University of Exeter
Department of Computer Science

Evolution of Robotic Behaviour Using Gene Expression Programming

Jonathan Mwaura

December 2011

Submitted by Jonathan Mwaura, to the University of Exeter as a thesis for the degree of Doctor of Philosophy in Computer Science , December 2011.

I certify that all material in this thesis which is not my own work has been identified and that no material has previously been submitted and approved for the award of a degree by this or any other University.

(signature) .....

Publications

Some of the material in chapter 2 has been published in:

> Mwaura, J. and Keedwell, E. (2009). Adative Gene Expression Programming Using a Simple Feedback Heuristic In *Proceedings of the Artificial Intelligence and Simulation of Behaviour(AISB2009)*, Endiburgh, April 2009.

Some of the material in chapter 3 has been published in:

> Mwaura, J. and Keedwell, E. (2010). Evolution of Robotic Behaviours Using Gene Expression Programming. In *Proceedings of the Congress on Evolutionary Computation (CEC2010)*, Barcelona, 19th-23rd July 2010.

Some of the material in chapter 4 has been published in:

> Mwaura, J. and Keedwell, E. (2011). Evolving modularity in robot behaviour using gene expression programming. *Proceedings of: Towards Autonomous Robotic Systems - 12th Annual Conference (TAROS 2011)*, Sheffield, UK, August 31 - September 2, 2011.

Acknowledgements

I would like to thank my supervisor, Dr. Ed Keedwell, for his help and support throughout my studies. I highly appreciate your precise criticism, timely advise and a genuine interest in my welfare throughout my studies as well as during my stay in England.

I would like to acknowledge my mother, Elizabeth Njeri Mwaura (deceased), for her conviction and insistence that education would lead to a bright future. You were right on the point. Similarly, I would like to thank Mr. John Gathinji Mucheru (my primary school teacher) and Mrs. Ann Njuguna (high school teacher) for it is only through them that I was able to access both high school and university education. Today I am educated because of their goodwill, support, encouragement and self sacrifice.

I would like to offer my gratitudes to Dr. Abel Nyamapfene and Dr. Richard Fredlund for their encouragement and friendship as well as helping with the proof reading.

I would also like to thank my siblings, Esther, Patrick and Margaret Mwaura for their unwavering love, friendship and encouragement. Special gratitudes goes particularly to Esther Mwaura for her humour and optimism. It has kept me going.

I would also like to offer my gratitudes to Susan Munene-Karuthui for her friendship, encouragements and unrelenting support before and during my research work.

Finally, I would like to offer my heartfelt gratitudes to Miss. Tutti Kandukira for her friendship, continuous encouragement and for listening to my ideas even when she did not exactly understand my research.

Abstract

The main objective in automatic robot controller development is to devise mechanisms whereby robot controllers can be developed with less reliance on human developers. One such mechanism is the use of evolutionary algorithms (EAs) to automatically develop robot controllers and occasionally, robot morphology. This area of research is referred to as evolutionary robotics (ER). Through the use of evolutionary techniques such as genetic algorithms (GAs) and genetic programming (GP), ER has shown to be a promising approach through which robust robot controllers can be developed

The standard ER techniques use monolithic evolution to evolve robot behaviour: monolithic evolution involves the use of one chromosome to code for an entire target behaviour. In complex problems, monolithic evolution has been shown to suffer from bootstrap problems; that is, a lack of improvement in fitness due to randomness in the solution set [103, 105, 100, 90]. Thus, approaches to dividing the tasks, such that the main behaviours emerge from the interaction of these simple tasks with the robot environment have been devised. These techniques include the subsumption architecture in behaviour based robotics, incremental learning and more recently the layered learning approach [55, 103, 56, 105, 136, 95]. These new techniques enable ER to develop complex controllers for autonomous robots.

Work presented in this thesis extends the field of evolutionary robotics by introducing Gene Expression Programming (GEP) to the ER field. GEP is a newly developed evolutionary algorithm akin to GA and GP, which has shown great promise in optimisation problems. The presented research shows through experimentation that the unique formulation of GEP genes is sufficient for robot controller representation and development. The obtained results show that GEP is a plausible technique for ER problems. Additionally, it is shown that controllers evolved using GEP algorithm are able to adapt when introduced to new environments.

Further, the capabilities of GEP chromosomes to code for more than one gene have been utilised to show that GEP can be used to evolve manually sub-divided robot behaviours. Additionally, this thesis extends the GEP algorithm by proposing two new evolutionary techniques named multigenic GEP with Linker Evolution (mgGEP-LE) and multigenic GEP with a Regulator Gene (mgGEP-RG). The results obtained from the proposed algorithms show that the new techniques can be used to automatically evolve modularity in robot behaviour. This ability to automate the process of behaviour sub-division and optimisation in a modular chromosome is unique to the GEP formulations discussed, and is an important advance in the development of machines that are able to evolve stratified behavioural architectures with little human intervention.

Contents

1 Introduction **15**

1.1 Research questions, aims, and claims . . . 17

1.1.1 Research questions . . . 17

1.1.2 Aims . . . 17

1.1.3 Claims . . . 18

1.2 Thesis overview . . . 18

2 Autonomous Robotics: Design and Control **20**

2.1 Robot control system . . . 21

2.2 Robot control approaches . . . 22

2.2.1 Hierarchical/Deliberative paradigm . . . 22

2.2.2 Reactive paradigm . . . 23

2.2.3 Hybrid deliberative/reactive paradigm . . . 26

2.2.4 Learning robot control . . . 29

2.2.5 Evolutionary robotics . . . 31

2.3 Evolutionary algorithms . . . 31

2.3.1 Evolutionary algorithm model . . . 31

2.3.2 Components of evolutionary algorithms . . . 32

2.3.3 Genetic Algorithms . . . 35

2.3.4 Genetic programming . . . 35

2.3.5 Gene expression programming . . . 38

2.4 Evolutionary robotics . . . 47

2.4.1 Evolving neural networks . . . 48

2.4.2 Evolving control programs . . . 52

2.4.3 Evolving intelligent behaviours using GEP . . . 54

2.4.4 Evolution platform . . . 55

2.5 Improving evolutionary robotics . . . 57

2.5.1 Incremental evolution . . . 57

2.5.2 Layered learning . . . 59

2.5.3 Co-evolution . . . 59

2.6 Conclusions . . . 60

3 Evolving Behaviours using GEP **63**

3.1 Obstacle avoidance with straight line navigation . . . 64

3.1.1 Related work . . . 64

3.2 Using GEP to evolve an obstacle avoidance behaviour . . . 65

3.2.1 Robot and environment implementation . . . 66

3.2.2 Algorithm parameters . . . 67
3.2.3 Experimental results . . . 69
3.3 Adaptation to new environments . . . 75
3.3.1 Effect of training and testing controllers in similar environments . . 76
3.3.2 Effect of training controllers in simple environment and testing in a complex environment . . . 77
3.3.3 Testing generalisation using the last population . . . 79
3.4 Sensor based velocity control . . . 81
3.4.1 Experimental results . . . 82
3.5 Evolving monolithic controllers using multigenic GEP . . . 85
3.5.1 Experimental results . . . 85
3.6 Conclusion . . . 94

4 Using Multigenic GEP in Robot Behaviour Sub-division 96
4.1 Modular architectures in ER . . . 97
4.2 Linking functions in multigenic GEP . . . 98
4.3 Evolution of a wall following behaviour . . . 98
4.3.1 Program implementation . . . 100
4.3.2 Fitness function . . . 100
4.3.3 Parameter settings and algorithm parameters . . . 101
4.3.4 Algorithm primitives . . . 102
4.3.5 Experimental results . . . 102
4.3.6 Discussion . . . 108
4.4 Evolving in a 3D world model . . . 109
4.4.1 Experimental set up . . . 109
4.4.2 Fitness function . . . 111
4.4.3 Experimental results . . . 112
4.5 Effect of behaviour coordination mechanism . . . 116
4.5.1 Experimental set up . . . 117
4.5.2 Experimental results . . . 119
4.5.3 Discussion . . . 122
4.6 Conclusion . . . 125

5 Evolving Modularity in Robot Behaviour 127
5.1 Multigenic GEP with Linker Evolution . . . 128
5.1.1 Motivation . . . 129
5.1.2 Experimental set up . . . 129
5.1.3 Results and Discussion . . . 131
5.2 Multigenic GEP with a regulatory gene . . . 133
5.2.1 Implementation . . . 134
5.2.2 Experimental set up . . . 135
5.2.3 Results and Discussion . . . 136
5.3 Algorithm performance analysis . . . 140
5.3.1 Performance comparison . . . 141
5.3.2 Statistical significance . . . 146

5.3.3 Solution convergence . . . 149
5.3.4 Controller performance . . . 152
5.3.5 Mutation effect . . . 164
5.3.6 Discussion . . . 166
5.4 Evolving in 3D environments . . . 167
5.4.1 Experimental set up . . . 168
5.4.2 Results and Discussion . . . 169
5.5 Conclusion . . . 173

6 Conclusions and Further Work **175**
6.1 GEP for automatic development of robot controllers . . . 175
6.2 GEP for automatic development of modular robot controllers . . . 176
6.3 Further work . . . 179
6.3.1 Simulator to on-board evolution . . . 179
6.3.2 RoboCup . . . 179
6.3.3 Complex robots . . . 180

Bibliography **181**

List of Tables

3.1 Terminal set and sensor positions 66
3.2 General algorithm parameter settings 68
3.3 Adaptation test: algorithm parameter settings 76
3.4 Phenotype lengths in the last generation controllers 80
3.5 Statistical comparison of the performance in the training and test environment 80
3.6 Sensor based velocity control: Parameter settings 82
3.7 mgGEP-multiple out: Parameter settings 86
3.8 Mann-Whitney U test between mgGEP-multiple out and ugGEP performances 93

4.1 Success predicate 101
4.2 Parameter settings 101
4.3 Comparison of average evaluations of the GP, ugGEP and mgGEP 103
4.4 3D experiments: Algorithm parameter settings 111
4.5 Robot foraging behaviour: Algorithm parameter settings 119

5.1 mgGEP-LE: Algorithm parameter settings 129
5.2 mgGEP-RG: Algorithm parameter settings 135
5.3 Algorithm parameter settings 141
5.4 mgGEP vs ugGEP 146
5.5 mgGEP-LE vs ugGEP 147
5.6 mgGEP-RG vs ugGEP 147
5.7 mgGEP vs mgGEP-LE 147
5.8 mgGEP vs mgGEP-RG 147
5.9 mgGEP-RG vs mgGEP-LE 148
5.10 Analysing the linking set 159
5.11 Mutation effect on GEP algorithms 164
5.12 Algorithm parameter settings 169

List of Figures

2.1 Hierachical model 23
2.2 Reactive model 24
2.3 Hybrid deliberative/reactive model 27
2.4 Example of uniform crossover 34
2.5 GP representation of a problem 36
2.6 Flowchart of a GEP algorithm 39
2.7 GEP chromosome, phenotype/expression tree (ET) and resulting coding and non-coding regions 41
2.8 sub-ETs in a multigenic chromosome 42
2.9 Expression Tree (ET) for a multigenic GEP 42
2.10 Biological structure of a Gene Regulation Network. 43
2.11 An example of IS Transposition. The | marks the end of head region. 45
2.12 An example of RIS Transposition. The | marks the end of head region. . . . 45
2.13 An example of Gene Transposition. The | marks the end of head region. . . 46
2.14 Evolutionary robotics methodology 48

3.1 Training and test environments 67
3.2 Example of a potential robot avoidance controller using terminals and functions as reported on Table 3.1. 68
3.3 Progression of the median fitness of the best individual and the median of the population mean fitness over the generations 70
3.4 Evolved obstacle avoidance controller with parameters listed on Table 3.2. The controller was evolved in environment test 2 as shown in Figure 3.1 . . 70
3.5 Success rate with varying chromosome length. 72
3.6 Variation of success rate with number of generations. A comparison using different chromosome lengths is shown 73
3.7 Progression of best individuals in the population over the number of generations using different chromosome lengths. 74
3.8 A comparison of the average population mean fitness across different chromosome lengths. 75
3.9 Comparison of frequencies with different phenotypical lengths when controllers are tested in similar environments 77
3.10 Comparison of frequencies of different controller lengths when tested in complex environment 78
3.11 Performance of all the final generation controllers in the 20 GEP runs in training environment and the resultant performance in test environment . . 79

3.12 Progression of the average mean fitness in the population as achieved using ugGEP and ugGEP with sensor based velocity control. 83
3.13 Obstacle avoidance controller evolved using ugGEP with sensor based velocity control. 84
3.14 Comparison of performance of best individuals in new environments. 84
3.16 Comparison of success rate over generations as achieved using ugGEP, ugGEP with sensor based velocity control and with multiple output gene. . 86
3.15 Comparison of success rates achieved using ugGEP, ugGEP with sensor based velocity control and with mgGEP multiple out. 87
3.17 Progression of best individual in the population as achieved using ugGEP, ugGEP with sensor based velocity control and with multiple output gene in environment test 2. 88
3.18 Progression of the mean population mean fitness as achieved using ugGEP, ugGEP with sensor based velocity control and mgGEP with multiple output gene in environment test 2. 89
3.19 Progression of best individual in the population as achieved using ugGEP and mgGEP with multiple output. 90
3.20 Mean of the population mean fitness in the population as achieved using ugGEP, ugGEP with sensor based velocity control and mgGEP with multiple output. 91
3.21 Comparison of performance of best individuals in new environments. 92
3.22 Comparison of performance of best individuals in new environments. 93

4.1 A behaviour modularity model 97
4.2 mgGEP behaviour sub-division using 2 genes and a linking function 98
4.3 Five different room types used in the experiment. These room types have been adapted from Lazarus and Hu [78]. 99
4.4 Comparison of ugGEP and mgGEP success rates across the five different room types. 104
4.5 Progression of the average fitness of the best individual in the population as achieved through ugGEP and mgGEP in the different room types. 105
4.6 Progression of the average population mean fitness as achieved through ugGEP and mgGEP in the different room types 106
4.7 An example of a controller generated using ugGEP while solving room type 4. 107
4.8 Simplified version of figure 4.7 107
4.9 Example of mgGEP controller evolved while solving room type 4 108
4.10 Robot environments as set up in simbad simulator 110
4.11 Comparison of ugGEP and mgGEP success rates across the five different room types. 112
4.12 Progression of success rates with number of generations using ugGEP and mgGEP in the four room types. 113
4.13 Progression of the average fitness of the best individual in the population as achieved through ugGEP and mgGEP in the different room types. 114

4.14 Progression of the average population mean fitness as achieved through ugGEP and mgGEP in the different room types 115
4.15 Progression of standard fitness with number of generations. 116
4.16 Robot world used in the foraging experiments. The round objects represents "food sources" that the robot picks to improve its energy while navigating. 117
4.17 A behaviour coordination model that relies on robot proximity to obstacles in order to choose a robot task. 120
4.18 Progression of the mean of the best individual in the population, and the average of the population mean fitness over generations as achieved through ugGEP and mgGEP (OA priority). 121
4.19 A behaviour coordination model that relies on robots energy level in order to select a robot task. 121
4.20 A behaviour coordination model that uses robots energy to determine behaviour selection. In this case sub-behaviour selection is determined via cascading priorities. 122
4.21 Progression of the mean fitness of the best individual in the population over generations. 123
4.22 Progression of average performance of the population over generations, with different behaviour coordination mechanisms. 124

5.1 Mean fitness of the best individuals along 200 generations for the population of mgGEP-LE and mgGEP chromosomes. 131
5.2 Mean of the population mean fitness as achieved through mgGEP and mgGEP-LE in the different room types. 132
5.3 mgGEP-RG implementation model. The output of the regulatory gene is a symbol that is then used for decision making 134
5.4 Comparison of success rates achieved in the four different room types using standard mgGEP, mgGEP-LE and mgGEP-RG algorithms. 136
5.5 Comparison of best individual performances in the population as achieved through standard mgGEP and mgGEP-RG in the different room types. . . 137
5.6 Comparison of progression of average fitness of the population as achieved through mgGEP and mgGEP-RG in the different room types. 138
5.7 an example of a controller evolved using mgGEP-RG 139
5.8 Comparison of success rates achieved using ugGEP, standard mgGEP, mgGEP-LE and mgGEP-RG algorithms. 142
5.9 Progression of the mean fitness of the best individual in the population as achieved through ugGEP, standard mgGEP, mgGEP-LE and mgGEP-RG in the different room types. 143
5.10 Progression of the mean of the population mean fitness as achieved through ugGEP, mgGEP, mgGEP-LE and mgGEP-RG in the different room types. 144
5.11 Progression of the median of the population mean fitness as achieved through ugGEP, mgGEP, mgGEP-LE and mgGEP-RG in the different room types. 145

5.12 P values generated using mann whitney two tailed test. The values are generated from 50 observations in each generation. Each curve shows the comparison with a different algorithms. A summary of all the parameters used in this experiment is shown on Table 5.3. 149
5.13 Q1, Q2 and Q3 values of the best individual in the population as achieved by ugGEP and mgGEP algorithms, plotted every generation. 150
5.14 Q1, Q2 and Q3 values of the best individual in the population as achieved by mgGEP and mgGEP-LE algorithms, plotted every generation 151
5.15 Q1, Q2 and Q3 values of the best individual in the population as achieved by mgGEP and mgGEP-RG algorithms, plotted every generation 152
5.16 An example of a multi-controller evolved using standard mgGEP 153
5.17 An example of a multi-controller evolved using mgGEP-LE. 155
5.18 A multi-controller evolved using mgGEP-LE 156
5.19 An evolved mgGEP-LE controller that replicates the behaviour of a controller evolved using mgGEP 157
5.20 Example of a multi-controller evolved using mgGEP-LE 158
5.21 An example of a multi-controller evolved using mgGEP-RG. 160
5.22 Example of a novel multi-controller evolved using mgGEP-RG 161
5.23 A strategy used by mgGEP-RG to solve wall following problem in room 5. . 162
5.24 A novel mgGEP-RG controller evolved to solve wall following problem in room 5 163
5.25 Mutation effect on ugGEP, mgGEP, mgGEP-LE and mgGEP-RG individuals 165
5.26 Average performance of the best individual in the population across rooms 2-5 evolved using mgGEP, mgGEP-LE and mgGEP-RG. 170
5.27 Average performance of the best individual in the population evolved using mgGEP-RG and different variations of mgGEP-LE algorithm. 171
5.28 Comparison of progression of the fitness of the best individual fitness in the population as achieved through mgGEP, mgGEP-RG and mgGEP-LE in the different room types. 172
5.29 Progression of the best individual in the population as achieved through mgGEP, mgGEP(L) and mgGEP-RG in room 2 and 4. 173
5.30 Progression of success rates with number of generations using mgGEP, mgGEP(L) and mgGEP-RG. 173

Acronyms

Acronym	Meaning
AI	Artificial Intelligence
ADF	Automatically Defined Functions
ANN	Artificial Neural Network
ARL	Adaptive Representation through Learning
AuRA	Autonomous Robot Architecture
BBR	Behaviour Based Robotics
BPGEP	Backtracking Parallel Gene Expression Programming
CGP	Cartesian Genetic Programming
CTRNN	Continuous Time Recurrent Neural Network
EA	Evolutionary Algorithms
EANN	Evolving Artificial Neural Networks
EC	Evolutionary Computation
EP	Evolutionary Programming
ER	Evolutionary Robotics
ES	Evolution Strategies
ET	Expression Tree
FFNN	Feed Forward Neural Network
FLC	Fuzzy Logic Control
GA	Genetic Algorithms
GEP	Gene Expression Programming
GP	Genetic Programming
GRNN	Global Recurrent Neural Network
HGP	Hierarchical Genetic Programming
IAS	Intelligent Autonomous Systems
IFLTE	If Less Than or Equal to
IS	Insertion Sequence Transposition
LGP	Linear Genetic Programming
LISP	LISt Processing
LRNN	Local Recurrent Neural Network
MA	Module Acquisition
MEP	Multi Expression Programming
mgGEP	Multigenic Gene Expression Programming
mgGEP (L)	Multigenic Gene Expression Programming (checking left sensor)
mgGEP - multiple out	Multigenic Gene Expression Programming with multiple output

Acronym	Meaning
mgGEP (OA Priority)	Multigenic Gene Expression Programming (with Obstacle Avoidance as Priority behaviour)
mgGEP-LE	Multigenic Gene Expression Programming with Linker Evolution
mgGEP-RG	Multigenic Gene Expression Programming with Regulator Gene
ORF	Open Reading Frame
RIS	Root Insertion Sequence
SFX	Sensor Fusion Effects
S-R	Stimulus-Response
TGP	Traceless Genetic Programming
ugGEP	Unigenic Gene Expression Programming

1 Introduction

Intelligent autonomous systems (IAS)[1] are designed "to do the right thing at the right time"[2] as well as to exhibit "intelligent" behaviour. Autonomous robots are used to assist human beings in dangerous tasks such as detecting and removing land mines, risk assessments in such areas as nuclear plants, performing domestic tasks such as vacuum cleaning as well as helping elderly persons. These tasks require the robots not only to be autonomous but also good decision makers. There are a lot of studies that have been carried out on how to develop these robots [74, 105]. The research is normally two fold: (a) development of robot morphology or body [84] and (b) development of the robot controller [105, 104]. The design and development of robot controllers involves the implementation of control routines that support flexible task execution and effective action planning. Hand-coding these control routines takes a lot of programmers' time and cannot be guaranteed to generate a robust control system that is able to help the robot adapt easily when confronted by new circumstances in the environment.

To solve the problems related with manual hand-coding of robot control systems, there have been various mechanisms to develop robot control systems automatically. Chief among these techniques is the nascent field of evolutionary robotics (ER). Briefly, ER is a technique to automate the design of robot controllers and sometimes morphology using evolutionary computation techniques. Evolutionary computation (EC) is a branch of artificial intelligence that involves the use of neo-Darwinian evolution techniques such as selection, mutation and crossover to solve optimisation problems. EC techniques are largely referred to as evolutionary algorithms (EAs) and include evolution strategies (ES), evolutionary programming (EP), genetic algorithms (GA), genetic programming (GP) and gene expression programming (GEP). In this thesis, GA, GP and GEP algorithms are described.

There are two common approaches used to evolve robot controllers, the first is to optimise a neural-based controller including the weights, learning rates and architecture using GAs [105], and secondly, the evolution of symbolic computer programs using GP [75]. The two approaches have shown great versatility in generating robot controllers for such tasks as obstacle avoidance, maze exploration, food foraging and homing behaviours. The most common evolutionary approach used in ER is monolithic evolution; that is, evolution of robot controllers using one gene or module to represent the entire target behaviour. For instance, in a robotic task such as foraging, only one gene is used to evolve the entire behaviour complete with its constituent tasks such as obstacle avoidance, exploration and food location [105, 56, 60]. Although this evolutionary approach has been utilised to evolve

[1] a glossary of acronyms can be found on page 13
[2] http://www.ias.uwe.ac.uk/

robot controllers with satisfactory results, the approach may not evolve optimum solutions when presented with complicated problems. This in turn may lead to scaling up problems in evolutionary robotics, that is situations where the evolved controllers do not have sufficient capability to solve the problem [105, 49, 136, 37]. There are various mechanisms that have been proposed to counter scaling up problems associated with monolithic evolution. Among these techniques is incremental evolution [105, 5], where the targeted behaviour is subdivided by the experimenter and then solved sequentially. The underlying problem with this mechanism is that once a controller is evolved, there is no way of distinguishing which part of the controller solves the problem. Also, the mechanism requires numerous changes of fitness and reconfiguration of the problem set up. Another mechanism that has been proposed is layered learning [117, 61], in this technique, the targeted behaviour is divided into simpler tasks and each task solved until a targeted fitness is achieved. Once this is done, the next sub task is evolved and so on. This technique is better than incremental evolution because each particular task has its own sub-controller and therefore it is easy to distinguish what part of the controller solves which task. Additionally this mechanism means that the controller is modular and therefore replicates the well known subsumption architecture [136]. However layered learning requires the use of multiple objective fitness which means that it is computationally expensive. Moreover, since learning on each module is stopped when a particular fitness is achieved, the robot is highly likely to have a hindered performance when new situations are encountered.

This thesis introduces GEP to the field of evolutionary robotics, by presenting a study on evolving robot controllers using the GEP algorithm. GEP is a new and more biologically plausible EA that incorporates the simplicity of GAs as well as the capability of GP [38, 93]. The main difference between GEP and its predecessors, GA and GP, resides in how the individuals are represented. In GEP the individuals are encoded as linear strings of fixed length (similar to GAs) which are later translated to non-linear entities of different sizes and shape (similar to GP individuals). The initial linear fixed length structure is known as the genotype and the ramified structure, derived from the translation of genotype is known as the phenotype. GEP is therefore said to be a genotype/phenotype algorithm. Another difference between GEP and its predecessors is in the genetic operations used. Similarly to GA and GP, GEP uses mutation and recombination for genetic operation. However, GEP incorporates additional genetic operations; gene recombination, insertion sequence transposition, root insertion transposition and gene transposition. The use of more genetic operators than GA and GP is likely to create a population of highly diverse individuals which in turn, may lead to reduced local minima problems. Another important aspect which this thesis addresses in detail is that GEP chromosomes can be composed of more than one gene of equal length; these chromosomes are referred to as multigenic GEP (mgGEP) chromosomes. These multigenic chromosomes can be utilised in two different ways. Firstly, in problems with multiple outputs, each gene or sub-expression tree (after gene translation) evolves its respective output. Secondly, in problems that require one output, the various genes interact with one another forming a more complex chromosome or controller. For instance, in robot problems involving more than one task, each task can be evolved using one gene and a special linking function used to enable interaction within the different tasks. This means that GEP chromosomes can be utilised to evolve modular

controllers similar to subsumption architecture. Since these modular structures are part of a whole chromosome, there is a concurrent evolution of different modules of behaviour, leading to robust controllers.

This thesis utilises GEP algorithm to evolve robot controllers using both monolithic and modular approaches. The thesis shows that GEP is not only a suitable algorithm for evolving mechanisms for robot control but also it is more biologically plausible than both GAs and GP. Also, this thesis extends the current implementation of multigenic GEP by proposing two more bio-inspired multigenic GEP variants to be used for evolution of modular controllers. From our investigation, this is the first time that GEP has been used in evolving robotic controllers. The thesis is an attempt to address the research questions and claims as discussed in the next section.

1.1 Research questions, aims, and claims

This section outlines the research questions, aims and claims that drive this thesis.

1.1.1 Research questions

- How can GEP be best utilised to evolve robot controllers?
- What are the effects of using controllers, evolved using GEP, in new environments?
- How can the linking functions used in multiple gene chromosomes in GEP be utilised to provide behaviour modularity?
- What are the possibilities of using GEP technique to generate more biologically plausible methods for ER problems?

1.1.2 Aims

- To explore the capability of GEP in evolving robot controllers. In particular to investigate whether GEP is a viable EA technique to be used in ER.
- To investigate the robustness of controllers evolved using GEP, when tested in new environments.
- To investigate the effect of manual behaviour sub-division using multiple genes evolved using GEP. In addition, to investigate whether a particular sub-behaviour position in the model affects how the whole controller solves a problem.
- To investigate whether multiple gene chromosomes implemented using GEP can be used to automatically evolve behaviour modularity. In particular, to study the effect of evolving the behaviour coordination mechanism or linking function in GEP and also to study the effect of using a regulatory gene to determine sub-behaviour activation and inhibition.

1.1.3 Claims

- GEP, a more biologically plausible EA technique than GA and GP, is a suitable mechanism for evolving robust robot control programs. In addition, GEP is the only EA technique which can be implemented using more than one gene. These multiple genes can be utilised to provide controllers in robotic problems requiring multiple output as well as provide mechanisms to evolve modular architectures.
- Modular robot control architectures implemented using GEP, can be used to develop better robot controllers than those evolved using a monolithic GEP implementation.
- Linking functions used in multiple gene chromosomes in GEP can be used as behaviour coordinators in modular robot behaviour implementations. In addition, this linking functions can be evolved in essence providing automatic behaviour subdivision. Moreover, one of the genes in the chromosome can be utilised as a regulatory gene to provide gene inhibition and activation control within the chromosome, similar to the process in natural biology.

1.2 Thesis overview

Chapter 2 presents the necessary background in robotics control. In this chapter, various control methods are reviewed starting from classical artificial intelligence (classical AI) control methods, behaviour based control techniques and hybrid control. The discussion then proceeds to adaptive control methods such as fuzzy control and robot learning, followed by a brief review of evolutionary robotics. The survey then proceeds to evolutionary computation algorithms where genetic algorithms (GA) and genetic programming (GP) are briefly discussed. This is then followed by a detailed review of gene expression programming (GEP); an evolutionary computation method that is preceded by both GA and GP. Techniques to evolve robot controllers are then discussed starting with evolution of artificial neural network based controls and evolution of computer programs. Mechanisms to improve the basic evolutionary robotics methodology and various platforms for evolution of robot behaviours are then commented upon.

Chapter 3 presents mechanism to evolve a robot behaviour using GEP. The first section of this chapter is largely based on [94]. However, the rest of the chapter explores new mechanism to evolve robot behaviour. A robot obstacle avoidance behaviour is presented and results discussed. The aim of the chapter is to show the capabilities of GEP in evolving robot behaviour and the abilities of evolved behaviours to adapt when introduced to new environments. Using the same examples, a mechanism to evolve a multiple output controller is discussed and results presented. Through this new technique, it is shown that multiple output controllers offer more capability in adaptation when compared to standard monolithic structures developed using GA and GP.

Modular control architecture such as subsumption architecture, incremental evolution and layered learning have been shown to outperform monolithic controllers [55, 105, 136]. Work

presented in Chapter 4 utilises multigenic GEP structures to evolve modular controllers for a robot. The obtained results show that GEP chromosomes that utilise multiple genes (mgGEP) can be used to evolve a global modular behaviour. This work is an extension of work presented by the author in [95]. In the presented experiments, each sub-behaviour is evolved using one gene and the overall structure is linked together using a pre-selected linking function. In this chapter, a wall following behaviour is implemented and the results show that mgGEP controllers are more robust than unigenic GEP (ugGEP) structures. Additionally, it is shown through a robot foraging behaviour that how the linking function is implemented to activate or inhibit execution of any of the sub-behaviours (genes) is of critical importance as it affects the overall performance of the evolved control program.

Chapter 5 extends the work carried out in Chapter 4 by investigating a mechanism that can be used to automatically regulate which gene/sub-behaviour is to be activated or inhibited during the control process. This chapter, introduces two main methods. Firstly, the standard multigenic GEP algorithm is modified by including an extra parameter set, that is, a linking set. In standard multigenic GEP a linker is normally specified *a priori*, however, in the proposed new mechanism, a linking function set (Linking Set) is provided together with the function and terminal sets. Therefore, individuals in a population are linked up using different linking functions. The linking functions also go through the evolution mechanism and as thus this new mechanism is named "multigenic gene expression with linker evolution" (mgGEP-LE). The results achieved using mgGEP-LE are compared to results from standard mgGEP and a conclusion drawn. Secondly, we explore new mechanisms where the sub-behaviour selection is determined by use of an extra gene. The GEP mechanism, as the name implies, is not only based on evolution but also the process of gene translation. Following this, we use the gene analogy to evolve a control structure where a regulatory gene is used to activate or inhibit the execution of the rest of the genes in the chromosome. This mechanism is named "multigenic gene expression programming with regulatory gene" (mgGEP-RG). This is a novel technique as behaviours evolved are as a result of not just the evolution mechanism but also the regulatory gene whose role is to activate and inhibit control. The evolution of this type of controller can be used to solve the problem of behaviour coordination that affects such modular structures as subsumption architecture and layered learning techniques [92, 142].

Finally, Chapter 6 discusses the contribution of the thesis to the field of robot control, discusses future work that can be carried out using standard GEP and the newly proposed mechanisms.

2 Autonomous Robotics: Design and Control

A robotic system is a machine that senses its environment by using sensors, "behaves autonomously" by processing received signals from the sensors to determine its next action and lastly acts by use of its actuators [92, 30]. Robot actuators are devices that provide a mechanism to activate process control equipment in a robot. The most common robot actuators are electrical motors. It is the work of actuator devices to convert energy to actions or behaviours by using robot effectors such as robotic arms, robot wheels or legs in case of a humanoid robot. Robot sensors come in different varieties depending on the intended function; infra red sensors, sonars and stereo vision are used for range detection, robot bumpers and touch sensors make the robot aware that it has come into contact with an external body, such as obstacles, while cameras provide visual information about the state of the environment. An autonomous system refers to a system that is capable of operating in the real world without any external control over a length of time [92, 124]. Biological organisms are good examples of autonomous systems. They live and adapt in dynamic environments, maintain their internal structures, automatically repair when damaged, gather information regarding their environment and exhibit behaviours that make them suitable to live in the environment they are in. These behaviours include locomotion, foraging, mating and escaping from predators.

Autonomous robots are machines that can perform desired tasks in unstructured and sometimes chaotic environments without explicit human control [49]. Many types of robots have some degree of autonomy and different robots can be autonomous in different ways. A fully autonomous robot must have the ability to gather information about its environment by the use of sensors, be able to process received signals in order to make decisions, be entirely mobile or have some of its part mobile (in the case of a robotic arm) and should be able to avoid situations that can harm people or itself.

Robot autonomy is particularly desirable in fields such as military applications [73, 114]. Examples of operations where the military has utilised robotics include transportation of military supplies, ammunition and war casualties [1]. In addition, robots have also been used by the military in warfare[2]. In nuclear technologies, autonomous robots are

[1] Examples of robots used for this purpose include autonomous platform demonstrator (APD) which is an unmanned ground vehicle. Details can be found on; `http://www.rec.ri.cmu.edu/projects/apd/`, iRobot's Warrior that can carry up to 70kg payloads , Squad Mission Support Systems(SMSS) which is an unmanned ground vehicle,details can be found on; `http://www.time.com/time/nation/article/0,8599,1884169,00.html`

[2] Armed robots are robots that are used to fire missiles, for instance, the General Atomics MQ-1 Predator which is an unmanned aerial vehicle used primarily by the US army to fire Hellfire anti-armor missile. Details can be found on `http://www.airforce-technology.com/projects/predator/`

desirable due to the risks involved if exposing personnel to the environment; an example is the iRobot's Warrior and packbots that are being used in the nuclear emergency in Fukushima in Japan[3]. Autonomous aerial vehicles have also been used increasingly for space exploration. These robots include planetary flybot probes such as voyager 1 and 2 [4]. In health care, autonomous robots have been used for transportation, for instance Speciminder and RoboCourier [5] are used to transport payloads of up to 50 pounds in hospitals. Additionally, semi autonomous robots are being used to assist surgeons in surgical procedures [140, 35]. Autonomous robots are also used in agriculture [12], search and rescue [16], home [129], mining [145, 129] and waste management. An important area of robotics research is to enable an autonomous robot to adapt to its environment as well as learn new capabilities like adjusting strategies for accomplishing its task(s) or adapting to changing surroundings. For any particular robot to accomplish any of the tasks discussed here, a robot control systems is required. The design and development of robot controllers involves implementation of control routines that support flexible task execution and effective action planning.

2.1 Robot control system

Robot control refers to the process of enabling a robot to make decisions, given some inputs. The robot control system or controller refers to the robots "brain" or the component that handles robot control. The controller could be a pre-programmed hardware microcontroller processor which is used to store information about the robot and its environment and to store and execute programs which operate the robot. Alternatively, it could be software programs that determine what the robot does when presented with different scenarios in its environment.

The control system contains programs, data algorithms, logic analysis and other processing activities which enable the robot to perform the required tasks. Additionally, the control system requires a set of rules to be able to analyse presented data in terms of signals received from sensors and internal feedback structures. The result generated from the processor directs the actuators to carry out some action.

To control robots to perform behaviours autonomously a number of tasks have to be addressed. These include:

[a] Selection of robot movement control mechanisms; e.g. kinematics, dynamics and odometer calculations.

[3]The Japanese earthquake and nuclear emergency started on 11th March 2011, where a great level of damage occurred and three nuclear reactors at the Fukushima nuclear plant were damaged. The effects of the reactors meltdown led to high risks of radiation exposure. Robots are thus being used in the cleaning up exercise and to investigate radiation levels inside the reactors `http://spectrum.ieee.org/static/japans-earthquake-and-nuclear-emergency`

[4]Voyager 1 and 2 are spacecraft sent to space by NASA in 1977 to explore the solar system, the initial mission was to Jupiter and Saturn but the mission has now been extended to cover Sun's outermost edge. There is also hope that they may be picked by intelligent extraterrestrials and communicate information about earth. Further reading can be found on `http://voyager.jpl.nasa.gov/mission/`

[5]Speciminder and RoboCourier are developed by swisslog health solutions (`http://www.swisslog.com/`)

[b] Selection of robot sensors; whether active or passive proximity sensors.

[c] Selection of robot's low-level control of actuators; this will help determine the action a robot executes in a given scenario.

[d] Selection of robot control architectures; this could be either traditional planning architectures, behaviour-based control architectures or hybrid architectures.

Autonomous robots are programmed to understand their environment and take independent action based on the knowledge they possess. Some autonomous robots may have memory and are able to "learn" from their past encounters. This means they can identify a situation, process actions which have produced successful/unsuccessful results and modify their behaviour to optimize success. This activity takes place in the robot controller. Due to the unstructured nature of the environments in which autonomous robots are employed, the robot controllers cannot always be programmed to execute predefined actions because it is hard to predict in advance the changes that might occur in the environment and the required robot sensorimotor transformations. In addition, the environment might contain dynamic characteristics that require the robot to modify its behaviour in order to fulfil the required task. Moreover, robot systems in dynamic environments have to deal with :

[a] Sensor noise and uncertainty; sensor readings could be imprecise and unreliable.

[b] Environment uncertainty; various aspects of the environment cannot be observed. Also the environment is initially unknown to the robot or may have certain dynamic characteristics that may require rapid real time modifications of the robot behaviour.

[c] Action uncertainty; actions taken could fail and also actions may have a non-deterministic outcome.

It is for this reason that researchers in robotics have been looking at different novel methods to prepare controllers for autonomous robots. The next section will describe different approaches to robotic control.

2.2 Robot control approaches

This section highlights different approaches that are utilised to control an autonomous robot.

2.2.1 Hierarchical/Deliberative paradigm

Minsky [88] argued that when faced by a problem, an intelligent agent would build an abstract model of its world. Once the model is built, the agent would explore solutions within the internal model of the environment and finally attempt a solution in the real world when the best strategy is devised. This is now what is referred to as a classical artificial intelligence (AI) approach or the deliberative control architecture. In this

paradigm, the robot operates in a top-down fashion with a lot of emphasis on planning. The paradigm is geared towards replicating human-like intelligence in solving a problem [92]. In this approach, the robot senses the environment and creates an internal world model, next, the robot reasons about the consequence of various actions to be taken within the environment. Once the best strategy is formulated, the robot executes it. Thus, the control process goes through a sequence of sensing (SENSE), model update, planning steps (PLAN) and execution (ACT). The key components in this paradigm are functions: i.e. perception, learning and planning. The Hierarchical paradigm,as shown by figure 2.1, has a horizontal decomposition with SENSE, PLAN and ACTION as distinctive components enabling robot action.

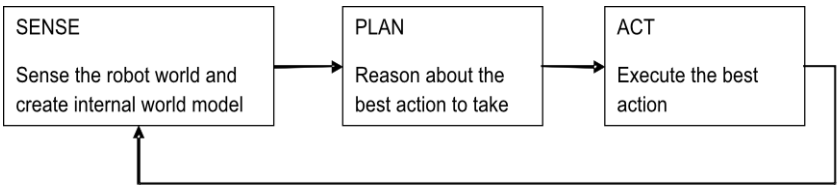


Figure 2.1: Hierachical model

There are various advantages associated with this approach. Firstly, the intelligent agent reasons about contingencies, that is, in every step the robot checks the internal world model and then uses the inbuilt control routine to make a decision on what action to execute. Therefore, the robot is unlikely to execute destructive actions such as high speed collisions. Secondly, the robot computes solutions to a particular task hence the solutions arrived at are the best achievable in the solution pool. Finally, this approach is good at achieving goal directed strategies, thus, it is successful in robot planning and search oriented techniques. The approach however has the following drawbacks: The solutions tend to be fragile in the presence of uncertainty, requires frequent re-planning, react relatively slowly to changes and unexpected occurrences and finally the fusing of all sensing into a global map supplemented with a knowledge base means that expert knowledge of the operating environment needs to be known *a priori*. Also this approach suffers from the frame problem (i.e. the problem of finding an accurate representation form that is able to ensure that an unstructured world is sufficiently represented)[118]. Thus, the approach is not sufficiently robust to address rapid changes to the environment as well as to model an unstructured environment where an autonomous robot needs to operate. For a comprehensive review of hierarchical paradigm, see [118, 92]

2.2.2 Reactive paradigm

The major drawbacks related to the hierarchical paradigm as discussed above, were slow reaction speed due to re-planning after every step and weak solutions in the face of uncertainty. It is impossible to create a knowledge base or an inference engine covering all angles in a non-structured environment. Due to this, there was a need to approach the problem of implementing robotic behaviours and investigating artificial intelligence differently. Reactive behaviour arose to address the problems with classical AI, particularly

the reliance on representation and symbols. Brooks [17] argued that the representation was the wrong building block particularly when implementing a large intelligent system. Brooks also noted that explicit representations and the building of a world model was prohibitive when examining simple level intelligence. The world, according to Brooks, is its own best model. In the reactive paradigm, the artificial agent senses the world through its sensors and reacts immediately using its motors. There is therefore no need for the planning phase as the agent does not create an internal model. Figure 2.2 shows the model of the reactive paradigm.

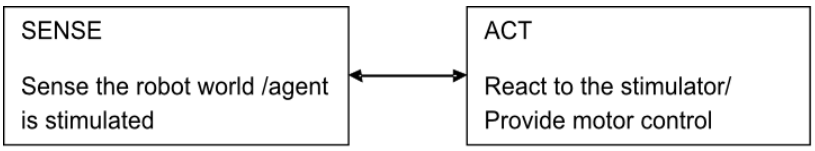


Figure 2.2: Reactive model

The reactive paradigm draws its inspiration from biological intelligence, particularly from the field of ethology (study of animal behaviour). In this approach, behaviour is the fundamental building block for intelligence, this is different from classical AI where functions are the basis for intelligence. Due to the use of behaviour as basis for intelligence, the reactive paradigm is also known as behaviour based robotics (BBR).

What is a behaviour?

A behaviour can be described as a mapping of sensory inputs to a set of motor activities which are used to accomplish a particular task [92]. The schema theory defines a behaviour as a schema composed of both motor and perceptual schemas [92]. The perceptual schema represent the template for sensing while the motor schema represents the template for motor activity. Thus in BBR, the "PLAN" component is removed. Behaviours in animals can be divided into three categories;

[a] Reflexive behaviours - These are stimulus-response behaviours (S-R). In this case the stimulus (an incentive that evokes action) is directly connected to the response. For example, if something is directed towards the eye, the eye lid closes immediately.

[b] Reactive behaviours - These are learned behaviours that are then committed to memory e.g. changing a gear while driving, riding a bicycle among others.

[c] Conscious behaviours - These are deliberate, planned behaviours e.g. planning a journey.

In BBR, the concentration is on reflexive behaviours where a stimulus is effected by the sensors and the response enacted by the motors. Reflexive behaviours are further divided into: (a) Reflexes: in this category, the response time is directly proportional to the intensity or the duration of the stimulus, for instance, a robot could turn right when the front sensor faces an obstacle. (b) Taxes: in this case, the response is to move towards

a particular orientation such as, move towards the light or if hungry (stimulus) look for food (response). (c) Fixed-action patterns: in this case the response time is longer than intensity of stimulus, for example, when a robot is faced by a certain situation such as a low battery and fleeing from predators. These categories of reflexive behaviours can be implemented on one robot consecutively [92]; that is, all the categories of the reflexive behaviours can be implemented on one autonomous robot. As can be seen in this section, in BBR the term reactive behaviour is used to mean animal reflexive behaviours.

In BBR, the robot is provided with a repertoire of simple behaviours. The behaviours are implemented in a vertical decomposition with lower layers/modules being the primitive survival behaviours (e.g. charge battery, evade obstacles etc.) and higher layers as goal accomplishing behaviours e.g. map building. The global behaviour emerges due to the interaction of these simple behaviours with the environment [17, 18]. An interlinking method is used to determine how the repertoire of these simple behaviours form the global behaviour. The BBR approach is a trial and error method with the designer building the set of simple behaviours to achieve a desired emergent behaviour. The designer has to explicitly program rules and design control structures that include knowledge concerning how the behaviour organization is achieved [142, 144]. The behaviours are built step by step with increasing complexity, that is, once a lower behaviour is designed and implemented the designer can then move to a new behaviour until all behaviours have been implemented.

An important issue in BBR and ethology is behaviour coordination or action selection [117, 142]. This refers to the process of determining which behaviour to activate at a particular moment in time. In BBR, methods of behaviour coordination have been categorised based on the method used in behaviour selection. Two major categories have been identified: (a) Arbitration methods; these are competitive based methods where the dominant behaviour affects the motor output. The selection of which behaviour is dominant is determined by the sensor readings and the internal state of the robot at a given time. (b) Cooperative methods; in these methods the different behaviours contribute towards a single motor action. In BBR, various architectures have been suggested in order to deal with behaviour organisation. These architectures include subsumption and potential fields methods [4, 92, 18].

For a comprehensive reading on BBR, please see [4, 92].

Subsumption architecture

The subsumption architecture is an arbitrative method of organising behaviour [142]. In this approach, the robot controller is divided into layers, where each layer is a human designed piece of hardware or software. Each layer works independently with higher layers subsuming and inhibiting the lower layers. Lower layers are used for simpler or default behaviours and the complex behaviours are left for the higher layers [18]. The behavioural modules operate concurrently and independently with potential behaviour conflicts being resolved in a winner-takes all format where the higher layer is always the

winner. A particular task is achieved by selecting the appropriate behaviour which then triggers the lower layers below it. Since the robot does not have to build a world model, the subsumption architecture solves the frame problem in classical AI.

