy Controllers by Genetic Algorithms*. International Journal of Approximate Reasoning, 1995. 12: p. 299-315.
113. Hill, R.R. *A Monte-Carlo Study of Genetic Algorithm Initial Population Generation Methods in Proceedings of the Thirty-First Conference on Winter Simulation: Simulation - a Bridge to the Future*, p. 543-547. 1999. Phoenix, AZ.
114. Hillis, D., *Co-evolving Parasites Improve Simulated Evolution as an Optimization Procedure*, in *Artificial Life II*, C.G. Langton, et al., Editors. 1992, Addison-Wesley.
115. Hinterding, R., H. Gielewski, and T.C. Peachey. *The Nature of Mutation in Genetic Algorithms* in *Proceedings of the Sixth International Conference on Genetic Algorithms*, p. 65-72. 1995: Morgan Kaufmann.
116. Hoffmann, F. and O. Nelles., *Genetic Programming for Model Selection of TSK-Fuzzy Systems*. Journal of Information Sciences, 2001. 136(1-4): p. 7-28.
117. Hoffmann, F. and G. Pfister, *Evolutionary Design of a Fuzzy Knowledge Base for a Mobile Robot*. International Journal of Approximate Reasoning, 1994. 11: p. 1-158.

118. Hoffmann, F. and G. Pfister. *A New Learning Method for the Design of Hierarchical Fuzzy Controllers Using Messy Genetic Algorithms* in *Proceedings of the Sixth International Fuzzy Systems Association World Congress (IFSA'95)*, Vol. 1, p. 249-252. 1995. Sao Paulo.
119. Hoffmann, F. and G. Pfister. *Evolutionary Learning of a Fuzzy Control Rule Base for an Autonomous Vehicle* in *Proceedings of the Fifth International Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems*, p. 659-664. 1996. Granada, Spain.
120. Holland, J., *Adaptation in Natural and Artificial Systems*. 1975, Ann Arbor: The University of Michigan Press.
121. Holland, J.H., *Escaping Brittleness: The Possibilities of General-Purpose Learning Algorithms Applied to Parallel Rule-Based Systems*, in *Machine Learning: An Artificial Intelligence Approach*, R.S. Michalski, J.G. Carbonell, and T.M. Mitchell, Editors. 1986, Morgan Kaufmann: Los Altos CA.
122. Homaifar, A., et al., *Genetic Programming Design of Fuzzy Controllers for Mobile Robot Path Tracking*. *International Journal of Knowledge-Based Intelligent Engineering Systems*, 2000. 4(1): p. 33-52.
123. Homaifar, A. and E. McCormick, *Simultaneous Design of Membership Functions and Rule Sets for Fuzzy Controllers Using Genetic Algorithms*. *IEEE Transactions on Fuzzy Systems*, 1995. 3(2): p. 129-139.
124. Hordijk, W., *A Measure of Landscapes*. *Evolutionary Computation*, 1996. 4(4): p. 335-360.
125. Hornby, G.S., et al. *Autonomous Evolution of Gaits with the Sony Quadruped Robot* in *In Banzhaf et al., eds, Proceedings of 1999 Genetic and Evolutionary Computation Conference*, p. 1297-1304. 1999: Morgan Kaufmann.
126. Hruschka, E.R. and N.F.F. Ebecken. *Applying a Clustering Genetic Algorithm for Extracting Rules from a Supervised Neural Network* in *Proceedings of the 2000 IEEE-INNS-ENNS International Joint Conference on Neural Networks*. 2000. Como, Italy.
127. Hsu, W.H., et al. *Empirical Comparison of Incremental Reuse Strategies in Genetic Programming for Keep-Away Soccer* in *Late Breaking Papers of the 2004 Genetic and Evolutionary Computation Conference*. 2004. Seattle WA.
128. Husbands, P., et al. *Artificial Evolution of Control Systems* in *I.C. Parmee, ed., Proceedings of the Second International Conference on Adaptive Computing in Engineering Design and Control*, p. 41-49. 1996. University of Plymouth.
129. Husbands, P. and J.-A. Meyer, eds. *Evolutionary Robotics*. First European Workshop. 1998, Springer-Verlag: Berlin.
130. Hutter, M. *Fitness Uniform Selection to Preserve Genetic Diversity* in *D.B. Fogel et al., eds, Proceedings of the 2002 Congress on Evolutionary Computation CEC2002*, p. 783-788. 2002. Honolulu HI: IEEE Press.
131. Hwang, W. and W. Thompson. *Design of Fuzzy Logic Controllers Using Genetic Algorithms* in *Proceedings of the Third IEEE International Conference on Fuzzy Systems*, p. 1383-1388. 1994. Piscataway NJ: IEEE Computer Press,.

132. Ibrahim, A., *Fuzzy Logic for Embedded Systems Applications*. 2003: Newnes.
133. Jang, J.-S., C.-T. Sun, and E. Mizutani, *Neuro-Fuzzy and Soft Computing*. 1997: Prentice Hall.
134. Jang, J.-S.R. *Fuzzy Controller Design Without Domain Experts* in *Proceedings of the IEEE International Conference on Fuzzy Systems*, p. 289-296. 1992.
135. Jang, J.-S.R., *Self-Learning Fuzzy Controllers Based on Temporal Back Propagation*. *IEEE Transactions on Neural Networks*, 1992. 3(5): p. 714-723.
136. Janikow, C.Z. and Z. Michalewicz. *An Experimental Comparison of Binary and Floating Point Representations in Genetic Algorithms* in R.K. Belew and L.B. Booker, eds, *Proceedings of the Fourth International Conference on Genetic Algorithms*. 1991. La Jolla: Morgan Kaufmann.
137. Jones, T., *One Operator, One Landscape*, in *Technical Report 95-02-025*. 1995, Santa Fe Institute: Santa Fe NM.
138. Jones, T. and S. Forrest. *Fitness Distance Correlation as a Measure of Problem Difficulty for Genetic Algorithms* in L.J. Eshelman, ed., *Proceedings of the Sixth International Conference on Genetic Algorithms*, p. 184-192. 1995: Morgan Kaufmann.
139. Jones, T. and S. Forrest, *Genetic Algorithms and Heuristic Search*, in *Technical Report 95-02-021*. 1995, Santa Fe Institute: Santa Fe NM.
140. Jones, T.C., *Evolutionary Algorithms, Fitness Landscapes and Search*, in *PhD Thesis*. 1995, University of New Mexico: Albuquerque NM.
141. Kaindl, H., *Tree Searching Algorithms*, in *Computers, Chess, and Cognition*, T.A. Marsland and J. Schaeffer, Editors. 1990, Springer: New York NY. p. 133-168.
142. Kandel, A., *Fuzzy Mathematical Techniques with Applications*. 1986: Addison-Wesley, Reading MA.
143. Kandel, A., ed. *Fuzzy Expert Systems*. 1992, CRC Press Inc.: Boca Raton FL.
144. Kappler, C. *Are Evolutionary Algorithms Improved by Large Mutations?* in H.-M. Voigt, W. Ebeling, I. Rechenberg, and H.-P. Schwefel, eds, *Parallel Problem Solving from Nature IV. Lecture Notes in Computer Science*, p. 346-355. 1996. Berlin: Springer-Verlag.
145. Karr, C., *Genetic Algorithms for Fuzzy Controllers*. *AI Expert*, 1991. 6(2): p. 26-33.
146. Kauffman, S.A., *Adaptation on Rugged Fitness Landscapes*, in *Lectures in the Sciences of Complexity*, D. Stein, Editor. 1989, Addison-Wesley. p. 527-618.
147. Kauffman, S.A. and S. Levin, *Towards a General Theory of Adaptive Walks on Rugged Landscapes*. *Journal of Theoretical Biology*, 1987. 128: p. 11-45.
148. Kinoshita, S. and Y. Yamamoto. *11 Monkeys Description* in Veloso et al., eds, *Proceedings of Robocup-99: Robot Soccer World Cup III. Lecture Notes In Computer Science, Vol. 1856*, p. 550-553. 2000: Springer-Verlag, London.
149. Kinzel, J., F. Klawoon, and R. Kruse. *Modifications of Genetic Algorithms for Designing and Optimizing Fuzzy Controllers* in *Proceedings of the First IEEE Conference on Evolutionary Computation*. 1994. Orlando FLA.

150. Kitano, H., *Designing Neural Networks Using Genetic Algorithms with Graph Generation System*. Complex Systems, 1990. 4: p. 461-476.
151. Kitano, H., et al. *The Robot World Cup Initiative in Working Notes of the 1995 International Joint Conference on Artificial Intelligence Workshop on Entertainment and AI/Alife*, p. 19-24. 1995. Montreal, Canada.
152. Kitano, H., et al. *RoboCup: The Robot World Cup Initiative in Proceedings of the First International Conference on Autonomous Agents*, p. 340-347. 1997. Marina Del Rey, CA.
153. Kitano, H., et al., *RoboCup: A Challenge Problem for AI*, in *AI Magazine*, 18(1): p.73-85. 1997.
154. Kitano, H., et al. *Grand Challenge AI Applications in Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence*, p. 1677-1683. 1993: Morgan Kaufmann, San Mateo, CA.
155. Kitano, H., et al. *The RoboCup Synthetic Agent Challenge 97 in Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence*, p. 24-29. 1997. Nagoya, Japan.
156. Klir, G.J. and T.A. Folger, *Fuzzy Sets, Uncertainty and Information*. 1988, Prentice Hall.
157. Kosko, B., *Neural Networks and Fuzzy Systems: A Dynamical Systems Approach*. 1991, Upper Saddle River NJ: Prentice Hall.
158. Koza, J.R., *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. 1992: The MIT Press.
159. Kruse, R., J. Gebhardt, and F. Klawonn, *Foundations of Fuzzy Systems*. 1994: Wiley.
160. Kuhlmann, G. and P. Stone. *Progress in Learning 3 vs. 2 Keepaway in D. Polani et al., eds, RoboCup-2003: Robot Soccer World Cup VII*. 2004: Springer Verlag, Berlin.
161. Langdon, W.B. and P. Nordin. *Evolving Hand-Eye Coordination for a Humanoid Robot with Machine Code Genetic Programming in Miller et al., eds, Proceedings of the Fourth European Conference on Genetic Programming. Lecture Notes in Computer Science, Vol. 2038, p. 313-324*. 2001: Springer-Verlag.
162. Langdon, W.B. and R. Poli, *Why "Building Blocks" Don't Work on Parity Problems*, in *Technical Report CSRP-98-17, School of Computer Science*. 1998, University of Birmingham.
163. Langdon, W.B. and R. Poli. *Why Ants are Hard in Proceedings of the Third Annual Conference on Genetic Programming*. 1998: Morgan Kauffman.
164. Langdon, W.B. and R. Poli, *Foundations of Genetic Programming*. 2002: Springer-Verlag.
165. Lazarus, C. and H. Hu. *Evolving Goalkeeper Behaviour for Simulated Soccer Competition in Proceedings of the Third IASTED International Conference on Artificial Intelligence and Applications*. 2003. Benalmádena, Spain.
166. Lee, C.-C., *Fuzzy Logic in Control Systems: Fuzzy Logic Controller - Part I*. IEEE Transactions on Systems, Man and Cybernetics, 1990. 20(2): p. 404-418.