Mataric [83] has used the subsumption architecture to program a controller that helps a robot to follow walls of an irregular room. In this work, the wall following behaviour was divided into four tasks, stroll (wander), avoid (avoiding obstacles), align (align robot to the wall avoiding collision) and correct (if going out of boundary). When the subsumption architecture is employed in BBR as used by Mataric, each behavioural layer is normally programmed by a human designer and embedded directly on the robot processor. Moussi and Madrid [91] have also used the subsumption architecture to enable a robot to evade obstacles and explore the environment while searching for a target.

Potential fields methodology

The Potential Fields approach employs vectors to represent behaviours or motor action [92]. Behaviour coordination is achieved by a cooperative method where the vector summation is used to combine different behaviours in order to produce an emergent behaviour. The method is also referred to as motor schemas [4] as the concentration is based on providing a motor schema or action template. A vector is described as a mathematical construct that has a magnitude (size or velocity) and direction. This can be represented as a tuple (m,d) where m is magnitude and d is the direction. A potential field is created using an array of vectors, this is referred to as a region or space. In robotics, the potential field is a 2D world representation (i.e. a grid environment) with each element of the array representing a square in the x,y robotic environment. In the representation, perceived objects in the environment exert a force comparable to a gravitational or magnetic field in the surrounding space. When a robot is placed in the environment, the implemented sensors perceive the environment while the motor action is guided by the force exerted to the robot by the environment. For example, an obstacle placed in the space will repel the robot while a particular goal or taxes may attract the robot.

The reactive/BBR technique has been the dominant technique in designing robot control systems. This has been motivated by fast reaction to changes and absence of requirement for planning and re-planning. Moreover, a world model does not need to be built; the world as posited by Brooks [18] is its best model. Nevertheless, it is also difficult to anticipate in advance what effect combinations of different behaviours will have, it is difficult to create a large repertoire of behaviours to cover all angles in an unstructured environment and there is a requirement for continuous redesign to include control systems for new tasks.

2.2.3 Hybrid deliberative/reactive paradigm

As can be seen above, the reactive paradigm eliminates planning and does not involve reasoning about the state of the robot in relation to the world it acts upon. Although this brings faster response times for the robot and eliminates the high cost of real-time

reasoning, it means that the robot cannot tackle path planning problems, monitor its performance in the environment nor build maps. This, ideally, means that the robot cannot learn or plan as it operates in the environment. The question, raised by Murphy [92] is whether the robot can be made intelligent enough to determine the best behaviour in a given situation and perform behaviour control over time. It is this type of question and the need to bring back planning and deliberation that led to the hybridisation of the deliberative and the reactive paradigms.

The hybrid system incorporates PLAN, then SENSE-ACT sequence in modelling the robot behaviour (see figure 2.3). The planning phase is decoupled from real time execution and involves building a world model and reasoning offline. Once this is done, the robot then turns to the reactor portion which incorporates sense-act as discussed in the reactive paradigm above. The behaviour (SENSE-ACT) would go on until the plan is completed.

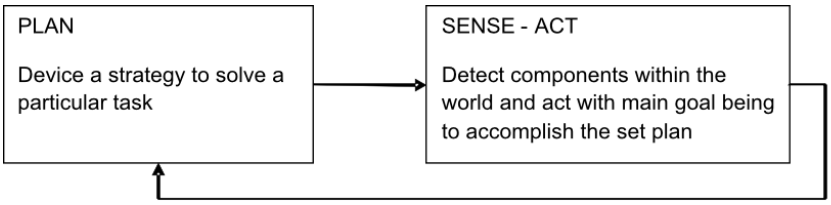


Figure 2.3: Hybrid deliberative/reactive model

The architectural components associated with the hybrid systems are thus; a deliberative layer, a reactive layer and a middle layer [110]. The reactive layer is behaviour based which means that the subsystem consists of a repertoire of behaviours that the robot can accomplish. Each of these behaviours represent a tight coupling from the sensors to motors. Modules in the reactive layers are stateless and they carry out calculations on raw data in near real-time for safety considerations. The deliberative layer handles planning, localization, reasoning and interactions with human operators. Modules in the deliberative layer are; a mission planner, which interacts with the human operator by receiving necessary commands, a cartographer module that is responsible for creating, storing and maintaining spatial information and techniques for accessing data. In addition, the resource manager module allocates resources to behaviours, for instance, if a robot has different range detection sensors, the resource manager may allocate the set of sensors that is sufficient for the tasks. The middle layer, also called the sequencer or supervisory layer, interlinks the deliberative and reactive layers. The sequencer generates the set of behaviours to use in order to accomplish a given task and determines the sequence and activation condition.

The behaviours created in a hybrid system are reflexive, innate and learned. This is more biologically plausible than those in the reactive paradigm which are only reflexive. There are different architectures implemented using the hybrid paradigm, these are divided into three main categories as discussed below:

Managerial architectures

The managerial architectures divides the deliberative part of the hybrid paradigm into layers depending on the level of abstraction required by that layer, for instance, a mission planning layer is less abstract than a path planning layer and hence the mission planning can control the path planning. The division is thus; the high level layers carry out the high level planning, this is passed to intermediate layers that may refine the plan and lastly this is passed to the reactive layers where the actual execution is carried out. This means that deliberative modules occupy the higher layers and the plans made are sent to reactive modules which are at the low level. Each module attempts to carry its tasks, identify the problems and solve them and if it cannot solve the problem, it seeks help from a layer above it, due to this, layers are said to fail upwards. Similarly to the subsumption architecture, a higher layer can only modify the layer below it.

Autonomous Robot Architecture (AuRA) [4] is an example of the managerial style architecture. This architecture was developed by Arkin [3] and is the oldest hybrid architecture. It has a deliberative layer consisting of a planner and a spatial reasoner (cartographer). The middle layer comprises a homoeostatic control, that influences which behaviours to execute by monitoring internal conditions of the robot and reporting to both higher level planning mechanisms and motor schemas [4]. This then leads to dynamic replanning particularly in environment with changing conditions. The reactive layer is composed of sensor and motor sub-systems which generate behaviours. Each behaviour is monitored by the schema controller and is associated with a perceptual schema which provides the stimuli for the behaviour. The Sensor Fusion Effects (SFX) architecture [92] started as an extension of the AuRA. The initial idea was to add modules that would specify how perception, sensor fusion and sensor failures would be handled. However, with time, the architecture has re-organised the deliberative and reactive layers, though the architecture remains closely related to AuRA.

State-Hierarchy architectures

State-hierarchy architectures use the state of a robot in relation to its environment at a particular moment in time in order to differentiate between deliberative and reactive layers. The robot states are divided into three layers: Past, Present and Future. With state hierarchy, just as in managerial style discussed above, the organization is broken down into layers with each layer having functions that carry out the required tasks. In addition, the higher layers can access the output of the lower layers as well as modify them.

An example of the State-Hierarchy architecture is the 3 Tiered Architecture [53]. As the name suggest, this architecture comprises three layers. The deliberative layer utilises past, present and future states to produce a plan for the sequencer or respond to sequencer requests. This layer runs as a separate thread on a separate processor and acts on time consuming computations. The sequencer layer interlinks the deliberative and the reactive layers. This layer utilises the past and present states when interpreting a plan, in order

to select which behaviours to activate and respond to a situation. The layer does not search into future states to decide actions. The reactive layer or the controller depends on the current state to execute the behaviour library presented to it. This layer acts fast, operates in real time and avoids internal states. The State-Hierarchy architecture has been used primarily by NASA for controlling planetary rovers, underwater vehicles and robot assistants for astronauts [92].

Model-Oriented architectures

The State-Hierarchy and managerial architectures are a direct extension of the reactive paradigm and thus operate on a bottom-up structure and emphasise behaviour as the core intelligent building block. The model-oriented architecture on the other hand is driven by the hierarchical paradigm. It has a top-down structure with symbolic manipulation around a global world model. The global world model supplies perception or sensing to the behaviours, this is unlike the State-Hierarchy and managerial structures where a global model is built in parallel.

Hybrid control techniques integrate the advantages of deliberative techniques, e.g. permitting goal based behaviours, and enriches this with reactive advantages such as fast reaction speeds. In addition to this, the complexity in planning is reduced [4, 92]. Nevertheless, the behaviours interactions have to be well modelled, also the choice of modelled behaviours limits what the robot can do. As a result, this technique is prohibited by the fact that it is very hard to model behaviours to cover all situations in an unstructured environment.

2.2.4 Learning robot control

The reactive and hybrid systems discussed above focus on building robust control architectures that address local uncertainties. In new environments or where new and previously unknown tasks needs to be accomplished, these approaches may not be adequate to solve a problem. In ever-changing environments, adaptation is key in order to generate suitable behaviour. Robot learning involves the use of a learning mechanism to acquire and to modify control strategies. In this approach, a robot control system (usually an artificial neural network controller [10, 72]) undergoes training using incomplete data and is then allowed to exploit the learned behaviour in new environments [105]. The controller can learn how to map sensory inputs to motor actions or the learning can be used to form new modules of the controller. There have been numerous machine learning techniques used to achieve this, the most popular of which are described below.

Reinforcement learning

Reinforcement learning is a special case of supervised[6] learning where the exact desired output is unknown. The learning is based on a global evaluation of the network response

[6]Supervised learning is based on direct comparison between the actual output and the desired correct output

after certain input is inserted and the output received [9, 132, 128]. In artificial agent learning, reinforcement learning enables an agent/robot to learn from interacting with its environment. In this framework, an agent uses its sensors to perceive the state of its environment and the motors/actions to change its state and that of its world in order to accomplish a set goal [33]. The robot acquires a reward for accomplishing the goal or a sub goal and is penalized for failure. The robot uses an existing control model that has been extended to include a learning mechanism or a control that integrates a learning mechanism can be built from scratch.

Fuzzy control

Fuzzy control or fuzzy logic control (FLC) is a closed-loop control system based on fuzzy logic: "a mathematical formalization of vague, ambiguous, imprecise, noisy, or missing input information" [111]. The system uses fuzzy logic to create rule-bases. These rules have the premise (IF) and the consequent (Then), for example: *If speed is low Then press the accelerator*. The antecedent (i.e speed is low) describes the state of the system and is used to activate a rule. The consequent (i.e press the accelerator) defines the action to be taken [49]. The values, such as speed in the example above is referred to as a linguistic variable. The uncertainty of the linguistic variable (such as "speed is low") is represented using fuzzy sets, i.e. a set whose boundaries between elements overlap. For instance, there are various speed values that can be defined as low, meaning, different values can be members of more than one fuzzy set. The membership of a value, such as speed, to a fuzzy set can also be defined by establishing a membership function, which is a function that maps a particular linguistic variable to a particular fuzzy set(s). A fuzzy control system is comprised of four components [111, 113]. Firstly, a rule-base or a knowledge base, this supplies the rest of the components with necessary information such as rules and membership functions. Secondly, an inference engine which is a decision making component. Thirdly, a fuzzification interface that converts controller inputs to fuzzy sets. Finally, a defuzzification interface that "interprets" the fuzzy sets to a format understandable by the process.

FLC has been used to develop a controller for a wall following and obstacle avoidance mobile robot [113]. Additionally, Floreano et al. [49] have used a fuzzy system to train a robotic neural network controller to learn the tasks of obstacle avoidance and wall following. In Gu and Hu [58] and Gu et al. [59], a genetic algorithm was used to optimise membership parameters for a fuzzy logic controller. Their approach was then tested in the robot football domain, where ball chasing and position reaching behaviours were implemented.

A major constraint with fuzzy control is choosing rules and membership functions. In addition, a database for the fuzzy set and an inference engine (decision making component) needs to be implemented [49].

2.2.5 Evolutionary robotics

Evolutionary robotics (ER) is a technique that employs evolutionary algorithms (EAs) to automatically create control systems for autonomous robots and occasionally the physical robot system (i.e. morphology) [84]. These controllers are either neural based with optimisation achieved through the use of an evolutionary algorithm (EA) or symbolic programs evolved using a range of evolutionary approaches [105, 142, 60, 143]. There have been numerous robotic behaviours that have been investigated using ER, these include photo axis [26], locomotion and obstacle avoidance [105], soccer playing [81], wall following [78], homing behaviour [45], path planning [71, 125] among others. Programming or evolving one program requires time and careful planning in order to ensure all interlinking function or tasks work correctly so as to generate the correct behaviour.

ER involves the automatic creation of the robot controllers and it could also be used as a learning mechanism to train robot control systems. This offers a great advantage to the designer as the main requirement is to set the required parameters describing the attributes of required behaviour and designing a fitness function that guides the process of evolution. The use of a population of controllers means that this system offers parallel development which could save time, i.e. a population of controllers means that a lot of behaviours are developed simultaneously and checked for the most suitable one. This is a better approach than behaviour based system where a trial and error method of designing a system, then testing and redesigning is used. ER can be carried out in simulation or on real robots. A detailed review of ER techniques and how evolutionary algorithms are formulated for controller development is outlined in Section 2.4.

In the next section, evolutionary algorithms and techniques are explored, the field of evolutionary robotics is then described in more detail and a discussion of the various mechanisms to evolve robots follows.

2.3 Evolutionary algorithms

Evolutionary Algorithms (EAs) are population based stochastic algorithms that mimic natural evolution. They include Evolution Strategies (ES) [11], Evolutionary Programming (EP) [51], Genetic Algorithms (GA) [89, 54], Genetic Programming (GP) [74], and more recently Gene Expression Programming (GEP) [38]. This section discusses the fundamentals of EAs, GAs, GP and finally GEP.

2.3.1 Evolutionary algorithm model

The general methodology used by a population based EA to solve a problem is as follows:

Step 1: Create an initial population of solutions P (0):= (P_1 (0),.., P_n (0)).

Step 2: Compute the fitness, f (P_i (t)), of each individual, P_i (t), of the current population P (t). P_i (t) refers to an individual i in population P(t). t is the number of

generations.

Step 3: Select a parent organism(s) by applying a selection method and/or replication.

Step 4: Apply genetic operations on the parent individuals to create offspring P (t + 1) that make up the next generation (generational) or replace individuals in the current population (steady state)

Step 5: Go to step 2; if maximum fitness has been achieved or maximum generation attained exit algorithm else go to step 3.

2.3.2 Components of evolutionary algorithms

Encoding techniques

An encoding scheme is a mechanism of specifying how the task at hand is encoded as an EA individual [7]. An encoding scheme can also be defined as a method of representing various required traits of phenotype (observable characteristics of an organism) by the genotype (genetic instructions, also referred to as a chromosome). For instance, when evolving a robot obstacle avoidance behaviour, the encoding attributes may include, robot sensors, motors and a function that help translate sensor readings to specific motor actions. Encoding techniques are very important in EAs as they represent the solution required; good encoding techniques are therefore more likely to lead the algorithm towards the correct solution [64]. If an inadequate encoding scheme is selected, the search space is most likely to converge prematurely, exhibit unrequired characteristics and prevent the evolution of a feasible solution [23].

Fitness function

The fitness function or objective function is an evaluation mechanism to determine the performance of an individual within a population P(t) on a given task. The fitness function is a mathematical function that calculates how the specific genetic trait represented by the individual controller performs in the search space or fitness landscape. A fitness function defines the task that needs to be achieved by the algorithm but not how the algorithm accomplishes it. The function guides evolution towards achieving the set objectives. For instance in evolutionary robotics, a controller can be rewarded for accomplishing the whole task (for instance following a wall) or for accomplishing sub-tasks such as obstacle avoidance (if the main task is wall following). Once the evaluation is done, the individual within the population is assigned a quantifiable fitness value, i.e. a measure of its performance. The decision on what fitness function to use depends on the problem that needs to be solved. Thus, a fitness function is normally formulated to suit a specific problem.

[7] The EA individual is also referred to as a genome, a chromosome, an organism or a decision vector. In evolutionary robotics, the individual is referred to as a robot controller. These terms have been used interchangeably in this thesis

In EAs, the fitness assignment can be accomplished in two main ways [133]: (a) **proportional fitness assignment** - in this case the absolute fitness, as evaluated in a task, is assigned to the individuals. (b) **Rank-based fitness assignment** - in this technique, once the objective fitness is calculated, the individuals are then sorted according to their measured objective values. Once sorted, a fitness value is assigned to each individual depending on its position in the individuals rank.

There is a range of evolutionary approaches that are used to solve optimisation problems with two or more objective functions [76]. These EA approaches are referred to as multi-objective evolutionary algorithms (MOEAs) and are capable of processing multiple fitness functions. However, their implementation and functionality is outside the scope of this thesis and therefore they will not be considered further.

A detailed survey of fitness functions used in evolutionary robotics can be found in [99].

Selection

This operation helps to determine which individuals within a population P(t) will be chosen for reproduction; that is, to create a new solution or offspring. In natural selection, Darwin proposes that the best individuals should survive and hence create offspring [31]. To determine which individuals are chosen, a fitness function (a mechanism to measure performance) is devised depending on the problem at hand and the individuals fitness values are computed and assigned. The selection operator selects parent organisms randomly: Therefore, to ensure that the selection is biased towards the best solutions, there have been various selection mechanisms proposed. These selection mechanisms include:

Roulette wheel selection : in this selection scheme, the population or individuals are mapped to contiguous segments of a line or a wheel, such that each individual's slice is proportionate in size to its fitness value. Once this is done, a random number is generated and the individual whose segment spans the random number is selected. The process is repeated until the required number of parents is selected. The mechanism is analogous to a roulette wheel with each slice representing an individual [89]. Individuals with high fitness values have a better chance of being selected for reproduction. This technique is also known as stochastic sampling with replacement [7].

Tournament selection : this is one of the most often used techniques in selection. In this case a certain number of individuals from the total population is selected at random and compared. This is called a tournament size or pool, whereby, the individual with the highest fitness value is selected from the pool and the process limited to select the next parent. Once the two individuals are selected they are used for the reproduction of the offspring. Large tournament sizes increase the selection pressure and lead to selection of only fairly convergent individuals, this can lead to faster convergence of the solution, with the attendant risk of premature convergence.

Since rank-based fitness is based on a position within the ranked population and not on raw fitness values, the probability of having one individual or a group of individuals

dominating the selection is eliminated. This also helps overcome scaling problems related to proportional fitness assignments. Scaling problems refers to local minima where high selection pressure has caused the search space to narrow down very quickly and stagnation in the case where there is low selection pressure. The technique provides a uniform scaling across the population and controls selection pressure effectively [7].

Genetic operations

In order for the EAs to create diversity within a population, the following genetic operators are normally used to varying degrees for different algorithm variants.

Replication : in this technique, once selection is performed, the individuals with higher fitness values are copied to the next generation without modification. The effect is that good individuals (with high fitness values) have a higher probability to be chosen and copied to the next generation than weaker individuals. However, this technique does not contribute to genetic diversity.

Mutation: in the process of evolution, it has long been discovered that variation of the genetic material is key to the adaptation of the population [31]. The same ideal is used in artificial evolution. A probability of mutation is normally chosen and a random bit/allele selected for mutation. High probability of mutation is often destructive to the individuals, thus, when performing mutation, a low probability is preferred. Mutation is a key operator used in genetic algorithms and most evolutionary algorithms [89].

Crossover : this involves transfer of materials from one parent individual to another akin to sexual reproduction in higher animals. Crossover can be implemented as a single point crossover, where only one point is randomly selected and genetic materials transferred from that point to the end of the individual, or it can be multi-point crossover. There are various forms of multi-point crossover, the most common being two point and uniform crossover. In two point crossover, a start and end points are randomly selected and the genetic materials between this two points swapped from one parent gene to the other. Uniform crossover, on the other hand, uses a fixed mixing ratio between the two parent organisms. For instance, if the mixing ratio is 0.5, then the resulting offspring will have 50% of the genetic materials from one parent and the rest from the other parents (Figure 2.4 shows an example of uniform crossover where the mixing ratio is 0.5).

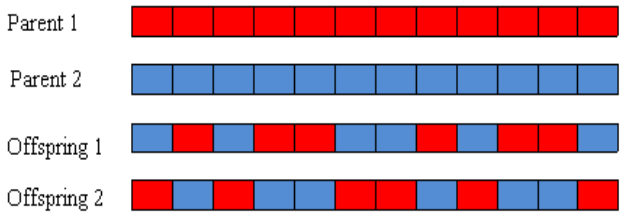


Figure 2.4: Example of uniform crossover

Elitism

When creating a new population by selection, crossover and mutation, there is a reasonable chance that the best individual can be lost. Elitism involves copying the best individual (or a few best individuals) to the new population. This ensures that best individuals are not lost during the process of evolution. Elitism can very rapidly increase the performance of EA, because it prevents the loss of the best found solution(s).

2.3.3 Genetic Algorithms

Genetic Algorithms (GAs) [89, 54] are a very popular EA technique. In this algorithm, the individuals in a population are fixed length linear structures. The main encoding scheme used in a GA is binary values. However, integers, floating points as well as symbols have also been used [85, 126]. In a GA, the individuals encode a solution to the problem being investigated. Thus the parameters that need to be provided to the algorithm must be part of the solution. The main genetic operator used in GA is mutation with a low probability, however, most often a crossover is implemented usually with a high probability. A GA follows the basic algorithm described above and incorporates the discussed components as set up by the designer.

2.3.4 Genetic programming

Genetic Programming (GP) [74] involves evolving computer programs to solve a problem or mathematical functions. Note that GA involves evolving solutions to solve a specific problem. A GP follows a similar algorithm to a GA, however GP individuals are ramified structures representing a computer program. The following section discusses a GP encoding and evolution operators.

GP encoding

The canonical GP uses tree like structures as individuals. This structure incorporates nodes which can either be functions or terminals. In tree or graph terminology, a function is a node on which other nodes can be connected to while a terminal is a leaf node (that is, terminal nodes are connected to function nodes and cannot have other nodes connected below them). Functions will either be arithmetic, boolean, conditional or logic while terminals may be real values, integers or symbols. Terminals provide the GP with values while functions provide the capability to either compute or connect the terminals. For instance, when evolving robot behaviour, sensor inputs and motor actions will form the terminals, while conditions that map the sensors to motor actions form the functions. Figure 2.5 shows an example of a GP structure.

The standard GP often uses crossover as the main genetic modification operator. This is unlike GA which, as previously mentioned, often uses mutation operator and crossover. The argument is that since GP individuals contains building blocks, given crossover, the

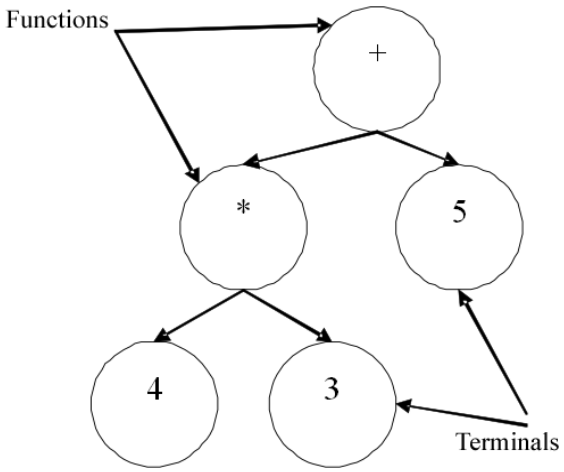


Figure 2.5: GP representation of a problem

system should be able to combine 'good' building blocks to form more fit individuals and that small building blocks combine to form large better building blocks [74]. The problem associated with the use of crossover only, is that it leads to premature convergence and there is technically nothing new being added to the population apart from elements from the existing individuals [40].

GP variants

Since the introduction of GP, there have been many variants that have been proposed. The main aims for the introduction of these variants are: "simpler implementation, higher speed, smaller memory requirements and the capability to work with particular hardware architectures" [109]. This section presents a brief description of common GP variants.

Linear GP

Linear Genetic Programming (LGP) represents the computer programs or individuals as linear variable length sequence of simple instructions [15, 116, 109]. The instructions in LGP may represent actual machine code (bits and ones) and be directly executed by the CPU or they can be interpreted instructions programmed using an imperative language such as C/C++. These instructions operate on one or two indexed variables, v, which are also referred to as registers/ memory locations. The result of an operation is placed into a destination register. An example of an instruction would be : $v_3 = v_1 + v_2$ (that is, an instruction operating on two registers). There is thus no distinction between terminals and functions as is common in standard GP.

LGP uses both mutation and crossover to introduce diversity to the initial population. However, these operations have to operate within instructions boundaries. The mutation operator can operate as a micro mutator where an operator of an instruction is changed or it can act as a macro mutator where an instruction is deleted or inserted. The crossover on the other hand, swaps continuous sequence of instructions.

Multi expression programming

Multi Expression Programming (MEP) [1, 107, 109] is a GP variant that uses linear string of genes to form its individuals. The genes in an MEP individual are formed using sub strings of variable lengths. A gene encodes either a terminal or a function. If a gene is formed using a function, then the gene also encodes pointers towards the function arguments. These pointers are indices or positions of other genes and they must be lower than the position of the calling gene. When initialising the MEP individual, the first gene always contains a terminal so as to ensure that the programs are syntactically correct. In any given problem, the function set, terminal set and the number of genes per chromosome is given. The number of genes remains constant during the evolutionary run and is thus used to calculate the length of the chromosome as shown by Equation 2.1.

$$Length = (n + 1) * (NumberOfGenes - 1) + 1 \tag{2.1}$$

where n represents the number of arguments of the function with the highest arity and NumberOfGenes represents the number of genes. A description on how this formula is derived can be found on [107, 109].

Each of the encoding genes in a chromosome is used to independently evolve a solution to the problem. The overall chromosome fitness is usually defined as the fitness of the best gene in the chromosome. This is unlike standard GP, where only one gene is evolved to provide solution to the problem.

MEP uses crossover (one point, two point and uniform) and mutation to introduce diversity. The crossover works by exchanging materials from the selected parents. The mutation operates by mutation of the genes to either a terminal or function. However, care has to be taken to ensure that the first gene is always a terminal.

Cartesian genetic programming

Cartesian Genetic Programming (CGP) [1, 87, 109] uses graph structures to represent individuals. This is unlike standard GP where tree structures are used. The main argument for using this algorithm is that unlike tree structures, graph structures can be used to encode more complex problems [109]. In this algorithm, the graph nodes are represented in a cartesian coordinate. Each node contains a function symbol and pointers to the function arguments. Each node has an output which can be used as an input by a different node. The CGP chromosomes are represented as arrays of integer values. Thus, the functions are provided with an integer label and the strings are encoded by reading the graph columns top-down and writing the inputs nodes first, followed by the function label for each node. The genetic operators, mutation and crossover, are used to modify the gene and to introduce diversity. The mutation can change a node function, a node input or even the connections in a gene.

Cartesian Genetic programming is similar in operation to Parallel Distributed Genetic Programming (PDGP) [116]. However in PDGP, the genetic operations work directly on the graph unlike in CGP where it works on the linear genome.

Traceless genetic programming

The main difference between Traceless Genetic Programming (TGP) [1, 108] and the standard GP is that TGP does not explicitly store the evolved chromosomes. Instead, the TGP individuals stores only the fitness value(s) achieved so far. For instance when solving a symbolic regression problem with m fitness cases, the TGP individual will be: (f_1, f_2, f_3 ... f_{m-1}, f_m), where f_i is a fitness value for the i^{th} fitness case. Behind this fitness values is a tree expression (similar to standard GP), however the TGP does not store the chromosome.

There are two genetic operators used with TGP. These are: (a) Crossover: In this genetic operation, a function operator is selected together with a number of parent chromosomes. The number of selected parent chromosomes has to be equal to the arity of the selected function. The crossover operation is completed by joining the selected parent chromosomes using the selected function. (b) Insertion: This operation is used to introduce diversity by introducing a new simple expression to the population.

In summary, as the described GP variants show, there have been many techniques proposed to improve the standard GP. These variants represent GP individuals either as fixed length linear strings, variable length and even graphs. Other GP mechanisms that have been introduced include Grammatical Evolution [1, 109], Stack-Based GP [109] as well as Probabilistic Genetic Programming [116]. The similarity with these algorithms is that the aim is to evolve computer programs with minimum overheads. A survey of various GP variants can be found in [1, 109, 116].

2.3.5 Gene expression programming

Gene Expression programming (GEP) [38], like GP, is an EA that is used for creation of computer programs. However, this algorithm differs from GP due to the representation of its individuals and the additional genetic variation operators that are used. In this algorithm the encoding scheme used is one of linear strings of fixed length (analogous to GA), which are later expressed as tree like structures similar to GP individuals [8]. Due to the use of tree like structures to represent GEP phenotypes, most literature in the evolutionary computation field present GEP as a GP variant [1, 109]. However, the GEP algorithm combines the simplicity of GA and the abilities of GP, and can be said to be a generalization of both techniques. The main advantage that it has over GP is that the expense of managing a tree structure (parse trees in GP) and ensuring correctness of programs is eliminated.

In GEP, the genotype is subjected to modification by means of mutation, transposition, root transposition, gene recombination, and one and two-point recombination as discussed in section 2.3.5. The genotype encodes expression trees which are the object of selection. It is this creation of separate entities with distinct functions that allows the algorithm to perform with high efficiency that is shown to surpass existing evolutionary algorithms

[8] In GEP, the fixed length linear strings are referred to as individual/organism/genome/chromosome or the genotype. The tree like structure, however are known as the phenotype or expression tree (ET)

[41] in particular tasks. Figure 2.6 shows the flowchart of GEP algorithm. The flowchart describes the implementation of a generational GEP algorithm as used in all the experiments described in this thesis. As shown by the flowchart, the process begins with the random generation of the chromosomes of the initial population. The chromosomes are then expressed and the fitness of each individual is evaluated. The individuals are then selected according to fitness to reproduce with modification, producing offsprings with new traits. The individuals of this new generation are then subjected to the same evolutionary process: that is, expression of the genotype, selection, and reproduction with modification. The process is repeated for a certain number of generations or until a solution has been found.

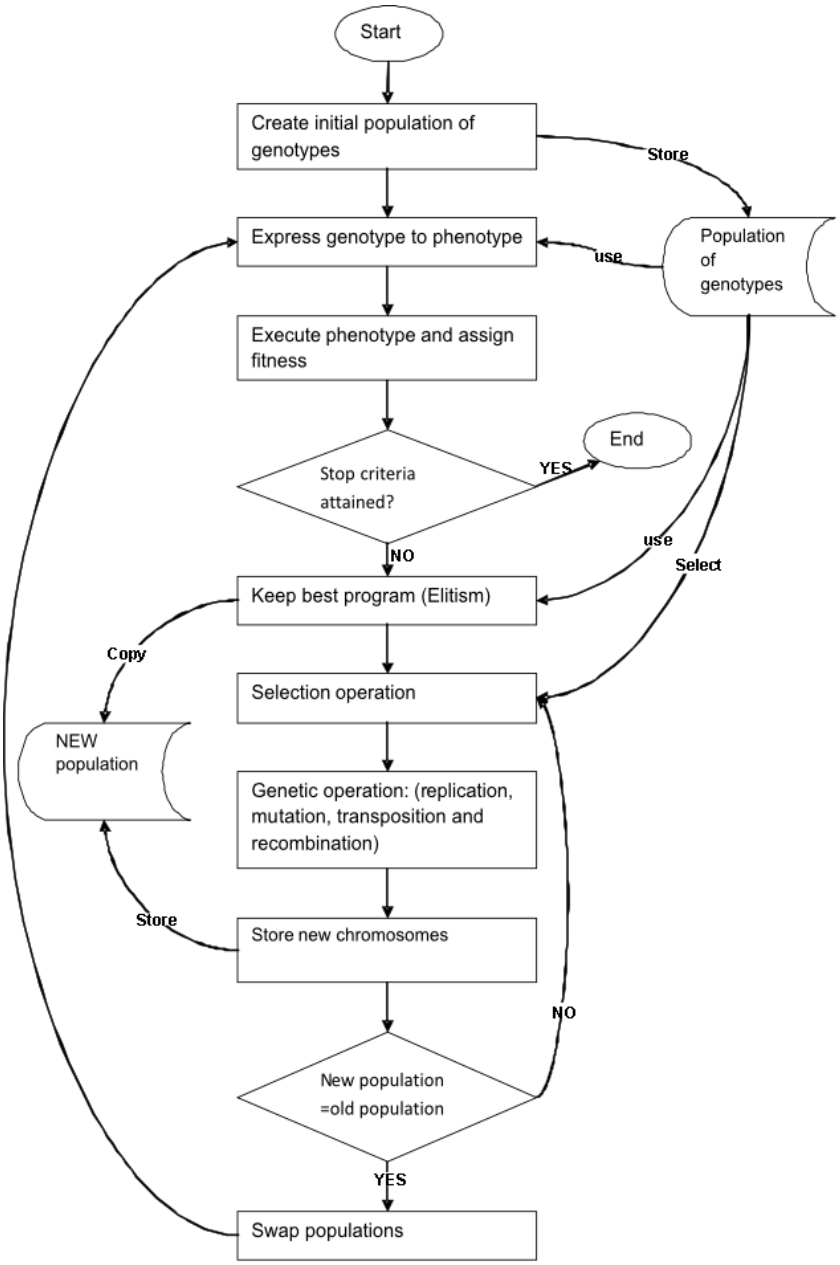


Figure 2.6: Flowchart of a GEP algorithm

GEP individuals

The individual of a GEP algorithm is made up of one or more genes [9]. A gene is a symbolic string composed of a head and a tail section. The head contains symbols that represent both functions, F, and the terminals, T. The tail section only contains terminals. Thus, there are two different alphabets in the two different regions of the gene.

In any given problem, the terminal set, function set and the length of the head, h is given. The length of the tail is a function of the length of head and the number, n, which denotes the number of arguments of the function, f, with the most arguments i.e. maximum arity of a function in the function set, F.

Equation 2.2 shows the formula used to calculate the length of the tail for any given GEP gene.

$$t = h(n-1) + 1 \tag{2.2}$$

Example: given the terminal set [a, b, c] and function set [*,+] and h=4 , the highest arity is 2 , i.e. both multiplication and addition can only take two arguments, we then formulate the tail length as shown below:

$$\text{t=h (n-1) +1=5.}$$

Thus the total number of arguments will be 9. An example of the expression formed with these parameters is as follows:

$$+*ab|2abc2$$

The vertical line indicates the beginning of the tail.

Once the genotype is formed as shown in the example above, it is then expressed/translated into the phenotype/expression tree (ET). This is done by placing the first function in the gene at the root of ET and then attaching as many branches as the arity of a function may allow. The formulation is complete when all functions have terminals attached to them and thus all branches end with terminals. Figure 2.7 shows the chromosome, the resulting phenotype after translation and the delineation between the materials used to form the phenotype and those that are not used.

The GEP tree is referred to as an Expression Tree (ET) or the phenotype. Reading from left to right, top to bottom, the expression: +*ab2 is derived. This is referred to as an open reading frame (ORF) or Karva (K) expression [38]. The ORF or K-expression is the region of the genome that is used to evaluate the fitness value of the gene. The genome elements, downstream from the ORF, form the non-coding region of the genome and it is this region that gives GEP its robustness in solving a problem (Please see figure 2.7, coding & non-coding regions, shown in the genotype above in bold). The GEP ORFs are similar

[9] In GA and GP, the individual/chromosome/genome/organism is composed of one gene. Therefore, in GA/GP literature, gene is used interchangeably with individual/chromosome/genome/organism. In GEP, however, the individual/chromosome/genome/organism can be made up of more than one gene, thus, extra care is taken when using the words

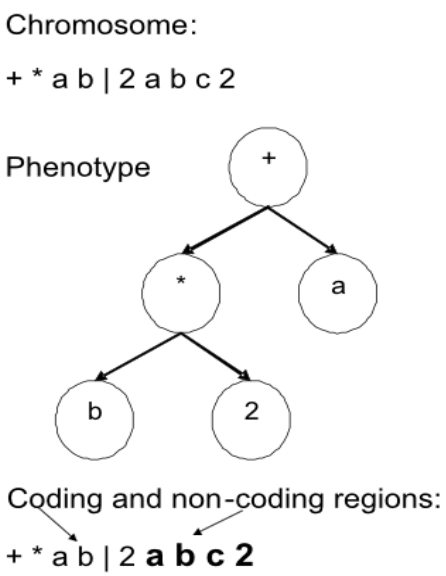


Figure 2.7: GEP chromosome, phenotype/expression tree (ET) and resulting coding and non-coding regions

in nature to the same elements in biology, the only difference is that in biology the ORF has non coding elements upstream and downstream while in GEP, the non coding region is only downstream [38]. During a GEP run, the genome undergoes the genetic operation while the phenotype (ET) undergoes the selection. Thus the genome is maintained as a fixed length string while the ETs can change in size and shape.

Multigenic chromosomes

In natural biology, a collection of genes forms a chromosome. Similarly, in GEP, a chromosome/genome with more than one gene can be formed. A GEP chromosome with more than one gene is referred to as multigenic GEP (mgGEP) chromosome. The genes combine to form a chromosome using a specified function known as a linker. For instance, they may be added together and hence the genes will appear as a concatenated single string of symbols [38]. The chromosome thus formed, has multiple ORFs each coding for a sub-ET. The advantage derived from such a structure is that each different sub-ET is a separate program and a part of the more complex whole ET. The sub-ET can be used in problems with multiple outputs to evolve a separate part of a problem, and selected depending on its individual fitness. Alternatively the different sub-ETs could be used as part of a more complex ET and selected based on the fitness of the whole ET.

For instance, if the following genes are given as part of a genome:

+*xy|xy32y

-/x*|yy23x

*-x3|xxy23

The following chromosome can be formed: +*xy|xy32y -/x*|yy23x *-x3|xxy23.

| indicates the beginning of the head and tail portions of the genes. The above genes, individually forms the sub-ETs and ORF shown by Figure 2.8. If the addition function (+) is used as a linking function, figure 2.8 can then form the ET as shown in figure 2.9

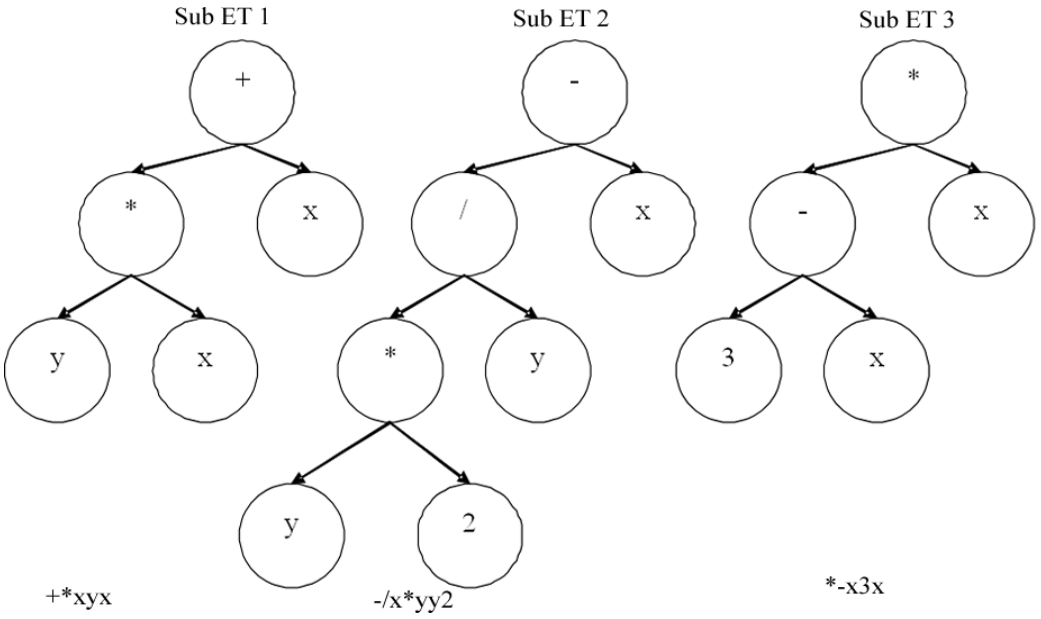


Figure 2.8: sub-ETs in a multigenic chromosome

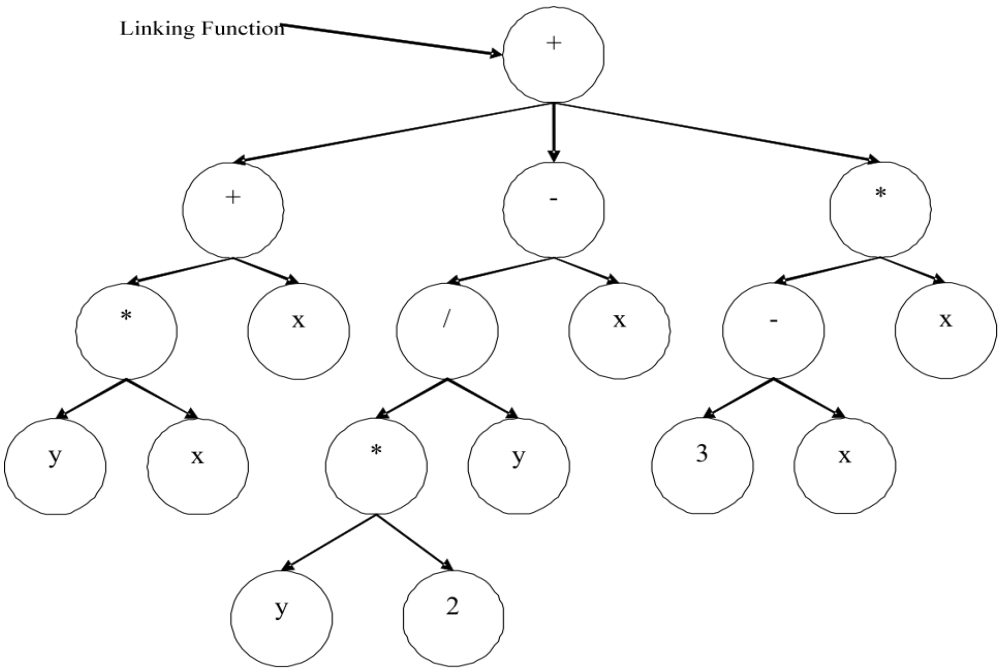


Figure 2.9: Expression Tree (ET) for a multigenic GEP

As observed, the GEP chromosomes can be formed by a simple chromosome with one gene, here on referred to as unigenic GEP (ugGEP), or it can be formed by multiple genes - multigenic GEP (mgGEP) with multiple ORFs. GEP can be used to encode simple problems using the unigenic structures or more complex problems using the mgGEP.

Gene Regulation in GEP

In natural organisms, the regulation of gene expression is achieved through genetic regulatory systems structured by networks between the genes, RNA, proteins and other cell molecules. The connection topology and interactions between these cell components forms an action-reaction chain that is referred to as a Gene Regulation Network (GRN). Thus, in systems biology, a GRN refers to the organisation of an interlinked set of genes in a cell, and how their interactions influence how member genes are transcribed into mRNA [20, 57]. The interaction within this set of genes is achieved indirectly through their RNA and protein expression products. Alternatively, the interaction can be achieved through other substances in the cell [121, 34, 82]. In natural organisms, once the genes are transcribed into mRNA, the mRNA is then translated into a protein or set of proteins [27]. These proteins can either be structural and hence utilised to give particular structural properties to a cell, or it could be an enzyme (that is, a chemical catalyst which is used to enable/speed up particular reactions within a cell). Conversely, these proteins products could be used primarily to activate or inhibit the transcription of other genes by binding to the promoter region at the start of the genes [121, 122]. Those proteins whose primarily role is regulation of gene activities are referred to as transcription factors [121, 27]. A simplified example of how gene regulation occurs in a biological organisms is shown by Figure 2.10.

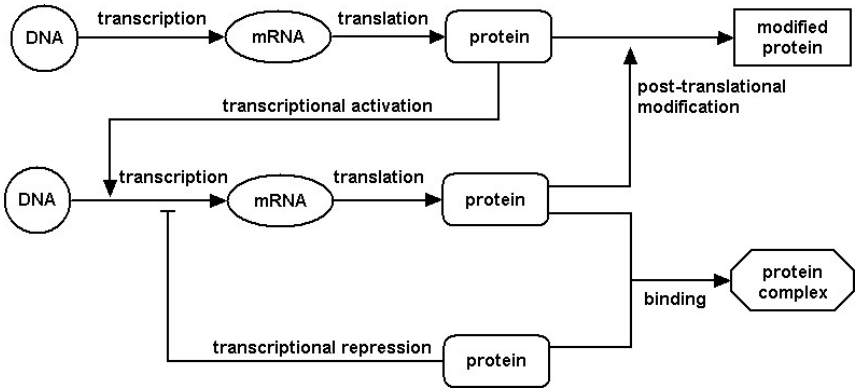


Figure 2.10: Biological structure of a Gene Regulation Network. Genes regulate each others activity through regulatory networks. Gene transcription into mRNA is influenced by transcription factors, themselves products of other genes. In addition, post-translational modifications lead to proteins with modified. This figure and description has been reproduced from [123].

The multigenic GEP (mgGEP) chromosome is inspired by the workings of the biological cell. Similar to a biological cell, the mgGEP chromosome is made up of multiple genes. These genes, as described earlier (Section 2.3.5), are usually interlinked using a linking function. In the standard GEP algorithm, the linking function is provided before the start of an evolutionary run and provides the mechanism for gene interactions. Additionally, when a logical or a conditional linking function is used, the conditions specifies when a particular gene in mgGEP chromosome will be expressed. Consequently, similar to GRN in systems biology, the linking function acts as a gene regulator or a transcription factor. Thus, roughly speaking, mgGEP chromosome with a linking function represents a GRN model, albeit in a very simplified way. This thesis shows how these linking functions

can be evolved together with the rest of genome. Additionally, the thesis proposes new techniques where one gene in a multigenic set can be used to evolve the conditions under which gene regulation should occur (see Chapter 5).