167. Lee, C.-C., *Fuzzy Logic in Control Systems: Fuzzy Logic Controller - Part II*. IEEE Transactions on Systems, Man and Cybernetics, 1990. 20(2): p. 419-435.
168. Lee, C.-Y., *Entropy-Boltzmann Selection in Genetic Algorithms*. IEEE Transactions on Systems, Man and Cybernetics, Part B, 2003. 33(1): p. 138-149.
169. Lee, M.A. and H. Takagi. *Integrating Design Stages of Fuzzy Systems Using Genetic Algorithms* in *Proceedings of the Second IEEE International Conference on Fuzzy Systems*, p. 613-617. 1993. San Francisco CA.
170. Leger, C., *Darwin2K: An Evolutionary Approach to Automated Design for Robotics*. 2000: Kluwer Academic.
171. Li, Y., P.M. Pardalos, and M.G.C. Resende, *A Greedy Randomized Adaptive Search Procedure for the Quadratic Assignment Problem*. In P. Pardalos and H. Wolkowicz, eds, *Quadratic Assignment and Related Problems*. DIMACS Series in Discrete Mathematics and Theoretical Computer Science, 1994. 16: p. 237-261.
172. Lima, P., et al., *RoboCup 2004 Competitions and Symposium: A Small Kick for Robots, a Giant Score for Science*, in *AI Magazine* 6(2). 2005.
173. Lipsitch, M. *Adaptation on Rugged Landscapes Generated by Iterated Local Interactions of Neighboring Genes* in *Proceedings of the Fourth International Conference on Genetic Algorithms*, p. 128-135. 1991. La Jolla CA: Morgan Kaufmann.
174. Luke, S. *Evolving SoccerBots: A Retrospective* in *Proceedings of the Twelfth Annual Conference of the Japanese Society for Artificial Intelligence*. 1998. Tokyo, Japan.
175. Luke, S. *Genetic Programming Produced Competitive Soccer Softbot Teams for RoboCup97* in J. Koza et al., eds, *Proceedings of the Third Annual Genetic Programming Conference*, p. 204-222. 1998: Morgan Kaufmann, San Francisco.
176. Luke, S. *When Short Runs Beat Long Runs* in *Proceedings of the 2001 Genetic and Evolutionary Computation Conference*, p. 74-80. 2001. San Francisco CA.
177. Luke, S., et al. *Co-evolving Soccer Softbot Team Coordination with Genetic Programming* in H. Kitano, ed., *RoboCup-97: Robot Soccer World Cup I. Lecture Notes in Artificial Intelligence*, p. 398-411. 1998: Springer-Verlag, Berlin.
178. Luke, S. and L. Spector. *Evolving Teamwork and Coordination with Genetic Programming* in J.R. Koza et al., eds, *Proceedings of the First Annual Conference on Genetic Programming*, p. 150-156. 1996. Cambridge MA: The MIT Press.
179. Lund, H.H. *Evolving Robot Control Systems* in J. T. Alander, ed., *Proceedings of the First Nordic Workshop on Genetic Algorithms*. 1995. University of Vaasa.
180. Lush, J.L., *Progeny Test and Individual Performance as Indicators of an Animal's Breeding Value*. Journal of Dairy Science, 1935. 18: p. 1-19.
181. Mackworth, A., *On Seeing Robots*. Vision Interface, 1992.
182. Macready, W.G. and D.H. Wolpert, *On 2-armed Gaussian Bandits and Optimization*, in *Technical Report SFI-TR-96-03-009*. 1996, Santa Fe Institute: Santa Fe NM.
183. Magdalena, L. and F. Monasterio-Huelin, *A Fuzzy Logic Controller with Learning Through the Evolution of its Knowledge Base*. International Journal of Approximate Reasoning, 1997. 16(3/4): p. 335-338.

184. Mamdani, E. and S. Assilian, *An Experiment in Linguistic Synthesis with a Fuzzy Logic Controller*. International Journal of Man-Machine Studies, 1975. 7(1): p. 1-13.
185. Manderick, B., M. de Weger, and P. Spiessens. *The Genetic Algorithm and the Structure of the Fitness Landscape* in *Proceedings of the Fourth International Conference on Genetic Algorithms*, p. 143-150. 1991. La Jolla CA: Morgan Kaufmann.
186. Maniadakis, M. and H. Surmann. *A Genetic Algorithm for Structural and Parametrical Tuning of Fuzzy Systems* in *Proceedings of the 1999 European Symposium on Intelligent Techniques*. 1999. Greece.
187. Merz, P. and B. Freisleben, *Fitness Landscapes and Memetic Algorithm Design*, in *New Ideas in Optimization*, D. Corne, M. Dorigo, and F. Glover, Editors. 1999, McGraw-Hill: London. p. 245-260.
188. Meyer, T.P. and N.H. Packard, *Local Forecasting of High-Dimensional Chaotic Dynamics*, in *Nonlinear Modeling and Forecasting*, M. Casdagli and S. Eubank, Editors. 1992, Addison-Wesley.
189. Ming, L., Y.-P. Wang, and Y.-M. Cheung. *A New Schema Theorem for Uniform Crossover based on Ternary Representation* in *Proceedings of the 2004 Intelligent Sensors, Sensor Networks and Information Processing Conference*, p. 235-239. 2004: IEEE.
190. Mirzaei, A., M. Moallem, and B. Mirzaei. *Designing a Genetic-Fuzzy Anti-lock Brake System Controller* in *Proceedings of the 2005 IEEE International Symposium on Intelligent Control*, p. 1246-1250. 2005. Limassol, Cyprus.
191. Mitchell, M., *An Introduction to Genetic Algorithms*. 1996: The MIT Press.
192. Mitchell, M., J.P. Crutchfield, and P.T. Hraber, *Evolving Cellular Automata to Perform Computations: Mechanisms and Impediments*. Physica, 1994. D75: p. 361-391.
193. Montana, D.J. and L. Davis. *Training Feedforward Neural Networks Using Genetic Algorithms* in *Proceedings of the International Joint Conference on Artificial Intelligence*, p. 762-767. 1989.
194. Moraglio, A. and R. Poli. *Topological Interpretation of Crossover* in *Deb et al., eds, Proceeding of the 2004 Genetic and Evolutionary Computation Conference, Part I*, p. 1377-1388. 2004. Seattle WA: Springer.
195. Mühlenbein, H., *Evolution in Time and Space - The Parallel Genetic Algorithm.*, in *Foundations of Genetic Algorithms*, G. Rawlins, Editor. 1991, Morgan Kaufmann: San Mateo CA. p. 316-338.
196. Nakashima, T., et al. *An Evolutionary Approach for Strategy Learning in RoboCup Soccer* in *Proceedings of the 2004 IEEE International Conference on Systems, Man and Cybernetics, Vol. 2*, p. 2023-2028. 2004.
197. Nilsson, N., *Artificial Intelligence: A New Synthesis*. 1998: Morgan Kaufmann, San Francisco.
198. Noda, I. *Soccer Server: A Simulator of Robocup* in *Proceedings of AI Symposium '95. Japanese Society for Artificial Intelligence*, pp. 29-34. 1995.
199. Noda, I., et al., *Soccer Server: A Tool for Research on Multiagent Systems*. Applied Artificial Intelligence, 1998. 12: p. 233-250.

200. Noda, I. and P. Stone. *The RoboCup Soccer Server and CMUnited: Implemented Infrastructure for MAS research* in T. Wagner and O. Rana, eds, *International Workshop on Infrastructure for Multi-Agent Systems (Agents 2000)*. Lecture Notes in Computer Science, p. 94-101. 2001. Barcelona, Spain.
201. Nolfi, S. and D. Floreano, *Evolutionary Robotics: The Biology, Intelligence, and Technology of Self-Organizing Machines*. 2000, Cambridge MA: The MIT Press.
202. Nolfi, S., et al. *How to Evolve Autonomous Robots: Different Approaches in Evolutionary Robotics* in R.A. Brooks and P. Maes, eds, *Proceedings of the Fourth International Workshop on Artificial Life*. 1994: The MIT Press.
203. Palmer, C.C., *An Approach to a Problem in Network Design using Genetic Algorithms*. 1994, Polytechnic University: Troy, NY.
204. Parker, G.B., T.S. Doherty, and M. Parker. *Evolution and Prioritization of Survival Strategies for a Simulated Robot in Xpilot* in *Proceedings of the 2005 IEEE Congress on Evolutionary Computation*. 2005. Edinburgh, UK.
205. Parker, G.B., M. Parker, and S.D. Johnson. *Evolving Autonomous Agent Control in the Xpilot Environment* in *Proceedings of the 2005 IEEE Congress on Evolutionary Computation*. 2005. Edinburgh, UK.
206. Parodi, A. and P. Bonelli. *A New Approach to Fuzzy Classifier Systems* in *Proceedings of the Fifth International Conference on Genetic Algorithms*, p. 223-230. 1993. San Mateo, CA: Morgan Kaufman.
207. Pedrycz, E. and F. Gomide, *An Introduction to Fuzzy Sets: Analysis and Design*. 1998: The MIT Press.
208. Pedrycz, W., ed. *Fuzzy Evolutionary Computation*. 1997, Kluwer Academic Publishers: Boston.
209. Pham, D.T. and D. Karaboga, *Optimum Design of Fuzzy Logic Controllers Using Genetic Algorithms*. Journal of Systems Engineering, 1991. 1: p. 114-118.
210. Pipe, A. and B. Carse. *Autonomous Acquisition of Fuzzy Rules for Mobile Robot Control: First Results From Two Evolutionary Computation Approaches* in *Proceedings of the 2000 Genetic and Evolutionary Computation Conference*, p. 849-856. 2000. Las Vegas, NV.
211. Poli, R. *Why the Schema Theorem is Correct also in the Presence of Stochastic Effects* in *Proceedings of the 2000 Congress on Evolutionary Computation*. 2000. San Diego CA.
212. Poli, R., *Exact Schema Theory for Genetic Programming and Variable-Length Genetic Algorithms with One-Point Crossover*. Genetic Programming and Evolvable Machines, 2001. 2(2): p. 123-163.
213. Poli, R. and N.F. McPhee. *Exact Schema Theory for GP and Variable-length GAs with Homologous Crossover* in L. Spector et al., eds, *Proceedings of the 2001 Genetic and Evolutionary Computation Conference*. 2001: Morgan Kaufmann.
214. Radcliffe, N.J., *Non-linear Genetic Representations*, in *Parallel Problem Solving from Nature 2*, R. Männer and B. Manderick, Editors. 1992: Amsterdam, Holland. p. 259-268.

215. Rechenberg, I., *Evolutionsstrategie: Optimierung technischer Systeme und Prinzipien der biologischen Evolution*. 1973: Frommann-Holzboog, Stuttgart.
216. Reeves, C.R., *Landscapes, Operators and Heuristic Search*. Annals of Operational Research, 1999. 86: p. 473-490.
217. Reidys, C.M. and P.F. Stadler, *Combinatorial Landscapes*. Society for Industrial and Applied Mathematics, SIAM Review, 2002. 44(1): p. 3-54.
218. Reilly, C.H. *Properties of Synthetic Optimization Problems* in D.J. Dedeiros, E.F. Watson, J.S. Carson, and M.S. Manivannan, eds, *Proceedings of the 1998 Winter Simulation Conference*, p. 617-621. 1998. Washington DC: IEEE.
219. Riedmiller, M., et al. *Brainstormers 2D - Team Description 2005* in *Team Description Papers, Proceedings of RoboCup 2005 (CD) (to appear)*. 2005.
220. Riedmiller, M., et al. *Karlsruhe Brainstormers - a Reinforcement Learning Approach to Robotic Soccer* in P. Stone, T. Balch and G. Kraetschmar, eds, *RoboCup-2000: Robot Soccer World Cup IV. Lecture Notes in Artificial Intelligence*. 2001: Springer Verlag, Berlin.
221. Right, A.H., J.E. Row, and J.R. Neil. *Analysis of the Simple Genetic Algorithm on the Single-peak and Double-peak Landscapes* in *Proceedings of the 2002 Congress on Evolutionary Computation*, p. 214-219. 2002. Hawaii, USA.
222. Riley, J. *The SimpleSoccer Machine Learning Environment* in S.-B. Cho, H. X. Nguen and Y. Shan, eds, *Proceedings of the First Asia-Pacific Workshop on Genetic Programming*, p. 24-30. 2003. Canberra, Australia.
223. Rothlauf, F., D. Goldberg, and A. Heinzl. *Bad Codings and the Utility of Well-Designed Genetic Algorithms* in *Proceedings of the 2000 Genetic and Evolutionary Computation Conference*. 2000: Morgan Kaufmann.
224. Rothlauf, F. and D.E. Goldberg, *Redundant Representations in Evolutionary Computation*. Evolutionary Computation, 2003. 11(4): p. 381-415.
225. Saffiotti, A., E.H. Ruspini, and K. Konolige, *Using Fuzzy Logic for Mobile Robot Control*, in *International Handbook of Fuzzy Sets and Possibility Theory*, H. Prade, D. Dubois, and H.J. Zimmermann, Editors. 1997, Kluwer Academic Publishers.
226. Sanchez, E., T. Shibata, and L.A. Zadeh, eds. *Genetic Algorithms and Fuzzy Logic Systems: Soft Computing Perspectives*. 1997, World Scientific.
227. Schraudolph, N.N. and R.K. Belew, *Dynamic Parameter Encoding for Genetic Algorithms*. Machine Learning, 1992. 9: p. 9-21.
228. Schwarz, T.J., *Fuzzy Systems in the Real World*. AI Expert, 1990. August.
229. Schwefel, H.P., *Numerical Optimisation of Computer Models*. 1981: John Wiley and Sons.
230. Setness, M. and H. Roubos, *GA-Fuzzy Modeling and Classification: Complexity and Performance*. IEEE Transactions on Fuzzy Systems, 2000. 8(5): p. 509-522.
231. Shaefer, C.G. *The ARGOT Strategy: Adaptive Representation Genetic Optimizer Technique* in *Proceedings of the Second International Conference on Genetic Algorithms*, p. 50-58. 1987. Cambridge MA: Erlbaum.

232. Shannon, C.E., *A Mathematical Theory of Communication*. Bell System Technical Journal, 1948. 27: p. 379-423; 623-656.
233. Shaw, I.S., *Fuzzy Control of Industrial Systems: Theory & Applications*. 1998, Boston: Kluwer Academic Publishers.
234. Skellett, B., et al. *Maximally Rugged NK Landscapes Contain the Highest Peaks* in Beyer et al., eds, *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO-2005*. 2005. Washington DC: ACM Press.
235. Smith, R.E. and J.E. Smith, *New Methods for Tunable, Random Landscapes*, in *Foundations of Genetic Algorithms 6*. 2001, Morgan Kaufmann: San Francisco CA.
236. Smith, S., *A Learning System Based on Genetic Adaptive Algorithms*, in *PhD Thesis, Department of Computer Science*. 1980, University of Pittsburgh: Pittsburgh, PA.
237. Smith, T., et al., *Fitness Landscapes and Evolvability*. *Evolutionary Computation*, 2002. 10(1): p. 1-34.
238. Soltoggio, A. *A Comparison of Genetic Programming and Genetic Algorithms in the Design of a Robust, Saturated Control System*. in *Proceedings of the Sixth Genetic and Evolutionary Computation Conference*. 2004. Seattle WA.
239. Spencer, H., *Social Statics, or, The Conditions Essential to Human Happiness Specified and the First of Them Developed*. 1868, London: Williams and Norgate.
240. Stabell, B. and K.R. Schouten, *The Story of XPilot*. *ACM Crossroads*, 1996. Winter 1996 - 3.2.
241. Stabell, B., et al., *The XPilot Page*. 2005, <http://www.xpilot.org/>.
242. Stadler, P., *Landscapes and Their Correlation Functions*. *Journal of Mathematical Chemistry*, 1996(20): p. 1-45.
243. Stephens, C.R. and H. Waelbroeck. *Effective Degrees of Freedom in Genetic Algorithms and the Block Hypothesis* in T. Bäck, ed., *Proceedings of the Seventh International Conference on Genetic Algorithms*. 1997. East Lansing: Morgan Kaufmann.
244. Stephens, C.R. and H. Waelbroeck, *Schemata Evolution and Building Blocks*. *Evolutionary Computation*, 1999. 7(2): p. 109-124.
245. Stolzenburg, F., O. Obst, and J. Murray. *Qualitative Velocity and Ball Interception* in *Proceedings of the Twenty-Fifth Annual German Conference on AI: Advances in Artificial Intelligence. Lecture Notes In Computer Science, Vol. 2479, p. 283-298*. 2002. Aachen, Germany: Springer-Verlag.
246. Stone, P., *Layered Learning in Multiagent Systems*, in *PhD Thesis, Computer Science Department, Technical Report CMU-CS98-187*. 1998, Carnegie Mellon University.
247. Stone, P. and R. Sutton. *Scaling Reinforcement Learning Toward RoboCup Soccer* in *Proceedings of the Eighteenth International Conference on Machine Learning*. 2001. Williamstown MA.
248. Stone, P., R.S. Sutton, and G. Kuhlmann, *Reinforcement Learning for RoboCup-Soccer Keepaway*. *Adaptive Behavior*, 2005. 13(3): p. 165-188.

- 249. Stone, P., R.S. Sutton, and S.P. Singh. *Reinforcement Learning for 3 vs. 2 Keepaway in Robocup 2000: Robot Soccer World Cup IV*. P. Stone, T.R. Balch, and G.K. Kraetzschmar, eds. *Lecture Notes In Computer Science*, vol. 2019, p. 249-258. 2001: Springer-Verlag, London.
- 250. Stone, P. and M. Veloso. *Team-partitioned, Opaque-transition Reinforcement Learning in Proceedings of the Third International Conference on Autonomous Agents*. 1999. Seattle WA.
- 251. Stone, P. and M.M. Veloso. *Layered Learning in Proceedings of the Eleventh European Conference on Machine Learning*, p. 369-381. 2000: Springer, Berlin.
- 252. Sugeno, M. and G. Kang, *Structure Identification of Fuzzy Model*. Fuzzy Sets and Systems, 1988. 28: p. 15-33.
- 253. Syswerda, G. *Uniform Crossover in Genetic Algorithms* in J.D. Schaffer, ed., *Proceedings of the Third International Conference on Genetic Algorithms*. 1989: Morgan Kaufmann.
- 254. Takagi, T. and M. Sugeno, *Fuzzy Identification of Systems and its Applications to Modelling and Control*. IEEE Transactions on Systems, Man, and Cybernetics, 1985. 15: p. 116-132.
- 255. Tan, K.C., et al. *A Messy Genetic Algorithm for the Vehicle Routing Problem With Time Window Constraints in Proceedings of the 2001 Congress on Evolutionary Computation*, p. 679-686. 2001. Seoul, Korea.
- 256. Thierens, D. *Adaptive Mutation Rate Control Schemes in Genetic Algorithms in Proceedings of the 2002 Congress on Evolutionary Computation*. 2002. Honolulu HI.
- 257. U.S. DOE, *Robotics and Intelligent Machines in the U.S. Department of Energy: A Critical Technology Roadmap*. <http://www.sandia.gov/isrc/RIMfinal.pdf>. 1998, U.S. Department of Energy.
- 258. U.S. DOE, *A DOE Success: The RIM Roadmap*. <http://www.sandia.gov/rim/documents/successtories.html>. 2004, U.S. Department of Energy.
- 259. Uchibe, E., *Cooperative Behavior Acquisition by Learning and Evolution in a Multi-Agent Environment for Mobile Robots*, in *PhD Thesis*. 1999, Osaka University.
- 260. Valenzuela-Rendón, M. *The Fuzzy Classifier System: Motivations and First Results* in H.P. Schwefel and R. Männer, eds, *Proceedings of the First International Conference on Parallel Problem Solving from Nature*, p. 330-334. 1991: Springer-Verlag, Berlin.
- 261. Varela, R., et al. *Heuristic Generation of the Initial Population in Solving Job Shop Problems by Evolutionary Strategies* in J. Mira and J.V. Sánchez-Andrés, eds, *Proceedings of the International Work-Conference on Artificial and Natural Neural Networks: Foundations and Tools For Neural Modeling*. *Lecture Notes in Computer Science*, vol. 1606, p. 690-699. 1999: Springer-Verlag, London.
- 262. Vassilev, V., T. Fogarty, and J. Miller, *Information Characteristics and the Structure of Landscapes*. *Evolutionary Computation*, 2000. 8(1): p. 31-60.

263. Vassilev, V.K. *Information Analysis of Fitness Landscapes* in P. Husbands and I. Harvey, eds, *Proceedings of the Fourth European Conference on Artificial Life*. 1997. Brighton, England: The MIT Press.
264. Velasco, J.R., *Genetic-based On-line Learning for Fuzzy Process Control*. International Journal of Intelligent Systems, 1998. 13(10-11): p. 891-903.
265. Venturini, G. *SIA: A Supervised Inductive Algorithm with Genetic Search for Learning Attributes Based Concepts* in P.D. Brazdil, ed., *Proceedings of the European Conference on Machine Learning*. 1993. Vienna, Austria: Springer.
266. Vose, M.D., *Generalizing the Notion of Schema in Genetic Algorithms*. Artificial Intelligence, 1991. 50(3): p. 385-296.
267. Weinberger, E., *Correlated and Uncorrelated Fitness Landscapes and How to Tell the Difference*. Biological Cybernetics, 1990. 63: p. 325-336.
268. Weinberger, E., *Fourier and Taylor Series on Fitness Landscapes*. Biological Cybernetics, 1990. 65: p. 321-330.
269. Weinberger, E., *Local Properties of Kauffman's N-K model: A Tunably Rugged Energy Landscape*. Physical Review A, 1991. 44(10): p. 6399-6413.
270. Weinberger, E., *Measuring Correlations in Energy Landscapes and Why it Matters*, in *Information Dynamics*, H. Atmanspacher and H. Scheingraber, Editors. 1991, Plenum Press: New York NY.
271. Westerberg, C.H. and J. Levine. *Investigation of Different Seeding Strategies in a Genetic Planner* in E.J. Boers, J. Gottlieb, P.L. Lanzi, R.E. Smith, S. Cagnoni, E. Hart, G.R. Raidl, and H. Tijink, eds, *Proceedings of the Evoworkshops on Applications of Evolutionary Computing. Lecture Notes In Computer Science*, vol. 2037, p. 505-514. 2001: Springer-Verlag, London.
272. Whiteson, S., et al. *Evolving Keepaway Soccer Players through Task Decomposition* in E. Cantu-Paz et al., eds, *Genetic and Evolutionary Computation - GECCO-2003, volume 2723 of Lecture Notes in Computer Science*, p. 356-368. 2003. Chicago IL: Springer-Verlag.
273. Whiteson, S., et al., *Evolving Keepaway Soccer Players through Task Decomposition*. Machine Learning, 2005. 59: p. 5-30.
274. Whiteson, S. and P. Stone. *Concurrent Layered Learning* in *Proceedings of the Second International Joint Conference on Autonomous Agents and Multiagent Systems*, p. 193-200. 2003.
275. Whitley, D., Mathias, K., & Fitzhorn, P. *Delta Coding: An Iterative Search Strategy for Genetic Algorithms* in *Proceedings of the Fourth International Conference on Genetic Algorithms*, p. 77-84. 1991. La Jolla: Morgan Kaufmann.
276. Whitley, L.D. *Using Reproductive Evaluation to Improve Genetic Search and Heuristic Discovery* in J.J. Grefenstette, ed., *Genetic Algorithms and Their Applications: Proceedings of the Second International Conference on Genetic Algorithms*. 1987: Erlbaum.