Mathematical and computational models of GRNs have been developed in order to analyse and predict how biological GRNs behave. Some of these modelling techniques in GRNs include: Boolean Networks [82], Coupled Ordinary Differential Equations (ODEs), Bayesian Networks, Graphical Gaussian Models, Stochastic and Process Calculi. A comprehensive discussion of these techniques can be found in [34].

GEP evolution Operators

Like all the other evolutionary algorithms, GEP utilises various genetic operations to create diversity within the population. In comparison to its predecessors GEP offers additional genetic operators as discussed below.

Mutation : as highlighted earlier, mutation involves randomly changing symbols in a chromosome. When performing mutation in GEP, elements in the head region can be changed to either a function or a terminal, while at the tail, terminals can only be changed into terminals. This way the structural organization of chromosomes is maintained and all the new chromosomes produced by mutation are structurally correct programs. Mutation is by far the single most important genetic operator and populations undergoing mutation display non-homogenizing dynamics [40, 41].

Transposition and insertion sequence elements : the transposon of GEP are elements of the chromosome that can be copied from one section of the chromosome and be transferred to a different section [38]. Three types of transposition are implemented:

Insertion sequence transposition (IS) : in this transposition operator, any sequence of alleles (fragments or symbols that form the gene) in the chromosome can be selected as an IS element, therefore, these elements are randomly selected throughout the chromosome. A copy of the transposon is made and inserted at any position in the head region, except the first position, pushing downstream the remaining elements in the head region. This means that the gene loses as many elements at the end of the head region as the displacing elements. The earlier implementation described in [38] has the transposition operator randomly selecting the chromosome, the start of the IS element, the target site and the length of the transposon. However, the IS described in [41] removes the requirement for the transposition length and incorporates the start and termination points instead, this makes sense as the knowledge of the two points nullifies the need for the length to be known. Figure 2.11 shows an example of how insertion sequence transposition occurs. The multi-gene chromosome shown previous has been used.

Root insertion sequence transposition (RIS) : the elements required for this transposition need to start with a function, as a result, the first element or the entire transposon has to be selected from the head section of a gene. To select the first element of the transposon, a point is randomly chosen in the head section and the head section scanned

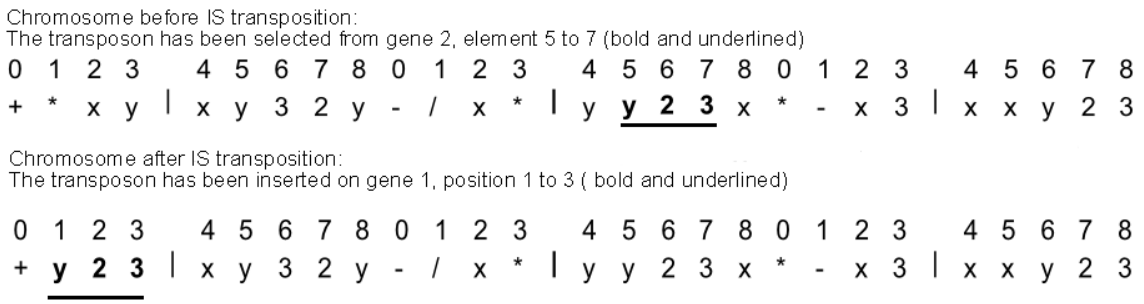


Figure 2.11: An example of IS Transposition. The | marks the end of head region.

downstream until a function is found [38]. This function becomes the start of the RIS element, if no function is found, it does nothing. The RIS operator randomly chooses the chromosome, the gene to be modified, the start of the RIS element, and its length. The RIS described in [41] is slightly modified as the operator does not choose the length, only the start and terminal points are chosen. During RIS transposition, the whole head shifts to accommodate the RIS element, losing at the same time, the last symbols of the head (actually as many as the transpose length). Figure 2.12 shows an example of how root insertion sequence transposition occurs. The multi-gene chromosome shown previous has been used.

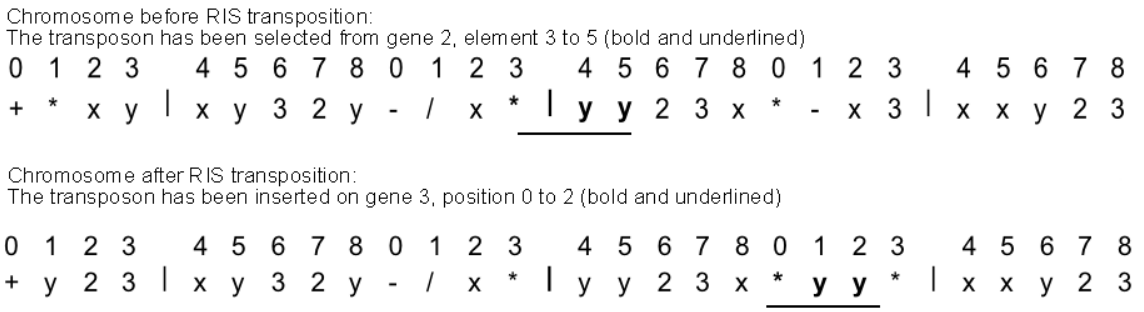


Figure 2.12: An example of RIS Transposition. The | marks the end of head region.

Gene transposition : this operation involves interchanging the positions of various genes in the chromosome. The transposon, i.e. the selected gene, is deleted at the place of origin and placed as the first gene of the chromosome pushing other genes downstream. Since the selected transposon is deleted at the place of origin, the size of the chromosome is maintained. In multigenic chromosomes joined by a linking function such as addition, this operator contributes nothing to the adaptation. Nonetheless, when expression trees are linked by a non- commutative function the order of the genes is important. In this case, the gene transposition operator works as a global mutation operator. The power of the gene transposition is seen when used in conjunction with recombination as it allows the duplication of genes, which plays an important role in evolution, and also allows a more generalized shuffling of genes or smaller building blocks [38, 40]. Figure 2.13 shows an example of how gene transposition occurs. The multi-gene chromosome shown previous has been used.

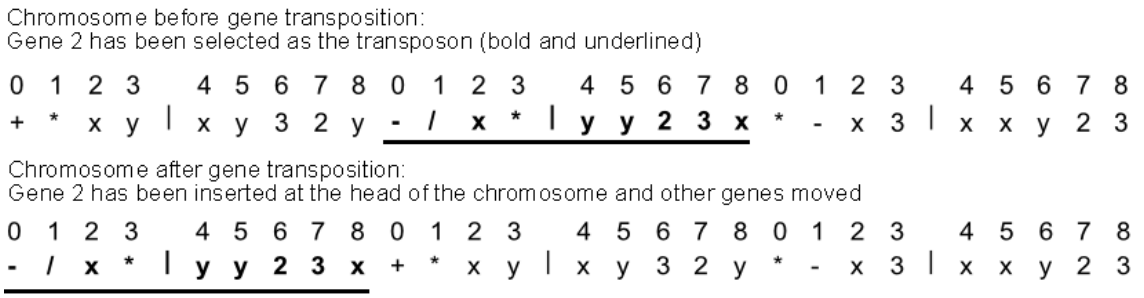


Figure 2.13: An example of Gene Transposition. The | marks the end of head region.

It is important to note that transposition does not affect the tail of the gene, only the head region, which is an important area as this determines the coding region of the gene.

Recombination : this can be one-point (the chromosomes are split in two and the corresponding sections are swapped) or two-point (chromosomes are split in three and the middle portion is swapped) or gene (one entire gene is swapped between chromosomes). Recombination whether 1 point, 2 point, or gene recombination is the least powerful of all GEP operators and populations undergoing recombination alone display homogenizing dynamics. For it to be useful it should be used in conjunction with the other operators.

A detailed description of GEP with examples on how the operators are used can be found on [42, 38].

GEP applications

The capabilities of the GEP algorithm have been demonstrated in solving various problems. Firstly, GEP has been used in function finding problems such as symbolic regression and sequence induction [38, 93]. Secondly, GEP has been used in time series prediction and logical synthesis [42]. Thirdly, the generation of emergent behaviours by evolving cellular automata rules for the density classification problem has been investigated. In this problem, GEP has been seen to outperform GP by four orders of magnitude [38]. GEP has also been used in boolean concept learning where the results obtained were compared to GP solutions and it was found that GEP is faster and more efficient in solving the problem [38, 42].

Other problems that GEP has been tested on, include but not limited to, sentence ranking functions for text summarization, event selection in high energy physics [134] and more recently on an automatic plan design [77] [10]. This thesis investigates and describes how GEP and novel variants thereof can be used to evolve efficient control programs for a robot.

The next section discusses how evolutionary algorithms are utilised in robotics research for automatic development of robot controllers. The section extends the brief introduction

[10] A bibliography of research carried out using GEP can be found on `http://www.gene-expression-programming.com/GEPBiblio.asp`

given in Section 2.2.5. Additionally, a discussion on how artificial neural networks (ANNs) and EAs are utilised in evolutionary Robotics (ER) is outlined.

2.4 Evolutionary robotics

As previously mentioned, evolutionary robotics (ER), is a mechanism to automatically generate robot controllers [105] and sometimes robot morphology [84] using artificial evolution techniques. The subject of evolution in ER, can be computer programs [74] or elements of a neural network such as synaptic weights, learning rules [105] or whole network architectures [149]. The ER process, just like an EA, starts with an initial population of individuals or computer programs. The individuals encode a set of synaptic weights which are then fed to a neural network controlling a robot, or a computer program that defines how the robot needs to act given a set of sensor readings. If the aim is to evolve a robot morphology, the controller encodes attributes needed in a physical robot. Once the controllers are initialised with the required parameters, the robot is set to act freely in the environment while its fitness is evaluated. The process of artificial evolution is then started by selecting two individuals using a previously chosen selection method. The individuals are allowed to reproduce by undergoing mutation, recombination and transposition. In a steady state algorithm, the offspring fitness is evaluated and they are either passed on to the next generation by replacing them with individuals with a lower fitness, or are discarded. In a generational algorithm, the offspring are used to construct a new population for the next generation. This process continues for a set number of generations or until a controller which satisfies the set performance criteria is met. Figure 2.14 shows the steps towards evolving a robotic controller using a generational algorithm. The process starts with the random generation of the chromosomes of the initial population (robot controllers). These chromosomes are then expressed and the fitness of each individual is evaluated in a robotic environment (that is, the controllers are tested for the target behaviours and their performance recorded). The individuals are then selected according to fitness to reproduce with modification, producing offsprings with new traits. The individuals of this new generation are then subjected to the same evolutionary process: that is, expression of the genotype, selection, and reproduction with modification. The process is repeated for a certain number of generations or until a solution has been found. All experiments described in this thesis have been carried out using a generational algorithm.

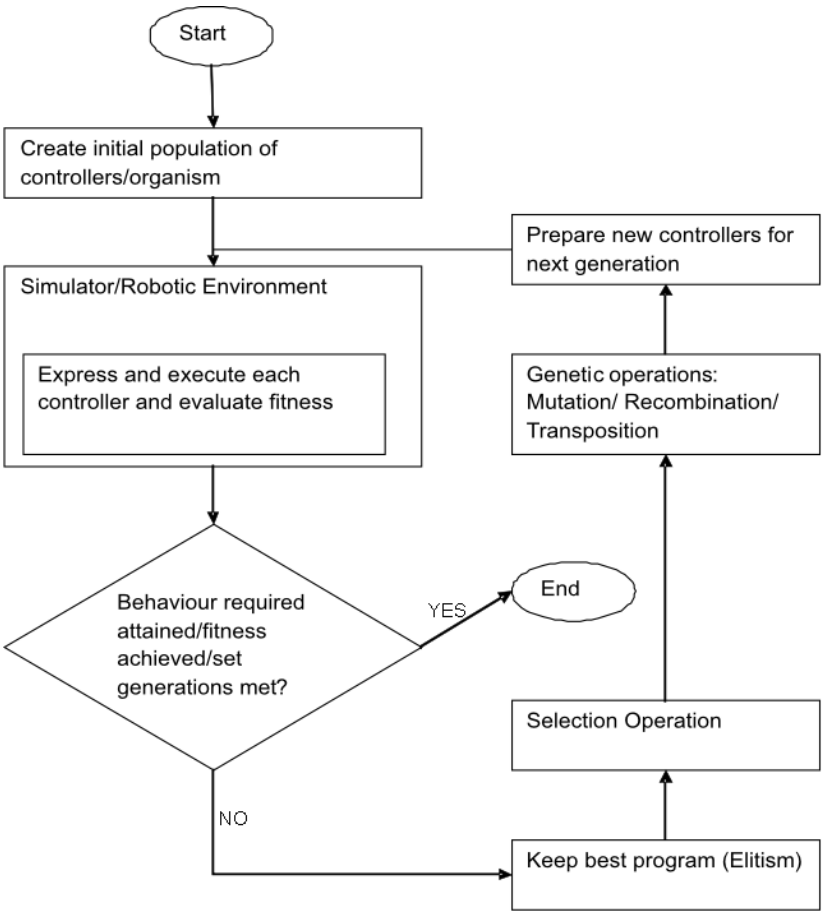


Figure 2.14: Evolutionary robotics methodology

This section focusses on how to evolve robot behaviours in ER. Two approaches are discussed: firstly evolving neural networks and secondly evolving computer programs. This section also discusses the various evolution platforms utilised in ER experiments.

2.4.1 Evolving neural networks

Artificial neural networks (ANN) are simplified models of the central nervous system. They are networks of highly interconnected neural computing elements that have the ability to respond to input stimuli and to learn to adapt to the environment. The architecture borrows the parallel nature possessed by the biological neurons to model computer systems. An ANN consists of a set of processing elements, also known as neurons or nodes, which are interconnected. It can be described as a directed graph in which each node i performs a transfer function f_i of the form:

$$y_i = f_i(\sum_{i=j}^{n} w_{i,j}x_j - \theta_i) \tag{2.3}$$

Where y_i is the output of the node i, x_j is the jth input to the node, and $w_{i,j}$ is the

connection weight between nodes i and j. θ_i is the threshold (or bias) of the node. Usually, f_i is non-linear, such as Heaviside, Sigmoid, or Gaussian function, n is the number of nodes. See [112, 149, 131, 102, 62].

The school of thought supporting the use of EAs to evolve neural networks argues that learning and evolution are the two fundamental forms of adaptation in intelligent natural systems and hence the great interest in combining the two. This particular class of study is known as Evolving Artificial Neural Networks (EANNs) [149] or neuroevolution [86, 136, 50]. The use of ANNs in tandem with EAs is normally justified by the following issues: ANNs are able to relate the input to the required outputs; in that case they can be used in predictions using historical data. They have also shown to be able to generalize between samples, in the way humans do [105], they also show 'graceful degradation' which means that removing one or more units results in reduced performance, not complete failure. They have also shown to be tolerant to noise in the data. Another advantage is that the use of the parallel nature of the neural networks and high speed computers could lead to very good results in solving problems [102, 62]. EAs are used to either evolve the architecture of the ANN, or to evolve weights, learning parameters or both as detailed below. For a detailed review on ANNs, please see [36, 102, 70, 62].

Weights and learning parameters

The primary way of learning in ANN is to adjust the weights using a set learning rule. When using EA for weights and learning parameters, the weights are directly encoded to the genotype either as a string of real values or string of binary values [146, 149, 105]. The main aim on evolving ANN weights is to find near optimal weights that can be used with a fixed ANN architecture to solve a given problem. Since most robot controllers are neural based, GAs have been used extensively to evolve suitable weights.

Floreano and Mondada [46] used a GA to evolve the weights and thresholds (activation function values) for a real robot controlled by a simple recurrent neural network. Their GA was encoded with real values denoting the weights and thresholds, each individual was in turn decoded to the corresponding neural network whose inputs was the robot sensors and the output unit set the velocity of the wheels. The evolved neural network guided the robot to accomplish obstacle avoidance and a homing behaviour successfully. Neruda and Slusny [101] have used a GA to evolve obstacle avoidance and maze following behaviours for a neural controlled robot. In their work, the GA encoded the weights. GAs have also been used to encode weights and activation function values in [141] where shooting and path planning behaviours are evolved. Nelson et al. [97, 98] has evolved the weights of a recurrent neural networks to evolve navigation and maze exploration behaviours. The experiment is carried out in simulation and later transferred to a colony of 8 real robots. In [135] a (1+1)-evolution strategy is used to evolve the synaptic weights and thresholds for subsumption like neural controller. Bajaj and Marcelo [5] used a GA to evolve the synaptic weights for a single layer recurrent neural networks. Their work uses an incremental evolution to evolve an obstacle avoidance behaviour which is then modified to a wall following one.

Architectures

The architecture of an ANN is made up by the weight connections and the transfer functions of each node in the network. Thus, a good network architecture is crucial to the success of solving a problem [149]. A key issue in evolving neural network architectures is to make a decision on how the architecture shall be encoded into a GA chromosome. Encoding techniques as mentioned in section 2.3.2 refers to the strategy utilised to represent a specific task (such as a network structure) as an EA chromosome (also called a genotype). The following two main encoding strategies have been identified.

- **Direct encoding:** This refers to an encoding strategy where all parameters that define an ANN architecture are encoded to an EA chromosome [89, 149, 21]. These parameters include the weight values, number of nodes, connectivities and activation function. In this strategy the connection topology is represented as a $N \times N$ matrix where each entry encodes the type of connection from the "from node" to the "to node" [89]. Once the matrix is formed it is then translated to a GA chromosome by placing the matrix rows in a sequence. A fitness function is then selected and various evolutionary parameters set before the evolutionary run. To compute the fitness, the chromosomes are decoded back to the connectivity matrix and used to form the neural network [89, 149]. The fine tuning of the weights is done by applying a learning algorithm to the decoded network and the weights are trained using, for instance, back propagation [149, 105].

  In comparison to GEP, direct encoding is a strategy to encode an ANN as a GA genotype while GEP is an EA similar, in concept, to a GA. The concept of direct mapping between genotype and phenotype is nevertheless similar in both direct encoding and GEP. Additionally, GEP as an algorithm can utilise direct encoding mechanism to form the genotype of a particular problem. Nevertheless, in direct encoding, utilised in ANNs, the entire genotype is utilised to form the phenotype while in GEP only the open reading frame (ORF) is utilised to form the phenotype. Additionally, the direct encoding strategy is only utilised to encode an ANN as a GA chromosome while GEP represents any particular problem using a genotype/phenotype representation.

- **Indirect encoding:** This strategy involves encoding only the most important parameters of an architecture [89, 149]. Thus, the memory requirements for indirect encoding scheme is less than than of direct encoding. Due to the limitation of size, the method is faster but often with sub-optimal results [149]. In most implementations, the neural networks are encoded as grammars and the GA is then used to evolve the grammars [89]. The fitness of the developed networks is then tested after a network is evolved from the grammar. That is, the grammar represents the genotype while the phenotype is the network derived from the decoded grammar. A grammar is this context refers to a set of rules utilised to encode a network structure.

  Similar to the direct encoding above, indirect encoding is similar to GEP in the division of genotype and phenotype. Also, GEP utilises string symbols to represent an attribute of a problem as a genotype. These string symbols, which include func-

tions and terminals, are used to represent a set of rules on how a problem should be solved. Thus, a GEP algorithm can utilises an indirect encoding mechanism to encode a problem as a GEP genotype. However, unlike GEP, all materials in the ANN indirect encoding genotype, are used to form the phenotype.

There is a considerable interest in evolving ANN architectures [149]. For instance, Capi and Doya [22] used a GA to evolve the architectures of a neural network. In their work, three types of architectures were explored. Firstly, a feed forward neural network (FFNN) was evolved where the GA chromosome encoded weight connections and the number of hidden units. Secondly, a global-recurrent neural network (GRNN) where the weight connections, initial weight values and number of hidden units were encoded into the chromosome. Thirdly a local-recurrent neural network (LRNN) where the genome encoded the weight connections, thresholds, initial values, memory neurons, interconnection between memory units and the number of hidden and memory units. In this work, a sequential navigation task requiring a robot to move to two rewarding sites was investigated. Results generated show that modularly organised recurrent networks, in this case the LRNN, are better than fully connected recurrent networks. The synaptic weights, threshold values and time constant for a continuous time recurrent neural networks (CTRNN) were evolved in [13] using a GA. The authors show that reinforcement learning like abilities can be generated using a CTRNN on a set of T-maze and double T-maze navigation tasks. Learning in this work is achieved from internal dynamics without modifying the synaptic strengths.

GAs have also been used to evolve neural network architectures in e.g. [61, 60, 28, 29, 67, 105, 137, 138].

Learning rules

The evolution of connection weights and network architectures deal with the components of the ANN. The evolution of the learning rate or weight updating rule on the other hand, has to deal with the training of the neural network and hence generates the dynamic behaviour displayed by the ANN [149]. The evolution of learning parameters such as the ones used in a back propagation or Hebbian rules can be described as a method of evolving rules, however these techniques tends to be more optimised towards the architecture of the ANN rather than learning [105]. If there is little knowledge regarding the type of ANN architecture that is being used in a problem, designing an optimal learning rule becomes very hard [149]. It is thus recommended to develop an automatic method to adapt the learning rule to the architecture and the problem being investigated. EAs are suited to evolve learning rules or the learning parameters as they can discover unique parameters that lead to robust adaptability of ANN [130].

Floreano and Urzelai [48] used a GA to encode and evolve different sets of learning rules. In their work the plain Hebbian learning rule is divided into rules that strengthen the connection proportionally to correlated activation and rules that weaken the connection if the activations do not correlate. The success of evolving these rules, instead of using static ones, is shown by evolving a behaviour where a robot is required to stay near a

light source. Stanley et al. [130] evolved the same rules developed by Floreano and Urzelai [48], however, in their work the Hebbian rule was not divided to separate rules, instead a single learning rule that combined excitatory and inhibitory characteristics was evolved. The success was shown through a food foraging behaviour. Radi and Poli [120], used GP to evolve a general learning rule that works like a standard back propagation rule whereas [24] has evolved a global learning rule similar to a delta rule.

Other researchers [47, 139] have used GA to encode and evolve learning rules. A comprehensive survey can be found in [149].

2.4.2 Evolving control programs

EA variants such as GP and GEP are used to evolve computer programs. Unlike a GA, they do not encode solutions to a specific problem, instead they are used to evolve a computer program to solve a problem. Due to this capability, GP has been used to evolve robot control programs. GP, as previously detailed, uses functions and terminal sets. In ER, the function set is used to encode conditionals (such as IF), logical operators (e.g. AND, OR) and equality/inequality conditions (such as $=$, $>$, $\leq$, $\geq$). The terminals are used to encode robot sensors and motor actions. The GP algorithm is then used to evolve a control program that maps the sensors to specific motor functions.

Koza [74] was the first to evolve robotic behaviour using GP. In his work, GP was used to evolve a subsumption architecture that helped a robot develop wall following behaviours in an irregularly shaped room. The terminals used in GP corresponded to the robot sensors and a few additional terminals such as the edge and the minimum safe distance. The functions used include move forward (MF), move backward (MB), Turn Right (TR), Turn Left (TL) and two additional functions, one being connective and the other conditional. The fitness function used gave value to a robot touching some 56 tiles placed along the perimeter of the room [74]. The work used a population size of 1000, the evolution operators used were reproduction with 10% probability and a 90% crossover rate. The crossover points also had a probability of 90% for functions and 10% for terminals. The GP was allowed to run for 101 generations and the stopping criteria was either finding an individual that hit all the 56 tiles or all the 101 generations were met. At the 57th generation, an individual was able to achieve the maximum fitness function by hitting all the 56 tiles.

Lazarus and Hu [78] used GP to evolve a wall following behaviour. The main difference between this and Koza's work can be described in terms of the evolution operators used; in this work only crossover was used and the selection criteria is tournament selection. The stopping criterion was met when 100 generations or the maximum fitness value was achieved. This work shall be discussed in depth in a later chapter as this is the basis for comparison with GEP.

GP was used in [79] to evolve a robot controller for a set of goal-keeper behaviours for the simulation league of the Robocup competition. Their work investigates the conditions needed to be met for an evolved goal-keeper agent to perform adequately in a defensive

situation. The work involves the optimisation of multiple objectives and uses five fitness measures for the evolution. Their work uses a very high probability of crossover which as discussed earlier does not add much variation in the population. The probability of the mutation used, though low, may not have been adequate for the small population of the individuals used. Due to this, the results achieved shows that convergence was reached at two thirds of the fitness range, meaning that though a good set of behaviours were evolved they are still short of optimum fitness by quite a large margin.

In [148, 14], GP was used successfully to co-evolve robot controllers for three Khepera robots, that simulated escape behaviour from a room during emergency. In this work, GP used a different sub-population for each robot. A migration policy was maintained among the sub-populations so as to allow exchange of good problem solving individuals. This is an example of a mechanism to improve the ER methodology using cooperative co-evolution.

Different variations of GP have also been used to solve robotic problems. For instance, Nordin and Banzhaf [106] used a linear GP to evolve an obstacle avoidance behaviour. Pilat and Oppacher [115] used a similar linear GP to evolve obstacle avoidance, wall following and light seeking behaviours. In their work, comparisons are made using a tree based GP and a hierarchical GP, that is, a GP that uses additional structures to enable modularity. Here adaptive representation through learning (ARL), module acquisition (MA) and automatically defined functions (ADF) were implemented on the linear GP. Results showed that linear GP with ARL outperformed module acquisition HGP, ADF and tree-based GP. Compared to GEP, ARL is costly to implement as it extends the standard GP by the introduction of parameterised functions, these functions are then stored in a global library to be accessible to all organisms [115]. In GEP, however, functions are chosen as a set and it is very easy to generate valid organisms. Additionally, the sub-ETs can be used as ADFs [43] and encode a self contained structure, with the added advantage that this ADF undergoes normal genetic variations without destroying their structure. In GP, care has to be taken when performing crossover in order to preserve the structure of valid ADFs [74, 115].

GP has been used in [71, 147] to evolve robot path planning behaviours. The task of path planning assumes the existence or creation of a map. Thus, given a start point and end point, the robot needs to check a map in order to plan a path to reach the destination. GP and the mechanisms that evolve computer programs, translate sensors to motor functions and therefore, need to incorporate a map in order to evolve articulate path planners. The view taken in this thesis is that mechanisms used in GP for path planning cannot specifically be termed as path planning as they have not used maps, the robot uses reaction to obstacles and a properly designed fitness function to find its path to the goal. GA, on the other hand, has been used to encode a path and the task given to the algorithm is to optimize the solutions in order to arrive at; firstly a correct solution that moves a robot from start point A to end point B, and secondly, to optimise the shortest path. Examples where GA has been used in this domain can be found in [2, 19, 126, 63, 96]. Other examples where GP has been used in evolving behaviours can be found in [69, 80].

In summary, the use of standard GP in evolutionary computation and in ER can suffer from early convergence and the need to maintain the correct tree structure after every operation. The use of replication, though essential in maintaining best performing organisms, does not contribute towards variation in the population. There is also the issue of the expense of managing the tree depth, for example, Koza [75] tried to limit the depth to 4 in order to be able to compensate on the computer time spent. In [78, 79] the tree depth has been maintained to a depth of 10. The limitation of the depth of the ramified structure (tree in LISP), although necessary to reduce computation overhead and to avoid bloating, means that the program is limited in terms of functionality. Therefore, although the algorithm is able to come up with the desired programs, it is constrained to a certain degree and the terminals and functions has to be chosen carefully in order to get the desired outcome. This, however, does not correspond to the ideas of natural evolution where no measures are taken towards regulating evolution. A good system should be able to ensure the validity of evolved organisms automatically. In GEP the tree depth is determined by the head size and the functions being used. There is no need to maintain or regulate the depth of the tree as the non-coding region ensures that evolved tree structures are always correct after every operation.

2.4.3 Evolving intelligent behaviours using GEP

To the best of our knowledge, GEP has not yet been used in ER. However, GEP or a variation of GEP has been used to evolve or help intelligent agents in decision making.

Backtracking Parallel Gene Expression Programming (BPGEP) has been used to solve a robot path planning problem [119]. In this work, the genotype is divided into two chromosomes, one representing the functions and the other the terminals. The functions have a one to one mapping with terminals, implying that the functions have unary arity. The only genetic operators used are one and two point crossover and mutation. Chromosomes are said to evolve separately but simultaneously, hence the parallel nature of the algorithm. Despite the reference to GEP in the name of this algorithm, its operation is different to the execution of standard GEP as presented here. As previously described, the separation of terminals and functions is distinctive of GEP to other evolutionary computation algorithms. It is difficult to determine how these parallel chromosomes are converted to phenotypic trees in the standard manner. Finally, the operations performed on those chromosomes are restricted to the standard EA methods of crossover and mutation and do not incorporate the additional methods described in standard GEP.

Laszlo [77] has evolved an automatic plan design using GEP. The work is based on the classical agent-environment interaction (classical AI approach). As previously discussed, in classical AI, an agent evaluates the best decision by computing the current state to the final state after the decision has been taken. The agent starts by sampling the environment and creating an internal model of the environment. Next, the agent generates all possible actions and computes the resulting final state. Finally, the agent modifies its state once the action with the best final state is devised. In the redefined classical agent-environment interaction proposed by Laszlo [77], the agent senses the global reality first and then

computes an internal representation of the environment, this is known as a local reality. Since there is only one interpretation of the global environment by the agent, there can only be one local reality (internal model). However, many global realities can result in a similar internal model, the agent is thus permitted to form fantasies or alternative global realities. The agent then produces a population of condition-action chains (controllers or agent doubles [77]) which are evaluated using the fantasies. This condition-action chain forms a GEP chromosome. A chromosome is made up of perceptive actions (activate a different action if a given condition of the local reality occurs), cognitive actions (evoke certain actions if a given condition occurs in the global environment) and modifying actions (modify the agent's state). The GEP chromosome undergoes the normal genetic operations to find the best overall solution. The best condition-action chain, is one that has the best average performance in all fantasies. This approach was tested using two artificial agents test problems: the wumpus world and table world. In both cases good results have been recorded.

The concept proposed by Laszlo [77] uses GEP to extend the classical agent-environment interaction paradigm. The population of conditional-actions are generated by the agent in its interaction with the environment. Additionally, the agent constructs the fitness function after creating an internal model. Fitness of a condition action is assigned after computing its 'goodness' in the final state of the internal model. It is thus clear that although GEP is used to guide the process, it is only used partially with a view to only helping to optimize the chromosomes generated. The agents behaviour, just as in classical AI, is deliberative rather than reactive. Also worth noting is that a new population is generated in every newly created local reality, therefore, there is a very high computation cost in solving the problem. In comparison to Laszlo [77], this thesis seeks to explore how GEP can be used as the sole mechanism to evolve controllers for autonomous robots.

2.4.4 Evolution platform

This section highlights various platforms utilised while evolving controllers for an autonomous robot.

On-board evolution

On-board evolution refers to the process of carrying out evolutionary robotics on physically situated robots [104]. This is advantageous as the robot encounters the real world first hand. The world as noted by [17] is its own best model and hence the robot evolves not only the required behaviour but also strategies to deal with noise in the environment. However, the experimenter has to deal with various issues. Firstly, evolutionary runs take a very long time to complete: When evolution is carried out on a physical robot the program transfer and evaluations of the robot in the environment have to be done in real-time, this can raise a time overhead during the run. Secondly, energy supply to the robot needs to be sufficient during the run. This means that if the program is being transferred on a serial cable to the robot with power derived from a nearby power source or charging

station, the robot environment cannot be made to be very big in order to allow the robot to move back and forth to the power station [105]. Thirdly, the design of a fitness function requires the environment to be factored in as different encounters within the environment may be beneficial to the overall behaviour but they might be hard to predict prior to the run. Finally, some controllers (particularly in early generations) may produce behaviours, such as high speed collisions [105], that can damage the robot.

Evolving in simulation

Evolving with simulation involves carrying out the evolution using a simulated robot in a simulated world. The advantages with this is that it greatly reduces the computation time required for the evolutionary run, no energy supply is required for the robot, there is no need to worry about initial controllers destroying the robot and finally, since the robot environment is already known the design of fitness function is much easier. Simulated evolution, however, poses the challenge of transferring the learned behaviour to a physical robot. Since, no simulator can effectively model the dynamics of the real world due to noise in the environment and possibilities of new scenarios emerging, the evolved controller may fail to work or may exhibit behaviours that are not required. Nevertheless, there have been experiments conducted in simulations that exhibited required behaviours when validated on real robots [105]. This means that, with carefully designed simulators, transfer to a real robot can be easily accomplished.

Different types of robotic simulators with different capabilities and functions exists. Khepera[11] and Evosim simulators are 2D robot simulators developed for testing evolutionary robotics algorithms easily and fast either before testing the controllers on real robots or with the aim of analysing results of various algorithms with no particular need for on-board testing or implementations. Webots[12] simulator is a powerful 3D simulation package used to model, program and simulate mobile robots. The package provides various 3D robotic worlds that a developer can use by just developing the controllers for the robots, specifying the controllers and testing within the simulator. In addition, the simulator offers the capability to create different robotic worlds, different variety of robots and the capability to program the controllers. Webots incorporates most robot platforms including Khepera, Nao, Aibo, e-puck and Koala among others. In addition, the software offers the capability to port the controller to the physical robot. Simbad[13] is a 3D multi-robots Java simulator that enables the designer to write controllers, modify the environment and use the available sensors or build new ones. It is simple to use for studying situated AI, evolutionary robotics and machine learning. The simulator offers a test kit for AI algorithms for autonomous agents. An introduction to Simbad can be found on [66]. Another widely

[11]The original Khepera GP simulator was designed by K-Team (`http://www.k-team.com/`) and run on Linux systems using C/C++ for development, This original version can be found on `http://diwww.epfl.ch/lami/team/michel/khep-sim/`. A windows based Khepera simulator can be found on `http://www.pilat.org/khepgpsim/index.html`

[12]The Webots simulator is a commercial software developed by K-Team and can be found at `http://www.k-team.com/`

[13]This is a free robot simulator found on `http://simbad.sourceforge.net` under GNU General Public License

used simulator is Player/Stage/Gazebo[14]. Player is a network server for robot control and it provides a simple interface to robotic sensors and actuators over the Internet Protocol (IP) network. Player acts as a link between the robot (actuators and sensors) and the control program written by the developer. The control programs can thus be written in any language as long as they support a TCP (Transmission Control Protocol) socket. It is designed to support any number of clients at the same time. Gazebo and Stage are multi robot simulators for outdoor 3D environments; they have various capabilities such as simulation of standard robot sensors, sonar, scanning laser range-finders, GPS (Global Positioning System) and IMU (Inertial Measurement Unit), monocular and stereo cameras. Player/Stage/Gazebo is therefore a package incorporating Gazebo and Stage as the simulator and Player acting as a link to the physical robot. 3D simulators provide a better world model than 2D and therefore the simulated robots require to have vision capabilities and object proximity sensors.

Most experiments conducted in ER using either of the mentioned evolution platforms, utilise the standard EA technique [105]. The standard EA technique utilises one EA genome to encode an entire robotic problem. However, this mechanism is susceptible to various problems such as local minima. The following section discusses various techniques that can be utilised to improve the basic ER methodology.

2.5 Improving evolutionary robotics

The basic evolutionary methodology, also known as monolithic evolution, uses one module to map all the sensors of an agent to the actuators. The advantage with this technique is that there is no need for the designer to identify the sub-behaviours of the target behaviour nor how these sub-behaviours interact. Nevertheless, the technique has several shortcomings, they include: a) Local minima - this is an equilibrium point where there is no more increase in fitness yet the problem has not been solved. b) Bootstrap problems - this is a situation where there is no improvement of fitness due to randomness of the solution set [103, 105, 100, 90]. In this case there is no solution that can start off the evolutionary process and hence the problem cannot be solved.

There have been various mechanisms devised to improve the basic ER method. Some of these techniques such as incremental and layered learning solves the problem by dividing the task into a set of simpler tasks. Thus, they employ modularity analogous to subsumption architecture to solve evolutionary robotic problems. Multi-robot techniques such as co-evolution uses multiple robots. In this section, the mechanisms to improve the basic ER methodology are discussed.

2.5.1 Incremental evolution

Incremental evolution is a behaviour organisation method without explicit behaviour arbitration [142]. The approach involves the alteration of the environment in which the

[14]The Player/Stage/Gazebo package is provided freely on `http://playerstage.sourceforge.net/`

evolution takes place, as well as the fitness function. The evolution is a step by step process, starting by evolution of a basic behaviour and then progressing to more complex behaviour while changing the environment and the fitness function. The technique starts by dividing the required behaviour into simpler sub-behaviours. The controllers are then allowed to solve the sub-behaviours sequentially with increasing complexity. For instance if the behaviour targeted is wall following, the behaviour is divided into obstacle avoidance and navigation with wall proximity. The controllers are then allowed to evolve an obstacle avoidance behaviour until a set fitness level is achieved. Once this is is accomplished, the fitness function and the robot environment is changed and the controllers used to evolve navigation with wall proximity behaviours. Gomez and Miikulainen [55] have utilised incremental evolution to evolve a predator-prey behaviour, they also show that this behaviour could not be generated using direct evolution.

Bajaj and Marcelo [5] used an incremental approach to evolve a robotic controller that enables a Khepera robot to explore an environment while avoiding obstacles. The converged organisms are then introduced to a more complex environment and later on fine tuned to generate a wall following behaviour. In this work, a single layer neural network architecture is used with the weights optimized using a genetic algorithm. Their work draws largely from [46] where the floating point number representation is used in the chromosomes. The experiment is carried out in three stages on increasing complexity. In the first phase, obstacle avoidance behaviour is investigated. A rectangular environment is provided with an obstacle in the middle, the robot is encouraged to explore this environment without colliding with the obstacle. An obstacle avoidance fitness function is used. After the first phase, the surviving individuals are then taken to the next stage where a more complex environment is used. In the third stage where wall following behaviour is investigated, individuals from the last generation in stage 2 are used; a new fitness function is used. Good results were reported in obstacle avoidance stage and this carried on to the second stage where increasing complexity is used. However, in the third stage, though some good results were achieved, individuals taken from stage two appeared to "forget" obstacle avoidance behaviour they had achieved earlier whereas individual taken from stage one learnt how to follow the walls.

Whereas the incremental approach has been shown to evolve suitable behaviours, the step by step process employed means that the process is computationally expensive. Moreover, there is only one module used to solve the problem and its therefore impossible to distinguish which part of the genotype accomplishes which problem. Nevertheless, as reported by [138, 5, 55], this approach has been shown to outperform a monolithic ER implementation.

This thesis outlines a different style of improving the basic ER approach by use of modules to specify behaviour. The approach is different to incremental learning in that the behaviour is divided into layers which are solved concurrently.

2.5.2 Layered learning

Layered evolution, like incremental evolution, uses multiple fitness functions during the evolution. The difference is that the layered approach combines incremental and modularized evolution (evolution where more than one neural network is used) with elements of subsumption architecture, to achieve robot behaviour. The controller is thus a subsumption styled controller where each layer is an evolutionary neural network (ENN). The neural network layers are evolved in sequence so that the lowest layers are evolved first, after a desired fitness is achieved, then another layer is added on top [117, 136]. Once a set fitness is achieved, its development is stopped and the cycle repeated until the required number of layers is achieved. Work presented in [25], investigated the use of genetic algorithms, the Nelder-Mead technique and policy gradient methods in the evolution of layers in a layered learning approach. In their work they tried to learn optimal parameters for basic routines, with behaviours and strategy selection for a robot involved in the robocup soccer competition. In [44] a layered approach has been used in parameter fine-tuning in order to evolve a grasping behaviour. Layered learning has also been used in [58, 59].

Whereas the layered learning divides the required behaviours into sub-behaviours, their development is stopped once a set fitness level is achieved, this means that if the autonomous agent encounters a new situation, its performance may be hindered. In addition, this algorithm requires multiple fitnesses to guide the evolution: Since evolutionary computation in general and evolutionary robotics in particular takes a long time to run, additional fitness evaluations add computing overhead during the run.

This thesis focuses on developing a modular based evolutionary robotics algorithm that requires only one fitness function and where each module continuously learns during the course of evolution. This is particularly desirable as it mimics natural biology where organism's functionality evolves concurrently.

2.5.3 Co-evolution

Co-evolution refers to the simultaneous evolution of more than one population. Co-evolution could be cooperative where two or more populations of the same species evolves together to solve a particular problem [14, 148]. Alternatively, it could be competitive where two or more populations of different species are evolved together with coupled fitness. Both techniques improve the basic ER methodology in different ways.

In cooperative co-evolution, the task(s) to be solved is the same for all the organisms [8]. Each robot/autonomous agent uses a different population but of the same species. The benefits of this technique is that migration of individuals across the populations is allowed with the hope that better solutions can be moved across the populations. This, in turn, helps to progress the evolution and can potentially prevent local minima. This technique is similar in nature to the implementation of island models [127] in evolutionary computation.

In competitive evolution, two or more populations of different species are used, e.g. preda-

tor and prey. The idea is that as one species evolves, it creates an increasing challenge to the other species requiring more complex solutions to be evolved. At the start of the evolution, the behaviours displayed are generally poor, however, as the evolution progresses, one species evolves a better strategy to evade the other species. This, in turn, leads the competing species to evolve a better strategy. The cycle is repeated as evolution progresses, eventually producing an evolutionary "arms race" [105]. The controllers' evolved using competitive co-evolution exhibit more general behaviours as they have encountered different generations of the competing species requiring them to have different strategies in different situations. Moreover, the changing search space driven by the changes occurring in the competing species could prevent local minima.

Although co-evolution is very interesting and has shown to drive evolution to better results, it is beyond the scope of this thesis, hence, it shall not be considered further. Literature on robot co-evolution can be found on [104] and a discussion in natural co-evolution can be found on [32, 31].

2.6 Conclusions

Developing a suitable control system for an autonomous robot, to carry out tasks in unstructured environments, is a challenging and difficult task. The control development approaches described in the first section show positive results in various aspects of control development, while at the same time shows shortcomings in different areas. Systems that use deliberative techniques (hierarchical paradigm) require the engineer to create a large database of different scenarios that a robot might find itself in. However, it is very difficult to know in advance the changes that might occur in a dynamic environment. As mentioned previously, these approaches are not suitable for systems that work in unstructured environments. Although purely reactive techniques are more successful than purely deliberative ones, they require the designer to implement all the behaviours required in carrying out the tasks that the robot may be needed to perform. Since the design and implementation of these behaviours occur before the deployment of the robot to the environment, it is not known in advance what changing circumstances the robot might need to undergo, as well as other sub-tasks that might arise when the robot is executing its main task. Another shortcoming of this method is that since motor schemas are implemented as a mapping from the perception schema, tasks that require the robot to plan in advance or build a strategy cannot be implemented. Moreover, creating a large repertoire of behaviours to cover all angles is a daunting task. Hybrid techniques present a better approach, however, the engineer is confronted with the task of designing a planning component that directs which behaviour to execute. Furthermore, the engineer is required to design a set of necessary behaviours. The approach has the advantages of deliberative and reactive techniques but carries their shortcomings as well. Approaches that use reactive techniques and combines learning mechanisms have been more successful in implementing control systems; this illustrates the benefits that arise from combining the two approaches. On the other hand, the complexity of designing a robotic behaviour and training the robot control to handle various situations in an unstructured environment persists. The use of

evolutionary robotics gives an alternative in designing robot controllers by enabling automatic programming. In this technique, the designers' work is restricted to providing a fitness function and algorithmic parameters (characteristics of required behaviour and EA parameters). ER techniques can be used to train a pre-specified neural network controller, design and train a neural controller or the controller can be a computer program. It has also been shown that evolution of a robot morphology is possible [105, 84].

Research in ER where genetic algorithms are utilised, involves the evolution of a neural network robot controller. Since GAs encode solutions to a problem, they are adequately suited to encode parameters such as weights, threshold values and parameters describing whole network architectures. The use of GA and ANN, employs both evolution and learning and generally leads to very good results. Although ANN can be seen to aid with the controller learning, the evolution is also affected by local minima problems. It is important to highlight that a GA encodes and evolves solutions to the problem being investigated; on the other hand, GP and GEP evolve a computer program or a strategy to solve the problem. From this view, a GA without hybridization with neural networks, cannot be used in problems where changes in environment are imminent. Robotic environments are dynamic in nature and even where the environment is static, the robot needs to take inputs from the environment, process the inputs and give out an output as an action that affects both the robot and the environment. GP and GEP are more suited to deal with these kind of problems.

Research carried out using GP requires the programmer to constrain the tree depth in order to avoid creation of non valid trees as well as to avoid bloating. This is quite cumbersome and the work will be seen to have a human influence. Additionally, GP can suffer from local minima problems because of the use of very high probability of crossover and very low probability of mutation.

As previously stated, GEP has not previously been used in the problem of ER to the best our knowledge. However, this new approach looks very promising due to these facts: firstly, GEP unlike GP can easily be modelled as the operators work on the linear chromosomes, analogous to GA, which makes it easy to perform genetic operations on the chromosome at the same time maintaining the validity of the organisms. Secondly, GEP encompasses several evolution operators and it can be said to be more 'nature inspired'. Operators like transposition and mutation are key evolution operators that can lead to very good results. Thirdly, multigenic chromosomes can be utilised either to evolve one behaviour with two or more genes effectively contributing to the overall emergent behaviour or using two or more genes to evolve different tasks forming one global behaviour. This capability can be utilised in developing complex behaviours with genes solving simple sub-behaviours. This is comparable to solving multiple problems simultaneously.

Most robotic implementations require the use of multiple behaviours in order to accomplish a particular task. Therefore, robot controllers require the capability to solve more than one task. As discussed earlier, the capability to solve more than one task can easily be accomplished using modular controllers. As described in section 2.3.5, GEP offers the capability to encode multigenic chromosomes (mgGEP). This thesis shows that these

multigenic chromosomes can be utilised to encode modular robot controllers. Additionally, the mgGEP approach is extended further to show that behaviour sub-division as well as coordination can be evolved using GEP. This is an important advance not only in robotic control but also in optimisation problems.

The rest of this thesis describes the experiments carried out in evolving robotic behaviours using GEP. The experiments carried out in Chapter 3 addresses the first two research questions posed in section 1.1.1. Additionally the chapter provides support for the first claim in section 1.1.3. In the reported experiments ugGEP chromosomes are utilised to evolve obstacle avoidance behaviours. Additionally, multiple output GEP chromosomes (mgGEP-multiple out) are utilised to evolve obstacle avoidance behaviours. The evolved obstacle avoidance controllers are then tested in new environments. Results achieved in these experiments are analysed and a comparison made to determine which technique is more robust. Following the results achieved in Chapter 3, Chapter 4 utilises mgGEP chromosomes to solve a robotic wall following problem as well as a food foraging problem. The experiments reported in Chapter 4 show how GEP can be utilised to evolve a modular controller by utilising the linking function to provide behaviour arbitration. Chapter 4 thus provides an answer to the third research question as well as support for the second research claim. Following the gene regulation discussion on section 2.3.5, Chapter 5 shows how the mgGEP linking functions can be evolved with the rest of the GEP genome. In addition, Chapter 5 shows that a gene in a multigenic set can be utilised to evolve conditions under which gene regulation in a mgGEP chromosome is achieved. Chapter 5 thus extends the standard GEP algorithm by evolving mechanism for gene regulation. Additionally, Chapter 5 contributes an answer to the fourth research question as well as provide support for the third research claim. Finally, Chapter 6 draws a conclusion on the research experiments reported in this thesis and highlights the plans for further work.

3 Evolving Behaviours using GEP

This chapter describes the evolution of robot behaviours using gene expression programming (GEP). The main aim of the described experiments is to explore the capability of unigenic GEP chromosomes (ugGEP) and multiple genes GEP chromosomes (mgGEP) in evolving monolithic controllers. Monolithic or standard evolution is an evolutionary mechanism used by genetic algorithms (GA) and genetic programming (GP) to evolve one-dimensional controllers that solves a given problem without sub-division. The ugGEP chromosome works in a similar way and is similar in structure to GP, in contrast mgGEP is a new mechanism of evolving solutions to problems using more than one gene in a chromosome. This chapter seeks to explore the capabilities of ugGEP and mgGEP in evolving monolithic controllers as is common in the evolutionary robotics domain.

An autonomous robot should be able to explore its environment without colliding with obstacles and causing harm to itself or people. As described in the previous chapter, there are many studies that have been conducted on developing obstacle avoidance behaviours using GAs and GP [74, 105, 106, 115]. Generally, hand-coding these behaviours takes a lot of programmer's time and cannot be guaranteed to generate a robust behaviour that is able to generalise easily when confronted by new circumstances in the environment. Evolutionary Robotics (ER), however, using a population of control programs has been shown to generate behaviours that are more robust than human coded ones [75, 60, 61, 46, 105, 45]. Furthermore, these evolved behaviours have been shown to adapt successfully in new environments [105].

In this chapter, a navigation and obstacle avoidance behaviour is implemented and evolved using GEP. In addition, the evolved controllers are tested for adaptation into new environments. Their performance is evaluated and compared to the fitness in the training environments. The rest of the chapter is divided as follows: Firstly, an introduction of the obstacle avoidance problem and related work is presented. Secondly, an implementation of obstacle avoidance behaviour using ugGEP is presented and a discussion follows. Thirdly, the evolved controllers are tested in previously unseen environments and their performance evaluated. Fourthly, mechanisms of evolving monolithic controllers using ugGEP algorithm that utilises the sensor readings to determine robot wheel velocity, is discussed and an example shown. Fifthly, techniques to evolve monolithic controllers using mgGEP is discussed and an example shown. Finally, a discussion of the viability of GEP in evolutionary robotics follows and a conclusion is drawn.

3.1 Obstacle avoidance with straight line navigation

Straight line navigation with obstacle avoidance is a basic behaviour that an autonomous robotic system needs to display. In the absence of this behaviour the robot may not be able to navigate its environment without human control, and hence cannot be truly autonomous. As a result, straight line navigation with obstacle avoidance is used as a test case in robotic behaviour as additional behaviours cannot be implemented if the robot cannot move. In this task a robot is placed in an environment with some obstacles and it is required to travel the longest possible distance without colliding with the obstacles, where obstacles can be static or dynamic. The robot uses infra-red sensors or stereo vision to determine its distance from an obstacle and uses its motors and wheels to move around.

The task of navigation and obstacle avoidance requires a suitable mapping between the sensors and motors. The actions taken by the robot in accomplishing the task, requires the robot to act given certain sensory information. The behaviour thus requires reactivity to obstacles as stimuli. Though simple in nature, this task requires the robot to perceive different sensory situations and determine what action to take. In an unstructured environment, it is not possible to list all different situations that a robot might encounter and develop a suitable motor action for. Evolutionary algorithms (EA) are suitable for a task like this due to their use of a population of programs or solutions (in the case of GA). With EA, the population of different programs or individuals are optimized in order to generate the best program (controller) to control the robot.

3.1.1 Related work

There have been a number of studies conducted involving the evolution of navigation and obstacle avoidance behaviours. The work reported in this chapter is closely related to the obstacle avoidance behaviour experiments in Pilat and Oppacher [115], where the authors investigated the performance of various hierarchical GPs in evolving obstacle avoidance behaviour. The set of GP variations tested included adaptive representation through learning (ARL) GP, module acquisition (MA), automatically defined functions (ADFs) GP, tree-based GP and linear genome GP. In all the GP implementations investigated, the computer programs (individuals) used the function-set constituting the arithmetic operators ADD, SUBTRACT, MULTIPLY, shift right (SLR), shift left (SLL) and the logical operators, AND, OR, XOR. The terminal set was made up of variables denoting the robotic infra-red sensors (s_0–s_7). The tournament selection was used with a selection pool of 7 and a two point crossover was used for the modification of solutions. The environment used was an irregular $70cm \times 90cm$ room with multiple angles which acted as dead ends. Additionally, an obstacle was also placed in the middle of the room. In this work and in Nordin and Banzhaf [106] the obstacle avoidance problem was solved as a symbolic regression problem where the fitness used to guide the system during evolution was a summation of the expected values versus the actual values generated by the robot. Thus, the following function was targeted:

$$f(s_1, s_2, s_3, s_4, s_5, s_6, s_7, s_8) = (m_1, m_2) \tag{3.1}$$

Where s_i represents the set of sensors and m_i represents the set of motors.

The function thus modelled a simple stimulus-response behaviour of the robot. Motor speeds in this case were generated directly from the sensors and thus from the interaction with the environment. The fitness function used is as shown by Equation 3.2.

$$fitness = \alpha(\mid m_1 \mid + \mid m_2 \mid - \mid m_1 - m_2 \mid) - \beta \sum_{i=1}^{8} s_i \tag{3.2}$$

The m_i refers to motor values (in the range -10 to 10) and s_i refers to proximity sensors values (in the range 0-1023). α and β are coefficients whose values were set to 10 and 1 respectively (see [115] for explanation with reference to values used). The first part of the fitness formula encourages the robot to move in a straight line while the second part penalised the robot for object proximity.

Work presented in this chapter is related to [115] though we do not optimize the behaviour as a symbolic regression problem. Instead the fitness formula that we use awards explorations to new areas and penalties for hitting obstacles. Nevertheless, the output of the GEP algorithms generate the motor behaviours in a similar way to the representation discussed above. As previously discussed, the work presented in [115] uses hierarchical GP (HGP) to solve obstacle avoidance behaviour. The results reported showed that ARL HGP outperforms MA HGP, ADFs GP, linear GP and tree-based GP. Compared to GEP, ARL is costly to implement due to the fact that it extends the standard GP by the introduction of parameterised functions. These functions are then stored in a global library to be accessible to all organisms. In GEP however, functions are chosen as a set and it is very easy therefore to generate valid organisms.

Additional examples of work involving the evolution of straight line navigation and obstacle avoidance can be found on [105].

3.2 Using GEP to evolve an obstacle avoidance behaviour

As previously mentioned, the main aim of these experiments was to investigate the viability of GEP in evolving simple robotic behaviours. To start with, an obstacle avoidance behaviour was investigated. Obstacle avoidance requires the development of suitable mapping from sensory data to motor actions, thus describing behaviour. The task is that of minimizing the values of the infra-red sensors while the robot performs exploratory behaviour.

This section outlines the required implementation and algorithm set-up and lastly presents the experimental results.

3.2.1 Robot and environment implementation

In this experiment, the *simbad* simulator[1][66] was used to simulate the robot and its environment. The simulator is used in background mode to avoid rendering the environments during the course of the run.

In all experiments where the Simbad simulator is utilised, the simulated robot is made up of a cylindrical body with a radius of 0.3 metres and a height of 0.5 metres. The robot mass is given as 50 kilograms. Additionally, the robot uses eight infra-red sensors placed 0.25 radians apart along the perimeter of the top of the robot. The robot movement is achieved using a simulated pair of wheels (left and right). Two methods are offered by simbad simulator to control the robot movements. These are:

[a] **default kinematic:** In this kinematic model, the robot movements are controlled by specifying rotational and translational velocities. Rotational velocity, given in radians per second, specifies the amount of rotation that a robot should undergo while translational velocity, given in meters per second, specifies the speed of linear motion.

[b] **differential drive kinematic:** In this kinematic model, the left and right wheel velocities are controlled independent of each other. The difference of the left and right wheel velocity produce an angular/rotational velocity while an equal output gives a linear velocity.

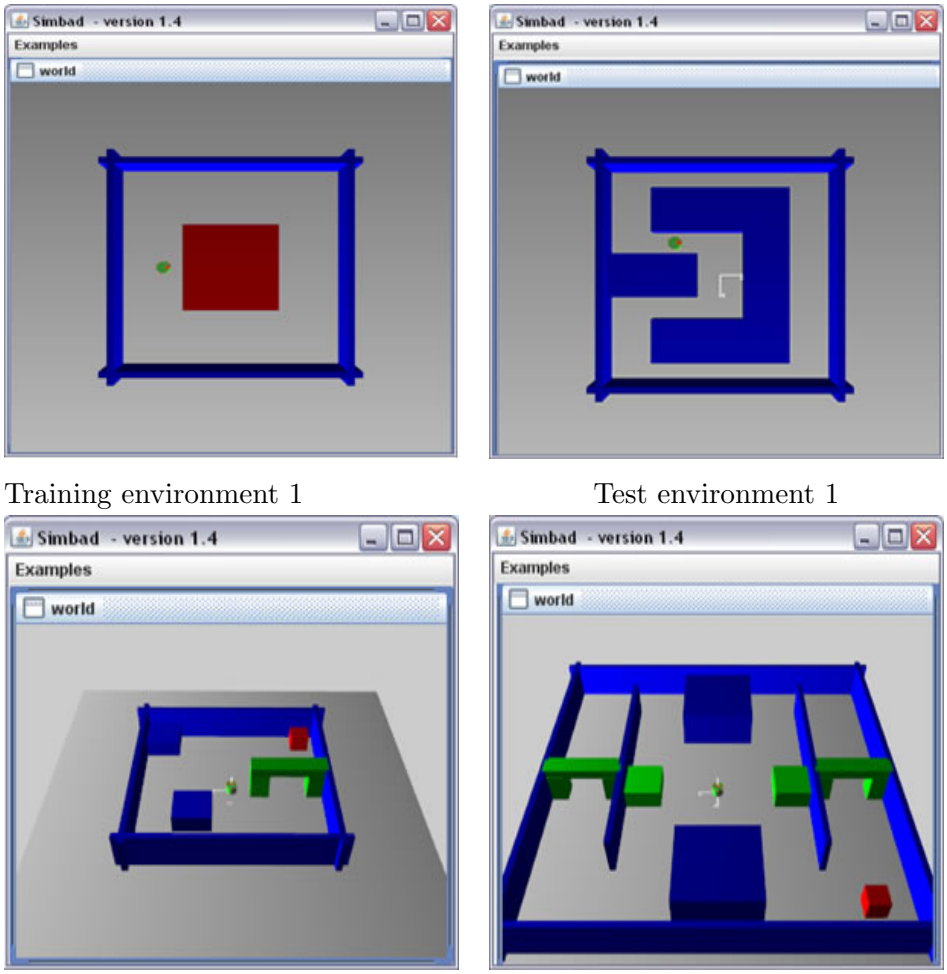
In the initial experiment, the robot movements were controlled using the default kinematic. The robots translation velocity was set manually to $2.0m/s$ for forward movement and $-2.0m/s$ for reverse movement. Therefore, the robot can move forward or back with aforesaid translation velocity and can also turn on its axis based on angles in Table 3.1. The robot uses 8 sensors, i.e. Front (F), Front right (FR), Front left (FL), left (L), Right (R), Back (B), Back right (BR), and Back left (BL), placed on specific angles as shown in Table 3.1. The robot sensors return a value between $0.0m$ and $1.5m$. A value of $0.0m$ means that the robot is adjacent to the wall, whereas a value of $1.5m$ means that the robot is a minimum of $1.5m$ away from the obstacle. The aim is thus to evolve a mechanism to map various sensor readings to specific motor functions.

Table 3.1: Terminal set and sensor positions

Symbol	**Represents(Terminal)**	**Represents(Motor)**	**Sensor position (Radians)**
F	Front Sensor	Move Forward at $2.0m/s$	0
B	Back Sensor	Move Back at $-2.0m/s$	π
L	Left Sensor	Turn Left	$\frac{\pi}{2}$
R	Right Sensor	Turn Right	$\frac{3\pi}{2}$
FR	Front Right Sensor	Turn Front Right	$\frac{7\pi}{4}$
BR	Back Right Sensor	Turn Back Right	$\frac{5\pi}{4}$
FL	Front Left Sensor	Turn Front Left	$\frac{1\pi}{4}$
BL	Back Left Sensor	Turn Back Left	$\frac{3\pi}{4}$

[1]Free download of simbad simulator can be found on http://simbad.sourceforge.net/

To perform the required experiments, four types of environments were used: two training environments and two test environments as shown by Figure 3.1. Training environment 1 is set up using a $10m \times 10m$ box with one square obstacle within it. Test environment 1 is a $15m \times 15m$ box with a U shaped obstacle covering most of the space (see Figure 3.1 Test environment 1). Training environment 2 is a $12m \times 12m$ box while Test environment 2 is a $20m \times 20m$ box with a more complex array of obstacles as shown in Figure 3.1.



Training environment 1 Test environment 1

Training environment 2 Test environment 2

Figure 3.1: Training and test environments

3.2.2 Algorithm parameters

The algorithm uses the following attributes to describe the terminal and function sets.

Function set

Only one function operator was used in these experiments : IFLTE ('If Less Than or Equal to'). This is a condition function which takes four arguments, w, x, y and z. The function performs a comparison on the evaluation of the first two sub-trees (less than or equal to ($\leq$)) and then executes the third sub tree if the comparison is true, or the fourth sub-tree

if the comparison is false. Figure 3.2 shows an example of a controller consisting of this function only.

Terminal set

Table 3.1 shows the terminal sets used in the experiment. The terminals represent different meaning depending on which part of the tree they are positioned in. As previously stated, the sensor terminals return a floating point value in the range of $0.0m$ and $1.5m$ to the algorithm. The symbols when executed on the right hand side of the tree produce a locomotive action while the left hand side returns the specified sensor readings. The motor terminals; L, R, BR, BL, FR, FL turns the robot with the number of radians shown on Table 3.1 while F moves the robot forward and B moves the robot back, i.e. the algorithm is set up such that similar alphabets are used for the sensor and motor functions, where the left part of the IF statement is the premise and the right hand side is the consequent.

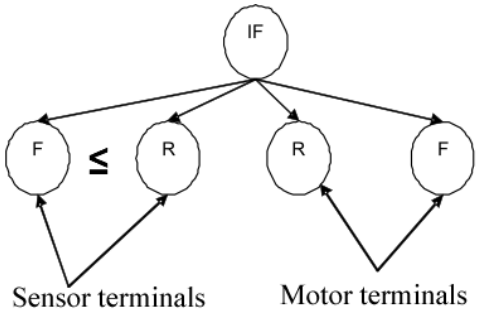


Figure 3.2: Example of a potential robot avoidance controller using terminals and functions as reported on Table 3.1.

Genetic operators and algorithm parameter values

All the GEP operators are used in the experiment with the probabilities set as per Table 3.2

Table 3.2: General algorithm parameter settings

Parameters	**GEP setting**
Maximum generations	100
Population size	50
No. of genes	1
Head size	4
Functions(IFLTE)	1
Terminals(see Table 3.1)	8
Mutation probability	2/chromosome size^2
1-Point Recombination Probability	0.7
2-Point Recombination Probability	0.2
IS Transposition probability	0.1
RIS Transposition probability	0.1
No. of randomly seeded runs	20

As described in section 2.3.5, GEP uses seven genetic operators. These operators are; mutation, 1-point recombination, 2-point recombination and gene recombination, insertion sequence, root insertion sequence and gene transposition. However, the experiments discussed in this section focus on evolving behaviours using unigenic GEP. This algorithm does not use gene recombination and gene transposition due to the fact that they are meaningless for single gene chromosomes, and hence are not implemented. The values of genetic operators probabilities, shown on Table 3.2 above, are derived from work carried out by the author in [93]. In the reported work, a simple feedback heuristic was utilised to adapt genetic probabilities depending on how well a certain value contributed towards the solution to a problem. In the achieved results the values used here were shown to generate the best performances. Similarly, these values generated best performances when utilised in [94, 38].

The algorithm is generational and uses replication to pass the best organism in the previous generation. Roulette wheel selection is used to select parent organisms.

Fitness function

The fitness is calculated as the summation of all new coordinates that a robot visits less any collisions. The robot is also penalised for remaining stationary. The following fitness function is used.

$$fitness = (\sum_{i=1}^{n} p_i(x_i, z_i)) - C - S \tag{3.3}$$

Where $\mathrm{p_i}(x_i, z_i)$= 1 whenever (x_i, z_i) has not previously been visited and 0 otherwise. In the experiment, the number of robot steps, n, is set to 1000 and the collision penalty, C, is set to 25. The stationary penalty, S, is set to 50 and is only applied if the robot does not move from initial starting point.

Fitness is awarded for every time-step and the maximum fitness is thus set to 1000. Lowest fitness is -50 which means that the robot did not move from the initial starting point.

3.2.3 Experimental results

In the initial experiment, the ugGEP was run using a chromosome with a head size *h=4*. A population of 50 individuals was run for 100 generations and the algorithm evaluated over 20 randomly seeded replications. The rest of the parameters were set as shown on Table 3.2. The experiment was carried out using environment test 2 (see Figure 3.1) as the simulated robot environment. Each individual controller was allowed a maximum of 50 virtual time seconds (1000 simulation steps) to control the robot. The results for this experiment are shown by Figures 3.3 and 3.4

[2]The chromosome size refers to the total number of string symbols in the chromosome. This is given by summation of tail and head lengths, see Equation 2.2

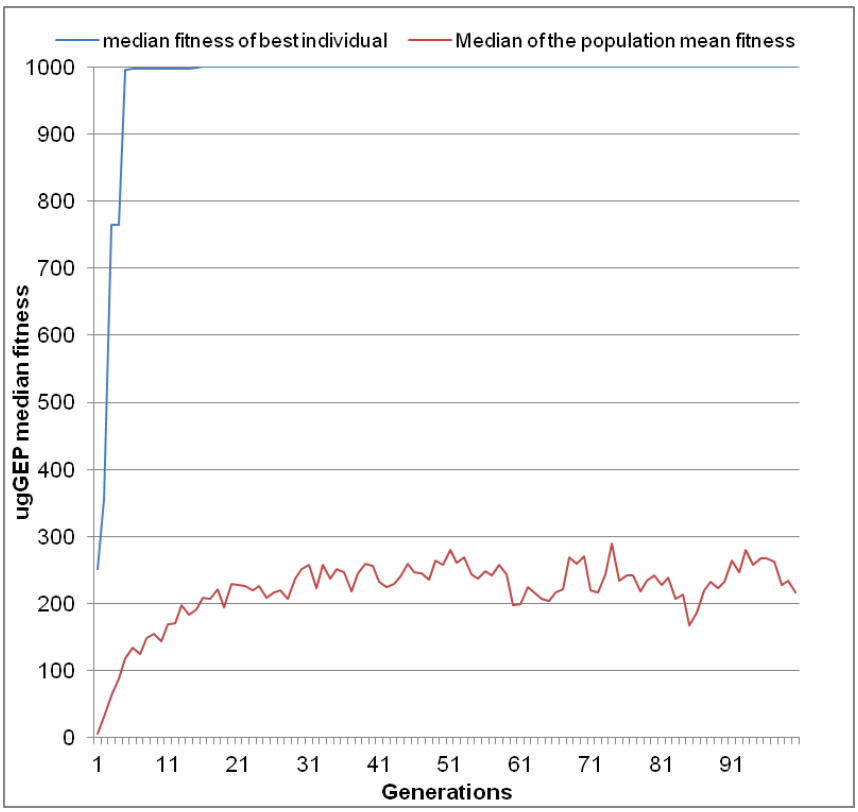


Figure 3.3: Progression of the median fitness of the best individual and the median of the population mean fitness over the generations. The median of the population mean fitness was computed from the population mean fitness achieved in each generation in all the 20 randomly seeded runs. Similarly, the best individual average performance is derived from the 20 randomly seeded runs. In each evolutionary run, a population of 50 organisms was evolved for 100 generations and the population mean and best individual fitness recorded in each generation. The rest of the parameters are as shown on Table 3.2.

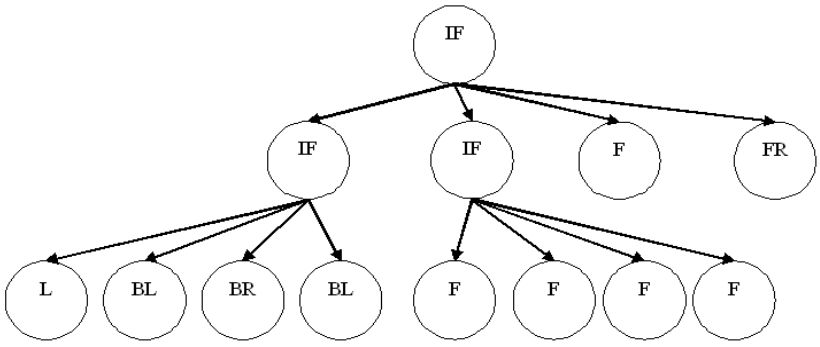


Figure 3.4: Evolved obstacle avoidance controller with parameters listed on Table 3.2. The controller was evolved in environment test 2 as shown in Figure 3.1

Figure 3.3 shows the progression of the median population mean fitness and the median fitness of the best individual in the population, along the generations. Additionally, Figure 3.4 shows an example of the tree structure of a single obstacle avoidance controller evolved in this experiment.

To control the robot using the controller shown in Figure 3.4, the robot starts by checking

obstacle proximity on the left hand side. If an obstacle is detected, the robot then checks obstacle proximity on the back-right area (the area between the right and back sensors). If there are obstacles on the back-right area, a comparison between the back-right and front sensor readings is done. If the front sensor reading is lower than the back-right sensor, the robot moves forward otherwise the robot makes a 45 degree rotation towards the right hand side; that is, FR. Alternatively, if there is no obstacle on the left hand side and robot is close to obstacles on the back-left area, a comparison between the back-left and front sensor readings is done. If the front sensor reading is lower than the back-left sensor, the robot moves forward, otherwise the robot makes a 45 degree rotation, towards the right hand side, to avoid obstacles. This control shows that the controller solves the problem in a logical manner, similar to a human approach. The described controller shows that GEP is a plausible technique to be used in ER applications.

As observed in Figure 3.3, the ugGEP evolves solves the obstacle avoidance problem within 15-20 generations. Nevertheless, the median of the population mean fitness shows that in general the rest of the individuals in the population are moving very slowly towards the maximum fitness. To better understand this, additional experiments were carried out in order to investigate whether the length of the chromosome affects the performance of the algorithm in solving the problem. The next section describes these experiments and the obtained results.

Effect of head sizes to the success of controllers in training environment

The goal in this experiment was to test the effect of varying the head size and hence the genotype length of the chromosome. A population of 50 individuals was run for 100 generations and the algorithm performance evaluated over 20 replications. The default kinematic was used to control the robot movements. The translational velocity was set at $2.0m/s$ if front movement **F** was returned or $-2.0m/s$ if back movement **B** was returned. Each individual controller in a population was allowed a maximum of 50 virtual time seconds (1000 simulation steps) to control the robot. The robot was then returned to the start point and the next controller evaluation carried out. The remaining parameters were as shown in Table 3.2. These experiments were conducted using test environment 2 (Figure 3.1).

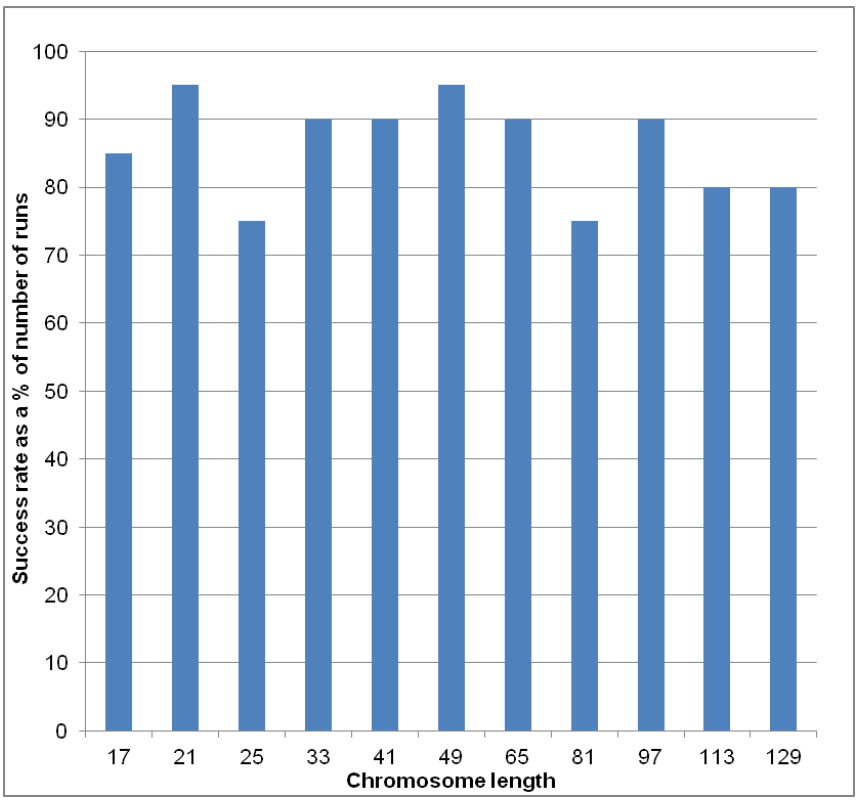


Figure 3.5: Success rate with varying chromosome length. Success rate refers to the number of runs which achieved maximum fitness (1000 steps) out of the 20 randomly seeded evolutionary runs conducted. In each evolutionary run, a population of 50 organisms was evolved for 100 generations and the best individual performance recorded in each evolutionary run. The rest of the parameters are as shown on Table 3.2.

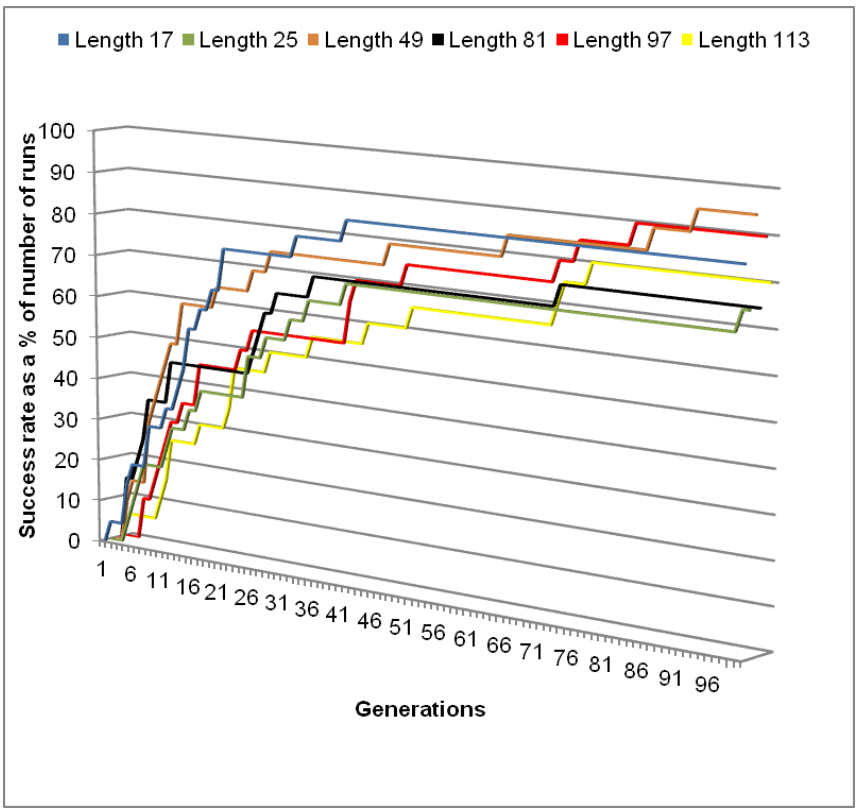


Figure 3.6: Variation of success rate with number of generations. A comparison using different chromosome lengths is shown. Success rate refers to the number of runs which achieved maximum fitness (1000 steps) out of the 20 randomly seeded evolutionary runs conducted. In each evolutionary run, a population of 50 organisms was evolved for 100 generations and the best individual performance recorded in each evolutionary run. The rest of the parameters are as shown on Table 3.2.

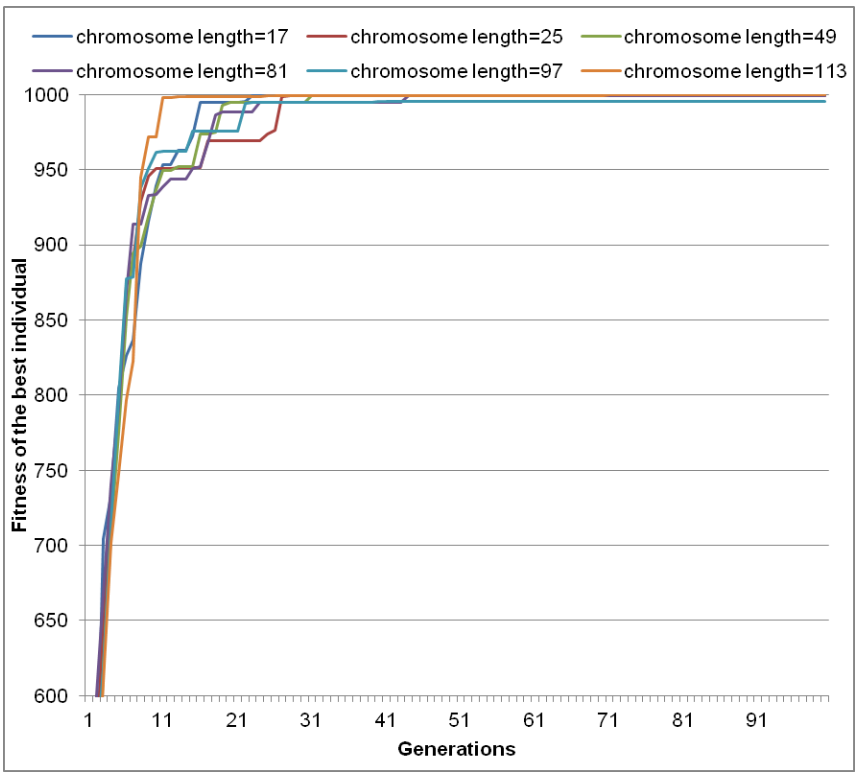


Figure 3.7: Progression of best individuals in the population over the number of generations using different chromosome lengths. Each individual curve represents the best performing evolutionary run out of the 20 randomly seeded runs. In each evolutionary run, a population of 50 organisms was evolved for 100 generations and the best individual performance recorded in each evolutionary run. The rest of the ugGEP parameters are as shown on Table 3.2.

Figure 3.5 shows the variation of success rate of organisms with chromosome length (determined as a percentage of those individuals that achieved the maximum fitness). Figure 3.6 shows variation of success rate with the number of generations. In addition, Figures 3.7 and 3.8 show the progression of the best individual in a population and the average fitness of the population over the number of generations respectively.

The comparison of success rates, Figure 3.5, shows that there is at least a 75% likelihood of evolving an obstacle avoidance behaviour with any of the presented chromosome lengths using the ugGEP algorithm. However, it does not show the overall performance of the algorithm in solving the problem during the course of the run. Moreover, it is very hard to determine the best chromosome length using this analysis. Figure 3.6 is possibly a better analysis as it shows the speed, in terms of generations, that the algorithm finds successful individuals. In this analysis, chromosomes with shorter phenotypes appear to achieve better success rates in early generations. However, after more generations the performance of the shorter chromosomes is improved upon by the longer chromosomes. This is to be expected as the longer chromosomes have a large search space and therefore requires a longer evolutionary run to generate a solution.

Figure 3.7 shows that the performance of best individual evolved using longer chromosomes was better than for the controllers evolved using shorter chromosomes. Additionally, Figure 3.8 shows that in terms of average fitness within the population, longer chromosomes

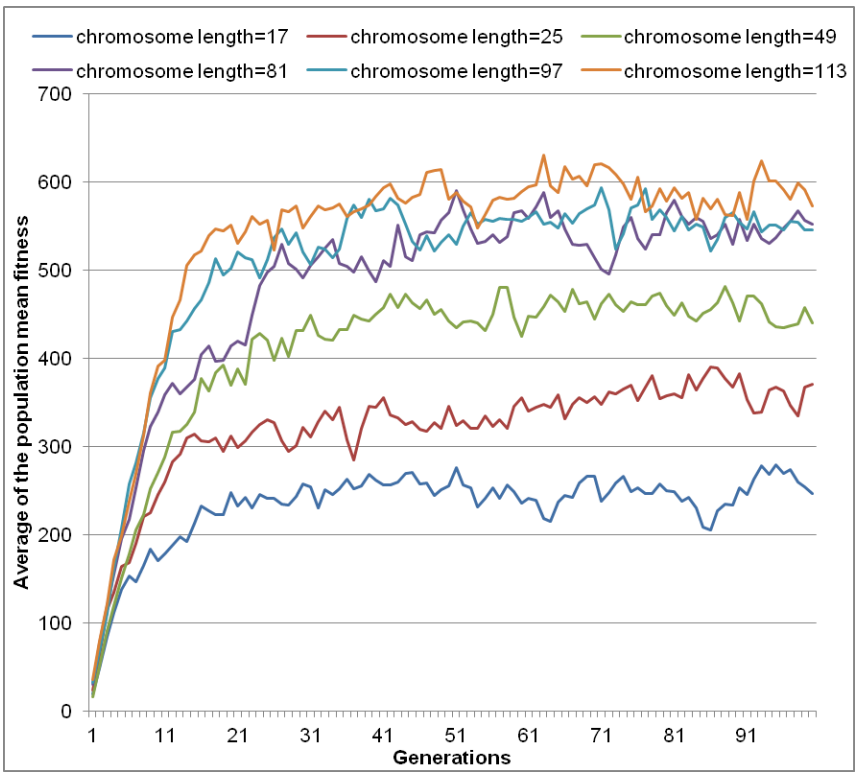


Figure 3.8: A comparison of the average population mean fitness across different chromosome lengths. The average of the population mean fitness was computed from the population mean fitness achieved in each generation in all the 20 randomly seeded algorithms. In each evolutionary run, a population of 50 organisms was evolved for 100 generations and the population mean fitness recorded in each generation. The rest of the parameters are as shown on Table 3.2.

perform better overall than shorter chromosomes. The performance of the longer chromosome in this problem, is likely to be as a result of an increase in diversity created by additional materials in the longer individuals. In GEP, since it is the coding region that determines the performance of the chromosome, genetic operators that act on the head region influence the overall individual performance greatly. There are five genetic operators acting on the chromosome (see Table 3.2). All these operators, apart from mutation, create diversity by shuffling elements of the individuals in the population. Mutation on the other hand, can occur in both head and tail regions. Chromosomes with a longer head length (that is, longer chromosomes) have a lower probability of mutation occurring at the root of the chromosome, hence reducing any negative effect associated with mutation and preserving better performing individuals in the process.

3.3 Adaptation to new environments

As shown in Figure 3.3, the algorithm is able to generate controllers which explore for 1000 simulation steps with no collisions within $15 - 20$ generations. Therefore, the following experiments concentrate on the ability of these controllers to generalise to a new environment and to discover whether the length of the phenotype of the controller has an effect on this ability. In effect, the aim of the experiment is to investigate the principle of Occam's

razor; that is, if two theories fit the problem, then the simpler theory is better [6]. In the experiments reported here, Occam's razor is tested using ontological simplicity; that is, whether the most parsimonious controllers that have been evolved, adapts best when introduced to previously unseen environments. Table 3.3 shows the specific parameters that were used in all the adaptation experiments. The rest of the algorithm parameters are as shown on Table 3.2.

Table 3.3: Adaptation test: algorithm parameter settings

Parameters	**GEP setting**
Maximum generations	100
Population size	200

The default kinematic was used to control the robot movements. The translational velocity was set at $0.8m/s$ for forward movement or $-0.8m/s$ for reverse movements. The slow robot speed was used in the experiment to ensure that the robot had enough time to react to sensor readings as well as to learn more about the environment. In the first two experiments reported below, the best unique controllers (*fitness = 1000*), were selected in the course of a GEP run and recorded. These best controllers were then tested in a new environment and their performance recorded. The resulting fitnesses were grouped into 50's and the midpoints versus their percentage frequency compared.

3.3.1 Effect of training and testing controllers in similar environments

The goal in this experiment was to determine whether the length of a controller affects its performance when the training and test environments are similar. Training and test environment 2 (shown in Figure 3.1) were used as the training and test environments respectively. The environments shown above have similar layouts and differ only in size and number of obstacles (training environment 2 is $12m \times 12m$ while test environment 2 is $20m \times 20m$). The aim is to check how well the controllers generalise in the new environment and compare the effects of different phenotype lengths. The head size, h, of the chromosome was given as 4 and hence, using Equation 2.2, the maximum phenotype length, L, can be calculated as :

$$L = hn + 1 \tag{3.4}$$

All viable chromosomes should have the root allele as a function. Thus, plausible chromosomes are made up of either 1, 2, 3 or 4 IFLTE functions. Since there is only one function used in the experiment, then substituting h in equation 3.4 with the number of functions in a chromosome, the plausible phenotype lengths are 5, 9, 13 and 17. The remaining parameters were as detailed in section 3.2.2. The results are shown in Figure 3.9.

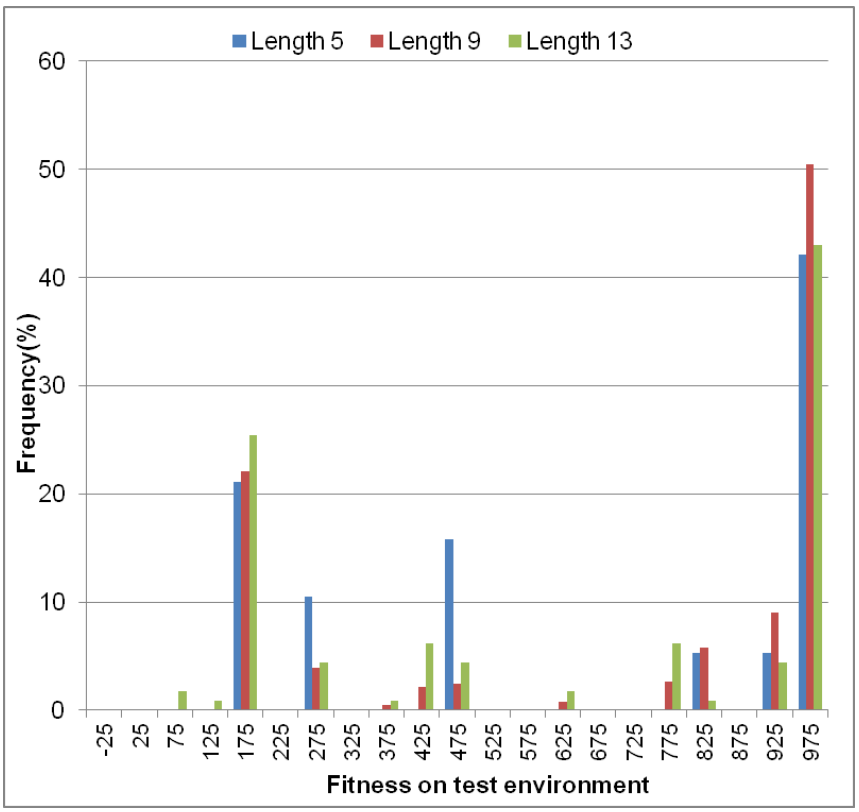


Figure 3.9: Comparison of frequencies with different phenotypical lengths when controllers are tested in similar environments

The results presented in Figure 3.9 show that controllers with length 9 had better overall performance in the test environment than those with lengths 5 and 13. The results also show that controllers with length 5 performed as well as those with length 13. Additionally, no controllers with length 17 were evolved. It is difficult to determine from these results whether shorter phenotypes adapt better than longer phenotypes when placed in a new environment. Since the training and test environments have similar layouts, it is likely that the controllers performance is equivalent in both environments. This is because the controllers are already trained on how to solve the problem and therefore any challenge presented in the new environment is likely to have been encoutered before. Subsequently, more experiments were required in order to investigate whether GEP controllers support Occam's razor.

3.3.2 Effect of training controllers in simple environment and testing in a complex environment

The goal in this experiment was to determine whether the length of a controller affects its performance when tested in a new and more complex environment. This experiment is similar to the one discussed in the previous section, with the exception that Figure 3.1 (training environment 1) was used as the training environment and Figure 3.1 (test environment 1) as the test environment. Results are shown by Figure 3.9.

Figure 3.10 shows that adaptation to a new environment has a direct correlation with

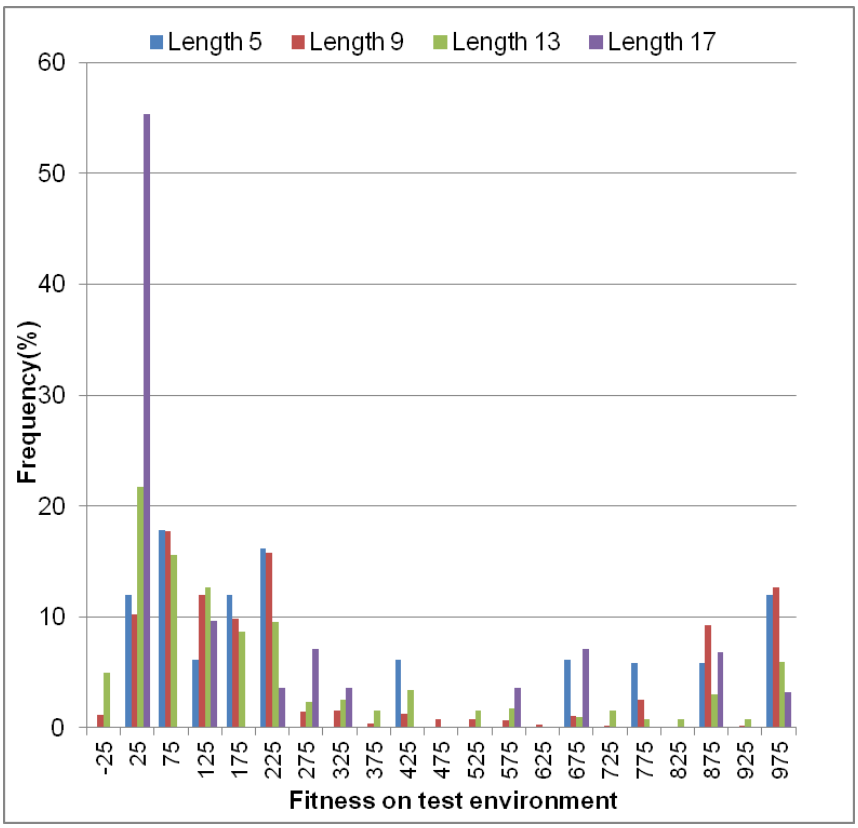


Figure 3.10: Comparison of frequencies of different controller lengths when tested in complex environment

the controller lengths. As observed, when introduced in a new environment, controllers with lengths 5 and 9 performed better than those with lengths 13 and 17. Across the first three quartiles, controllers with long phenotypes had high frequencies, this means that majority of these controllers did not perform very well in the new environment. The shorter controllers had low frequencies indicating that majority of these controllers performed better. At the upper quartile length 5 and length 9 appear to be dominant followed by controllers with lengths 13 and 17.

Longer phenotypes contain more building blocks and may also have redundancy built into them. These additional materials may contribute positively or negatively to the test performance, depending on the problem at hand. If the problem is significantly small, a longer phenotype could lead to low performance, this can potentially explain the performance of controllers with lengths 13 and 17. On the other hand, shorter phenotypes are parsimonious and contain only the necessary materials to solve a given problem. Although compact genomes are not always efficient in solving a problem [38], this experiment shows that more compact genomes adapt better when introduced in a new environment than longer phenotypes. This shows that GEP controllers support the principle of Occam's razor.

3.3.3 Testing generalisation using the last population

The goal of the experiment reported in this section was to investigate the generalization capability of all the controllers in the final population of a GEP run. The GEP was run using the parameters shown on Tables 3.3 and 3.2. The experiment set up was as reported in section 3.2.2. In the experiment, the final population of a GEP run was introduced to the test environment and the controllers performance in this new environment evaluated. The simple training environment, Figure 3.1 (training environment 1) was used for training and test environment 1 was used for testing. A comparison of performance in the test environment with respect to the training environment was then carried out. The results of the experiment is as shown by Figure 3.11 and Tables 3.4 and 3.5.