277. Whitley, L.D. *The Genitor Algorithm and Selection Pressure: Why Rank-Based Allocation of Reproductive Trials is Best* in J.D. Schaffer, ed., *Proceedings of the Third International Conference on Genetic Algorithms*. 1989: Morgan Kaufmann.
278. Wooldridge, M. and A. Haddadi. *Making It Up As They Go Along: A Theory of Reactive Cooperation* in W. Wobcke, M. Pagnucco, and C. Zhang, eds, *Agents and Multi-Agent Systems- Formalisms, Methodologies, and Applications. Lecture Notes in Artificial Intelligence*. 1998: Springer-Verlag.
279. Wright, S. *The Roles of Mutation, Inbreeding, Crossbreeding and Selection in Evolution* in D.F. Jones, ed., *Proceedings of the Sixth International Congress on Genetics*, p. 356-366. 1932. Brooklyn NY: Brooklyn Botanic Garden.
280. Wu, C.-J. and G.-Y. Liu. *Design of Fuzzy Logic Controllers Using Genetic Algorithms* in *Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics, Vol. 6*, p. 104-109. 1999.
281. Yang, C. and K.E. Nygard. *The Effects of Initial Population in Genetic Search for Time Constrained Traveling Salesman Problems* in *Proceedings of the 1993 ACM Conference on Computer Science*, p. 378-383. 1993. Indianapolis IN: ACM Press, New York.
282. Yao, X. and Y. Liu. *Fast Evolutionary Programming* in L.J. Fogel, P.J. Angeline, and T. Bäck, eds, *Evolutionary Programming V: Proceedings of the Fifth Annual Conference on Evolutionary Programming*, p. 451-460. 1996. Cambridge MA: The MIT Press.
283. Zadeh, L., *Fuzzy Sets*. Journal of Information and Control, 1965. Vol. 8.
284. Zadeh, L., *Outline of a New Approach to the Analysis of Complex Systems and Decision Process*. IEEE Transactions on Systems, Man, and Cybernetics, 1973. 3(1): p. 28-44.
285. Zadeh, L., *The Concept of a Linguistic Variable and its Application to Approximate Reasoning - part I*. Information Sciences, 1975. 8: p. 199-249.
286. Zadeh, L., *The Concept of a Linguistic Variable and its Application to Approximate Reasoning - part II*. Information Sciences, 1975. 8: p. 301-357.
287. Zhou, Y.-S. and L.-Y. Lai. *Optimal Design for Fuzzy Controllers by Genetic Algorithms* in *Proceedings of the IEEE/IAS Conference on Industrial Automation and Control: Emerging Technologies*, p. 429-435. 1995.
288. Zufferey, J.C., et al. *Evolving Vision-based Flying Robots* in Bülthoff et al., eds, *Proceedings of the Second International Workshop on Biologically Motivated Computer Vision. Lecture Notes in Computer Science, Vol. 2525*, p.592-600. 2002: Springer-Verlag.

Appendix A

Function Syntax	Returns	Description
(home)	v	A vector to my home (my starting position).
(ball)	v	A vector to the ball.
(findball)	v	A zero-length vector to the ball.
(block-goal)	v	A vector to the closest point on the line segment between the ball and the goal I defend.
(away-mates)	v	A vector away from known teammates, computed as the inverse of $\sum_{m \in \{\text{vectors to teammates}\}} \frac{\max - \ m\ }{\ m\ } m$
(away-oppo)	v	A vector away from known opponents, computed as the inverse of $\sum_{o \in \{\text{vectors to opponents}\}} \frac{\max - \ o\ }{\ o\ } o$
(squad1)	b	t if I am first in my squad, else nil.
(opp-closer)	b	t if an opponent is closer to the ball than I am, else nil.
(mate-closer)	b	t if a teammate is closer to the ball than I am, else nil.
(home-of <i>i</i>)	v	A vector to the home of teammate <i>i</i> .
(block-near-opp <i>v</i>)	v	A vector to the closest point on the line segment between the ball and the nearest known opponent to me. If there is no known opponent, return <i>v</i> .
(mate <i>i v</i>)	v	A vector to teammate <i>i</i> . If I can't see him, return <i>v</i> .
(inv <i>v</i>)	v	<i>v</i> rotated 180 degrees.
(if- <i>v b v1 v2</i>)	v	If <i>b</i> is t, return <i>v1</i> , else return <i>v2</i> .
(sight <i>v</i>)	v	Rotate <i>v</i> just enough to keep the ball in sight.
(ofme <i>i</i>)	b	Return t if the ball is within $\frac{i}{\max}$ units of me, else nil.
(ofhome <i>i</i>)	b	Return t if the ball is within $\frac{i}{\max}$ units of my home, else nil.
(ofgoal <i>i</i>)	b	Return t if the ball is within $\frac{i}{\max}$ units of the goal, else nil.
(weight-+ <i>i v1 v2</i>)	v	Return $\frac{v1(i) + v2(9-i)}{9}$.
(far-mate <i>i k</i>)	k	A vector to the most offensive-positioned teammate who can receive the ball with at least $\frac{i+1}{10}$ probability. If none, return <i>k</i> .
(mate-m <i>i1 i2 k</i>)	k	A vector to teammate <i>i1</i> if his position is known and he can receive the ball with at least $\frac{i2+1}{10}$ probability. If not, return <i>k</i> .
(kick-goal <i>i k</i>)	k	A vector to the goal if the kick will be successful with at least $\frac{i+1}{10}$ probability. If not, return <i>k</i> .
(dribble <i>i k</i>)	k	A "dribble" kick of size $\frac{i}{10}(\max)$ in the direction of <i>k</i> .
(kick-goal!)	k	Kick to the goal.
(far-mate!)	k	Kick to the most offensive-positioned teammate. If there is none, kick to the goal.
(kick-clear)	k	Kick out of the goal area. Namely, kick away from opponents as computed with (away-oppo), but adjust the direction so that it is at least 135 degrees from the goal I defend.
(kick-if <i>b k1 k2</i>)	k	If <i>b</i> is t, return <i>k1</i> , else return <i>k2</i> .
(opponent-close <i>i</i>)	b	Return t if an opponent is within $\frac{\max}{(1.5)^i}$ of me.
0,1,2,3,4,5,6,7,8,9	i	Constant integer values.

Table 27 GP functions used in the soccer evaluation runs for [175] (see section 2.7.3 on page 65). Other functions tried (but not used in the final runs) included internal state, magnitude and cross-product comparison, angle rotation, boolean operators, move history, etc. *max* is the approximate maximum distance of kicking, set to 35. *k* is a kick-vector, *v* is a move-vector, *i* is an integer, and *b* is a boolean. Vectors (for either kicking or moving) are a pair of floating-point values. *Reproduced from [175]*.

Appendix B

Graphs and tabulated data of the results of the experiments listed in Table 19 on page 182 which were not presented in Chapter 7 are presented in this appendix. Refer to section 7.6 on page 177 for a general commentary on the experiments listed in Table 19.