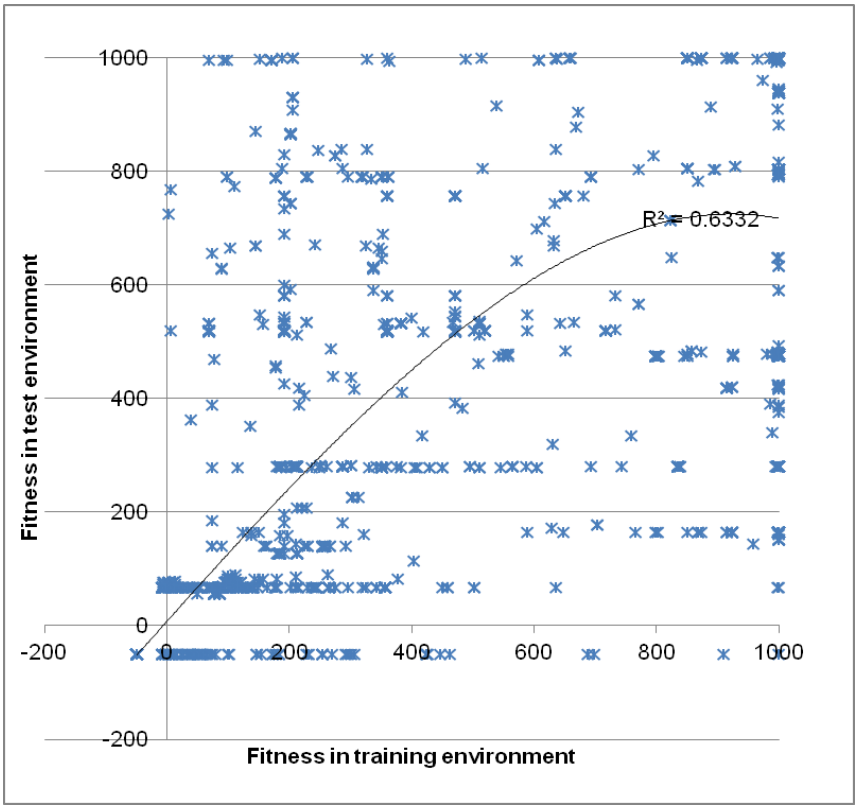


Figure 3.11: Performance of all the final generation controllers in the 20 GEP runs in training environment and the resultant performance in test environment. The fourth order polynomial regression line shows that there is a linear association of the performance in the training and test environment up to a fitness of approximately 650.

Figure 3.11 shows the performance of the final generation controllers in the test environment. The calculated correlation coefficient between train and test fitness was 0.77. A fourth order polynomial regression line, drawn on the scatter graph, shows the linear association of the performance in the training and test environment. Additionally, the calculated R^2 value suggests that 63% of the variability of the data could be explained by the linear regression. The calculated correlation and R^2 value suggests that fitness achieved during the training affects the test fitness. Also it can be seen from the graph that there are a large number of controllers that did not perform well in the training set but performed well in the test environment. This means that we possibly do not need to

have highly converged controllers in the training fitness, this is to be expected as a fitness of 1000 on the simple environment will not necessarily translate to the more complex environment.

Table 3.4: Phenotype lengths in the last generation controllers

Phenotype Length	Average number of controllers in the last population	Success rate as % of controllers with fitness ≥ 950 in the test environment
5	93	27
9	74	18
13	18	6
17	3	0

Table 3.4 compares the average number of controllers in both training and test environments based on their phenotype lengths. As reported previously, the GEP algorithm was run 20 times. The presented values are therefore averaged from the 20 runs. The results suggest that the last population in a run had a higher number of controllers with shorter phenotypes than longer ones. In fact, there is an inversely proportional relationship between length of phenotype and the number of controllers. Similarly, the results suggest that controllers with shorter phenotypes had a higher percentage of controllers with a fitness ≥ 950 in the test environment. These results concur with results shown in section 3.3.2 where shorter phenotypes outperformed the longer ones.

Table 3.5: Statistical comparison of the performance in the training and test environment

Phenotype Length	Mean of the last pop. in Training environment	Std Dev. of the last pop. in Training environment	Mean of the last pop. in Test environment	Std Dev. of the last pop. in Test environment	Correlation between train and test fitness
5	574.30	458.96	439.59	403.91	0.74
9	420.46	345.766	421.30	384.57	0.77
13	200.33	165.97	324.99	294.76	0.76
17	123.18	47.43	278.02	182.09	0.77

The descriptive statistics shown by Table 3.5 suggests that there is a relationship between the length of the phenotype and their performance in both training and test environments. The results suggests that controllers with shorter phenotypes performed best in both the training and test environments. Additionally, the results suggests that controllers with longer phenotypes had an improved overall average fitness in the test environment. The improved average fitness shown by the controllers with longer phenotypes is likely to have been as a result of the redundancy in the long phenotypes as well as the fact that these controllers were fewer than those with shorter phenotypes. The large standard deviation shown by the fitness of controllers with short phenotypes suggests that there was a large diversity in the number of controllers with lengths 5, followed by 9 and decreasing with increase in phenotype length. Thus, the standard deviation suggest an inverse proportion between diversity of controllers with phenotype lengths. This diversity is to be expected as there are different ways of forming a controller with a shorter length while there is restricted ways to form long controllers. The statistics also show a good correlation between the

performance in training and test environments. This as discussed previously, suggests the performance in the training environment affected the performance in the test environment.

The generalization results described in section 3.3.2 and 3.3.3 suggests that shorter phenotypes were more robust at adapting to the new environment. However, the experiment is simple and thus conclusive results can possibly be derived when investigation is carried out in a more complex environment. Nevertheless, in this type of experiment, the ugGEP phenotypes support the Occam's razor principle.

3.4 Sensor based velocity control

In Section 3.2 above, the default kinematic model (discussed in Section 3.2.1) was used to control the robot movements. Subsequently, the robot's translational velocity was manually set to $2.0m/s$ for forward movement (*motor terminal,* $\boldsymbol{F}$) and $-2.0m/s$ for reverse movement (*motor terminal,* $\boldsymbol{B}$). Similarly, in section 3.3 the wheel's translational velocity was set to $0.8m/s$ for forward movement and $-0.8m/s$ for reverse movement. In all the previous experiments, the angular or rotation velocity was not set, however, the robot could turn based on the radians as reported in Table 3.2 above. This mechanism, though successful in solving the robot problem as shown above, does not cater for situations when the robot may need to reduce or increase speed given different environmental conditions. For instance, a robot may need to slow down when close to obstacles and move at top speed when there are no obstacles in the vicinity.

This section focusses on an experiment carried out to investigate the effect of evolving controllers that sets the velocity of a robot based on the infra-red sensor readings. The proposed controller is similar to controllers developed using neural networks [3]; that is, the controller takes the sensors readings as inputs and translates them to a motor output. However, the proposed model is simpler and contains only one output that effects robot motor controls.

In the described experimentation, the default kinematic model was used to control the robot movements. The output of the evolved GEP controller was used to set the robots translational or rotational velocity. As previously discussed, when a GEP controller is executed the output is one terminal symbol. In the experiment, the output of the controller was used to set the translational velocity (in m/s) with the *Front* or *Back* sensor readings if **F** or **B** terminals were returned. Similarly, the rotation velocity in radians per second (rad/s) was set to any of the other sensor readings depending on the terminal returned by the controller output (see Table 3.2 for the sensor positions). This means that the rotation angles could be small or large depending on the returned sensor reading and the robot accelerates or decelerates given the *Front* or *Back sensor* readings. In this case, the robot learns how and when to turn and the required speed. This is a more natural mechanism of control. In this experiment, therefore, the robot learns the mapping between specific

[3]Similarity with neural-based controllers is only based on the sensors generating input data and this input being translated to motor output. Most neural based controllers have a multiple output depending on the number of motors/wheels (see [105])

sensors to various translational and rotational velocities.

3.4.1 Experimental results

To conduct the experiments using the technique described in this section, the algorithm parameters shown on Table 3.6 were used. All the other parameters were set as shown on Table 3.2.

Table 3.6: Sensor based velocity control: Parameter settings

Parameters	**GEP setting**
Maximum generations	100
Population size	200
Head size	8

Two experiments were conducted as described below.

Experiment 1 : Sensor based control

The aim of the first experiment was to test the viability of this technique in evolving robust controllers that would guide a robot to explore a simulated environment while avoiding obstacles. In this experiment, the robotic sensors were set to return a value between 0.0m and 2.0m. This meant that the robots maximum speed was set at $2.0m/s$, similar to the experiments reported in Section 3.2.3 above. Similarly, the experiment was conducted using test environment 2 (Figure 3.1) as the simulated robot environment. Figures 3.15, 3.16, 3.17 and 3.18 show the achieved results.

From the results (Figures 3.17) the algorithm was able to evolve a suitable controller within 25 generations and the average performance population was better than when standard ugGEP was used (see Figure 3.18). This means that in general, the algorithm evolved better controllers than ugGEP and with a fewer number of generations. This can be attributed to the use of sensors to determine speed and rotation-angles particularly when close to obstacles. However, Figure 3.15 and 3.16, shows that when compared to standard ugGEP this mechanism had less success in evolving controllers that achieved maximum fitness (it was observed that most controllers had a high fitness (> 900)). This is likely attributed to numerous turning (at slight angles) where points were not awarded. Nevertheless, these results show that this technique is a viable mechanism to evolve controllers that require close interactions with the environment in order to function.

Experiment 2 : Adaptation in new environments

In the second experiment, the controllers were tested for robustness in adapting to new environments. In this experiment all the algorithm parameters were set as per Section 3.2.2 above. The robot sensors were set to return a value between 0.0m and 2.0m, giving a maximum speed of $2.0m/s$ for the robot. Similarly, the standard ugGEP was set such

that the robot always moved at $2.0m/s$. Training environment 2 (see Figure 3.1) was used for the training phase while test environment 2 (Figure 3.1) was used in the testing phase. The obtained results is as shown by Figures 3.12, 3.13 and 3.14.

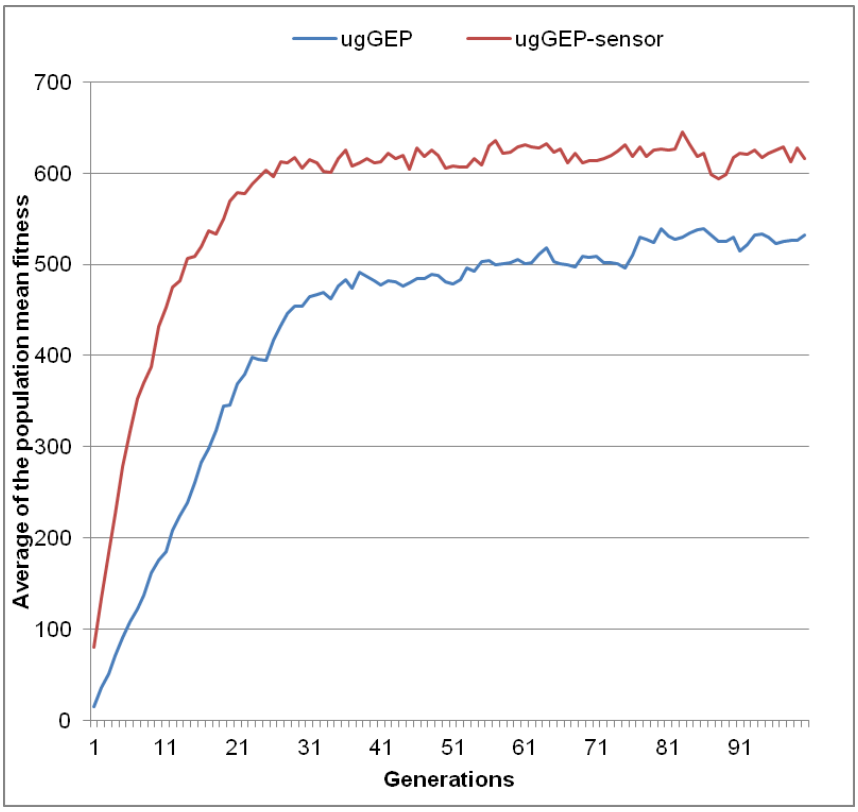


Figure 3.12: Progression of the average mean fitness in the population as achieved using ugGEP and ugGEP with sensor based velocity control. The average of the population mean fitness was computed from the population mean fitness achieved in each generation in all the 20 randomly seeded algorithms. In each evolutionary run, a population of 200 organisms was evolved for 100 generations and the population mean fitness recorded in each generation. The rest of the parameters are as shown on Table 3.2.

Figure 3.12 shows that in terms of average fitness, the ugGEP with sensor based velocity control started off faster and continued to perform better than the standard ugGEP. The likely explanations with regard to the initial 'surge' is that; since the robot turns only slightly and the speed near the obstacles is lower, it is less likely to collide with obstacles than when controlled by standard ugGEP controllers. Initial controllers, thus, performed better than their standard ugGEP controllers and as can be seen the trend continued on throughout the ensuing generations. This is similar to the observations made in Figure 3.18.

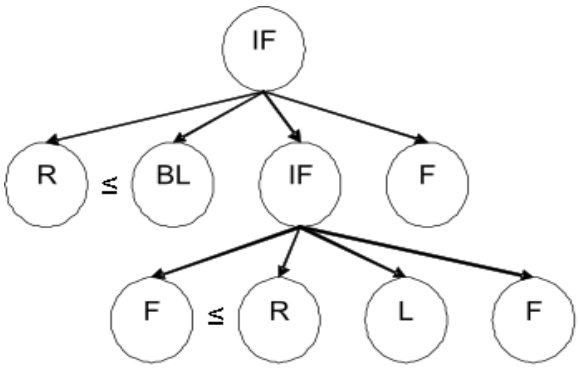


Figure 3.13: Obstacle avoidance controller evolved using ugGEP with sensor based velocity control. The algorithm was set up using parameters listed on Table 3.2. The controller was evolved in training environment 2 as shown in Figure 3.1.

Figure 3.13 shows a controller evolved using the ugGEP with sensor based velocity control. This controller shows that robot continued to move forward by checking the right and back left sensors and turned left if an obstacle was detected by forward and right sensors. This examples shows that evolved controllers approximated the type of controller that a human designer would program to control a robot.

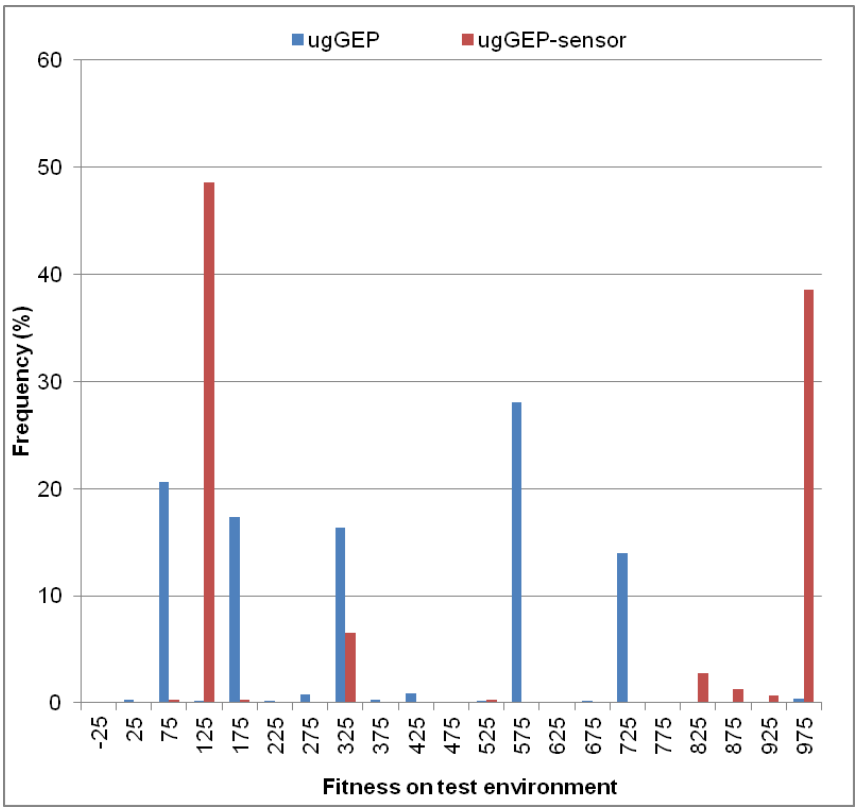


Figure 3.14: Comparison of performance of best individuals in new environments. The comparison is based on velocity based sensor control (ugGEP-sensor) and standard motor action ugGEP discussed above. In each algorithm, the evolutionary run was conducted using a population of 200 organisms evolved for 100 generations. The rest of the parameters are as shown on Table 3.2.

Figure 3.14 shows a comparison on the performance of best individuals evolved in training

environment 2 using ugGEP and ugGEP with sensor based velocity control. The results suggest that controllers that evolved the mechanism to accelerate and decelerate as well as rotate the robot, also performed satisfactorily when placed in new environments .

3.5 Evolving monolithic controllers using multigenic GEP

The previous sections outlined how to use ugGEP to evolve monolithic robot controllers. In this section, experiments were carried out with the aim of investigating how the multiple genes in GEP (mgGEP) can be utilised to evolve monolithic robot controllers that produce multiple output. The previous chapter mentions that the GEP organism can be constructed by the spatial organisation of different sub-ETs [38]. This means that GEP can be used to evolve controllers that provide multiple outputs yet are still part of the same organism. The evolved controllers (or partial controllers) are independent entities that work together to accomplish the set objectives. In the experiments reported in this section, GEP is utilised to evolve two sub-ETs that work together to evolve an obstacle avoidance controller. The output of the ETs control the velocity of the robot. The conducted experiments used a differential drive kinematic model to control the robot movements; that is, the left and right wheel velocities were controlled independent of each other during the course of the simulation. The robot, thus, had to establish a link between linear and angular velocity in order to accomplish exploration and obstacle avoidance tasks. The output of the first sub-ET controls the velocity of the left wheel while the output of the second sub-ET control the right wheel. The difference of the two outputs in terms of sensor readings produce an angular velocity while an equal output gives a linear velocity. The two genes are part of the whole mgGEP chromosome and are thus evolved as such. Note that there is no linking function provided for the genes and that every gene has an independent contribution to the overall robot behaviour. The fitness achieved is for the collective performance. In this the robot is required to evolve a strategy to control its velocity and to avoid obstacles. The output of each sub-ET is the sensor reading of that particular terminal as shown in Table 3.1.

The task of coordinating the wheel speeds is non-trivial since if one gene does not evolve a good solution or if it always returns a zero, the robot will continue rotating in the same place. The robot also has to achieve maximum speed when far from obstacles and decelerate when near obstacles, as well as evolving the mechanism to start turning when obstacles are encountered.

3.5.1 Experimental results

The experiments were carried out using a population of 200 organisms and run for 100 generations with head size h set to 4. The total length of the whole organism was therefore 34 alleles. The algorithm was evaluated over 20 randomly seeded runs. Each controller in the population was allowed to control the robot for 50 virtual seconds. Gene transposition and gene recombination was used with 0.1 probability for each operation. Table 3.7 shows

a summary of the parameter settings. The rest of the parameters were set as defined in Table 3.2. In all the experiments the robot sensors were set to return a value between $0.0m$ and $2.0m$. Hence, the maximum speed that the robot could move at was $2.0m/s$ (unless otherwise specified). Two experiments were conducted as described below.

Table 3.7: mgGEP-multiple out: Parameter settings

Parameters	**ugGEP**	**ugGEP-Sensor**	**mgGEP-multiple out**
Maximum generations	100	100	100
Population size	200	200	200
Head size	8	8	4
No. of genes	1	1	2
Gene transposition probability	0.0	0.0	0.1
Gene recombination probability	0.0	0.0	0.1

Experiment 1: Evolving a multiple output controller

The aim of the first experiment, in this section, was to test the viability of the multiple output GEP to control the robot. The obstacle avoidance problem was used as a test case to test the technique. In the first experiment, environment test 2 (Figure 3.1) was used. The obtained results are as shown in Figures 3.16, 3.15, 3.17 and 3.18

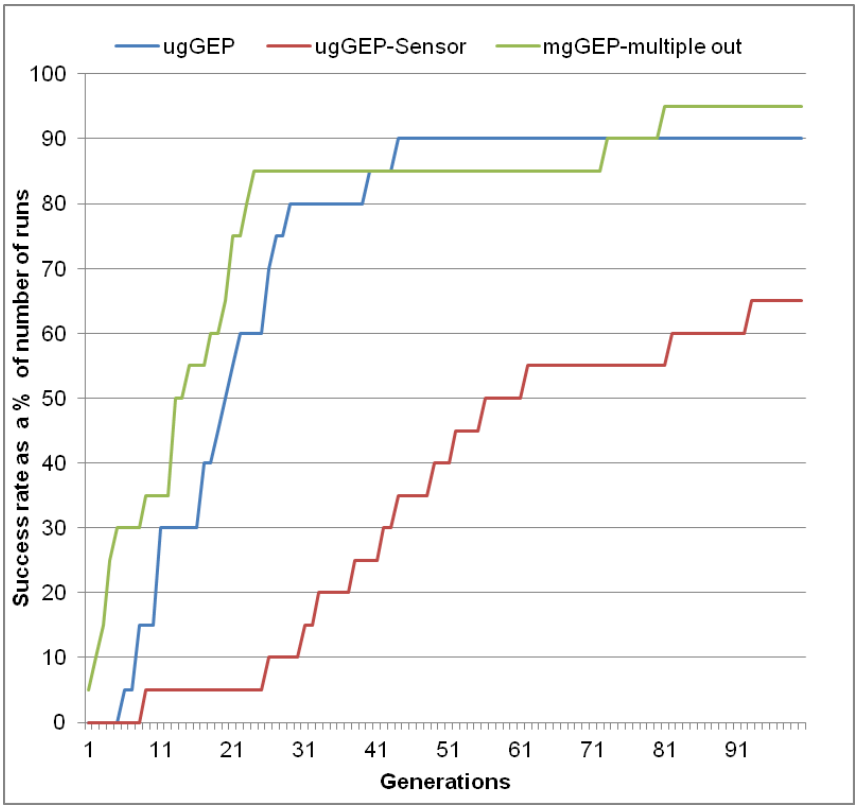


Figure 3.16: Comparison of success rate over generations as achieved using ugGEP, ugGEP with sensor based velocity control and with multiple output gene. Success rate refers to the number of runs which achieved maximum fitness (1000 steps) out of the 20 randomly seeded evolutionary runs conducted. In each evolutionary run, a population of 200 organisms was evolved for 100 generations using the parameters shown on Tables 3.2 and 3.7.

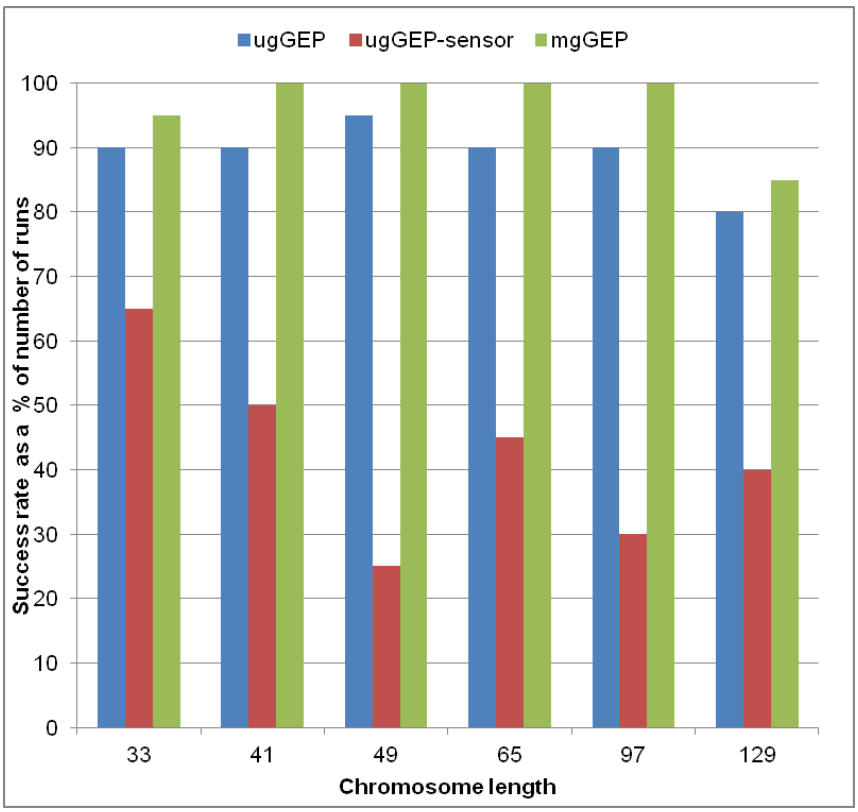


Figure 3.15: Comparison of success rates achieved using ugGEP, ugGEP with sensor based velocity control and with mgGEP multiple out. These success rates are shown over different number of chromosome lengths run using the aforesaid algorithms. Success rate refers to the number of runs which achieved maximum fitness (1000 steps) out of the 20 randomly seeded evolutionary runs conducted. In each evolutionary run, a population of 200 organisms was evolved for 100 generations using the parameters shown on Tables 3.2 and 3.7.

As observed in this experiment, the mgGEP with multiple output performs better that standard ugGEP and ugGEP with sensor based velocity control. Figure 3.16 shows that in all the different chromosome lengths tried the mgGEP-multiple output did not only have a better success rate but also evolved successful individuals much more quickly than the ugGEP approaches. Overall, the ugGEP with sensor based velocity control had a lower success rate in this task. During the experiments, it was observed that ugGEP with sensor based velocity control, had a majority of controllers achieving a fitness just slightly lower than the maximum fitness possibly because of losing points as the robot turned and therefore, though the general performance was better than standard ugGEP, the success rate was not as good. Figures 3.15 and 3.16 shows that there is a greater probability of evolving successful controllers using the mgGEP technique.

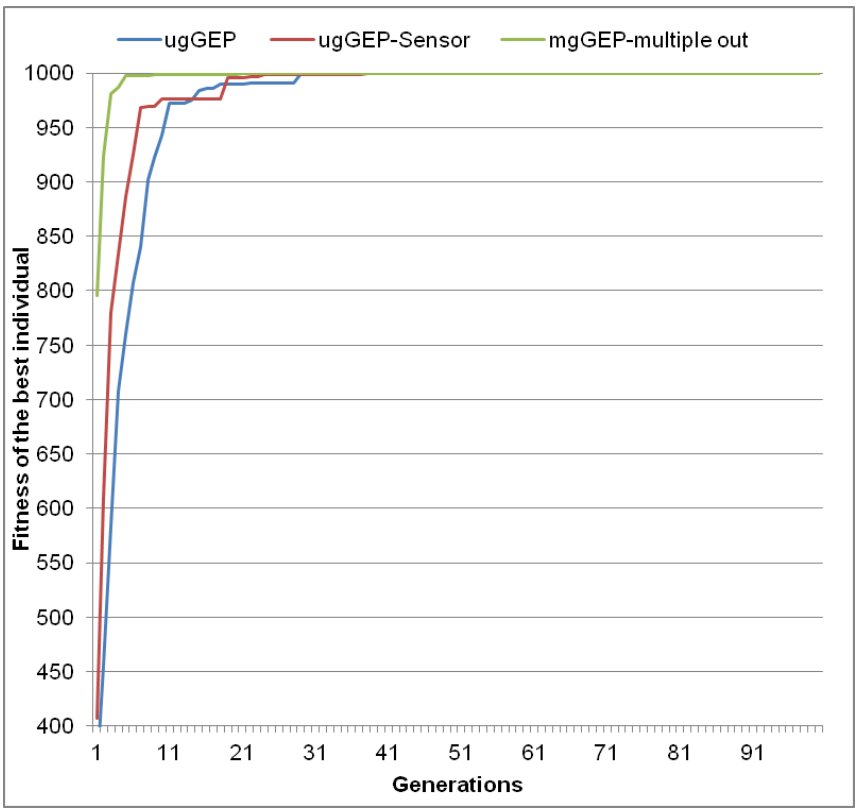


Figure 3.17: Progression of best individual in the population as achieved using ugGEP, ugGEP with sensor based velocity control and with multiple output gene in environment test 2. Each individual curve represents the best performing evolutionary run out of the 20 randomly seeded runs. In each evolutionary run, a population of 200 organisms was evolved for 100 generations and the best individual performance recorded in each evolutionary run. The rest of the ugGEP parameters are as shown on Tables 3.2 and 3.7.

Figure 3.18: Progression of the mean population mean fitness as achieved using ugGEP, ugGEP with sensor based velocity control and mgGEP with multiple output gene in environment test 2. The mean of the population mean fitness was computed from the population mean fitness achieved in each generation in all the 20 randomly seeded algorithms. In each evolutionary run, a population of 200 organisms was evolved for 100 generations and the population mean fitness recorded in each generation. The rest of the parameters are as shown on Table 3.2.

Figure 3.17 shows that mgGEP algorithm used approximately $5-10$ generations to evolve an individual that solved the problem. Additionally, Figure 3.18 shows that the mgGEP had a better overall average performance than the standard ugGEP and ugGEP with sensor based velocity control. The average performance across the entire population also shows continuous learning with number of generations. The nature of the environment used was such that there were a lot of open spaces, (see test environment test 2 in Figure 3.1 above), this suggests that the mgGEP-multiple output learns how to navigate this environment better given that it is able to move at a maximum speed in the open spaces and slower near obstacles. Slowing near obstacles also means that the robot can easily avoid obstacles. This idea of accelerating and decelerating also explains the good performance shown by ugGEP with sensor based velocity control.

Experiment 2: Adaptation in new environments

In the second experiment, the controllers were tested for robustness in adapting to new environments. The experiments were conducted using similar parameter settings as reported in section 3.5.1. Training environment 1 (see Figure 3.1) was used for the training

phase while test environment 1 (Figure 3.1) was used in the testing phase. The results for this experiment are shown by Figures 3.19, 3.20, 3.21 and Table 3.8

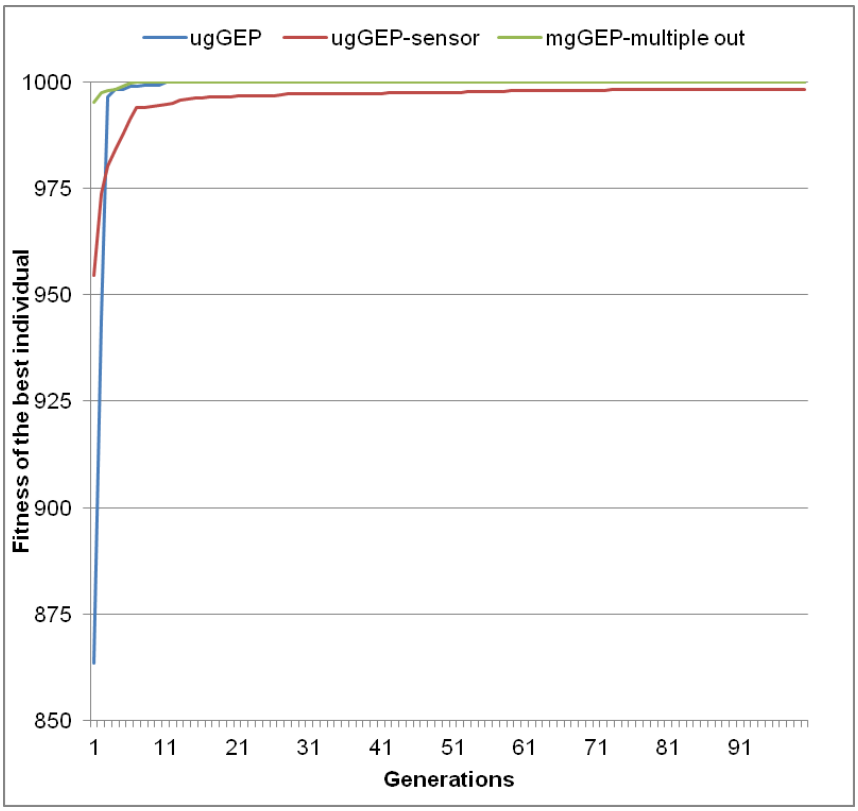


Figure 3.19: Progression of best individual in the population as achieved using ugGEP and mgGEP with multiple output. Each individual curve represents the best performing evolutionary run out of the 20 randomly seeded runs. In this experiment, the robot was trained in training environment 1 and tested on environment test 1. In each evolutionary run, a population of 200 organisms was evolved for 100 generations and the best individual performance recorded in each evolutionary run. The rest of the parameters are as shown on Table 3.2.

Figure 3.19 shows that on average, mgGEP with multiple output and the standard ugGEP evolved the best individual in less than 10 generations. In addition, mgGEP was quicker in evolving better controllers than the ugGEP approach. Given the nature of the environment, the task targeted is that of circumnavigating around the obstacle. As a result, the mgGEP controllers needed to learn a mechanism of attaining good angular velocity in order to turn and navigate the environment. It is also important to note that the behaviour emerges from the interaction of the individual controllers and the environment. This problem is thus not a simple task. The performance of mgGEP with multiple output shows that GEP has a great potential of evolving self organising mechanism needed for robot control.

The progression of average fitness over generations (Figure 3.20) shows that the mgGEP controllers performed equally as well as the ugGEP approach.

When the best solutions were introduced into new environments, it was noted that in general all the solutions from the different algorithms did not perform very well. It is also

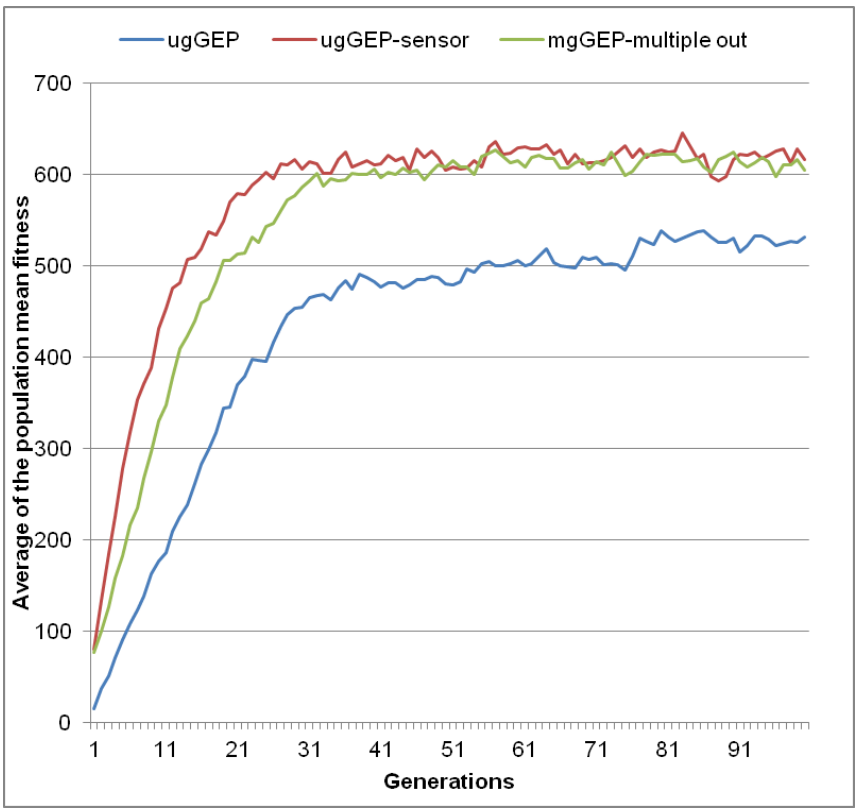


Figure 3.20: Mean of the population mean fitness in the population as achieved using ugGEP, ugGEP with sensor based velocity control and mgGEP with multiple output. The mean of the population mean fitness was computed from the population mean fitness achieved in each generation in all the 20 randomly seeded algorithms. In this experiment, the robot was trained in training environment 1 and tested on environment test 1. In each evolutionary run, a population of 200 organisms was evolved for 100 generations and the population mean fitness recorded in each generation. The rest of the parameters are as shown on Table 3.2.

important to note that the robot solves an obstacle avoidance problem in the training environment (Figure 3.1, training environment 1) while it is tested to perform a wall following task when introduced to the new environment (Figure 3.1, test environment 1). Results shown by Figure 3.21 suggests that controllers evolved using mgGEP-multiple output performed slightly better in the new environment than controllers evolved using standard ugGEP. Additionally, calculated mean fitness value in the new environment was 175.84 for mgGEP-multiple output and 132.26 for the standard ugGEP. This slight increase in performance can possibly be attributed to the reliance on the environment to determine linear and angular velocities dynamically. This shows that in problems requiring multiple output, the mgGEP mechanism can be used to evolve suitable solutions.

Another important factor that was realised during the experiment is that the robots speed in the training environment is of significant importance in determining how the robot performs in the test environment. To investigate this, the experiment discussed above was repeated with varying speed. For the standard ugGEP controllers, the robots translational velocity was set to $0.8m/s$ for forward movement and $-0.8m/s$ for reverse movement. In the mgGEP-multiple output and ugGEP with sensor based velocity control, the robot

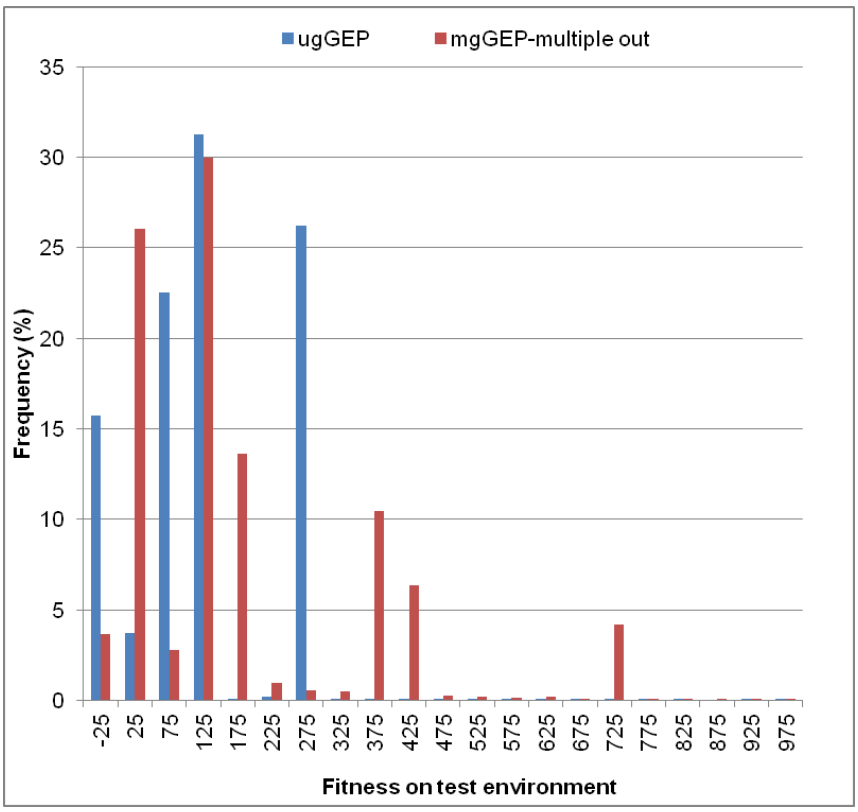


Figure 3.21: Comparison of performance of best individuals in new environments. The comparison is based on standard motor action ugGEP discussed above and the mgGEP multiple output. In all the algorithms, the evolutionary run was conducted using the parameters shown on Tables 3.7 and 3.2. In this experiment, the robot was trained in training environment 1 and tested on environment test 1. The robot translational velocity was set to $2.0m/s$ for standard ugGEP and mgGEP.

sensors were set to return a value between $0.0m$ and $1.5m$ translating this to a linear velocity between $0.0m/s$ and $1.5m/s$. These speeds are slightly lower than maximum speed set at $2.0m/s$ discussed above. Figure 3.22 shows the results achieved.

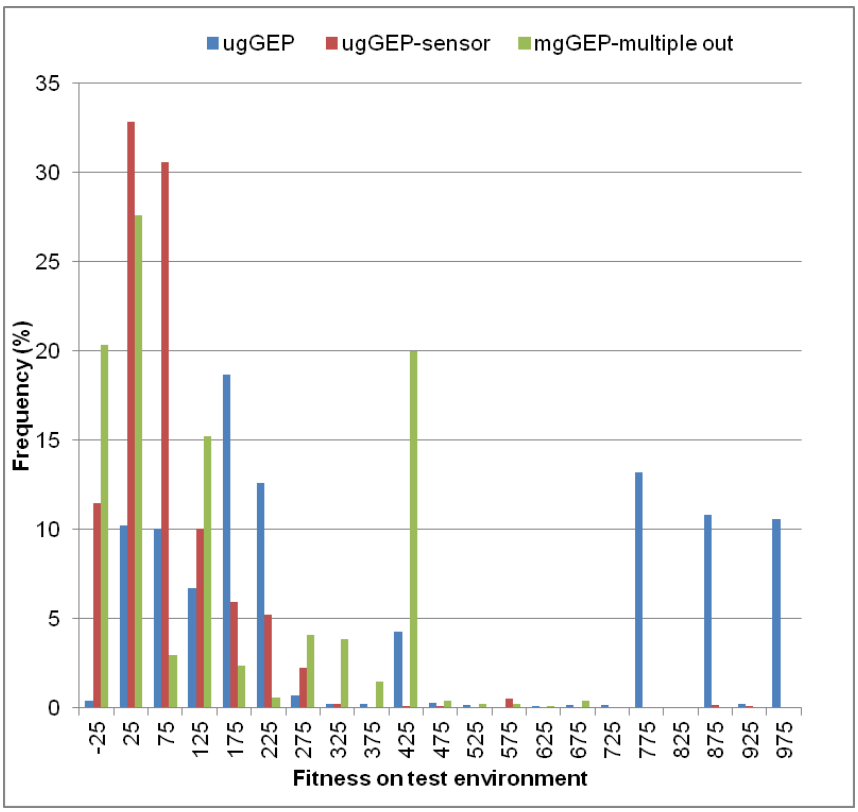


Figure 3.22: Comparison of performance of best individuals in new environments. The comparison is based on standard motor action ugGEP discussed above and the multiple output mgGEP. In all the algorithms, the evolutionary run was conducted using the parameters shown on Tables 3.7 and 3.2. In this experiment, the robot was trained in training environment 1 and tested on environment test 1. The robot translational velocity was set to $0.8m/s$ for standard ugGEP and mgGEP.

Figure 3.22 shows that controllers trained using standard ugGEP with the translational velocity set at $0.8m/s$ were more robust in new environments than controllers evolved using ugGEP with sensor based velocity control and the mgGEP with multiple output. Controllers using the latter two approaches had the speed ranging between $0.01m/s$ and $1.5m/s$. This shows speed plays an important part in determining the generalising capability of the algorithm. This effect is possibly attributed to the idea that lower speeds give the robot enough time to learn the environment and evolve an obstacle avoidance strategy. As a result, mechanisms like mgGEP-multiple output that can set speed automatically might be better positioned in evolving more robust controllers.

Statistical comparison

Table 3.8: Mann-Whitney U test between mgGEP-multiple out and ugGEP performances

	Z values	P_1	P_2	Signi. Level
ugGEP	3.16	0.0004	0.0008	2%
ugGEP (sensor based control)	5.4	< 0.0001	< 0.0001	2%

Table 3.8 shows the results of a Mann-Whitney U test between the mgGEP-multiple output

and the two ugGEP approaches on the mean of best individuals' fitness. In an evolutionary algorithm, it is not always possible to say with certainty how many generations are required to solve a specific problem. Thus, to make sure that the Mann-Whitney U test results were not skewed, the mean of best individual fitness in each generation in the evolutionary run were used in the test sample. Since the algorithms were run 20 times, 20 sample values were used in the Mann-Whitney U test. The default hypothesis, (H_0, for both the one tailed and two tailed tests is that there is "no difference in the performance of the presented algorithms in the obstacle avoidance problem". The alternative hypothesis, (H_A, for the two tailed test (P_2) is that "there is a difference in performance between the mgGEP-multiple output and the ugGEP approaches" while the alternative hypothesis for the one tailed test (P_1) is that "mgGEP-multiple output performs better than ugGEP approach in the obstacle avoidance problem". The P_2 values suggests that the alternative hypothesis is correct and that the statistical significance is beyond 2%. Additionally, the P_1 values suggest that the one tailed alternative hypothesis is correct and the statistical significance is beyond 2%. The results of the significance suggests that, in this problem, mgGEP-multiple output is better than the ugGEP approaches.

3.6 Conclusion

This chapter has introduced techniques of evolving robotic behaviours using GEP. The aim was two fold. Firstly, to investigate the performance of unigenic GEP (ugGEP) structures in a robot task and secondly, to investigate the performance of the more complex multigenic GEP (mgGEP) with multiple output. The performance of these algorithms in training and test environments was recorded and analysed. The results generated and the discussion that follows shows that the presented algorithms were successful in solving the problem. Moreover, when introduced into new environments, the trained controllers were robust enough to navigate the environment successfully. The results demonstrate that GEP is a suitable algorithm that can be used in evolutionary robotics to develop controllers with an ability to adapt to new environments.

Results in Figure 3.8 shows that the choice of a head size (chromosome length) affects the performance of the controller and the algorithm. As can be seen longer chromosomes (larger head sizes) tend to produce better controllers. A similar observation was seen when the mgGEP with multiple output was used (see Figure 3.15). In simple environments and simple problems these longer chromosomes tend to have redundancy built into them. These redundancies built into the genome provide more diversity in the structure of the phenotype. As shown in Figures 3.10, 3.9 and Tables 3.5 and 3.4, GEP tries to build simple parsimonious solutions depending on the size of the search space. Thus for a simple problem as presented here, shorter phenotypes are robust enough to adapt to new environments. The evolution of these simple parsimonious solutions is definitely easier and computationally cheaper than using GP where the depth has to be chosen in advance and regularly checked in order to avoid creating non-valid organisms.

As reported earlier, evolutionary robotics does not require human influence in generating

robotic behaviour. In the discussed experiments, only the attributes of behaviour such as sensor and motor terminals were supplied in addition to a conditional function. The fitness function awarded the robot for continuous movement and penalised the robot for hitting obstacles. The fitness function, therefore, exploited the information derived from the sensory information. The result was a robust behaviour that adapted when introduced to new environments. It is also clear that generating these behaviours manually would take a long time to implement all the conditions that the robot would need to tackle. In fact when manual hand coding is preferred, the robot environment needs to be known *a priori* in order to develop robust controllers.

Evolving behaviours using GEP has various advantages as compared to previous techniques mentioned in the previous chapter. For instance, in canonical GP [115, 106, 78] the depth of the tree has to be set so as to prevent the generation of non-valid individuals, however, in GEP there is no possibility of generating non-valid individuals. GEP regulates itself to definite phenotypical lengths that are always valid controllers. Setting the head size to 4 will only give a maximum depth of 4 from which a set of 4 different valid controllers can be achieved. For simple problems like obstacle avoidance, shorter phenotypes appear to be robust in both training and test environments.

GEP capabilities extend beyond the ugGEP structures. As shown in the experimentation above, GEP can be used to evolve multigenic chromosomes that can be used to solve problems requiring multiple output. In this scenario, the presented problem shows that mgGEP-multiple out was more successful in solving the problem and was also more robust in the new environment. This is of significance as there are various problems, particularly in robot control, where multiple output is required. In addition to this mechanism, multigenic GEP structures can be modelled such that a linking function is used to co-join them. This linking function can be used to form a concatenated chromosome where the sub-ETs contribute to the overall performance. Alternatively, if each sub-ET in a mgGEP codes for a certain unique function or module, then the linking function can be used as a function selection mechanism to determine which function or module to select next. The next chapter investigates mechanism of evolving mgGEP using a linking function as an action selection mechanism.

4 Using Multigenic GEP in Robot Behaviour Sub-division

In the previous chapter, the capability of unigenic GEP (ugGEP) in solving robotic problems was shown using an obstacle avoidance behaviour. Additionally, the capability of GEP to combine multiple genes (mgGEP) to generate multiple open reading frames (ORFs) whose outputs can be used to solve a problem requiring multiple outputs, was discussed and an example implemented. The ensuing results showed that the multiple output GEP (mgGEP multiple output) was more robust and evolved better structures than the monolithic ugGEP approach. It was also discussed that the mgGEP ORFs can be linked together to form a concatenated ORF that can be used to effect control on a robot. The mgGEP structures are unique to GEP and they can be used to evolve multiple output controllers as shown in the previous chapter, or the multiple genes could be linked together using a pre-selected linking function (for instance logical or mathematical functions) and then evolved simultaneously as modules. In this case, the linking function acts as a behaviour organiser and selects which gene/controller to execute in order to effect motor control at each specific time.

Many approaches to AI in robotics use a multi-layered approach to determine levels of behaviour from basic operations to goal-directed behaviour, the most well-known of which is the subsumption architecture. In this chapter, the multigenic gene expression programming (mgGEP) is used to evolve multiple genes corresponding to a layered architecture. Results are presented on a number of wall following tasks using ugGEP and mgGEP. Comparison is made with the similar behaviour evolved using standard genetic programming (GP). In addition to the wall following behaviour, a robot foraging behaviour is implemented with the aim of investigating whether the position of a specific module (sub-expression tree (ET)) in the overall ET is of importance when coding for a problem.

The rest of the chapter is organised as follows: Firstly, a discussion of modular architectures in ER is presented. Secondly, the linking mechanisms in a standard mgGEP are discussed. Thirdly, a robot wall following problem is presented and results in a 2D structure discussed. Fourthly, a similar problem is presented in a 3D world model and experimental results discussed. Fifthly, a robot foraging behaviour is presented and results discussed. Finally, a conclusion regarding the work presented is drawn.

4.1 Modular architectures in ER

In Chapter 2, mechanisms used to improve the basic ER algorithm were discussed. Among these mechanisms, is the process of 'divide and conquer' where the required behaviour is divided into simpler tasks. As previously discussed, there are two main techniques employed. Firstly, incremental evolution; this involves dividing the problem into different simple tasks and then evolving a controller to solve the problem sequentially. In this case, the controller is evolved to solve one task, then the objective function is fine-tuned and the controller re-evolved in this new task. For instance, in a wall following problem, a robot controller could be evolved for an obstacle avoidance task, then once a certain fitness is achieved, the objective function is fine-tuned to suit a wall following behaviour. There is thus only one genome that has undergone various evolutionary runs to learn the required behaviour. Secondly, layered learning; this approach is closely related to subsumption architecture. In this approach, the targeted behaviour is divided into various tasks, then the controller is evolved in different modules sequentially (see Figure 4.1 below for a sub-division model). For instance in the wall following behaviour mentioned above, the controller would have two modules; obstacle avoidance and navigation with wall proximity. The obstacle avoidance behaviour module would be evolved first and once the robot learns to navigate without obstacles, learning within this module would be stopped and the evolution of the second module started. After these modules are generated, an action selection or behaviour organising mechanism is then selected and used to coordinate the interaction of behaviours. The overall behaviour is thus emergent (that is, the global behaviour is as a result of the robot interacting with the environment and the interaction of the sub-behaviours).

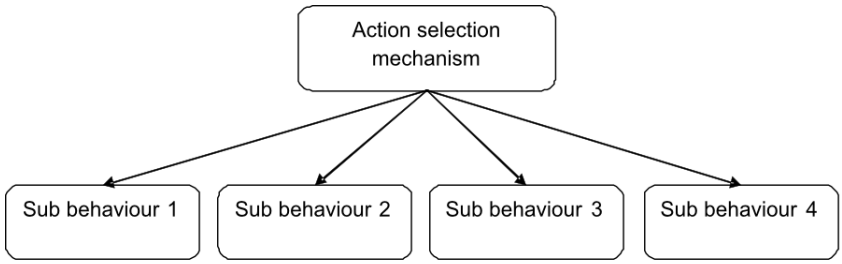


Figure 4.1: A behaviour modularity model

As previously discussed, the two approaches have been shown to generate better controllers than the canonical ER mechanism [55, 105, 136, 142]. However, there are various shortcomings with this approaches. For instance, with incremental evolution there is only one uni-dimensional controller, it is therefore impossible to say with confidence which behaviour a particular controller is displaying, as the actual controller is not divided into various modules [5]. Although layered learning provides functionality to sub-divide behaviours into various modules, learning in a particular module is stopped when the task is learnt. This means that there is a high likelihood that if the robot encounters a new scenario, it may need to re-learn how to solve the first task. In addition, layered and incremental evolution approaches use multiple fitness functions in order to achieve learning, this can lead to an increase in computational overhead.

As previously discussed, multigenic GEP (mgGEP) is a technique where multiple genes can be evolved together either as independent modules with each set output directed to a particular problem, or, they can be used as parts of one entire open reading frame (ORF) where coordination among the genes is achieved using a pre-defined linking function. For instance in figure 4.1 each of the sub-behaviours could be evolved as genes in an mgGEP set up. This chapter focusses on how the mgGEP approach can be used to successfully evolve control structures where the behaviour has manually been sub-divided by a designer.

4.2 Linking functions in multigenic GEP

In mgGEP, a linking function is pre-defined by the designer before the start of an evolutionary run. As discussed earlier, this linking function defines how the genes interact with each other to evolve the targeted behaviour. In effect, the linking function acts as a behaviour organiser and thus allows the activation or inhibition of a particular gene (each gene specifies a particular sub-behaviour). For instance in figure 4.1, the 'action-selection' box can be replaced with a linking function such as IF, AND, OR, XOR among others. Given a certain linking function and multiple genes it is thus possible to set the linking function such that each of the genes is targeted towards accomplishing a certain objective. The overall behaviour is achieved by the interaction of these genes with each other as defined by the linking function and the robot environment. Figure 4.2 shows an example of how mgGEP can be used with a linking function to evolve a modular architecture.

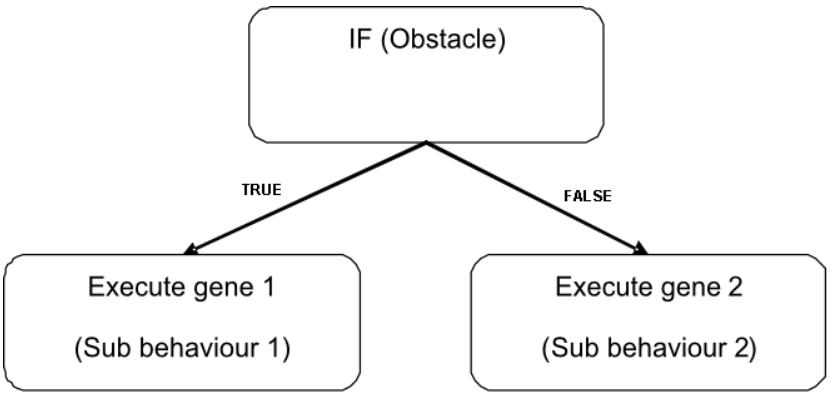


Figure 4.2: mgGEP behaviour sub-division using 2 genes and a linking function

4.3 Evolution of a wall following behaviour

The main aims of the following experiments were firstly to compare GEP and GP in solving a wall following problem with increasing complexity, and secondly to determine whether the mgGEP can be used to evolve a layered controller architecture.

The experimentation replicates that of Lazarus and Hu [78] where a robot wall following behaviour was presented. In their work, Lazarus and Hu [78] investigated the capabilities of GP in evolving wall following behaviours in environments of increasing complexity. Their

experiments involved a robot moving in five room types where each room is a $16m \times 16m$ cell map (see Figure 4.3). The room types progress from the simple room 1 with an extrusion added in each subsequent room to add complexity. The outer cells represent the walls of the room. The robot was allowed to move in any empty cell and was awarded a fitness point for moving in the inner cells adjacent to the walls. The experiment was conducted on five different room types as shown by figure 4.3.

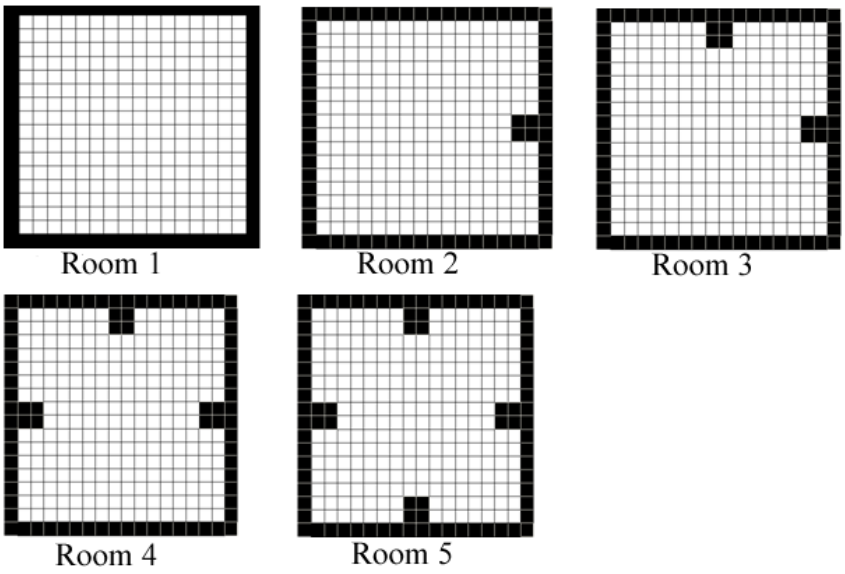


Figure 4.3: Five different room types used in the experiment. These room types have been adapted from Lazarus and Hu [78].

Lazarus and Hu [78] used eight sensors to help the robot detect any obstacles. The GP terminals and functions were defined as follows:

Terminal set: divided into two:

Sensor terminals: n, ne, nw, s, sw, se, w, e - in their work a sensor terminal returned a 0 if no obstacle was observed and 1 if there was an obstacle in the vicinity.

Action terminals: Move North, Move South, Move East and Move West. These terminals did not return any values to the calling program, however they altered the position of the robot and their execution caused the program to terminate. The robot 'hopped' from one cell to the other given the action terminal.

Function set: The function set consisted of the following:

- If (x, y, z) = y if x=1, z otherwise
- And(x, y) =0 if x =0 else y
- Or (x, y) =1 if x=1, else y
- Not(x) =0 if x=1, else 1

To run the algorithm the ramped half and half method was used to create and initialize the initial population. Ramped half and half method is a chromosome (trees in GP) generation method where half of the trees are generated randomly with no tree exceeding

the set maximum depth (i.e. grow method) and the other half is produced with all the trees with equal depth (i.e. full method) [74]). A population of 1000 organisms was used with maximum generations set to 100. The maximum nodes created per each organism was set to 100 while the depth of the tree was set to 10. Tournament selection was used and the only genetic variation operator was crossover. The algorithm forced the evaluation of the program to terminate when an action terminal was executed for the first time. Further calls to the same action terminal were ignored. Though being affected by local minima problems, results achieved showed that GP was able to evolve the wall following behaviour. Approximate average evaluations achieved across the five room types are as presented in Table 4.3.

4.3.1 Program implementation

In the work reported in this chapter, a 2D Java simulator was implemented to simulate the robot and its environment. Figure 4.3 shows the rooms that were used in the experiments. The robot was allowed to move into any empty cell and detected any obstacle adjacent to its cell location. The sensor terminals returned a 0 when there was no obstacle located at the particular sensor location around the robot, and 1 if there was an obstacle. In the implementation discussed here, the ugGEP and mgGEP algorithms were allowed to evolve a strategy to solve the problem and only terminated when the total number of allowed steps were exhausted. The robot state was set to collision when the intended movement would have led to the robot moving into an occupied cell (i.e. wall), when this was detected the robot earned a collision penalty and then stopped to move until the next step. In all the experiments carried out, the robot was allowed 100 steps in the environment. The robot was penalized for any wandering behaviour that did not award it any fitness points. To make sure that the algorithm evolved exploration capability, a high penalty was set if no movement was recorded in the first ten steps and the program terminated. Following the settings used by Lazarus and Hu [78], each robot controller was tested by starting the robot at 10 different starting points. The total fitness achieved was thus the sum of fitness achieved in each of the 10 start points.

4.3.2 Fitness function

The fitness was calculated as the summation of all new squares, next to the wall that a robot visited. The robot was penalised for moving into squares that were far away from the wall and also if any collision occurred.

$$\begin{aligned} f_j &= (\sum_{i=1}^{m} p_i(x_i, y_i)) - C - Wp \\ fitness &= \sum_{j=1}^{n} f_j \end{aligned} \tag{4.1}$$

Where f_j is the fitness value achieved when a start point (x_j, y_j) is used as the initial robot position. In the experiment, the total number of start points, n, was set to 10. Thus, the overall organism fitness is given by the summation of all fitness values, f_j, achieved from $j = 1$ to the total number of points, $j = n = 10$. The fitness point, $p_i(x_i, y_i) = 1$ whenever (x_i, y_i) has not previously been visited and 0 otherwise. The location (x_i, y_i) is a cell adjacent to the wall. The total number of cells next to the wall, m, is adjusted before a run depending on which room type being used in the experiment. See Table 4.1 for the maximum number of cells next to the wall. The collision penalty C, was set to 5 and the wandering penalty, Wp, was set to 10. The collision penalty was incurred when the intended movement would have led to the robot moving into an occupied cell (i.e. wall) whereas a wandering penalty was incurred when the robot visited any empty cells that were not adjacent to the wall. Table 4.1 shows the success predicates; maximum fitness achievable for each room.

Table 4.1: Success predicate

Room Type	Optimal result
1	$60 \times 10 = 600$
2	$64 \times 10 = 640$
3	$68 \times 10 = 680$
4	$72 \times 10 = 720$
5	$76 \times 10 = 760$

4.3.3 Parameter settings and algorithm parameters

Table 4.2: Parameter settings

Parameters	ugGEP	mgGEP
Maximum generations	200	200
Population	500	500
No. of Genes	1	2
Head size	16	8
Parent organisms	2	2
Mutation	0.041	0.041
One Point Recombination	0.7	0.7
Two Point Recombination	0.2	0.2
Gene Recombination	0.0	0.1
Insertion Sequence Transposition	0.1	0.1
Root Insertion Sequence Transposition	0.1	0.1
Gene Transposition	0.0	0.1
Selection range	5%	5%
Functions(If, And, Or, Not)	4	4
Terminals(8 sensors, 4 motors)	12	12
No. of starting points	10	10
No. of randomly seeded runs	50	50

Table 4.2 shows the algorithm parameters, population size, number of generations in a run and the probabilities of each of the genetic operations used by the algorithm. As previously mentioned, each controller in the population was tested from 10 starting points and the

fitness evaluated using Equation 4.1. The maximum fitness for each room is as shown by Table 4.1. In all the experiments, 50 randomly seeded algorithm runs were conducted. An evolutionary run terminated when the total number of generations was reached.

4.3.4 Algorithm primitives

The wall following problems presented in this chapter used the following terminals and function sets. The terminal set was categorised to either robot sensors or motor/action terminals.

Robot sensors: Following the representation described in [78], the robot was implemented with eight sensors described as (front (F), front right (FR), front left (FL), back (B), back left (BL), back right (BR), right (R), and left (L)).

Action terminals: There were four types of movements that the robot could achieve. This were; Move Forward (MF), Move Back (MB), Move Right (MR) and Move Left(ML). These action terminals moved the robot one cell step in either direction.

Functions set: The function set consisted of: And (A), Or, Not (N), If - as defined in section 4.3.

In addition to the aforesaid functions, the algorithm was also set such that, if the left branch of the tree (first argument to a function) was an action/motor terminal, then the function would return that particular action terminal.

4.3.5 Experimental results

Two experiments were conducted in this section:

- TEST 1: In the first experiment, the ugGEP algorithm was used to evolve the wall following behaviour using parameter settings as defined on Table 4.2 (Test 1). The main aim of this experiment was to compare the performance of a monolithic (ugGEP) algorithm to that of a GP.
- TEST 2: The second experiment involved the use of multiple genes (mgGEP) chromosomes to evolve the wall following problem. The main aim of this experiment was firstly to investigate how a modular algorithm (mgGEP) would scale to the wall following problem and secondly to compare the performance of a modular algorithm to that of a monolithic GEP (ugGEP) and GP. As shown on Table 4.2, the mgGEP was implemented using 2 genes each with a head size = 8 while the ugGEP had the head size = 16. This meant that the chromosomes were of equal length and differed only in the number of genes used. A logical IF was used as the linker with three arguments. The first argument was the FRONT sensor F, and the second argument was the first gene and third argument was the second gene. The linker decides which of the two genes effects motors control by checking the sensor reading of the front sensor. If Front sensor returns 0, that is no obstacle in the adjacent cell, then gene

one effects the robot's motor control, however if F = 1, meaning an obstacle is detected then gene 2 effects motor control. The rest of the parameters are as described in Table 4.2 above.

The next section describes the obtained results.

Comparison with GP

Table 4.3: Comparison of average evaluations of the GP, ugGEP and mgGEP

Room type	1	2	3	4	5
GP[1]	3000	90000	20050	50000	**14000**
ugGEP	**500**	30616	**19871**	19425	15825
mgGEP	**500**	**27611**	21652	**18809**	19885

Table 4.3 shows the mean fitness evaluations performed in all the runs to solve the wall following problem using ugGEP and mgGEP algorithms, alongside the results of the GP algorithm reported in [78]. The best results are shown in bold. It is clear from Table 4.3 that in general, GEP evolves successful controllers (with fitness as shown on Table 4.1) with fewer evaluations than GP. The ugGEP and mgGEP algorithms outperform the GP with large margins across rooms, 1, 2, and 4. The performance is very close in room type 3 and GP performs slightly better in room type 5.

The ugGEP and mgGEP chromosomes are made up of fewer total number of alleles than the GP. As mentioned previously, the GP's total number of alleles was set to 100 while the ugGEP and mgGEP's total number of alleles was set to 49 and 50 alleles respectively. Therefore, in this particular problem, the obtained results suggests that the GEP algorithms are capable of both outperforming (as shown by performances in rooms 1, 2 and 4) and approximating (as shown by performances in rooms 3 and 5) the GP performance, using approximately half the number of alleles. Also, the number of evaluations required to achieve success is much more consistent using the GEP runs. In the GP case, rooms 3 and 5 appears to have been solved with fewer generations whereas $50,000$ and $90,000$ evaluations are required for rooms 2 and 4. In the experiments reported here, the GEP results have been computed from the mean of all successful runs (those that achieved the set optimum fitness) over 50 random seeds, thus, these results give a clearer picture on how the GEP performs. It is also important to note that all genetic operations as describe in table 4.2 above were used compared to the GP where only crossover was used.

Performance comparison: ugGEP versus mgGEP

Table 4.3 shows that the ugGEP and mgGEP algorithms outperformed the GP in the wall following problem. These results suggest that in this problem domain, the GEP algorithms are a suitable alternative to GP. The ugGEP algorithm appears to have evolved solutions

[1]The GP results shown here are approximate values derived from [78]. Experiments described in this chapter do not duplicate the GP experiments conducted by Lazarus and Hu [78], however, the GEP experiments replicates the environments, algorithm settings, fitness function and chromosome representations as used in the GP.

to the problem using fewer evaluations than mgGEP in room type 3 and 5. However, the mgGEP used fewer evaluations than ugGEP in room 2 and 4. Both algorithms evolved a successful controller for room type 1 after only one generation (500 evaluations). Therefore, in these experiments, the obtained results suggests that ugGEP and mgGEP algorithms require almost an equal number of evaluations to evolve a successful solution. This is the likely outcome since the ugGEP and mgGEP chromosomes were of equal lengths. However, results above only focus on the successful controllers (those with a maximum fitness as defined on Table 4.1) and not the overall performance of the best individual in the population. This section thus presents further performance comparisons and analysis for ugGEP and mgGEP algorithms. Since both ugGEP and mgGEP solved room 1 after only one generation, further analysis on results based on this room shall not be considered.

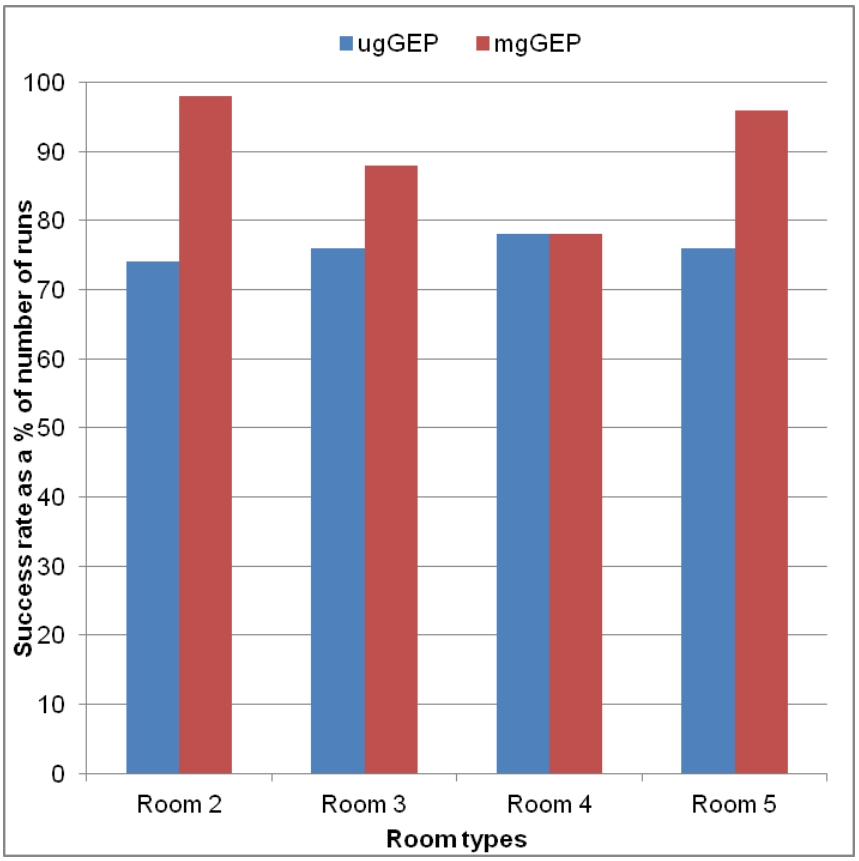


Figure 4.4: Comparison of ugGEP and mgGEP success rates across the five different room types. The success rate refers to the percentage of the total number of runs where the target fitness was achieved. Each evolutionary run had a population of 500 organisms and lasted for 200 generations. In all the experiments, 50 randomly seeded algorithm runs were conducted. The rest of the parameters are as shown on Table 4.2.

Figure 4.4 shows the percentage of actual number of runs out of the 50 runs that successfully solved the problem. The result shows that mgGEP algorithm was more successful than ugGEP in room types 2, 3, 5 while both algorithms had equal success rates in rooms 1 and 4. These results suggests that there was a high likelihood of obtaining a successful controller while using mgGEP, although both algorithms may have required an equal number of evaluations to solve the problem. The good performance shown by mgGEP is likely to have been as a result of using behaviour sub-division mechanisms to solve the

problem. The obtained result suggests that the use of modular controllers as implemented through mgGEP is likely to lead to more robust controllers than monolithic controllers evolved using ugGEP.

Best individual performance

The results presented in Figure 4.5 shows the mean of the best controller fitness in every generation across all the 50 runs. These graphs presents a clear picture of the performance of the best individual evolved by GEP as the run progressed.

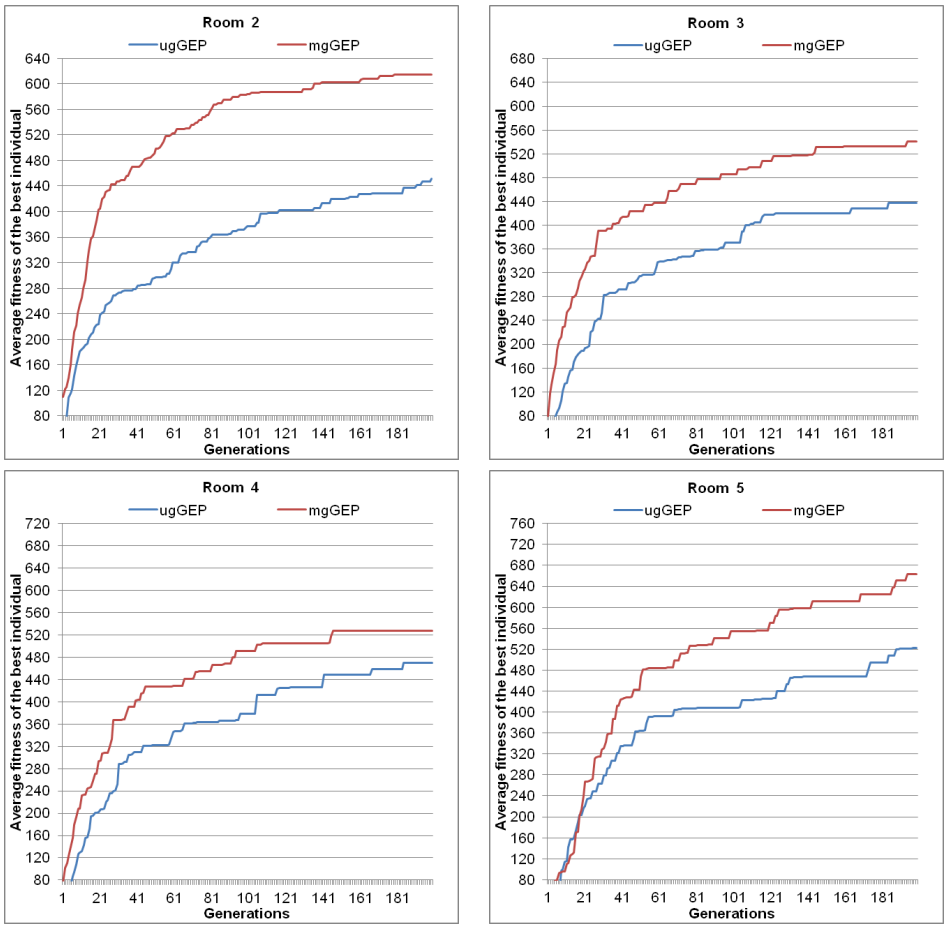


Figure 4.5: Progression of the average fitness of the best individual in the population as achieved through ugGEP and mgGEP in the different room types. The average fitness has been computed from the best individual fitness achieved in each generation in all the 50 randomly seeded algorithms. In each evolutionary run, a population of 500 organisms was evolved for 200 generations and the best individual fitness recorded in each generation. The rest of the parameters are as shown on Table 4.2.

The results show that the best individual evolved using mgGEP had higher fitness on average than those evolved using ugGEP, in every generation across all the room types. Additionally, the mgGEP algorithm evolved the best individual in the population quicker than ugGEP.

Average population fitness

Figure 4.6 shows the progression of average fitness of the population as achieved by ugGEP and mgGEP. Across all the room types mgGEP achieved better average fitness compared to the ugGEP.

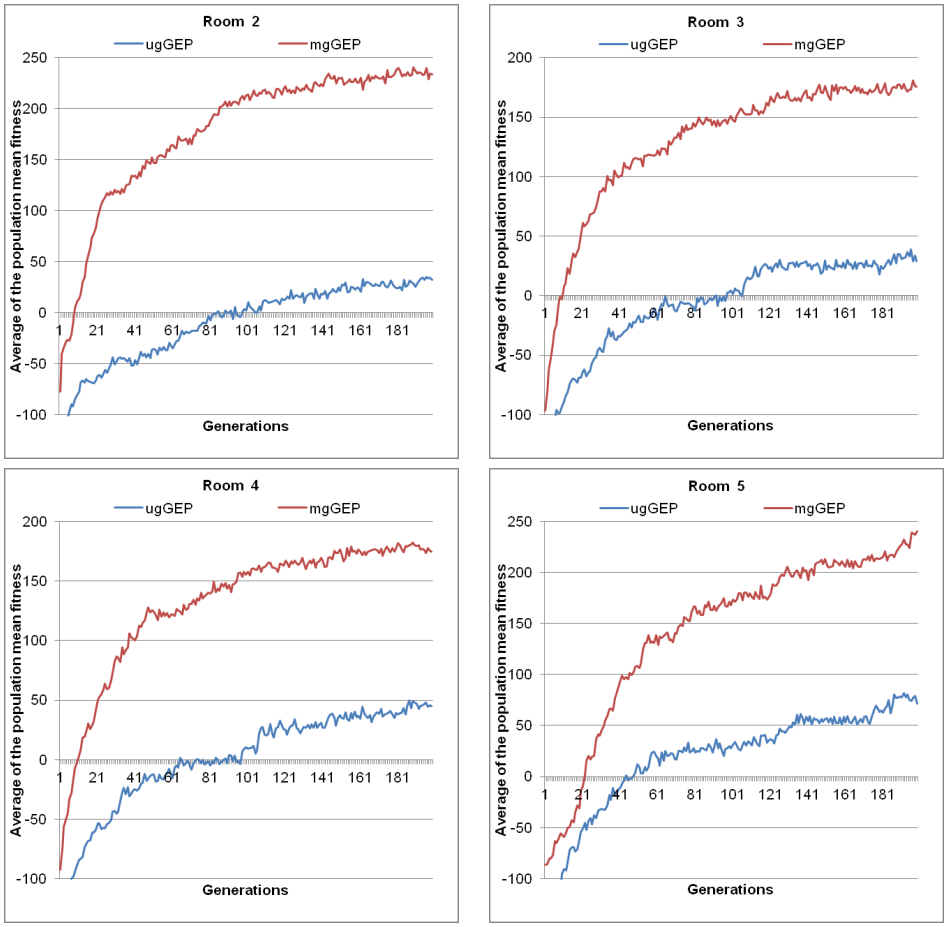


Figure 4.6: Progression of the average population mean fitness as achieved through ugGEP and mgGEP in the different room types. The average population mean fitness was computed from the population mean fitness achieved in each generation in all the 50 randomly seeded algorithms. In each evolutionary run, a population of 500 organisms was evolved for 200 generations and the population mean fitness recorded in each generation. The rest of the parameters are as shown on Table 4.2.

The results shown by Figures 4.5 and 4.6 suggests that modular controllers evolved using mgGEP achieved a better performance than those evolved using ugGEP, across all room types. The success of the modular controllers is likely to have been due to the evolution of specialised modules to solve the wall following problem. Since controllers evolved using ugGEP had to solve the entire problem using only one gene, they are likely to have been more general and hence less robust. Below we investigate the evolved controllers to demonstrate the improvements possible in the mgGEP controllers.

Evaluating the genome

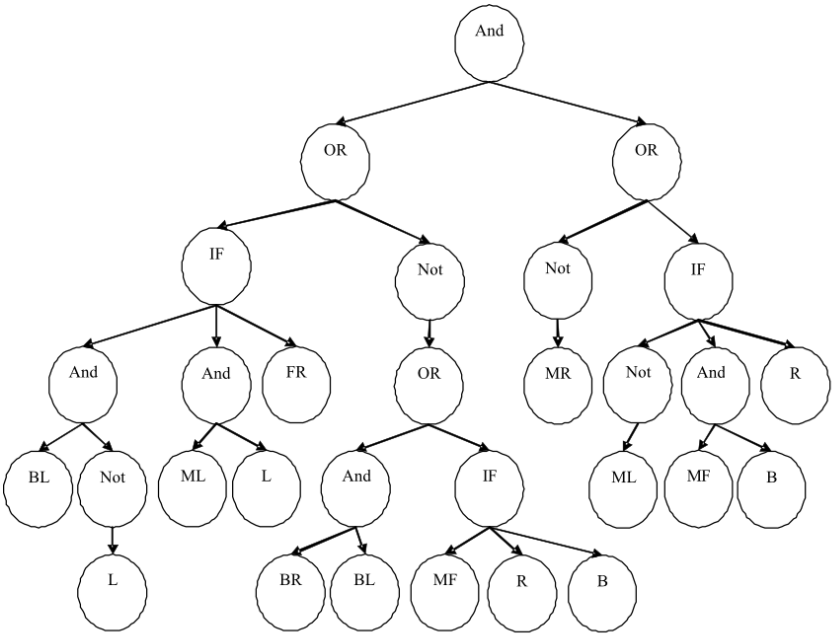


Figure 4.7: An example of a controller generated using ugGEP while solving room type 4.

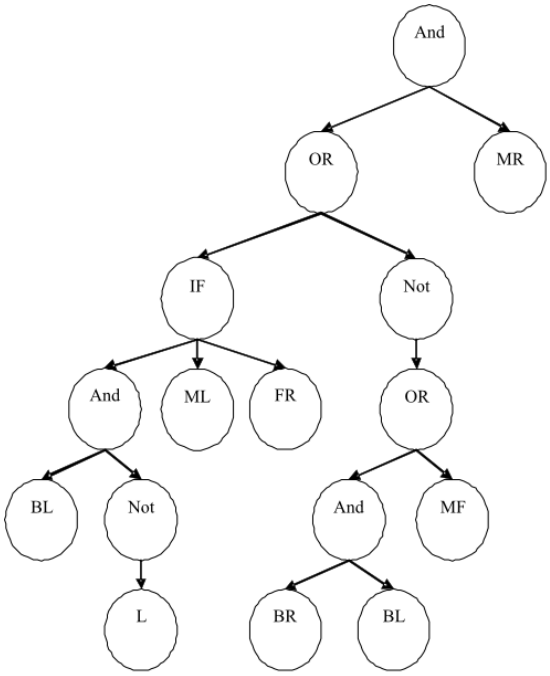


Figure 4.8: Simplified version of figure 4.7

The full ugGEP controller, Figure 4.7, shows that a lot of redundancy was required by ugGEP to solve the problem. Due to its modularity, the mgGEP controller is able to achieve the same behaviour with many fewer nodes in the tree (15 as opposed to 31 for the unigenic controller). This explains the largely superior performance of the mgGEP controller in the number of evaluations required to optimise the rooms. Detailed analysis

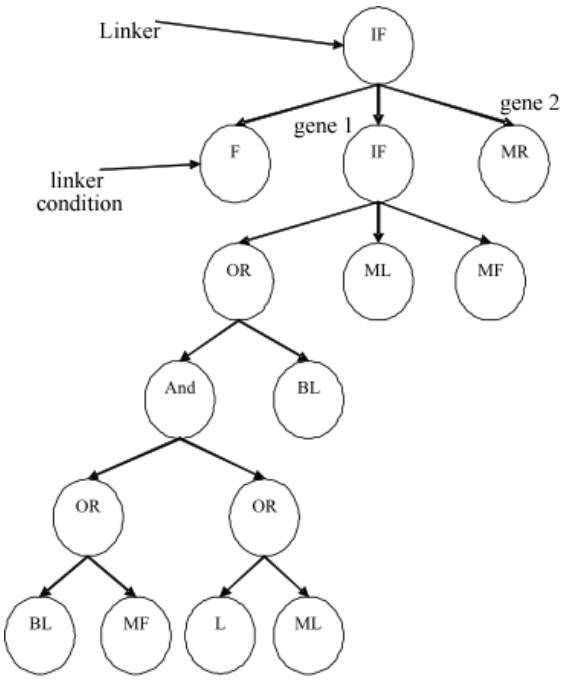


Figure 4.9: Example of mgGEP controller evolved while solving room type 4

done by simplifying the controller (see figure 4.8) shows that the ugGEP algorithm divided the problem into two branches, the left hand side tackled wall following problem and the right hand side obstacle avoidance. This is akin to evolving subsumption architecture.

In the mgGEP phenotype shown by Figure 4.9, the second gene has been evolved to turn the robot right whenever the forward sensor detects an obstacle, this means that the second gene guides the robot to evolve obstacle avoidance behaviour. The first gene enables the robot to follow the wall paying attention to only left and back left sensors which are required in order for the robot to navigate around the extrusions in the rooms. This shows that mgGEP evolved the controller into two modules, each independent but whose interaction with the environment resulted to the wall following behaviour.

4.3.6 Discussion

Two main theories could be used to explain the success of mgGEP algorithm in this problem. Firstly, the increased diversity in its chromosome due to gene transposition and gene recombination; these two operators when used in conjunction have a great transforming power in the solution set [38]. Secondly, and perhaps most importantly mgGEP divides the task into its two constituent elements, obstacle avoidance and wall following. This division of tasks into separate modules has been a feature of many successful robotic controller systems and the unique formulation of mgGEP allows each element of the task to be evolved separately whilst solving the global problem. As previously reported, many approaches to AI robotics use modular architectures such as incremental learning [142, 55, 5, 46], layered learning [117, 136] and subsumption architecture [83], to determine levels of behaviour. The results shown here suggests that mgGEP algorithm is a powerful approach that can be used to evolve modular robot controllers that have a predetermined action

selection (behaviour coordination) mechanism. In comparison to the existing approaches, the mgGEP algorithm offers continuous learning during the training stage as well as a more biologically plausible approach to evolution.

4.4 Evolving in a 3D world model

Following the performance improvements achieved using ugGEP and mgGEP algorithms in a 2D discrete model, the experiments were extended to a 3D model using the simbad simulator [2] [66]. Experiments in a 3D world model are important because its easier to model the real world. Additionally, noise can be added during the simulation in order to evolve more robust controllers. In the 3D simulator used in this experiment, the robot sensors report distance away from obstacles within a set range (in this case its set to $0.0m$ and $1.5m$), this means that the sensors are not constrained to return a false/true evaluation. Additionally, to solve the wall following problem, the robot has to maintain close proximity to the walls whilst avoiding a collision with them.

4.4.1 Experimental set up

Previously, it was reported that controllers for room type 1 were easily evolved as the structure is simple in nature since it has no extrusions. Following this, the 3D experiments focussed on rooms type 2-5. Similar to the 2D environments, the 3D rooms were $16m \times 16m$ with extrusions as shown by Figure 4.10.

The robot movements were controlled using the default kinematic model describe in section 3.2.1. Subsequently, the translational velocity was set at $2.0m/s$ for forward movement (**F terminal**) and $-2.0m/s$ for reverse movement (**B terminal**). Similar to the settings used by the standard ugGEP in the previous chapter, the angular or rotation velocity was not set. However, depending on the terminal returned by the algorithm, the robot rotated based on the radians as reported in Table 3.1. In the experimentations reported here, the robot used eight infra-red sensors and eight wheel motor actions as shown on Table 3.1. Thus, the robot actions and its perception of the world is similar to the set up in the obstacle avoidance problem discussed in Chapter 3. Due to the change of environments (from 2D discrete environments to 3D environments) and use of robot sensors that return a range of values between $0.0m$ and $1.5m$, three float constants (0.1, 0.2, 0.3) were added to the terminal set to ensure that the algorithm had the capability to evolve viable and successful controllers. An 'If Less Than or Equal to' (IFLTE), function was used as a sole conditional function in the function set. As shown, in Chapter 3, the combination of the reported terminal set and IFLTE function is robust enough to steer the robot and generate behaviours using a very short phenotype. Following this, the ugGEP was set up with head size, h, set to 8 whereas each gene in the mgGEP had the head size set to 4, the chromosomes were thus equal in length. For each algorithm investigated, 20 randomly seeded algorithms runs were conducted for each of the rooms. The rest of the algorithm

[2] http://simbad.sourceforge.net/

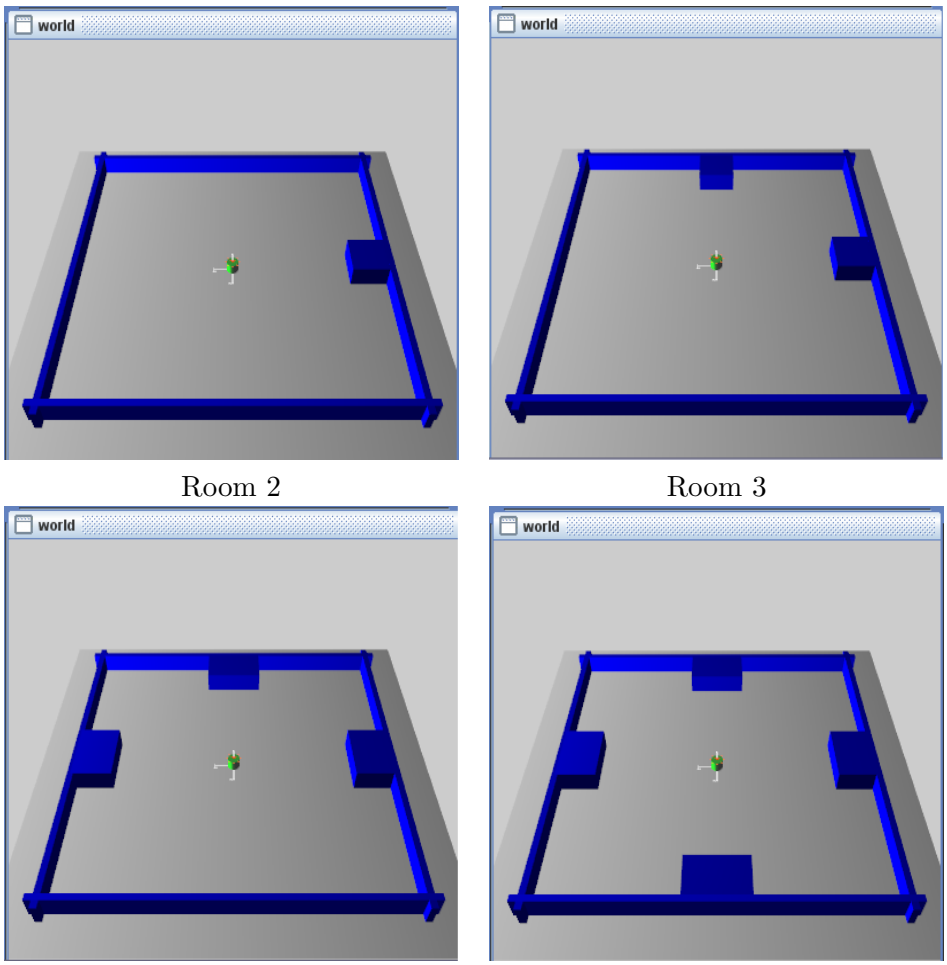


Figure 4.10: Robot environments as set up in simbad simulator

parameters, including the number of generations and size of population, were set up as shown by Table 4.4 below.

Algorithm parameter settings

Table 4.4: 3D experiments: Algorithm parameter settings

Parameters	ugGEP	mgGEP
Maximum generations	200	200
Population	500	500
No of Genes	1	2
Head size	8	4
Parent organisms	2	2
Mutation probability	0.061	0.061
1-Point Recombination probability	0.7	0.7
2-Point Recombination probability	0.2	0.2
Gene Recombination probability	0.0	0.1
IS Transposition probability	0.1	0.1
RIS Transposition probability	0.1	0.1
Gene Transposition probability	0.0	0.1
Selection range	5%	5%
Functions(IFTLE)	4	4
Terminals(8 sensors/motors, 3 float values)	11	11
No. of starting points	10	10
No. of randomly seeded runs	20	20

4.4.2 Fitness function

In these experiments, the wandering penalty term as reported in section 4.3.2 was removed from the fitness function. Instead, the robot was allowed to move in the environment for 50 simulation seconds during which fitness points were awarded for visiting new coordinates within 1 metre from the wall. The robot is also penalised for stagnation (when odometer reading is 0.0 for a duration of 5 simulation seconds) and for hitting obstacles; in the both cases the robot is penalised and the algorithm stopped. The fitness function is as shown in Equation 4.2.

$$\begin{aligned} f_j &= (\sum_{i=1}^{m} p_i(x_i, z_i)) - C - S \\ fitness &= \sum_{j=1}^{n} f_j \end{aligned} \tag{4.2}$$

Where f_j is the fitness value achieved when a start point (x_j, z_j) is used as the initial robot position. In the experiment, the total number of start points, n, was set to 10. Thus, the overall organism fitness is given by the summation of all fitness values, f_j, achieved from $j = 1$ to the total number of points, $j = n = 10$. The fitness point, $p_i(x_i, z_i) = 1$ whenever

(x_i, z_i) has not previously been visited and 0 otherwise. The location (x_i, z_i) has to be less than 0.5 metres away from the wall for a point to be earned. The total number of locations where a fitness point can be earned, m, is adjusted before a run depending on which room type being used in the experiment. See Table 4.1 for the maximum number of cells next to the wall. The collision penalty C, was set to 5 and the stagnation penalty, S, was set to 10. Table 4.1 shows the success predicates; maximum fitness achievable for each room.