Exp. 4: Composite Fitness; Base Skills Only; Default Action = Hunt Action 2

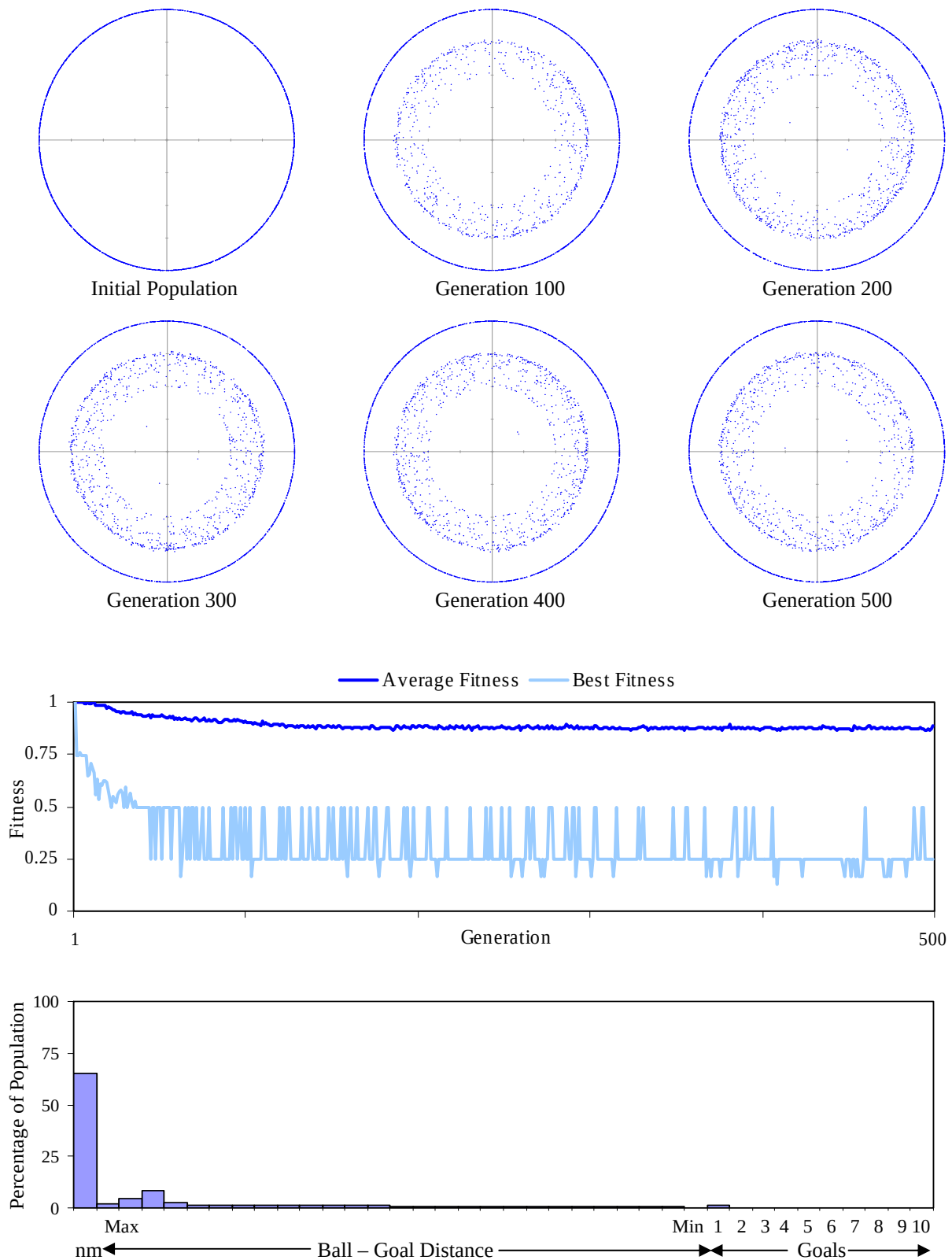


Figure 96 Composite Fitness; Base Skills Only; Default Action = Hunt Action 2

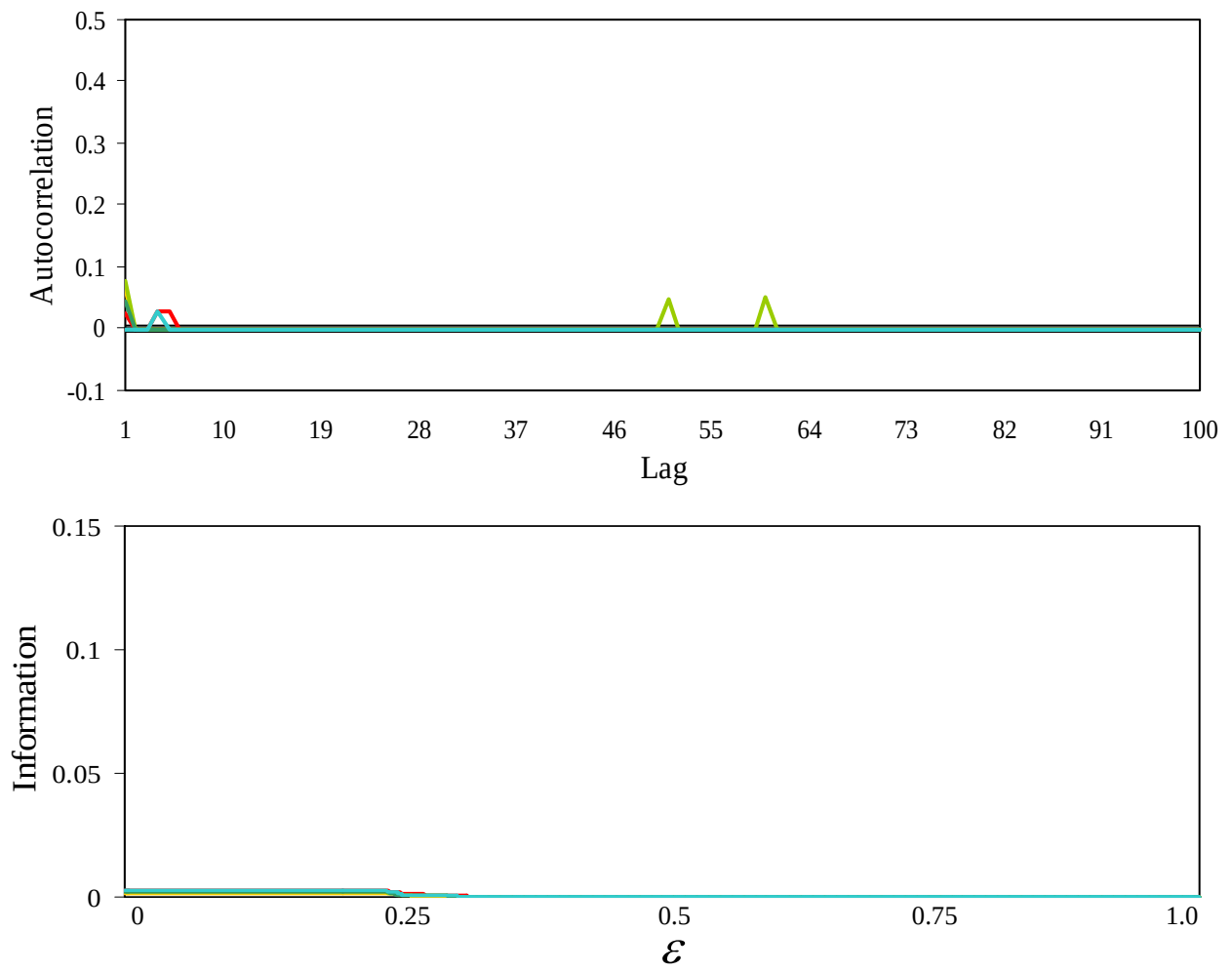
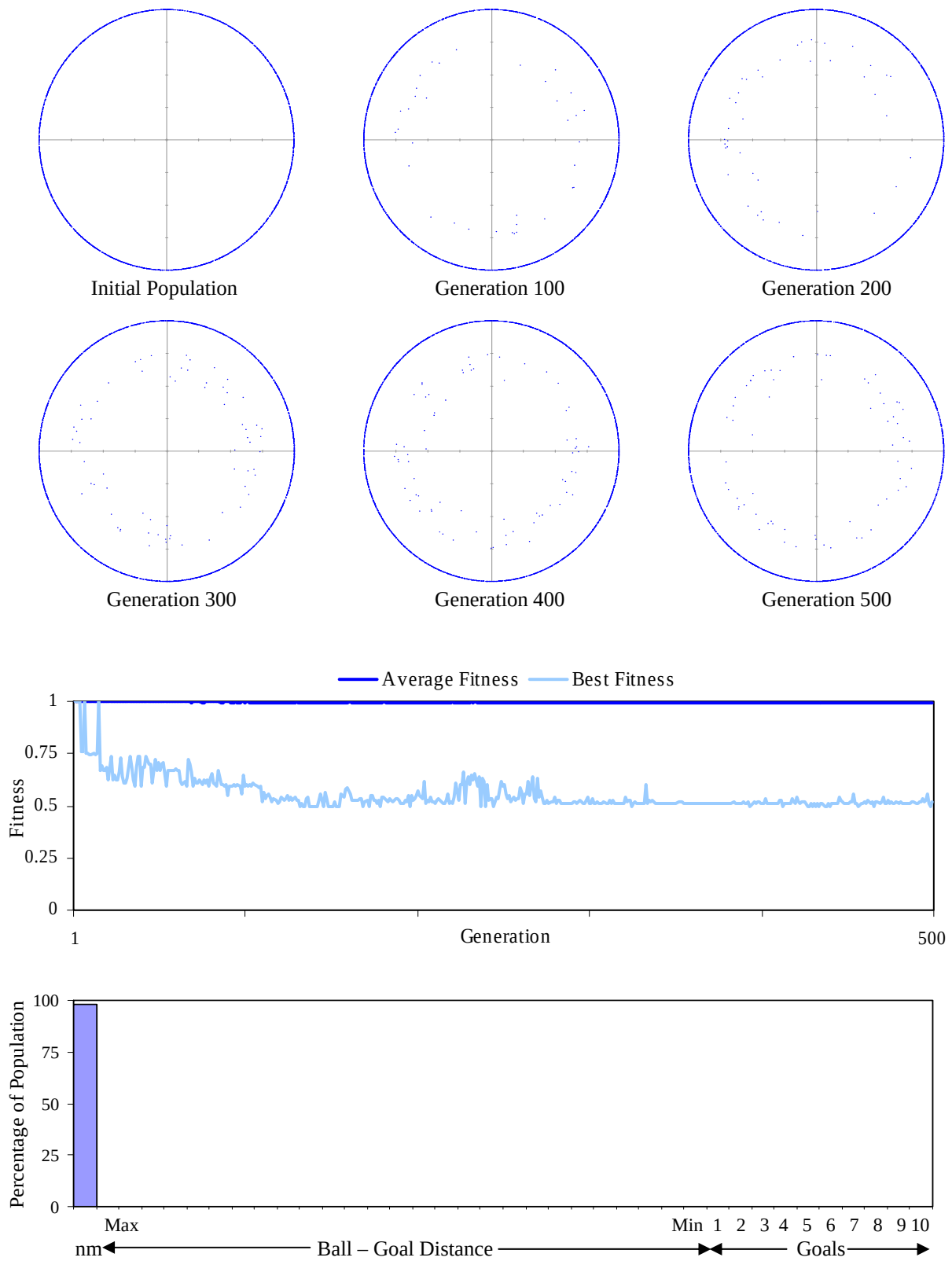


Figure 97 Composite Fitness; Base Skills Only; Default Action = Hunt Action 2

Random Walk Statistics (Average over 5 walks)								
Mean Fitness	0.99994							
Fitness Variance	0.00002							
Individual Fitness	No Movement	Ball Movement	Goals					
			1	2	3	4	5	>5
Number of Individuals	99977	22	1	0	0	0	0	0
Autocorrelation $r(1)$	0.02589							
Correlation Length	1							
$\mathcal{E}$	Information Content $H(\mathcal{E})$		Partial Information Content $M(\mathcal{E})$					
0.0	0.00215		0.00023					
0.1	0.00215		0.00023					
0.2	0.00215		0.00023					
0.3	0.00035		0.00003					
0.36	0.00000		0.00000					

Table 28 Composite Fitness; Base Skills Only; Default Action = Hunt Action 2

Exp. 6: Composite Fitness; Base Skills Only; Default Action = Hunt Action 3**Figure 98** Composite Fitness; Base Skills Only; Default Action = Hunt Action 3

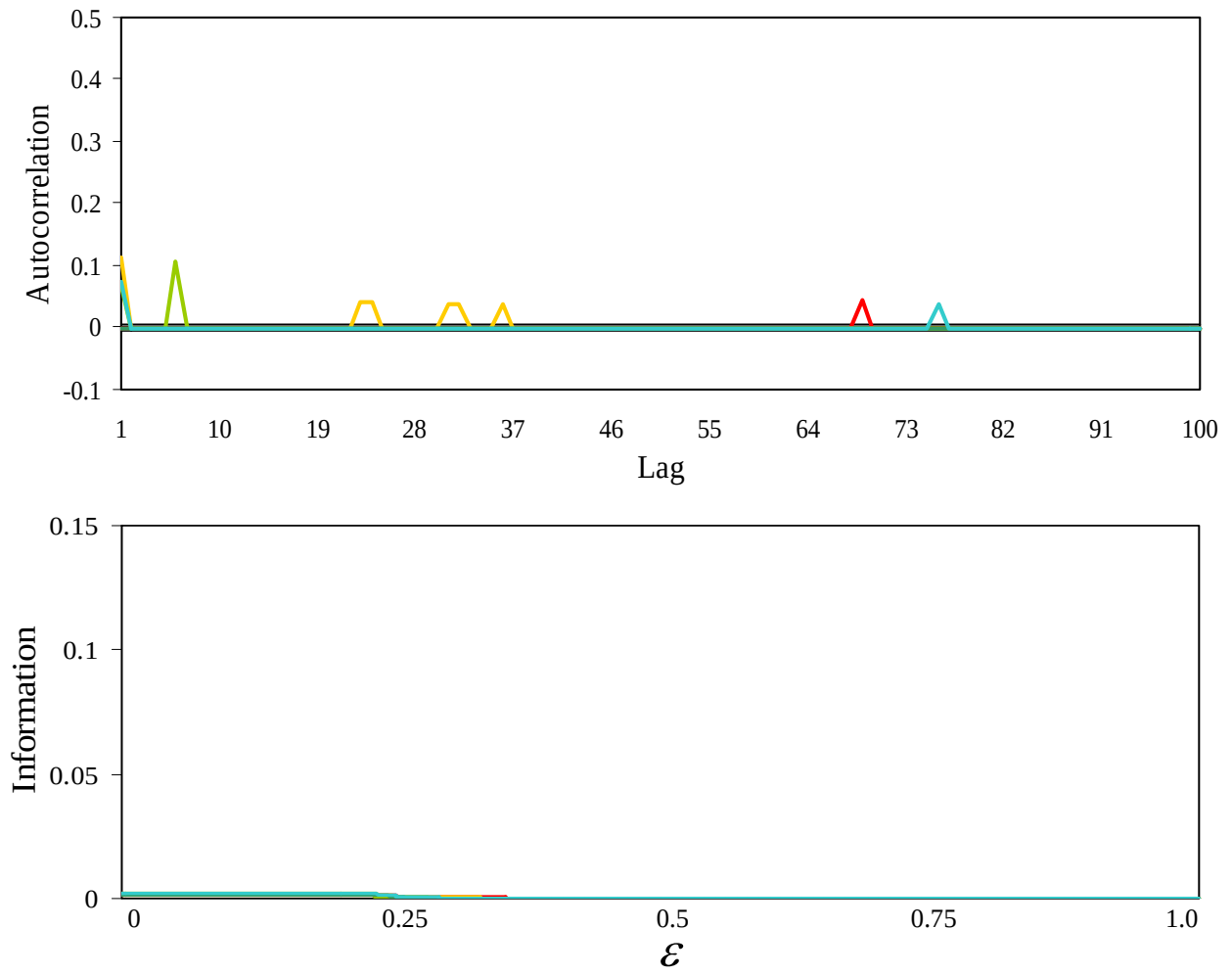
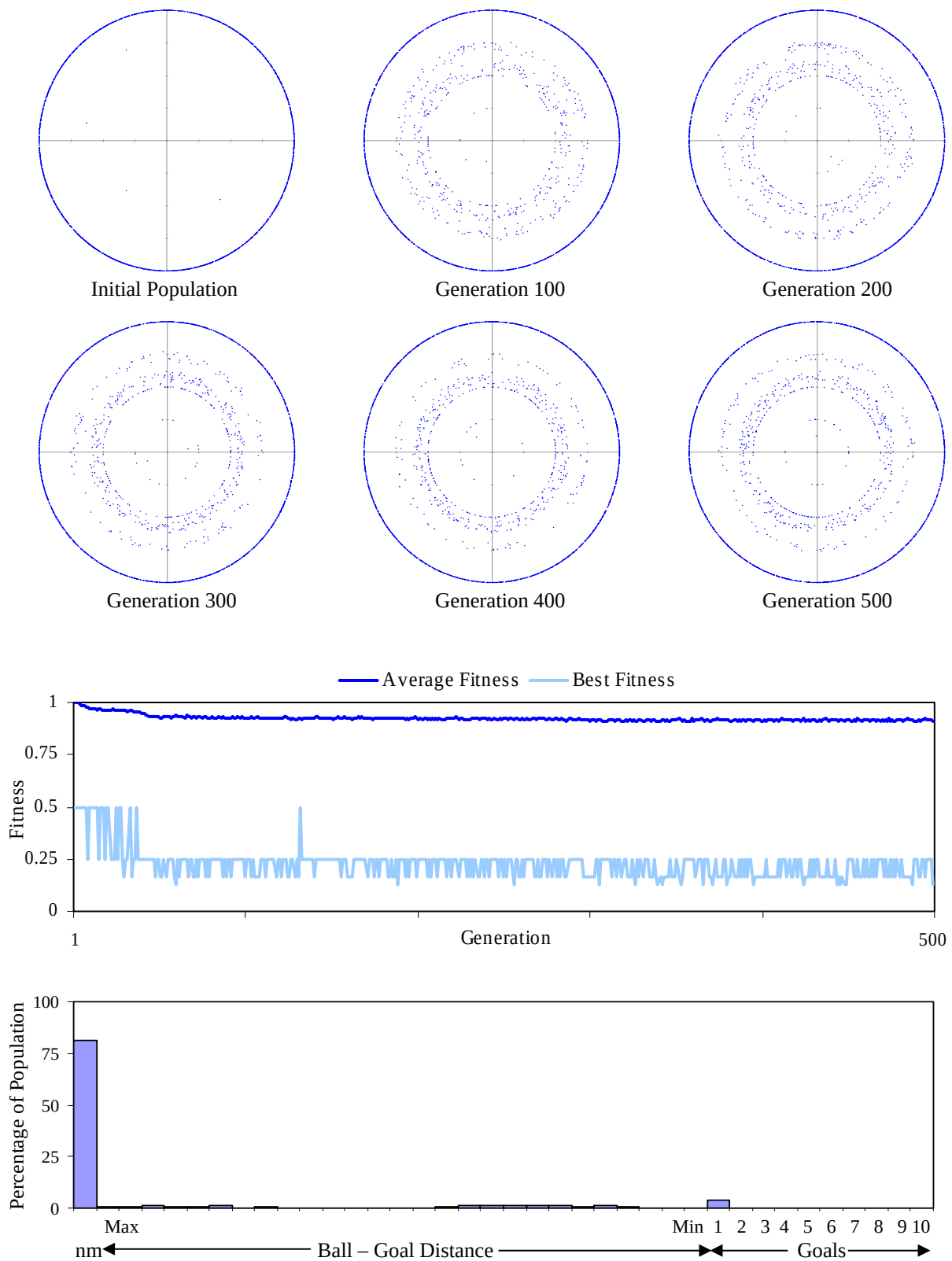