4.4.3 Experimental results

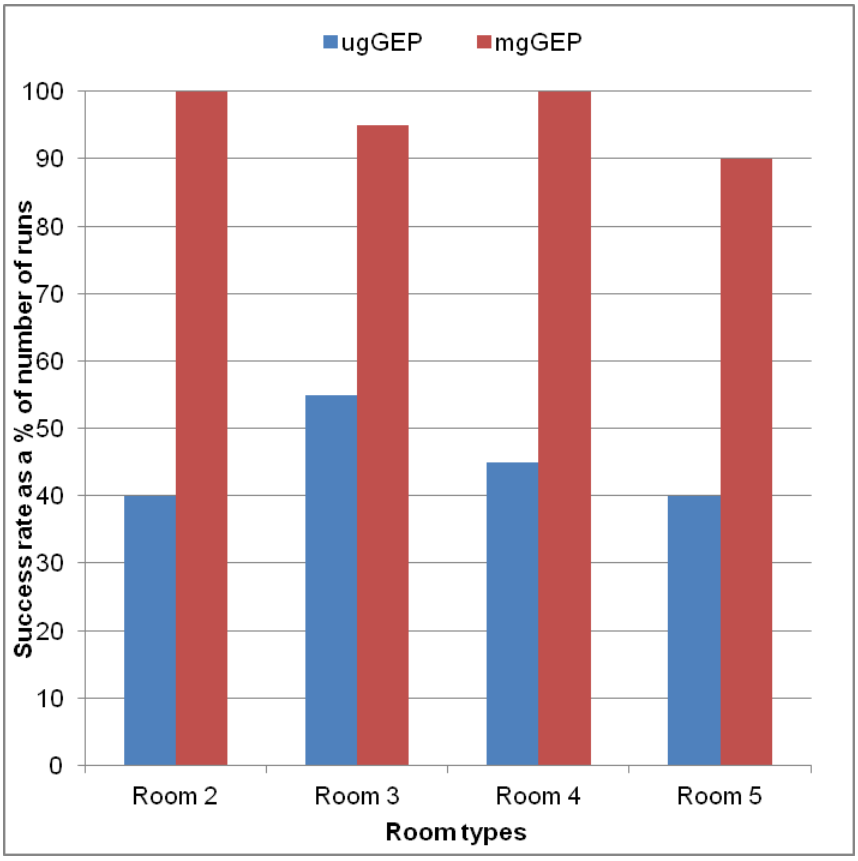


Figure 4.11: Comparison of ugGEP and mgGEP success rates across the five different room types. The success rate refers to the percentage of the total number of runs where the target fitness was achieved. Each evolutionary run had a population of 500 organisms and lasted for 200 generations. In all the experiments, 20 randomly seeded algorithm runs were conducted. The rest of the parameters are as shown on Table 4.4.

The results shown by figure 4.11 strengthens the conclusions reported in section 4.3.5 above: In this problem domain, the mgGEP algorithm is a better technique than ugGEP algorithm. In all the four room types under investigation, there was at least 90% success rate by the mgGEP compared to the 40% achieved by ugGEP. As reported earlier, the mgGEP algorithm evolves the problem in two modules, that is, wall following and obstacle avoidance. The obtained results suggests that behaviour sub-division gives mgGEP controllers a competitive edge, compared to the monolithic structures evolved using ugGEP.

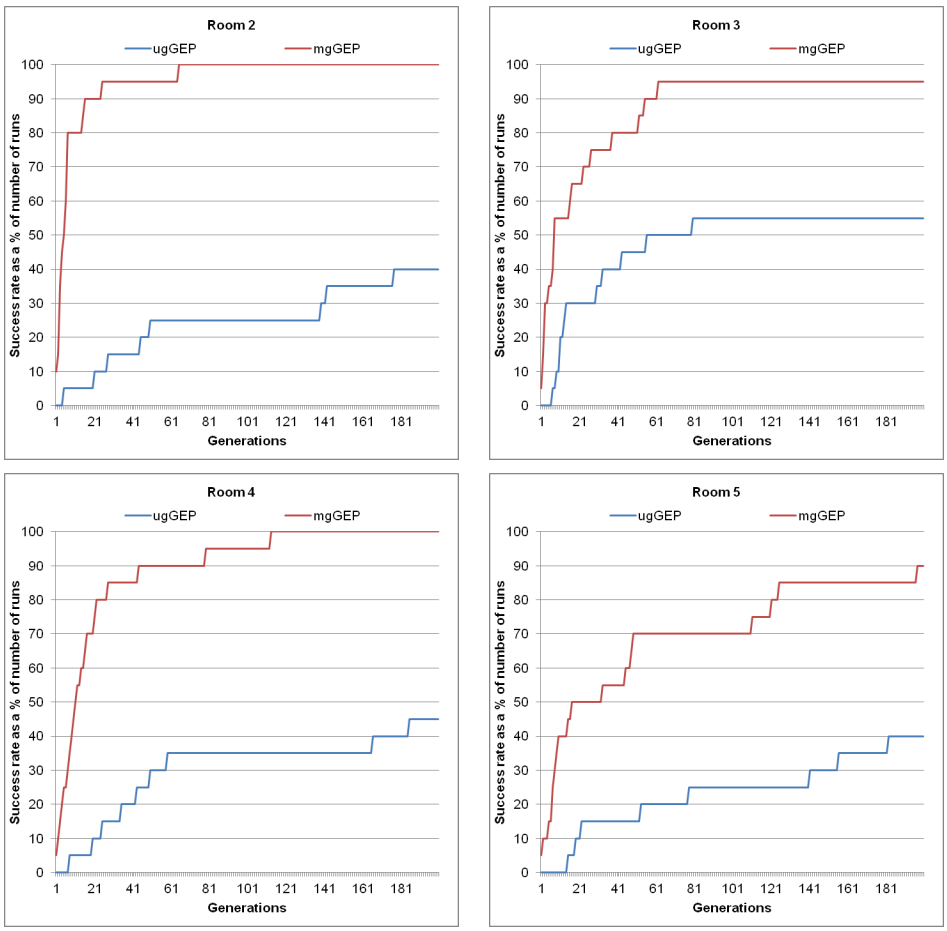


Figure 4.12: Progression of success rates with number of generations using ugGEP and mgGEP in the four room types. The success rate refers to the percentage of the total number of runs where the target fitness was achieved. In each evolutionary run, a population of 500 organisms was evolved for 200 generations. In all the experiments, 20 randomly seeded algorithm runs were conducted. The rest of the parameters are as shown on Table 4.4.

The comparison of success rates shown by Figure 4.12 suggests that mgGEP algorithm evolved an optimal controller on the first generation across the four room types. The ugGEP on the other hand took between $5-10$ generations to evolve an optimal controller. The progression also shows that the ugGEP takes a longer time than mgGEP to evolve optimal controllers. Additionally it was observed that all the evolved controllers, solved the problem when they were introduced in rooms other than the one they were evolved in. This strengthens the conclusions in the previous chapter regarding the adaptation capabilities for controllers evolved using GEP.

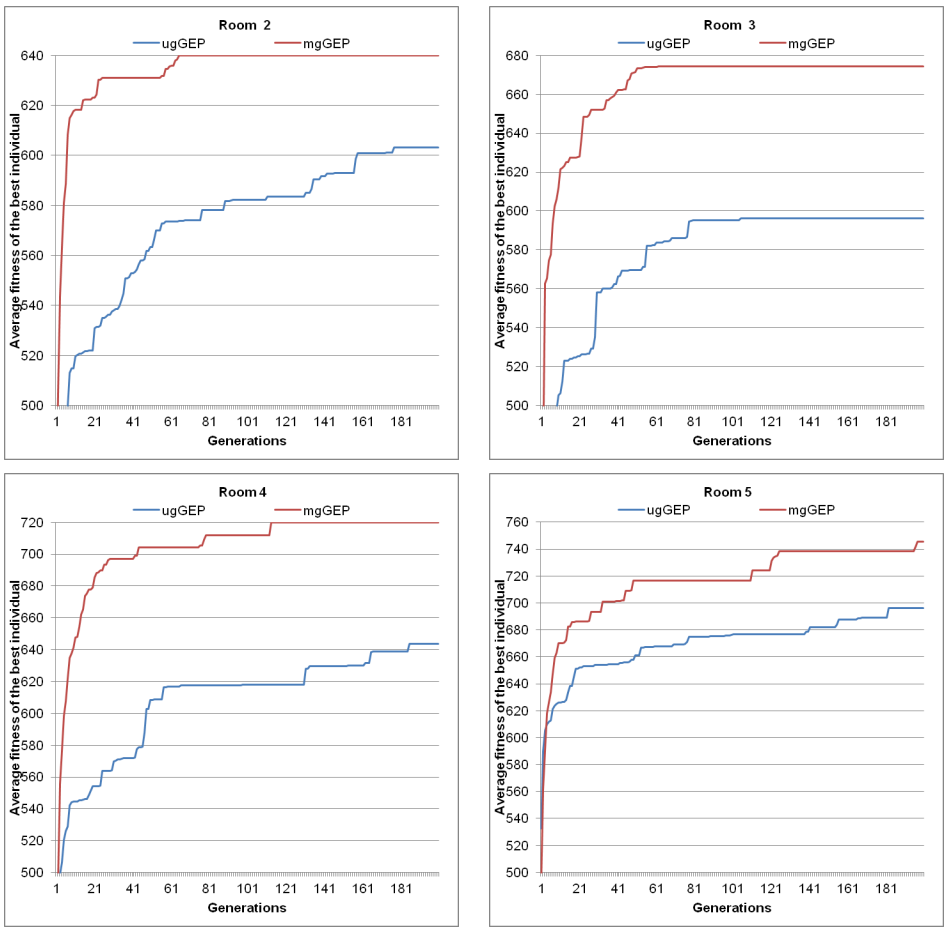


Figure 4.13: Progression of the average fitness of the best individual in the population as achieved through ugGEP and mgGEP in the different room types. The average fitness has been computed from the best individual fitness achieved in each generation in all the 20 randomly seeded algorithms. In each evolutionary run, a population of 500 organisms was evolved for 200 generations and the best individual fitness recorded in each generation. The rest of the parameters are as shown on Table 4.4.

Figure 4.13 shows that the mgGEP algorithm was quicker in evolving the best individual than the ugGEP. In addition, it was observed that the evolved best individual (at the end of a generation run) was either optimal or near-optimal solution. The ugGEP on the other hand is slower and overall the achieved best individual does not solve the problem fully. The division of the task, particularly having a module that focuses on how close the robot moves to the wall means that it is easy for the robot to pick up fitness points without having to concentrate on obstacle avoidance.

The average performance of the population, Figure 4.14, shows that on average mgGEP outperformed ugGEP across all the room types. The good performance can be attributed to the behaviour sub-division as well as genetic operations. As shown by Figure 4.14, the progression of average performance is less stable in ugGEP across the four rooms. This may have been brought about by negative genetic operations in a previous generation, a situation that appears not to affect the mgGEP controllers. As reported earlier any negative effect that may affect a gene after an operation, is shared amongst the two genes in mgGEP. This "sharing" of negative effects means that the performance is more

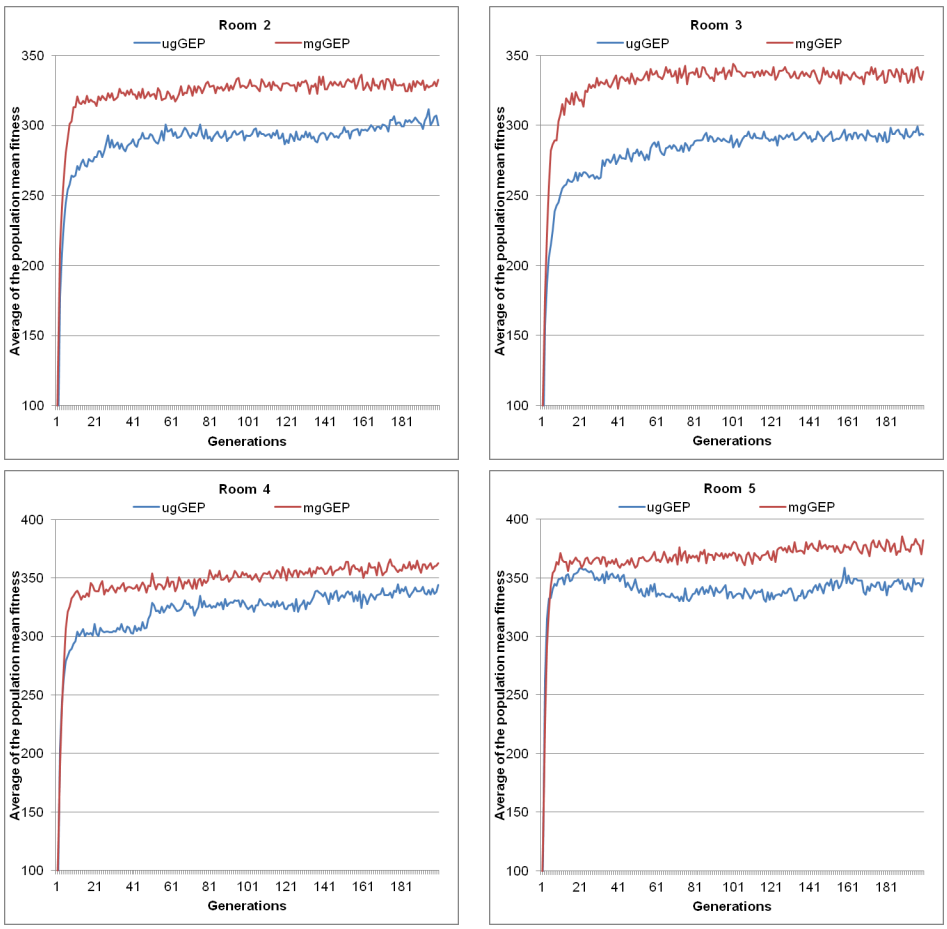


Figure 4.14: Progression of the average population mean fitness as achieved through ugGEP and mgGEP in the different room types. The average population mean fitness was computed from the population mean fitness achieved in each generation in all the 20 randomly seeded algorithms. In each evolutionary run, a population of 500 organisms was evolved for 200 generations and the population mean fitness recorded in each generation. The rest of the parameters are as shown on Table 4.4.

stable than in ugGEP. Additionally, diversity created by use of gene recombination and transposition is likely to have led to the success of the mgGEP algorithm.

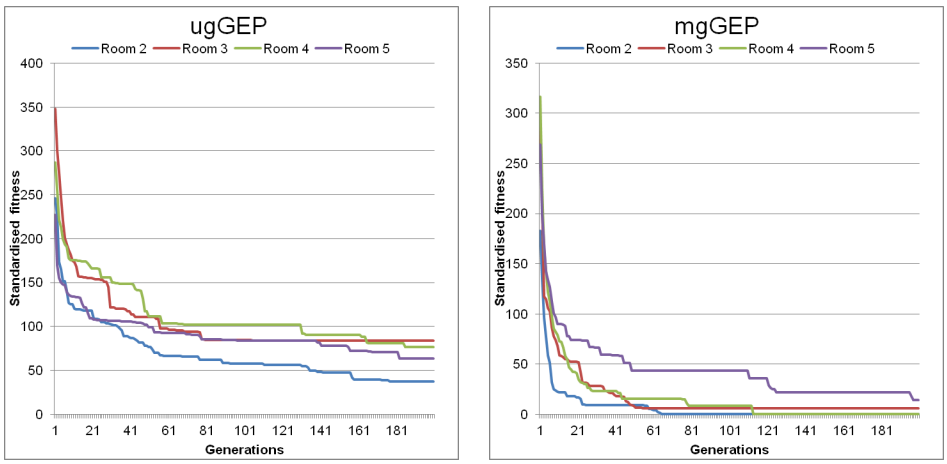


Figure 4.15: Progression of standard fitness with number of generations. The standard fitness is derived from the difference of the target fitness in each room and the averaged fitness of the best individual in each generation. A low standard fitness means that the algorithm is converging towards the target fitness. In this experiment, a population of 500 organisms was evolved for 200 generations and the fitness of the best individual recorded in each generation. A total of 20 randomly seeded evolutionary runs were conducted as shown on Table 4.4.

In carrying out this experiment, Lazarus and Hu [78] aimed to investigate the performance of GP with increasing complexity. In their results, they concluded that the GP was not affected by the complexity since it was hard to determine a pattern on the performance. Similarly, as shown by Figure 4.15, it was hard to determine a particular pattern on how the ugGEP and mgGEP algorithms solved the problem. For instance, up to the $80th$ generation, mgGEP appears to solve room 2 easily, followed by room 3 and progression on to room 5, however room 4 is by then solved completely, the performance surpassing the performance in room 3. For the ugGEP case, there is no pattern that can be determined at all. This suggests that both GEP controllers are not affected by increase in complexity of the environment.

4.5 Effect of behaviour coordination mechanism

The previous sections focussed on dividing the global behaviour to simpler tasks and then evolving the sub-controllers simultaneously in order to solve the problem. The achieved results suggests that this mechanism is better than monolithic evolution and has more advantages vis-a-viz other divide and conquer techniques such as incremental evolution. In this section, we raise the question: "Does the position of sub-controller in the overall controller matter?". More specifically, the aim is to investigate the effect of action selection mechanism or behaviour coordination strategy in the performance of a controller.

To attempt an answer to this question, a robot foraging behaviour was evolved using both ugGEP and mgGEP algorithms. The robot was required to navigate around the environment looking for "food sources" placed in various locations within the robot world

(see figure 4.16).

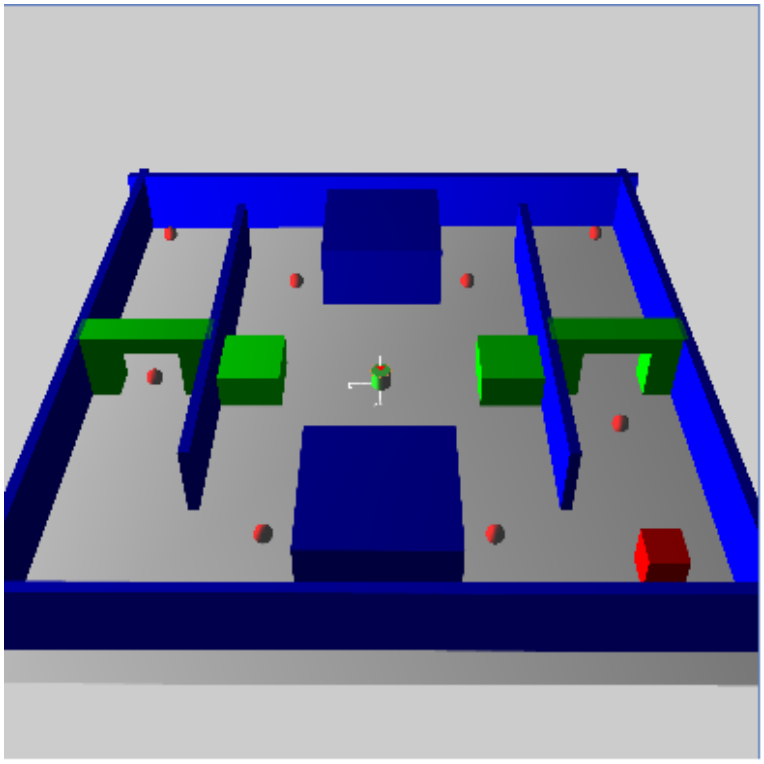


Figure 4.16: Robot world used in the foraging experiments. The round objects represents "food sources" that the robot picks to improve its energy while navigating.

The next sections outline the implementation and experimental set-up, presents the experimental results and lastly a discussion of the relevance of the obtained results to modular robot control architectures.

4.5.1 Experimental set up

Implementation

In the experiments reported here, the robot moves within the environment looking for food sources placed in various locations in the environment. The simbad simulator, introduced in sections 2.4.4 and 3.2.1, was used to simulate the robot and its environment. Simbad's default kinematic model was used to control the robot movements. Subsequently, the wheels translational velocity was set at $2.0m/s$ for forward movement and $-2.0m/s$ for reverse movement.

The robot was set to alter its position and orientation using 8 motor functions and perceived its environment using 8 infra-red sensors to measure obstacle proximity (the set up for motors and sensors is as reported on Table 3.1). The robot was also equipped with a virtual battery, whose energy output enabled the robot to act on the environment. Additionally, the robot was provided with a "food detector" sensor to enable the robot detect the food sources. The food detector sensor was set to be always "ON"; this means that the robot was always active to pick up any food source once detected. Once the robot detected a food source, the robot's energy was increased and the food source was removed

from the environment. In addition, the robot lost some amount of energy every time the controller effected motor control. Thus, the robot had to minimize energy lost yet at the same time maximise time spent in the environment. To do this, the robot increased its battery level if a food source was found and continued to lose energy for every timestep spent in the environment. The global behaviour can thus be divided into three sub-tasks; obstacle avoidance, exploration and foraging (that is, searching for food source to improve the robots energy).

At the start of every experiment the robot battery level was set to 1. The robot incurred a loss of 0.001 on its battery level each time the controller was executed to effect motor control on the robot. The initial energy level means that the robot could execute 1000 steps if no collisions occurred and if it did not stagnate. For every food source found, the robots battery level was increased by 0.2. The robot's maximum life span in the environment was set to 150 virtual seconds or 3000 steps: The inter step delay in simbad is $0.05s$ (please see [65]).

Algorithm set up

The algorithm was implemented using the same function and terminal sets as describe in section 3.2.2. However, an addition energy terminal "E" was provided: this terminal when executed returned the robot's battery level. Thus, the algorithm used 1 function and 9 terminals.

In all the described experiments, the mgGEP algorithm was implemented using three genes. Each gene was executed to solve a particular task depending on the behaviour coordination mechanism utilised. In the mgGEP algorithm, each gene had the head size, h, set to 4. Thus, using Equation 2.2 the total gene length was 17 alleles and therefore the mgGEP chromosome length was 51 alleles. Since the mgGEP genes were formed using a head size = 4, the ugGEP algorithm was implemented using a head size = 12 resulting to a length of 49 alleles. This is the closest chromosome length to the 51 alleles in mgGEP, since using a head size = 13 would result to a chromosome length of 53 alleles.

In all the experiments, a population of 100 chromosomes was run for 200 generations. To ensure that the overall results were not affected by random occurrences, 20 randomly seeded algorithm runs were conducted for each experiment. Table 4.5 provides a summary of the utilised parameters.

Table 4.5: Robot foraging behaviour: Algorithm parameter settings

Parameters	ugGEP	mgGEP
Maximum generations	200	200
Population	100	100
No. of genes	1	3
Head size	12	4
Parent organisms	2	2
Mutation probability	0.041	0.041
1-Point Recombination probability	0.7	0.7
2-Point Recombination probability	0.2	0.2
Gene Recombination probability	0.0	0.1
IS Transposition probability	0.1	0.1
RIS Transposition probability	0.1	0.1
Gene Transposition probability	0.0	0.1
Selection range	5%	5%
Functions(IFLTE)	1	1
Terminals(R, L, F, B, FR, FL, BR, BL, E[3])	9	9
No. of randomly seeded runs	20	20

The fitness function shown by Equation 3.3 was utilised in the experiments reported here. This fitness awarded the robot 1 point for visiting unique positions in the environment and penalised the robot 25 points for hitting obstacles and 50 if the robot did not move for the first 5 virtual seconds. In addition to the penalties, the controller execution terminated if the robot did not move with 5 virtual seconds and if a collision occurred. The maximum fitness was set to 3000 corresponding to the maximum life span of the robot.

4.5.2 Experimental results

The mgGEP algorithm was used in three experiments to determine the effect of a particular sub-controller position to the overall performance of the whole controller. A comparison of the results of the ugGEP algorithm and the various action selection mechanism was then conducted.

Experiment I

In the first experiment, the mgGEP algorithm was implemented using sub-behaviour model shown by Figure 4.17. This model was referred to as mgGEP with obstacle avoidance as a priority behaviour: mgGEP (OA priority) because the root IF statement considers obstacle avoidance the dominant behaviour.

[3]Please see section 3.2.2 on how the sensors and motor terminals are implemented. E terminal refers to the robot battery.

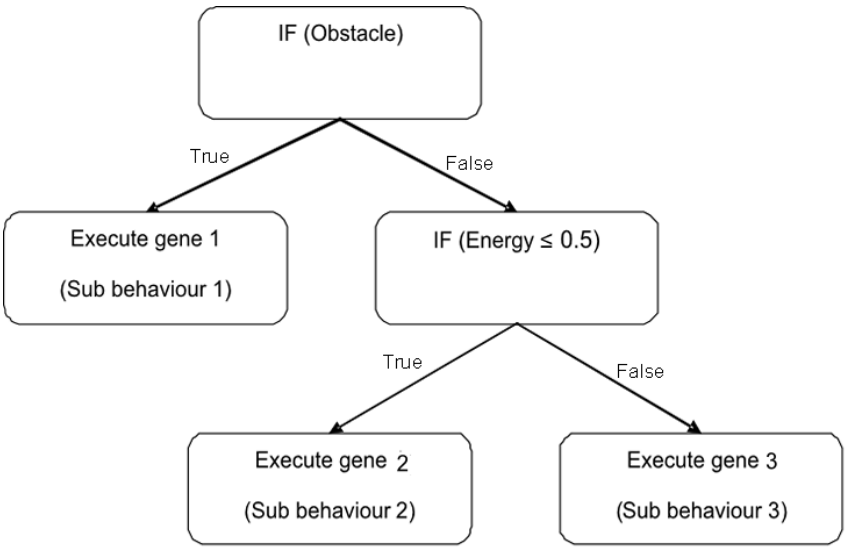


Figure 4.17: A behaviour coordination model that relies on robot proximity to obstacles in order to choose a robot task.

In this behaviour coordination mechanism, the first gene (gene1) was executed if the robot was within $0.5m$ of an obstacle, the second gene was executed if the robots energy fell within 0.5 and the last gene was executed when the above two conditions did not exist. Figure 4.18 shows a comparison of the performance of mgGEP (OA priority) and the ugGEP algorithm in evolving suitable controllers for a foraging robot.

Experiment II

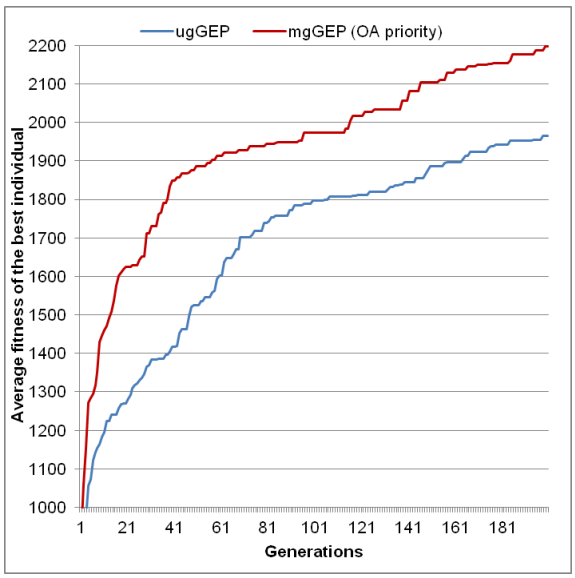
The second experiment employed the sub-division model shown by Figure 4.19. In this context, the robot's energy was set as the priority concern for the robot; that is, the robot checked the energy before it decided which actions (in terms of genes) to execute. This model was referred as mgGEP with energy level as a priority concern: mgGEP (Energy priority).

With this approach, if the robot's energy fell within 0.5, gene 1 would continuously be executed. However, If the energy was above 0.5, then the robot would check its proximity to obstacles before deciding whether to execute gene 2 or gene 3. Results for this experiment as compared to the first experiment is shown by Figures 4.21 and 4.22.

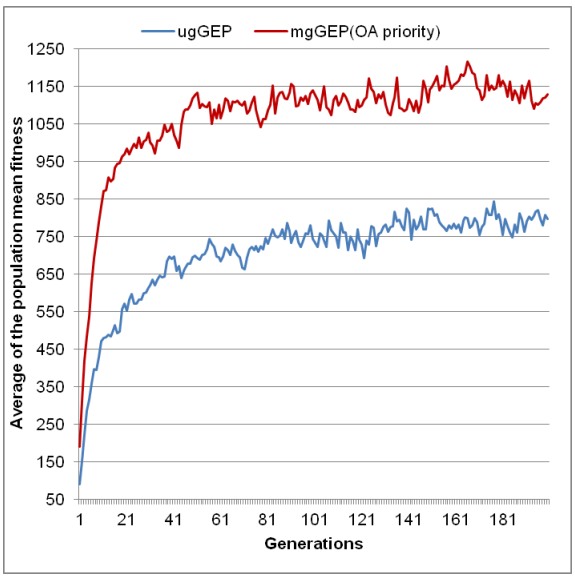
Experiment III

The third experiment, implemented the behaviour sub-division model shown by Figure 4.20. This model implemented a cascading order of priorities in order to choose a task to execute. The model was referred as mgGEP with cascaded priorities: mgGEP (Cascaded priorities).

This behaviour coordination model is similar to mgGEP (Energy priority); that is, the



Best individual performance



Population average performance

Figure 4.18: Progression of the mean of the best individual in the population, and the average of the population mean fitness over generations as achieved through ugGEP and mgGEP (OA priority). The average population mean fitness was computed from the population mean fitness achieved in each generation in all the 20 randomly seeded runs. Similarly, the mean of the best individual fitness was derived from the 20 randomly seeded runs. In each evolutionary run, a population of 500 organisms was evolved for 200 generations and the population mean and best individual fitness recorded in each generation. The rest of the parameters are as shown on Table 4.5.

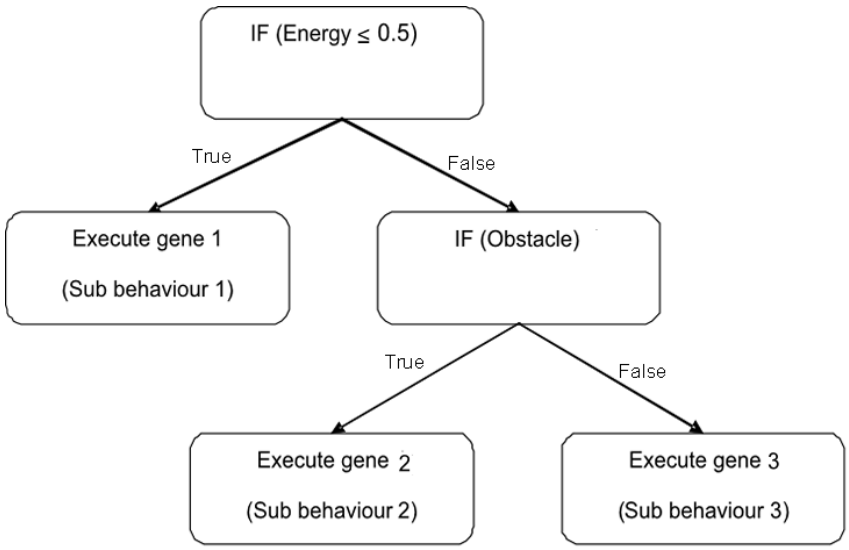


Figure 4.19: A behaviour coordination model that relies on robots energy level in order to select a robot task.

robot's energy was set as the priority concern for the robot. However, in mgGEP (Cascaded priorities) when the robot's energy was within 0.5, then the robot would check its proximity to obstacles before deciding whether to execute gene 1 or gene 2. Gene 3 was continuously executed if the robot's energy was above 0.5. The results achieved is as shown by figure

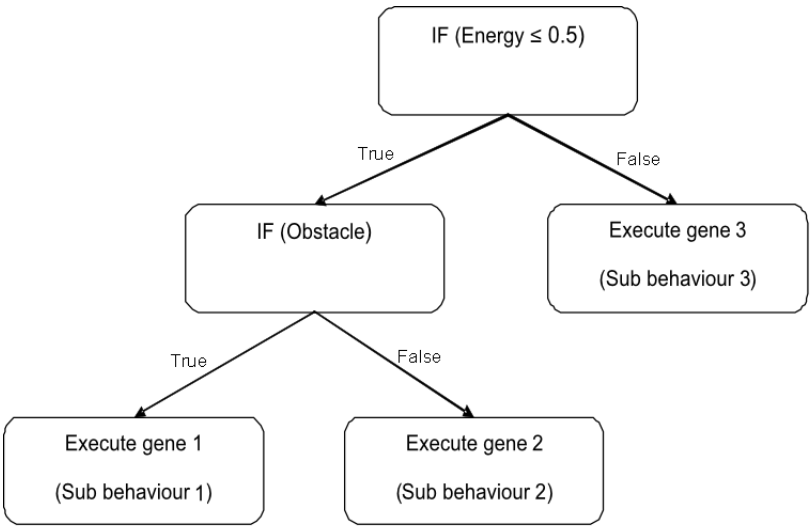


Figure 4.20: A behaviour coordination model that uses robots energy to determine behaviour selection. In this case sub-behaviour selection is determined via cascading priorities.

4.21 and 4.22.

4.5.3 Discussion

Similar to the results obtained in sections 4.4 and 4.3, Figure 4.18 shows that the mgGEP mechanism outperformed the ugGEP algorithm in solving the foraging behaviour problem. The performance of the mgGEP algorithm is likely to have resulted from the use of behaviour sub-division in solving the problem. The evolved modular controller has the potential to evolve various techniques to solve a problem. For instance, when the robot is close to an obstacle, the obstacle avoidance sub-controller (gene 1) can either turn the robot right, left or to any of the other radians as shown in Table 3.1. Similarly gene 2 and gene 3 have numerous ways to steer the robot. This evolution of specialised modules is likely to lead to a more robust controller than general controllers evolved using ugGEP. Additionally, any negative effects that might be incurred when root transposition, insertion sequence transposition as well as mutation are executed are going to have less impact in the mgGEP than they would have in ugGEP. For instance, in the case of the 3 genes chromosome above, there is only a $1/3 \times 0.1$ probability of root or insertion sequence transposition occurring in the genome, while the probability is 0.1 in the ugGEP.

Figures 4.21 and 4.22 show that the progression of the best individual in the population with the number of generations, is much quicker in mgGEP (OA priority) than it is when the robots energy level determines behaviour coordination (that is mgGEP(Energy priority) and mgGEP(Cascaded priorities)). This high performance by mgGEP (OA priority) can be attributed to a good behaviour coordination mechanism that utilised all the genes in the chromosomes fully to solve the problem. In the implementation reported here, obstacle avoidance is an important behaviour as the robot incurs a high penalty 25 for

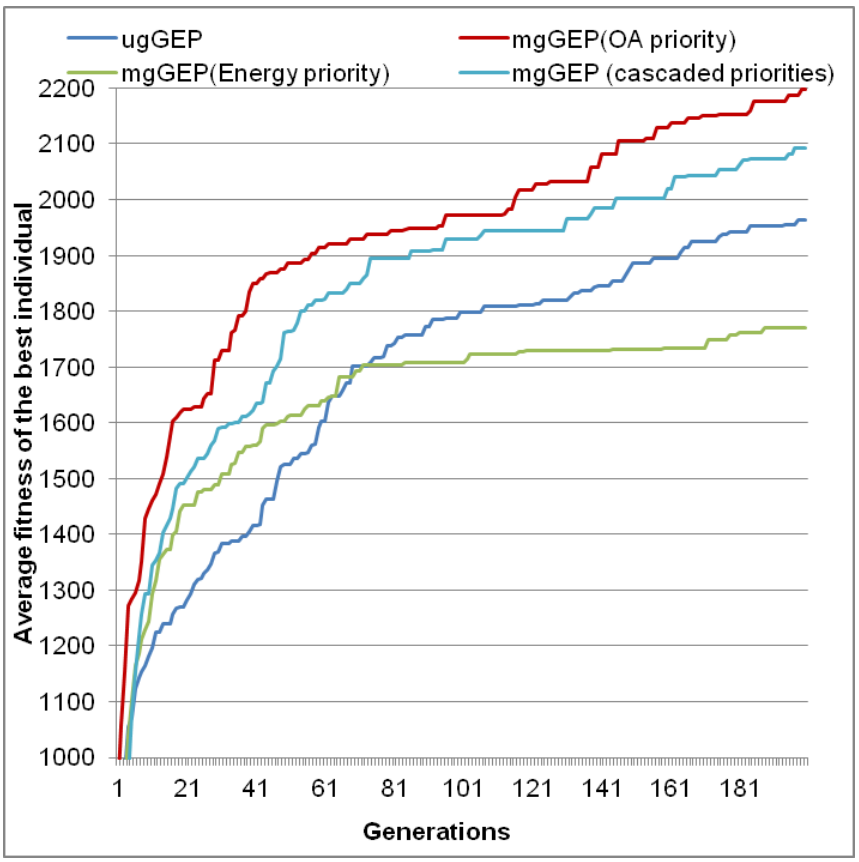


Figure 4.21: Progression of the mean fitness of the best individual in the population over generations as achieved using ugGEP algorithm as well as mgGEP with different behaviour coordination mechanisms. The average fitness has been computed from the best individual fitness achieved in each generation in all the 20 randomly seeded algorithms. In each evolutionary run, a population of 100 organisms was evolved for 200 generations and the best individual fitness recorded in each generation. The rest of the parameters are as shown on Table 4.5.

hitting an obstacle and the controller execution is terminated. In mgGEP (OA priority), obstacle avoidance is set as the primitive behaviour; this means that the first gene (gene 1) is specialized for obstacle avoidance tasks only. In absence of obstacles the robot then checks the energy before making a decision whether to look for food sources (gene 2) or to continue maximizing its exploration in the environment (gene 3). This behaviour arrangement is likely to lead to a robust controller with specialised modules, hence the high performance.

Results shown by Figure 4.21 shows that mgGEP (Energy priority) performed the worst in this task. The low performance can be attributed to various factors. Firstly, in mgGEP (Energy priority), when the robot's energy falls below 0.5, the first gene is executed continuously; this means that this gene has to evolve mechanisms for exploration, obstacle avoidance and foraging in as much the same way the ugGEP does. This is disadvantageous to the whole controller as the functions or sub-behaviours of the other two genes have to be duplicated. Secondly, when the energy is below 0.5 the robot is expected to start looking for food sources in order to increase its energy level. However, the energy level continues to fall as the controller is not specialised for energy sourcing. Results shown by

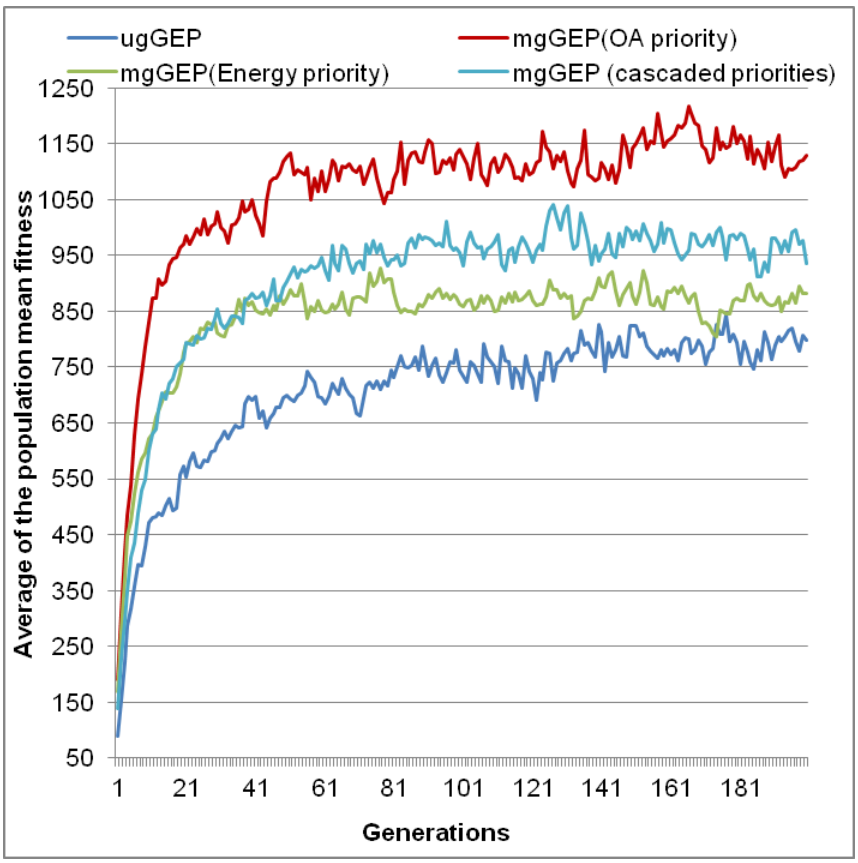


Figure 4.22: Progression of average performance of the population over generations as achieved using ugGEP algorithm as well as mgGEP with different behaviour coordination mechanisms. The average population mean fitness was computed from the population mean fitness achieved in each generation in all the 20 randomly seeded algorithms. The rest of the parameters are as shown on Table 4.5.

Figure 4.22 shows that mgGEP (Energy priority) was outperformed by the other mgGEP algorithm and only performed slightly better than ugGEP. The low performance in the overall populations are likely to be attributed to similar reasons as the low performance of the best individual (Figure 4.21). In comparison to the ugGEP, the results suggest that with an increased number of generations, the ugGEP is likely to perform as well as the mgGEP (Energy priority). In conclusion, the low performance of the mgGEP (Energy priority) is caused by the use of a behaviour coordination strategy that does not utilise all the three genes fully.

The mgGEP (Cascaded priorities) outperforms both ugGEP and mgGEP (Energy priority), however its outperformed by mgGEP (OA priority). The performance for this behaviour coordination mechanism can be explained using similar observations as reported above. In the mgGEP (Cascaded priorities) coordination model, the action selection mechanism uses the robot's energy level to determine task selection. If energy is below 0.5 and the robot's proximity to obstacles is below 0.5 then gene 1 is executed. This means that a vital behaviour such as obstacle avoidance is only executed when the energy is low, this means that gene 1 does not solve the obstacle avoidance problem completely. Nevertheless, gene 2 is likely to specialise to a foraging behaviour. Gene 3, on the other hand

is required to evolve the dual tasks of obstacle avoidance and exploration. Since gene 3 is executed when the energy is higher than 0.5, this gene is likely to evolve the dual sub-behaviour capabilities as they are closely related. The mechanisms utilised by this coordination model is thus more specialised that mgGEP (Energy priority) and hence as the results show, it outperforms it in this task. However, in comparison to mgGEP (OA priority) this mechanism is more generalised.

As shown by the results and the above discussion, the difference in performance of these behaviour coordination mechanism is affected by the action selection mechanisms used. This shows that the position of a sub-controller in an overall modular control architecture is of great importance to the overall performance of a modular controller.

4.6 Conclusion

In this chapter the performance of GEP in evolving mechanisms to sub-divide and coordinate sub-behaviours was investigated. Two behaviours, wall following and foraging, were implemented and results discussed. The obtained results show that unigenic GEP is able to evolve the two behaviours. Also, it was shown that ugGEP performs better in the wall following behaviour than standard GP as presented by Lazarus and Hu [78]. In addition to this, the mgGEP algorithm was used to solve the wall following problem. Multiple genes evolved by GEP performed better than both GP and ugGEP. A further analysis to the controllers evolved using the mgGEP, reveals that one gene was evolved to solve the obstacle avoidance problem while the other gene evolved straight line navigation with proximity to the walls (wall following). Similarly, the mgGEP algorithm outperformed ugGEP algorithm in the food foraging problem. The conclusions made from both experiments shows that mgGEP chromosomes have an advantage over the single chromosome used in ugGEP. The mgGEP is shown to evolve specialised modules while ugGEP evolves a more general purpose controller. The capability of GEP to code the chromosome into multiple genes shows that mgGEP can be used to evolve modularity in evolutionary robotic problems. The technique evolved is one of manually dividing the problem and then allowing each gene to solve specific tasks. The overall behaviour then emerges due to the interaction of the two genomes and the environment. This can be extended to various mathematical and robotic problems with promising results.

An important observation made in behaviour sub-division is that the behaviour coordination and organising mechanisms utilised, can affect the ability of a modulator controller to solve a problem successfully. Results obtained through the foraging behaviour showed that some action selection mechanisms were more successful than others. As reported in Chapter 2, action selection or behaviour coordination is an important issue both in animal behaviour and in robotics. In the literature, the proposed behaviour coordination mechanisms are either competitive or arbitrative (please see [117, 142] for an overview). The manual mechanism implemented in this chapter using mgGEP is arbitrative in nature, i.e. dominant behaviour determines which gene effects motor control (for instance presence of an obstacle). This mechanism as described here (mgGEP OA priority), though successful,

requires the designer to specify exactly how the behaviour is sub-divided. The outcome is therefore a product of the designers intelligence. As shown in the conducted experiments, this process requires the designer to be very careful in designing the behaviour coordination mechanism as a particular error may affect the evolution process and might make it hard for the algorithm to evolve the targeted behaviour. The questions that remain are firstly whether there exists a mechanism to coordinate behaviours for an optimal effect and secondly whether there exists any guidelines to be followed by a designer when evolving a particular behaviour in order to produce optimal controllers.

Techniques that can be used to generate behaviour sub-division and coordination automatically, are likely to reduce the need for human expertise in behaviour sub-division and coordination. Moreover, such techniques are likely to lead to development of more robust controllers than those developed using human expertise; the human developer may not always know all the various attributes in unstructured environments. Additionally, this may lead to substantial reduction in the development time: Architectures such as subsumption architecture and layered learning require one behaviour to be developed and tested before other behaviours can be implemented, this leads to an increase in development time. Since EAs encode a population of controllers, given the right parameters, they have shown great capabilities in evolving robot controllers (see [105, 45, 94] as well as results presented Chapter 3 and this chapter). However, for successful evolution of behaviour sub-division, an evolutionary algorithm that can be divided into modules is required. As shown by the experimentation presented in this chapter, mgGEP algorithm provides the capability to successfully formulate behaviour sub-division, unlike GAs and GP. The next chapter explores the potential for this behaviour sub-division and action selection mechanism to be evolved automatically using the mgGEP approach.