Figure 99 Composite Fitness; Base Skills Only; Default Action = Hunt Action 3

Random Walk Statistics (Average over 5 walks)								
Mean Fitness	0.99995							
Fitness Variance	0.00001							
Individual Fitness	No Movement	Ball Movement	Goals					
			1	2	3	4	5	>5
Number of Individuals	99987	13	0	0	0	0	0	0
Autocorrelation $r(1)$	0.07522							
Correlation Length	1							
$\mathcal{E}$	Information Content $H(\mathcal{E})$		Partial Information Content $M(\mathcal{E})$					
0.0	0.00129		0.00013					
0.1	0.00129		0.00013					
0.2	0.00129		0.00013					
0.3	0.00013		0.00001					
0.34	0.00000		0.00000					

Table 29 Composite Fitness; Base Skills Only; Default Action = Hunt Action 3

Exp. 7: Composite Fitness; All Skills & Directions; No Default Action; Hunt Consequent Allowed**Figure 100** Composite Fitness; All Skills & Directions; No Default Action; Hunt Consequent Allowed

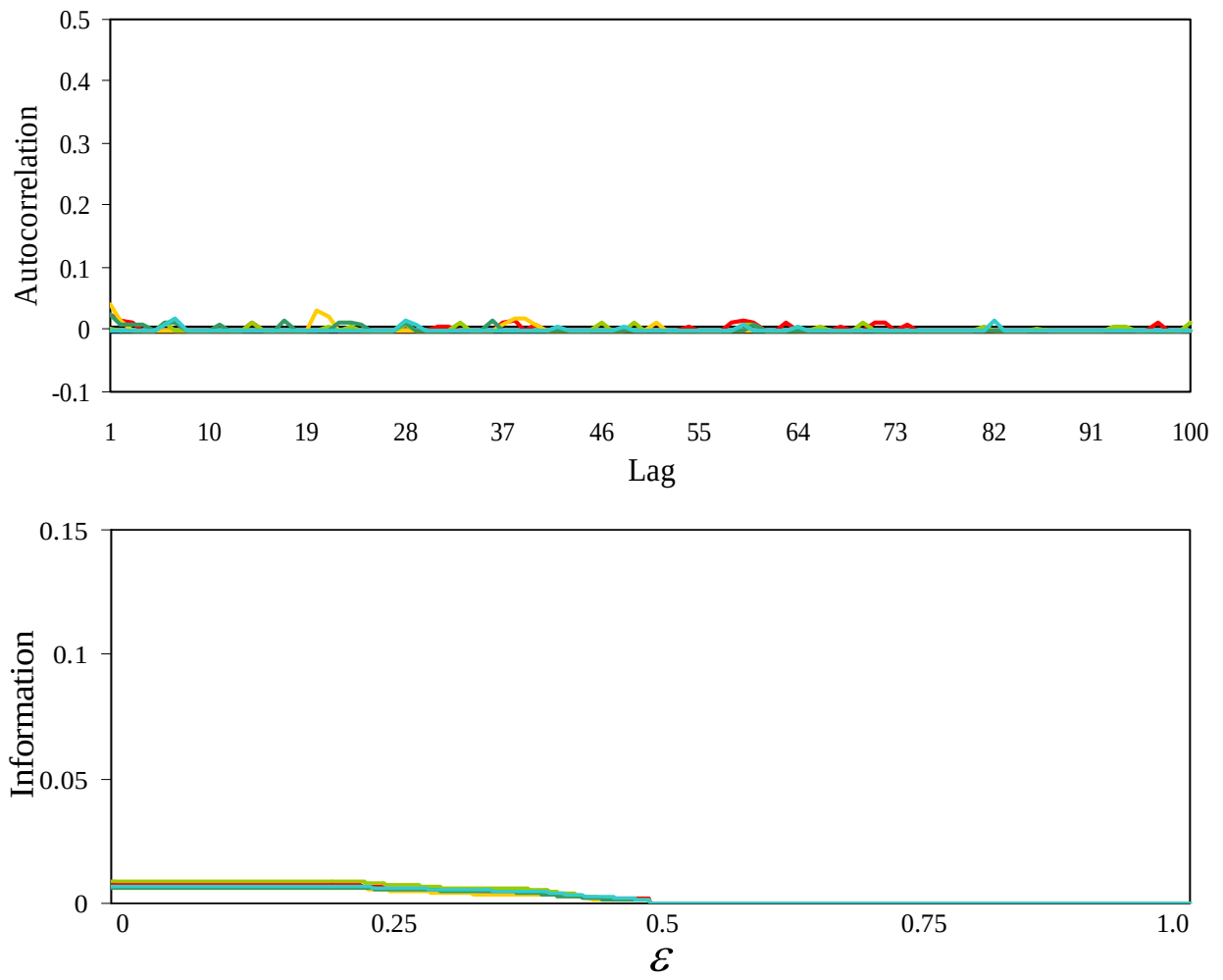
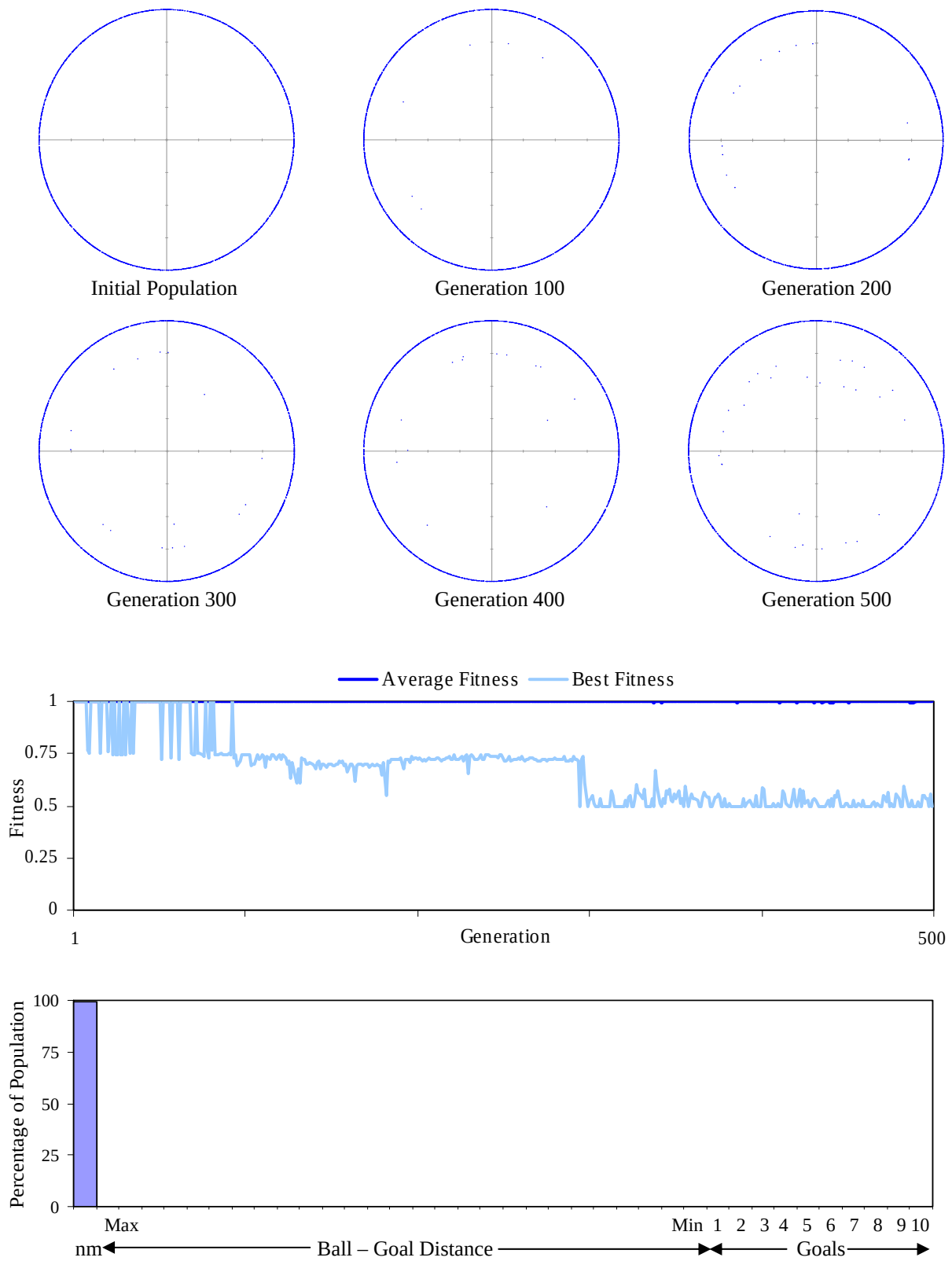


Figure 101 Composite Fitness; All Skills & Directions; No Default Action; Hunt Consequent Allowed

Random Walk Statistics (Average over 5 walks)								
Mean Fitness	0.99969							
Fitness Variance	0.00013							
Individual Fitness	No Movement	Ball Movement	Goals					
			1	2	3	4	5	>5
Number of Individuals	99923	44	33	0	0	0	0	0
Autocorrelation $r(1)$	0.01822							
Correlation Length	3							
$\mathcal{E}$	Information Content $H(\mathcal{E})$		Partial Information Content $M(\mathcal{E})$					
0.0	0.00595		0.00066					
0.1	0.00595		0.00066					
0.2	0.00595		0.00066					
0.3	0.00525		0.00064					
0.4	0.00372		0.00043					
0.5	0.00000		0.00000					

Table 30 Composite Fitness; All Skills & Directions; No Default Action; Hunt Consequent Allowed

Exp. 8: Composite Fitness; Base Skills Only; No Default Action; Hunt Consequent Allowed**Figure 102** Composite Fitness; Base Skills Only; No Default Action; Hunt Consequent Allowed

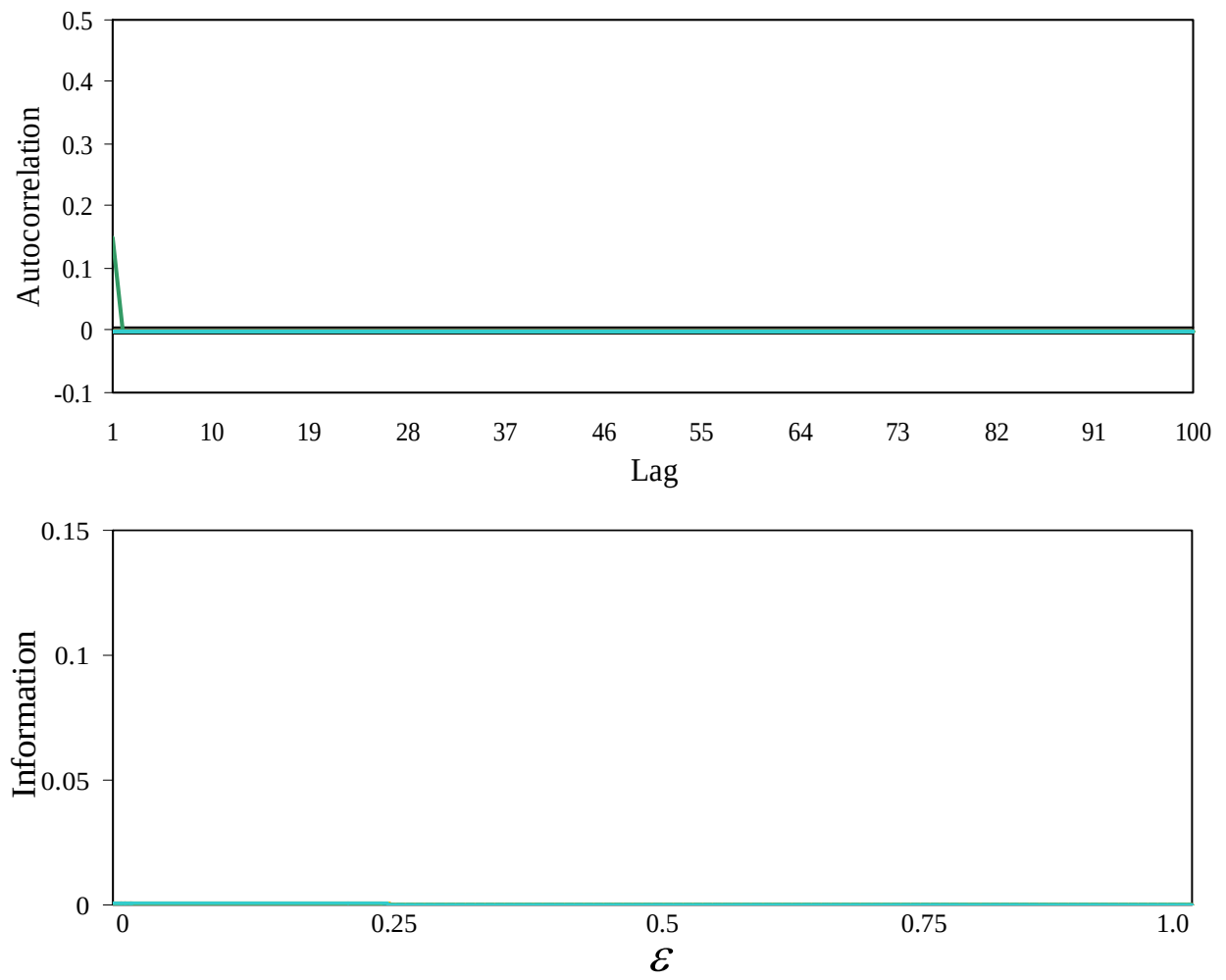
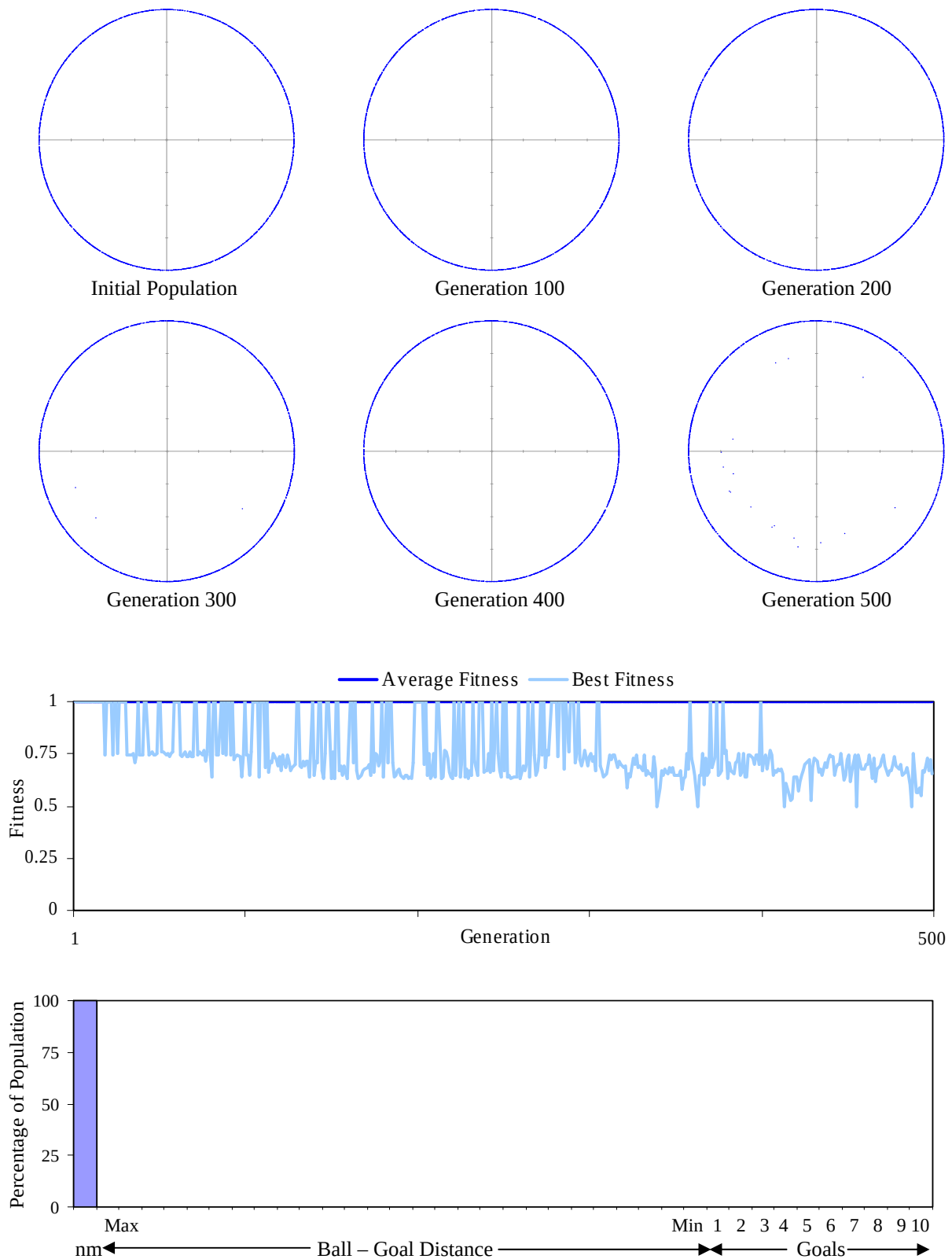


Figure 103 Composite Fitness; Base Skills Only; No Default Action; Hunt Consequent Allowed

Random Walk Statistics (Average over 5 walks)								
Mean Fitness	0.99998							
Fitness Variance	0.00001							
Individual Fitness	No Movement	Ball Movement	Goals					
			1	2	3	4	5	>5
Number of Individuals	99993	7	0	0	0	0	0	0
Autocorrelation $r(1)$	0.02981							
Correlation Length	1							
$\mathcal{E}$	Information Content $H(\mathcal{E})$		Partial Information Content $M(\mathcal{E})$					
0.0	0.00065		0.00006					
0.1	0.00065		0.00006					
0.2	0.00065		0.00006					
0.29	0.00000		0.00000					

Table 31 Composite Fitness; Base Skills Only; No Default Action; Hunt Consequent Allowed

Exp. 10: Composite Fitness; Base Skills Only; No Default Action; No Hunt Consequent



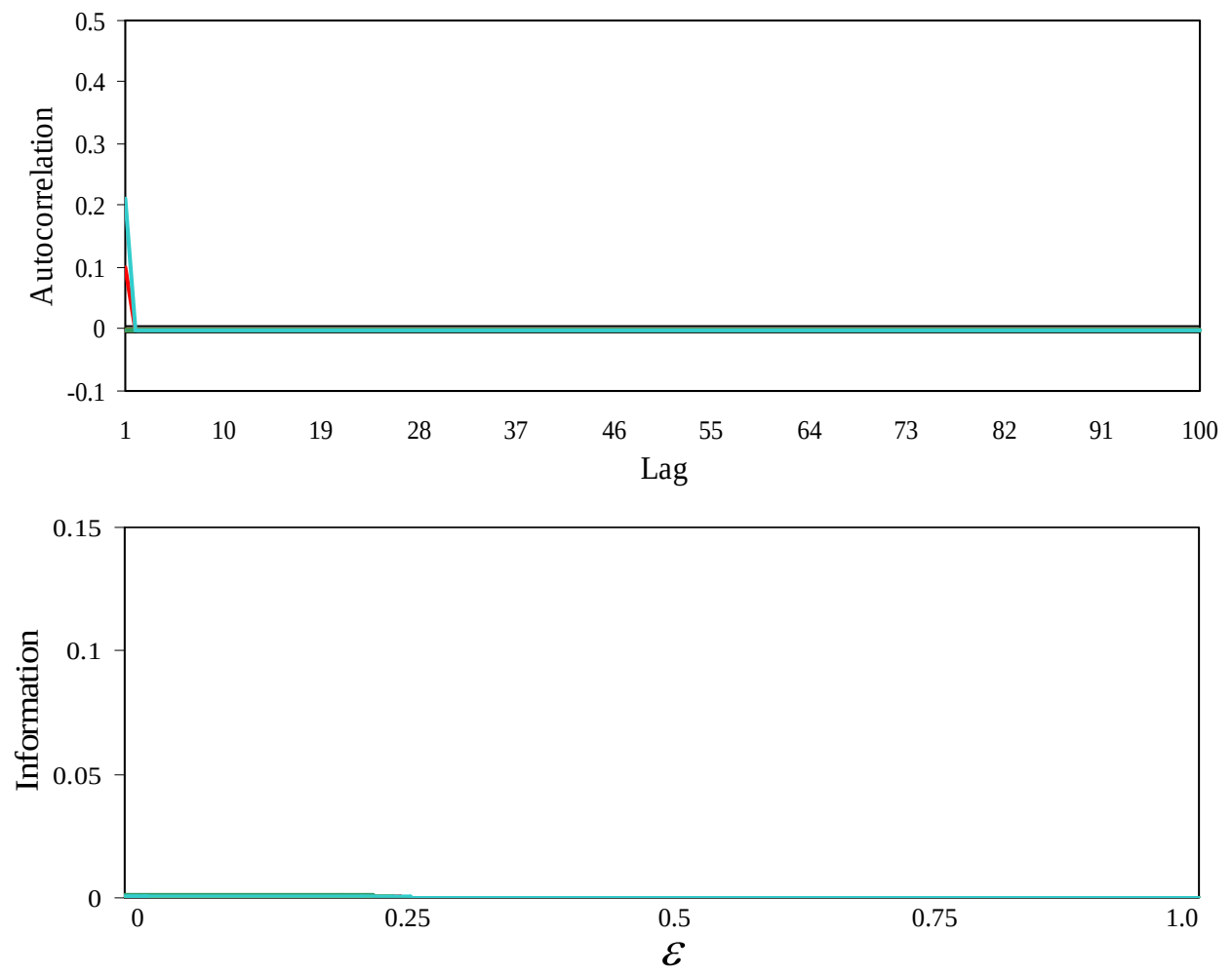
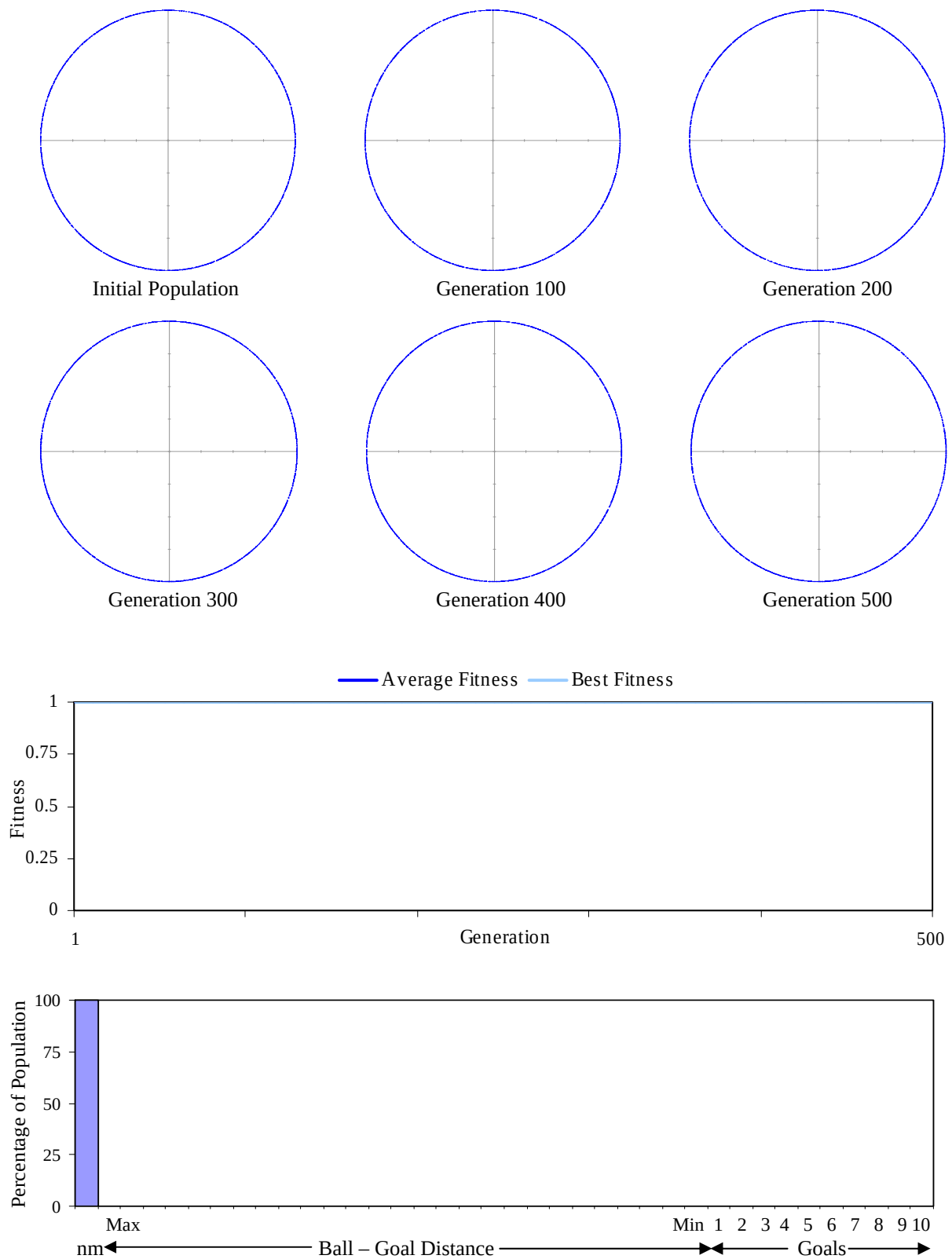


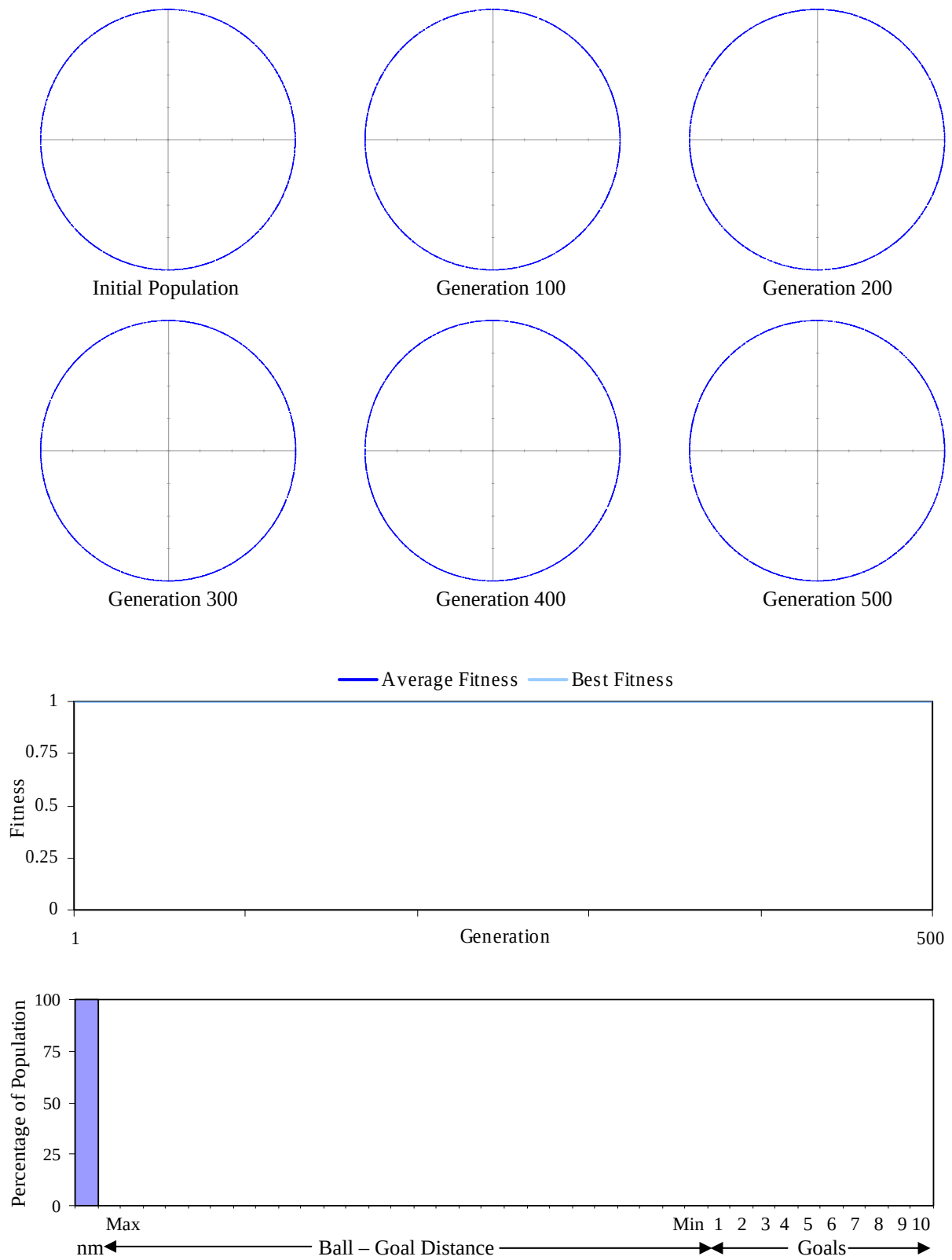
Figure 105 Composite Fitness; Base Skills Only; No Default Action; No Hunt Consequent

Random Walk Statistics (<i>Average over 5 walks</i>)								
Mean Fitness	0.99999							
Fitness Variance	0.00000							
Individual Fitness	No Movement	Ball Movement	Goals					
			1	2	3	4	5	>5
Number of Individuals	99995	5	0	0	0	0	0	0
Autocorrelation $r(1)$	0.06229							
Correlation Length	1							
$\mathcal{E}$	Information Content $H(\mathcal{E})$		Partial Information Content $M(\mathcal{E})$					
0.0	0.00055		0.00005					
0.1	0.00055		0.00005					
0.2	0.00055		0.00005					
0.29	0.00000		0.00000					

Table 32 Composite Fitness; Base Skills Only; No Default Action; No Hunt Consequent

Exp. 13: Goals-only Fitness; Base Skills Only; No Default Action; Hunt Consequent Allowed**Figure 106** Goals-only Fitness; Base Skills Only; No Default Action; Hunt Consequent Allowed

No individual evaluated on the random walk for this experiment exhibited goal-scoring behaviour, and since the fitness function implemented for this experiment is the goals-only fitness function the landscape described by the random walk is completely flat. Since there is no useful landscape information available no landscape graphs or tabulated data are presented.

Exp. 14: Goals-only Fitness; Base Skills Only; No Default Action; No Hunt Consequent**Figure 107** Goals-only Fitness; Base Skills Only; No Default Action; No Hunt Consequent

No individual evaluated on the random walk for this experiment exhibited goal-scoring behaviour, and since the fitness function implemented for this experiment is the goals-only fitness function the landscape described by the random walk is completely flat. Since there is no useful landscape information available no landscape graphs or tabulated data are presented.