5 Evolving Modularity in Robot Behaviour

In the previous chapter, using multigenic GEP (mgGEP) in robot behaviour sub-division, it was shown that a unigenic GEP (ugGEP) outperformed genetic programming (GP) in solving a robot wall following problem. It was further shown that the mgGEP approach outperformed ugGEP. The approach used by the mgGEP was that of manual behaviour sub-division; therefore, the success achieved by mgGEP was partially due to human expertise in creating modularity. Analysing the evolved chromosomes, it was observed that mgGEP divided the problem at hand into various subtasks; essentially evolving a subsumption like architecture. Using a robot foraging behaviour, it was observed that behaviour coordination mechanisms can affect the ability of a modular controller to solve a problem successful. This finding is in line with most research in this area (see [18, 4, 117, 92, 92, 142]). As a result, there is a need to develop mechanisms to automatically formulate behaviour sub-division as well as behaviour coordination. As discussed in section 4.6, automatic behaviour sub-division and coordination would likely reduce the need for human expertise, lower development time as well as lead to more robust robot controllers.

This chapter is aimed at introducing new evolutionary techniques that can be utilised to automatically evolve behaviour modularity using variants of the mgGEP algorithm. Two main ideas are introduced. Firstly, the mgGEP algorithm is extended by introducing an additional set of alphabets; that is, in addition to the function and terminal set, a new set made up of linking functions is introduced. In this technique, the linking function (linker) is evolved with the rest of the mgGEP genome. This removes the need for the human designer to specify the exact linking function to use in a particular mgGEP chromosome. This new technique is referred as multigenic GEP with linker evolution (mgGEP-LE). Secondly, the mgGEP idea is extended by evolving a regulatory gene as part of the GEP chromosome, the regulatory gene, just as in systems biology, determines which of the genes in the chromosome to express and therefore how the controller solves the problem. This new technique is referred as multigenic GEP with regulatory gene (mgGEP-RG). These proposed algorithms are implemented for the wall following problem, discussed in the previous chapter, and their results compared to that of ugGEP and mgGEP.

The rest of the chapter is divided as follows; firstly, the mgGEP-LE algorithm is introduced and implemented for a wall following problem using the 2D environment. Secondly, the mgGEP-RG is introduced and implemented for the wall following problem using the 2D environment. Thirdly, the results of these new algorithms are compared to that of ugGEP and mgGEP and a discussion follows. Fourthly, the two techniques are implemented

in a robot following problem in a 3D world model and their results compared to that of mgGEP. Finally a conclusion on the suitability of these methods in development of modular controllers is drawn.

5.1 Multigenic GEP with Linker Evolution

The standard mgGEP algorithm reported in Chapter 4, evolves multiple genes linked together using a specialised function called a linker [38, 42, 95]. In standard mgGEP, the designer selects the linking function before the evolutionary run. This linking function is then used by all chromosomes in a population. As described previously, the function of the linker is important for the overall performance of the modular controller since it regulates which genes are expressed under which circumstances. This linking function is thus supplied with a condition on which decisions to express or inhibit a gene are contained. For instance, in the wall following problem described in Chapter 4, the condition supplied for the IF linker was the output of the front sensor. As the obtained results show, this algorithm works better than the monolithic ugGEP algorithm. Since the linking function and conditions can affect the ability of a controller to solve a problem successfully, the designer has to choose carefully what linking function and condition to implement for a given problem. However, the designer does not always know how the evolved controllers will try to solve the problem and the various scenarios the robot may encounter. Thus, it is possible that there could be various other linking functions that may be suited to the problem and that may improve the overall performance of the algorithm. This means that providing a set of linking functions to the algorithms may potentially lead to a better performance. Additionally, this removes the pressure of choosing a specific linking function from the designer; the designer can select suitable linking functions and provide these to the algorithm for the evolutionary process to generate the best controllers with the best linking function.

The proposed multigenic GEP with Linker Evolution (mgGEP-LE) technique is similar in implementation to the standard mgGEP. However, instead of supplying the algorithm with one pre-determined linking function, the mgGEP-LE is supplied with a **Linking set** comprising all the linking function that the designer thinks can be used to regulate the genes. All the linking functions have a pre-set condition that allow the particular linker to regulate activation or repression of gene activity. Therefore, different chromosomes in a population use different linking functions for gene regulation. To generate the population of organisms using the mgGEP-LE algorithm, three alphabetic sets are provided: **Function set**, **Terminal set** and a **Linking set**. The linking function in a particular chromosome is subjected to evolution operations along with the rest of the genome: Due to the nature of the GEP algorithm, the implementation reported here allows the linker to only undergo mutation and one or two point recombination as the more complex operators are not required.

5.1.1 Motivation

The motivation behind this new algorithm is to reduce human influence in the overall evolutionary process. Additionally, the algorithm is aimed at enabling the automatic development of modular controllers through the evolutionary process.

By focussing on creating a set of linking functions, the algorithm reduces the overall human influence as the controllers evolved are not based on a particular linking function. The human input is thus limited to selecting candidate linking functions, similar to the formulation of candidate terminals and functions. Additionally, since the evolved chromosomes are not based on a particular linker then the technique can be said to automatically evolve modularity with reduced human input.

5.1.2 Experimental set up

In the experiment reported in this section, the mgGEP-LE algorithm was implemented to evolve controllers for a wall following robot as described in section 4.3. The results of the experiments were compared to the standard mgGEP performance. The algorithm was implemented using the parameters, function and terminal set as described in section 4.3 and Table 5.1.

Table 5.1: mgGEP-LE: Algorithm parameter settings

Parameters	mgGEP	mgGEP-LE
Maximum generations	200	200
Population	500	500
No. of Genes	2	2
Head size	8	8
Parent organisms	2	2
Mutation probability	0.041	0.041
1-Point Recombination probability	0.7	0.7
2-Point Recombination probability	0.2	0.2
Gene Recombination probability	0.1	0.1
IS Transposition probability	0.1	0.1
RIS Transposition probability	0.1	0.1
Gene Transposition probability	0.1	0.1
Selection range	5%	5%
Function set(If, And, Or, Not)	4	4
Linking set(If, And, Or)	–	3
Terminal set(8 sensors, 4 motors)	12	12
No. of start points	10	10
No. of randomly seeded runs	50	50

The linking function set used in the mgGEP-LE algorithm comprised of AND, OR and IF functions. The mode of operation for the linking functions is defined as follows:

- **IF linker :** this linker was implemented as per the standard mgGEP scenario in the previous chapter; that is, If front sensor returns 1 (close to obstacle) then express gene 1 else express gene 2.
- **OR linker :** this linker checks the results of the expression of the first gene; if the result is a movement terminal or if there is an obstacle (i.e. if expression of the first gene returns a 1), then the first gene effects motor control else the second gene is expressed.
- **AND linker:** this linker checks the results of the first gene and executes the first gene if the results are a movement terminal or if there is no obstacle (i.e. if expression of the first gene returns a 0), else second gene is expressed.

Similar to the mgGEP experiment described in section 4.3, 50 randomly seeded algorithm runs were carried out. The population was made up of 500 organisms and each run lasted for 200 generations. The performance of the evolved controllers were evaluated using Equation 4.1. Table 5.1 shows a summary of all the parameters utilised in the experiment.

5.1.3 Results and Discussion

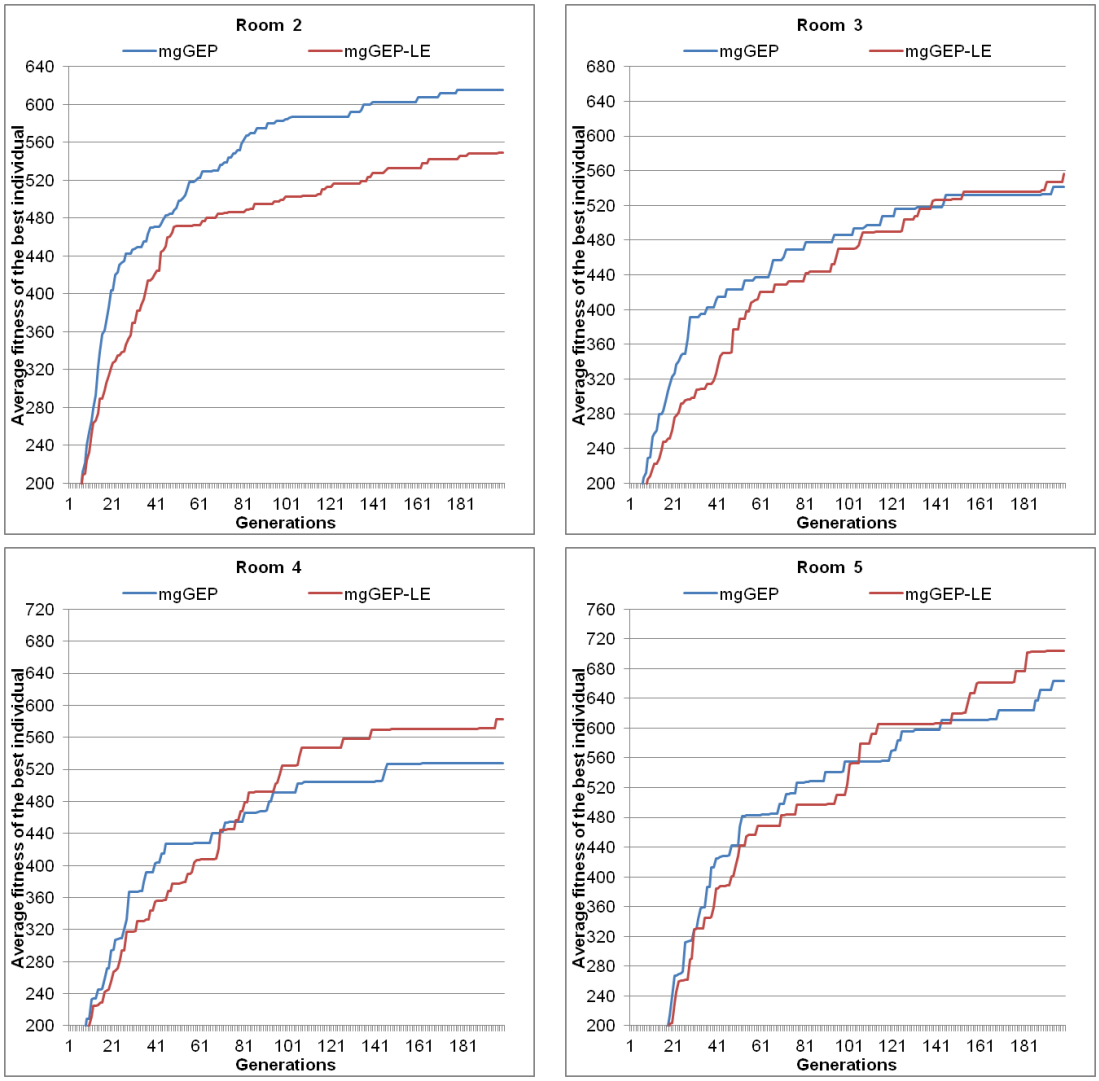


Figure 5.1: Average fitness of the best individuals along 200 generations for the population of mgGEP-LE and mgGEP chromosomes. The mean fitness has been computed from the best individual fitness achieved in each generation in all the 50 randomly seeded algorithms. In each evolutionary run, a population of 500 organisms was evolved for 200 generations and the best individual fitness recorded in each generation. The rest of the parameters are as shown on Table 5.1.

Figure 5.1 shows that the mean fitness of the best individual evolved using mgGEP-LE algorithm was higher than mgGEP in rooms 3, 4 and 5. However the mean fitness of the best mgGEP-LE individual was outperformed by mgGEP in room 2. The mean fitness of the population mean fitness as shown by Figure 5.2, shows that chromosomes in both algorithms are increasing in fitness over the generations. In general, the results suggest that the controllers evolved using mgGEP-LE performed better in this problem domain than the mgGEP. Since both chromosomes have the same structure and are of equal lengths, the improved performance shown by mgGEP-LE could be attributed to an increase in diversity due to the addition of evolvable linking function.

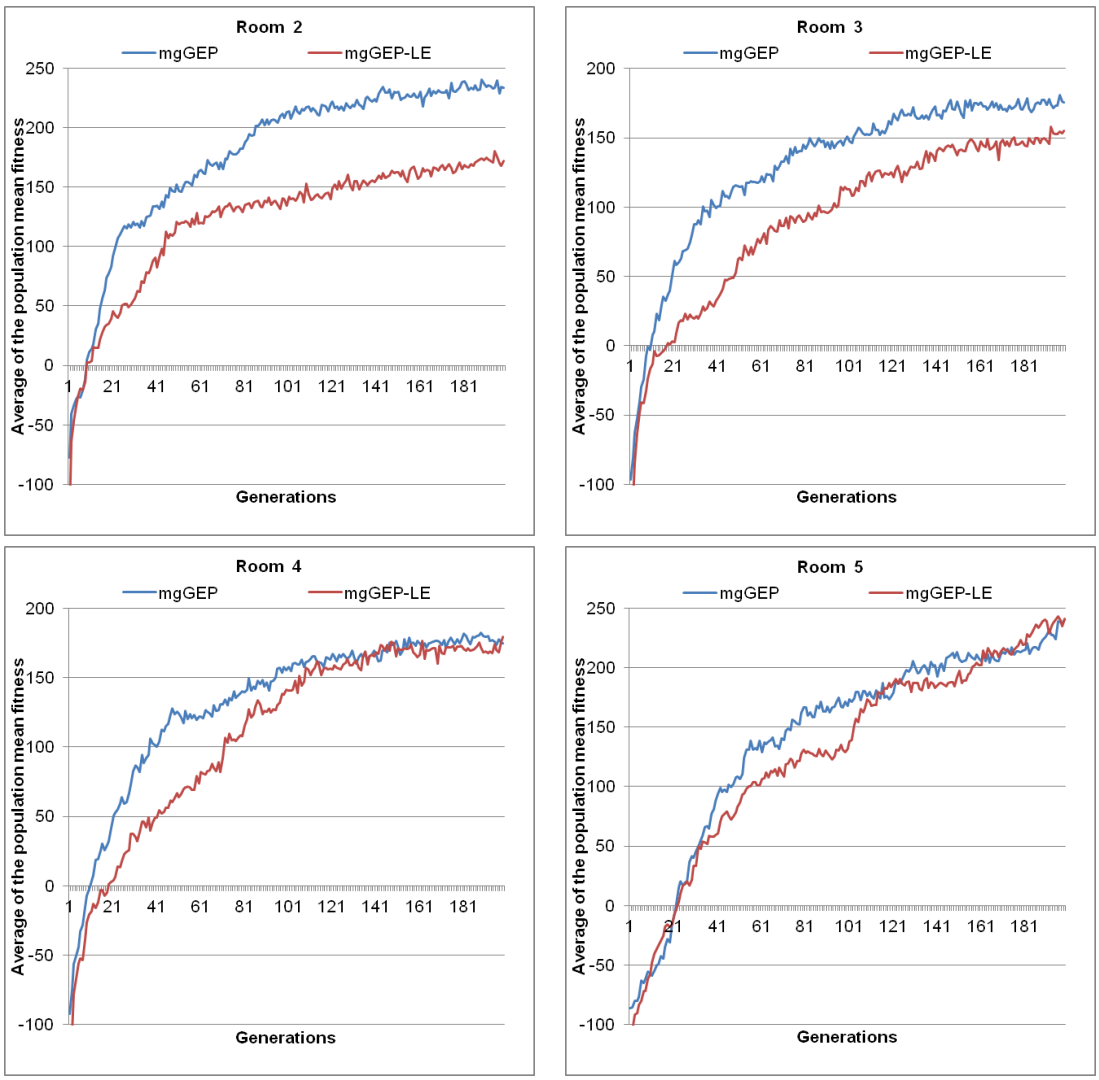


Figure 5.2: Mean of the population mean fitness as achieved through mgGEP and mgGEP-LE in the different room types. The mean of the population mean fitness was computed from the population mean fitness achieved in each generation in all the 50 randomly seeded algorithms. In each evolutionary run, a population of 500 organisms was evolved for 200 generations and the population mean fitness recorded in each generation. The rest of the parameters are as shown on Table 5.1.

As mentioned in section 5.1.2 above, the IF linker works similarly to the standard mgGEP, however, the OR and AND linking functions cause the first gene to be used as a regulatory gene. The first gene is thus used to evolve the condition for gene coordination. In the mgGEP-LE described here, the use of the first gene for regulation may not be very beneficial as the first gene functions as both regulatory and structural gene, thus it is not specialised to a particular task. This lack of specialisation may lead to low performance and could potentially explain the performance of best individual in rooms 2 as well as the average performance of the population in rooms 2 and 3 (see Figure 5.2). Nevertheless, the fitness of the best individual in rooms 3, 4 and 5 as well as average fitness of the population in rooms 4 and 5, shows that mgGEP-LE is able to approximate the performance of the standard mgGEP. This original and novel technique is not only easy to implement but can be used in various applications where linking or action selection mechanisms are

not always known.

5.2 Multigenic GEP with a regulatory gene

The average best individual performance obtained using mgGEP-LE (Figure 5.1), showed only a slight increase in the performance of the GEP. Consequently, a new technique was implemented with the main aim of investigating whether results achieved above could be improved further.

As in natural biology, a mgGEP chromosome can be compared to the interaction within a cell; the extent to which a gene is expressed (converted into protein) is dependent on many factors within the cell including the activity of other genes within the cell. Therefore the cell is able to switch the transcription of a specific gene or a group of genes on or off. Additionally, the cell is able to identify environmental conditions in which the activation or repression of transcription of a particular gene or a group of genes will be required. A cell is able to do this by using one or a group of genes in a multigenic set to regulate the activation or repression of expression of the other genes [57]. Also, in natural biology, the nature of organisms means that a regulatory gene undergoes evolution just like the rest of the chromosome [42]. In standard mgGEP, as described in Section 2.3.5 and Chapter 4, gene regulation (activation and repression of expression of genes) is accomplished using a linking function (**Linker**).

The multigenic GEP with a regulatory gene (mgGEP-RG) is a new evolutionary technique, proposed here for the first time, that seeks to extend mgGEP algorithm by incorporating a self-regulatory mechanism. Similarly to mgGEP, the proposed technique uses multiple genes of equal lengths to form a chromosome and a pre-determined linking function to determine control. In mgGEP, the pre-determined linking function contains a pre-set decision making criteria that allows it to regulate the activation or repression of a gene in the given set of genes. However, in the proposed technique, an extra gene referred to as a regulatory gene influences how genes are expressed (selected for activation) by providing a condition to the linking function. Thus, similar to natural biology, the regulatory gene provides the conditions under which a particular gene or group of genes should be activated or repressed. The regulatory gene is therefore a part of the chromosome but it is not involved in the direct control of the robot actions. Figure 5.3 shows an example of the proposed model.

The regulatory gene is equal in length to the rest of the genes in the chromosome and is formed like any other GEP gene; with head and tail regions and using the provided set of functions and terminals. In the proposed implementation, the regulatory gene is placed as the first gene in the chromosome, though it could be placed in any position with the gene set. As part of the chromosome, the regulatory gene undergoes all genetic operations like the rest of the genes. Note that the regulatory gene can change during the course of evolution particularly if a gene transposition occurs.

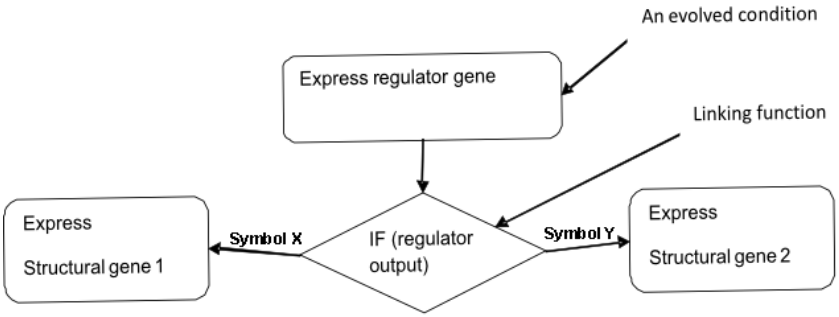


Figure 5.3: mgGEP-RG implementation model. The output of the regulatory gene is a symbol that is then used for decision making

5.2.1 Implementation

In this experiment, 3 genes each with a head size, $h = 8$, were used to form the mgGEP-RG chromosome. The genes are linked using an IF linker that uses the output of the first gene to decide which of the remaining two genes will be expressed to effect robot motor control. Thus the first gene acts as a regulatory gene. The rest of the genes which effectively control the robot actions are referred to as structural genes. In the implementation reported in this section, the first structural gene (second gene in the chromosome) was activated if the output of the regulatory gene was a motor action terminal while the second structural gene (third gene in the chromosome) was activated if the output of the regulatory gene was a sensor terminal. The algorithm is therefore symbol based. Algorithm 1 shows how the mgGEP-RG was implemented for this problem.

Algorithm 1 using mgGEP-RG in the wall following problem

Require: regulatory gene ⇒ gene1
Require: Structural genes ⇒ gene2, gene3
Require: Motor Terminals⇒ MF, ML, MB, MR
Require: Sensor Terminals⇒F, FR, FL, B, BR, BL, L, R
  determineController← Translate (gene1)
  {*The variable "determineController" is the string symbol output from the translation of the regulatory gene*}
  if determineController ⊂ Motor Terminals **then**
    motorEffectControl ← Execute (gene2)
    { *The variable "motorEffectControl" gets the output of the execution of a structural gene and convert it to a robotic action*}
  else
    motorEffectControl ← Execute (gene3)
  end if

5.2.2 Experimental set up

Table 5.2: mgGEP-RG: Algorithm parameter settings

Parameters	mgGEP	mgGEP-RG
Maximum generations	200	200
Population	500	500
No. of Genes	2	3
Head size	8	8
Parent organisms	2	2
Mutation probability	0.041	0.027
1-Point Recombination probability	0.7	0.7
2-Point Recombination probability	0.2	0.2
Gene Recombination probability	0.1	0.1
IS Transposition probability	0.1	0.1
RIS Transposition probability	0.1	0.1
Gene Transposition probability	0.1	0.1
Selection range	5%	5%
Function set(If, And, Or, Not)	4	4
Linking set(If, And, Or)	–	3
Terminal set(8 sensors, 4 motors)	12	12
No. of start points	10	10
No. of randomly seeded runs	50	50

Table 5.2 , shows the algorithm parameters, population size, number of generations in a run and the probabilities of each of the genetic operations used by the algorithm. As shown on the table, 3 genes were used in this experimentation. Also, similar to the experimentations in section 4.3, each controller in the population was tested from 10 starting points and the fitness evaluated using Equation 4.1. The maximum fitness for each room is as shown by Table 4.1. In all the experiments, 50 randomly seeded algorithm runs were conducted. An evolutionary run terminated when the total number of generations was reached.

5.2.3 Results and Discussion

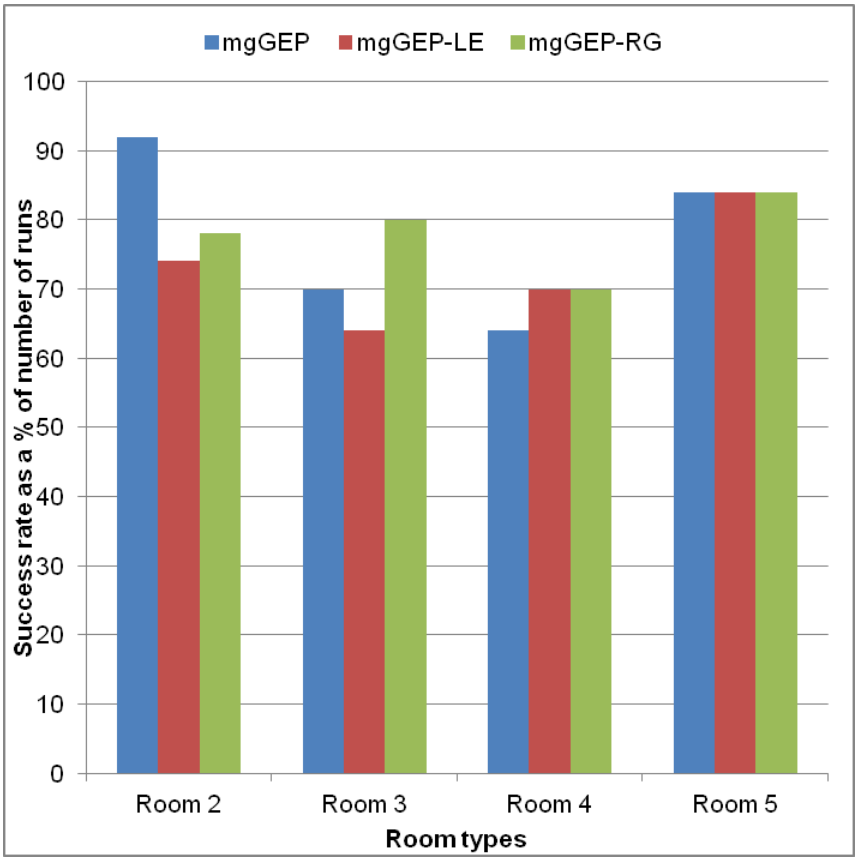


Figure 5.4: Comparison of success rates achieved in the four different room types using standard mgGEP, mgGEP-LE and mgGEP-RG algorithms. The success rate refers to the percentage of the total number of runs where the target fitness was achieved. Each evolutionary run had a population of 500 organisms and lasted for 200 generations. In all the experiments, 50 randomly seeded algorithm runs were conducted.

Figure 5.4 shows that standard mgGEP was more successful in room 2 than both mgGEP-LE and mgGEP-RG algorithms. In room 3, the mgGEP-RG had a better success rate than standard mgGEP while mgGEP-LE had the lowest success rate. In room 4, the mgGEP-RG had equal success rate with mgGEP-LE while standard mgGEP had the lowest success rate. All the three algorithms had equal success rates in room 5. Thus in general, these three algorithms appear to have solved the problem with almost equal success with mgGEP-RG appearing to be slightly better. This performance suggests that the new algorithms, mgGEP-RG and mgGEP-LE, evolve controllers that approximate the performance of mgGEP algorithm which, as previously mentioned, uses human expertise for sub-behaviour sub-division. Of importance is that the behaviour sub-division, the position of particular modules and behaviour selection mechanism is developed automatically in mgGEP-RG and mgGEP-LE. This offers better capability for the robot as well as less work for the designer.

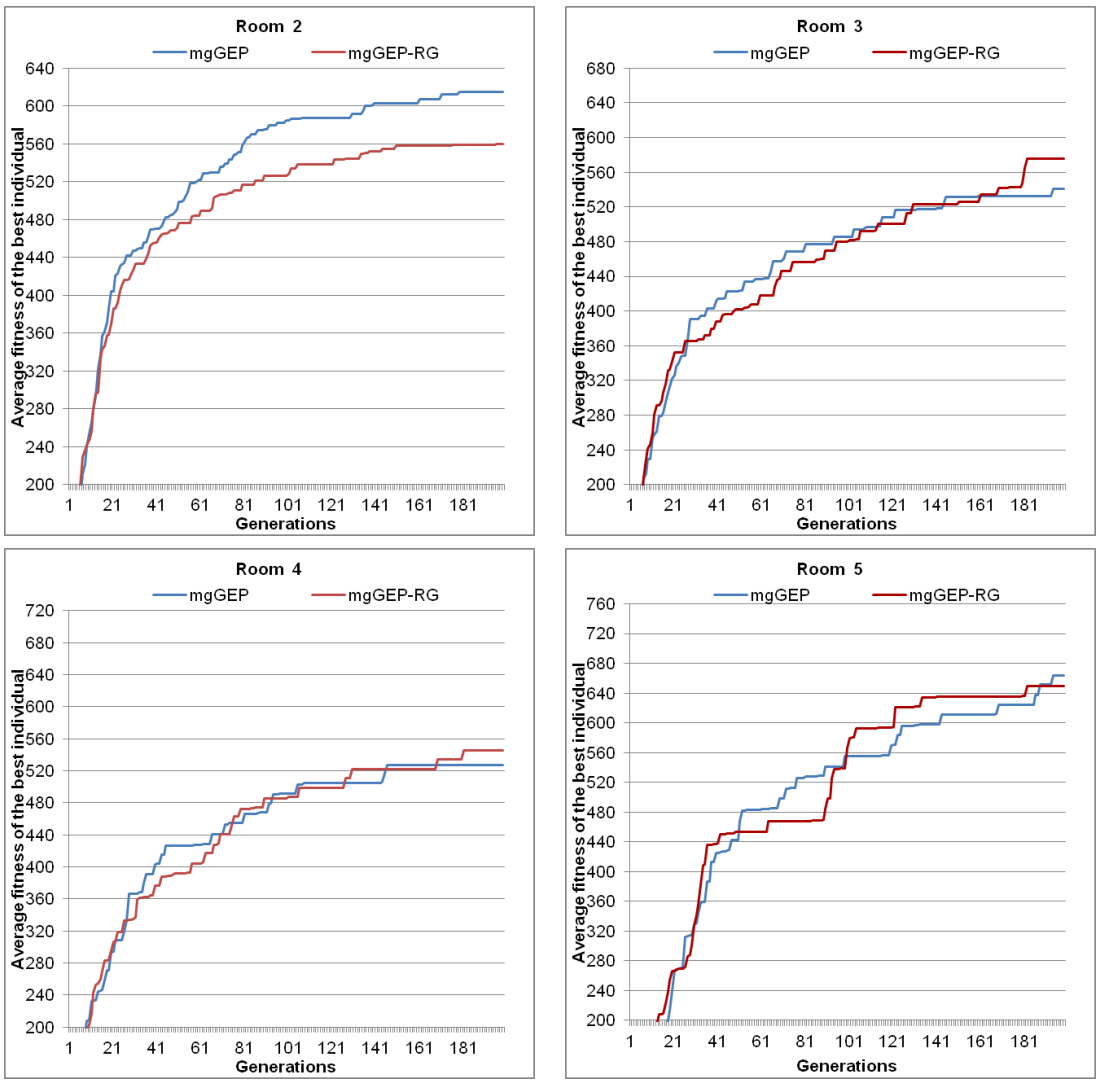


Figure 5.5: Comparison of best individual performances in the population as achieved through standard mgGEP and mgGEP-RG in the different room types. The mean fitness has been computed from the best individual fitness achieved in each generation in all the 50 randomly seeded algorithms. In each evolutionary run, a population of 500 organisms was evolved for 200 generations and the best individual fitness recorded in each generation. The rest of the parameters are as shown on Table 5.2.

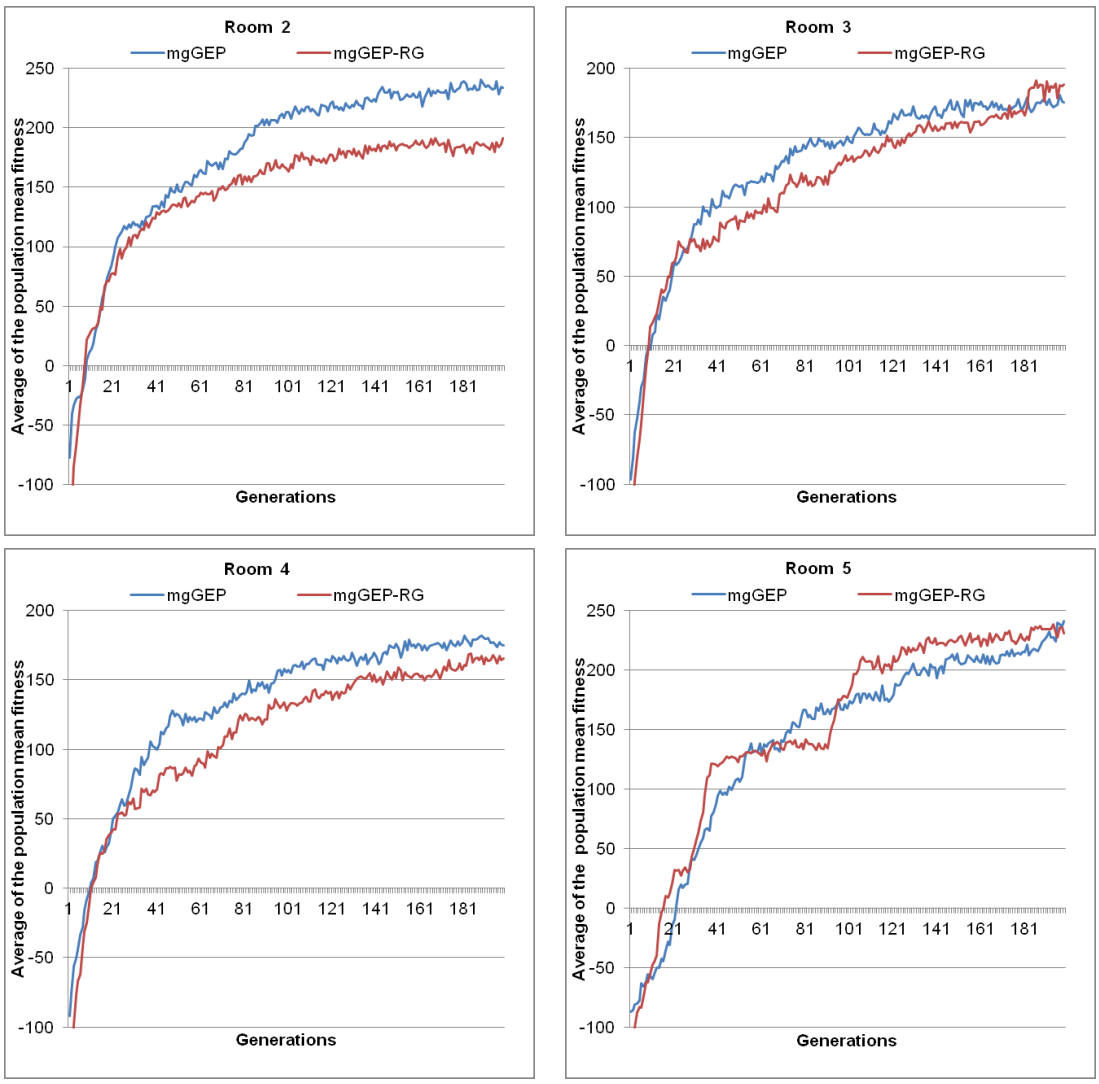


Figure 5.6: Comparison of progression of average fitness of the population as achieved through mgGEP and mgGEP-RG in the different room types. The average of the population mean fitness was computed from the population mean fitness achieved in each generation in all the 50 randomly seeded algorithms. In each evolutionary run, a population of 500 organisms was evolved for 200 generations and the population mean fitness recorded in each generation. The rest of the parameters are as shown on Table 5.2.

Figures 5.5, 5.6 shows the comparison of the performance between standard mgGEP and mgGEP with a regulatory gene (mgGEP-RG). The results shown by figure 5.5 suggests that the performance of mgGEP-RG was slightly better than standard mgGEP in rooms 3 and 4, slightly lower in room 5 and was outperformed in room 2. There could be various contributing factor to the slight success in rooms 3 and 4: Firstly, all the three genes in the chromosome have a head size h=8, this means that the mgGEP is only 2/3rds as long as the mgGEP-RG, the increase in the genome size, increases the search space and diversity. This increase is good for the search and could lead to the somewhat better performance as shown above. However, the increase in the search space could also lead to an increase in the computing time. Since the mgGEP-RG is longer in length than the mgGEP the number of generations used in the experiment may not be sufficient to enable the mgGEP-RG to converge, this could potentially explain the mgGEP-RG performance

in rooms 2 and 5. Additionally, in all the rooms the fitness of the best individual does not seem to converge near the optimal fitness. This means that both techniques require longer generation runs in order to solve the problem fully.

The IF linker used in the standard mgGEP, decides which gene to express based on the value of the front sensor. Since the robot is always facing forward, the mgGEP chromosomes learns easily to use the first gene for obstacle avoidance and use the second gene for the wall following. Therefore, as suggested by the results, the search is faster and yields good performance. In essence the standard mgGEP uses problem specific human expertise to determine the behaviour sub-division. The mgGEP-RG on the other hand is required to evolve the behaviour sub-division mechanism during the evolutionary run. As previously described, the mgGEP-RG has to express the first gene and use its output to determine which of the two genes to express. The output of the first gene could be either a movement terminal which lead to the third gene being expressed or it could be a sensor terminal leading to the expression of the second gene. Thus, the mgGEP-RG algorithm has the added task of learning which of the eight sensors is more significant to the problem. Therefore, the algorithm is actually trying to solve three tasks, one of finding a sensor that contribute effectively towards the solution of the general problem, i.e. wall following, and then subdividing the actual problems to the task of obstacle avoidance and exploration with wall proximity.

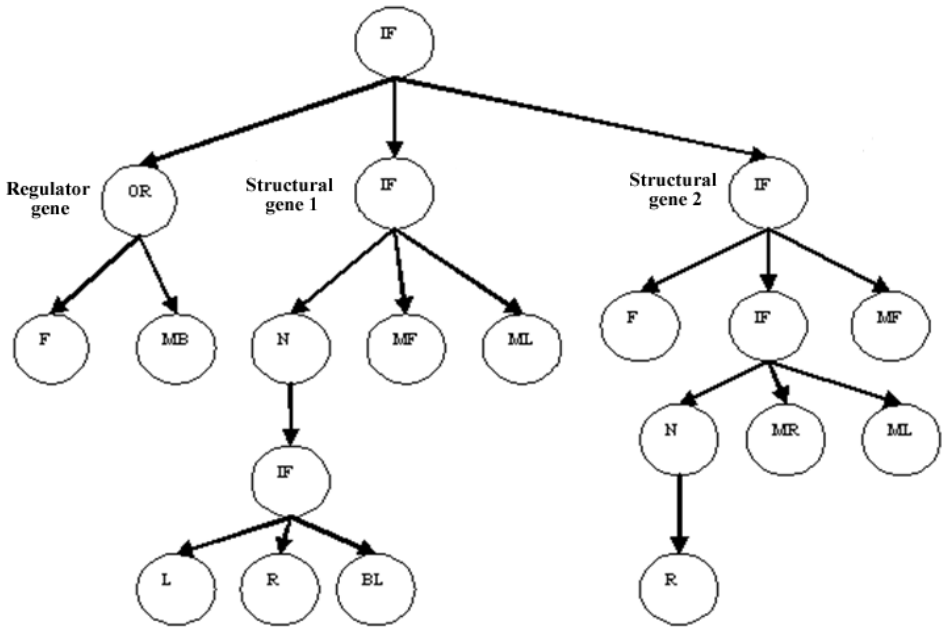


Figure 5.7: an example of a controller evolved using mgGEP-RG

Figure 5.7 shows an example of a controller evolved using mgGEP-RG. Although the shown controller was evolved using room 2 (figure 4.3), it adapts successfully in all the other rooms. Across the entire range of controllers evolved successfully, it was observed that the first gene (regulatory gene) evolved a structure that selects between a sensor and a movement action. The genome above shows that the regulatory gene selected the front sensor and the move back action. The expression of regulatory gene output a symbol

which is used by the linking function to determine which of the two structural genes to effect motor control. Structural gene 2 effects robot motor control when the output of the regulatory gene is a sensor, while structural gene 1 is executed when the output of the regulatory gene is a move action, **MB**, which is only possible if the front sensor outputs a 1 (i.e. obstacle ahead). When structural gene 2 is expressed, it checks the right sensor and then the robot can either turn right or left, thus avoiding the obstacle ahead. Note that the MF terminal (in structural gene 2) will never be executed as the regulatory gene will only allow structural gene 2 to be expressed when there is an obstacle ahead. Structural gene 1 is expressed when the regulatory gene outputs the MB terminal, again it can be observed that structural gene 1 guides the robot to move forward, it also checks the sensors on both the left and right directions, this is critical in guiding the robot on the extrusions as the front sensor cannot be used for this and keeps the robot effectively near the walls. Thus the evolved chromosome not only evolves the best sensor to use in obstacle avoidance (e.g. checking the existence of an obstacle ahead and modifying behaviour accordingly) but also the obstacle avoidance behaviour and straight-line motion with wall proximity.

In this approach, only one fitness function has been used (Equation 4.1). The fitness function encourages exploration near the wall as well as discouraging wandering and collision. The fitness function is utilised by the whole organism and called once, this lowers computation time and awards the robot for the dual behaviours of obstacle avoidance and exploration with wall proximity. This is unlike incremental evolution and layered learning approaches where unique fitness evaluation is required for behaviours in the set. Also, no additional time is taken to evolve separate behaviour or a layer. Since the major concern is the overall behaviour, the specification for lower behaviours is not taken into consideration; this means there is a global outlook in providing attributes to the desired behaviour.

5.3 Algorithm performance analysis

Results presented in figures 5.1 and 5.2 suggest that the mgGEP-LE algorithm performed slightly better than the mgGEP algorithm. Additionally, the results shown in figures 5.5 and 5.6 suggest that the mgGEP-RG algorithm performs equally as well as the mgGEP. These results suggest that these mgGEP variants are capable of evolving mechanisms to change the way a genome is expressed to solve a problem.

In this section an analysis is carried out to determine whether the reported results can be improved further by extending the generational run. The performances of these algorithms is then compared to that of standard mgGEP and ugGEP. Additionally, a test of statiscal significance is carried using the Mann-Whitney U test and the level of significance between the results obtained by different algorithms compared. Similarly, a solution convergence test is carried out to determine how fast these algorithms converge to the target fitness. Further, a description of controllers evolved using the mgGEP, mgGEP-LE and mgGEP-RG is presented. Finally, the effect of mutation operation on the performances of the mgGEP-LE and mgGEP-RG is analysed and compared to that of the mgGEP and ugGEP.

A discussion of the suitability of these algorithms in ER is then presented.

5.3.1 Performance comparison

The results presented in the previous sections suggest that mgGEP-RG and mgGEP-LE may have required longer generation runs in order to solve the wall following problem fully. Subsequently, the experiments reported in the previous sections were repeated using similar settings and parameters but with longer generation runs, in this case 500 generations were used. A summary of the parameters used in this experiments is as shown on Table 5.3 below.

Table 5.3: Algorithm parameter settings

Parameters	ugGEP	mgGEP	mgGEP-LE	mgGEP-RG
Maximum generations	500	500	500	500
Population	500	500	500	500
No. of Genes	1	2	2	3
Head size	16	8	8	8
Parent organisms	2	2	2	2
Mutation probability	0.041	0.041	0.041	0.027
1-Point Recombination probability	0.7	0.7	0.7	0.7
2-Point Recombination probability	0.2	0.2	0.2	0.2
Gene Recombination probability	0.0	0.1	0.1	0.1
IS Transposition probability	0.1	0.1	0.1	0.1
RIS Transposition probability	0.1	0.1	0.1	0.1
Gene Transposition probability	0.0	0.1	0.1	0.1
Selection range	5%	5%	5%	5%
Function set(If, And, Or, Not)	4	4	4	4
Linking set(If, And, Or)	–	–	3	–
Terminal set(8 sensors, 4 motors)	12	12	12	12
No. of starting points	10	10	10	10
No. of randomly seeded runs	50	50	50	50

The mgGEP-RG and mgGEP-LE experimental results were compared to the performances of ugGEP and mgGEP algorithms under similar settings. The results achieved in these experimentations were used to analyse the performance of the algorithms in solving the wall following problem.

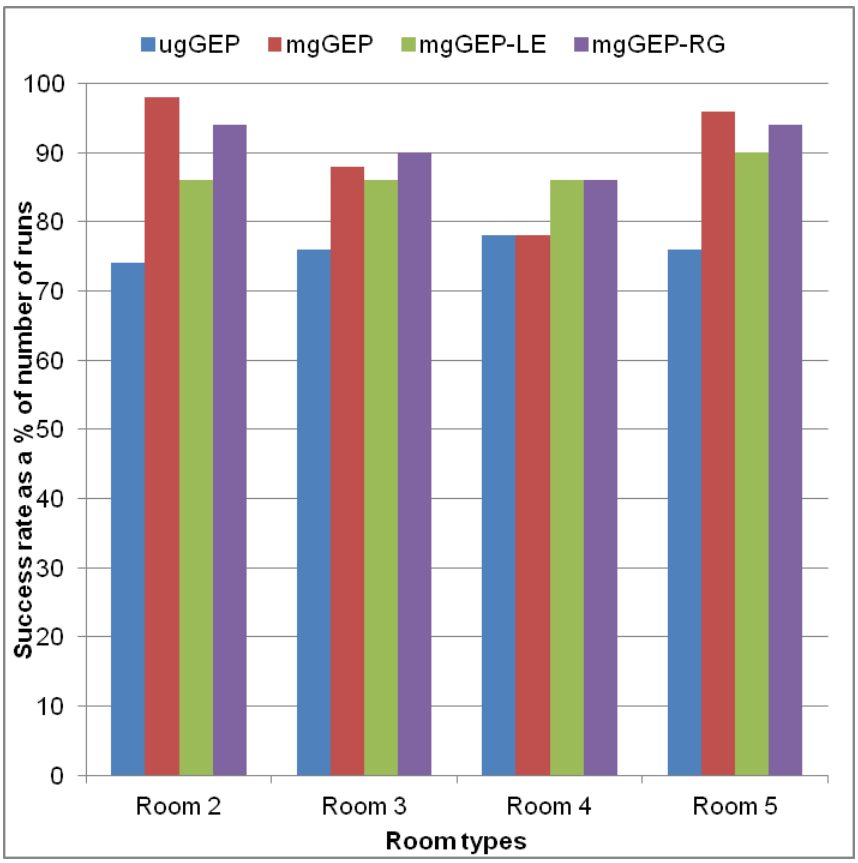


Figure 5.8: Comparison of success rates achieved using ugGEP, standard mgGEP, mgGEP-LE and mgGEP-RG algorithms. The success rate refers to the percentage of the total number of runs where the target fitness was achieved. Each evolutionary run had a population of 500 organisms and lasted for 500 generations. In all the experiments, 50 randomly seeded algorithm runs were conducted. The rest of the algorithm parameters are as shown on Table 5.3.

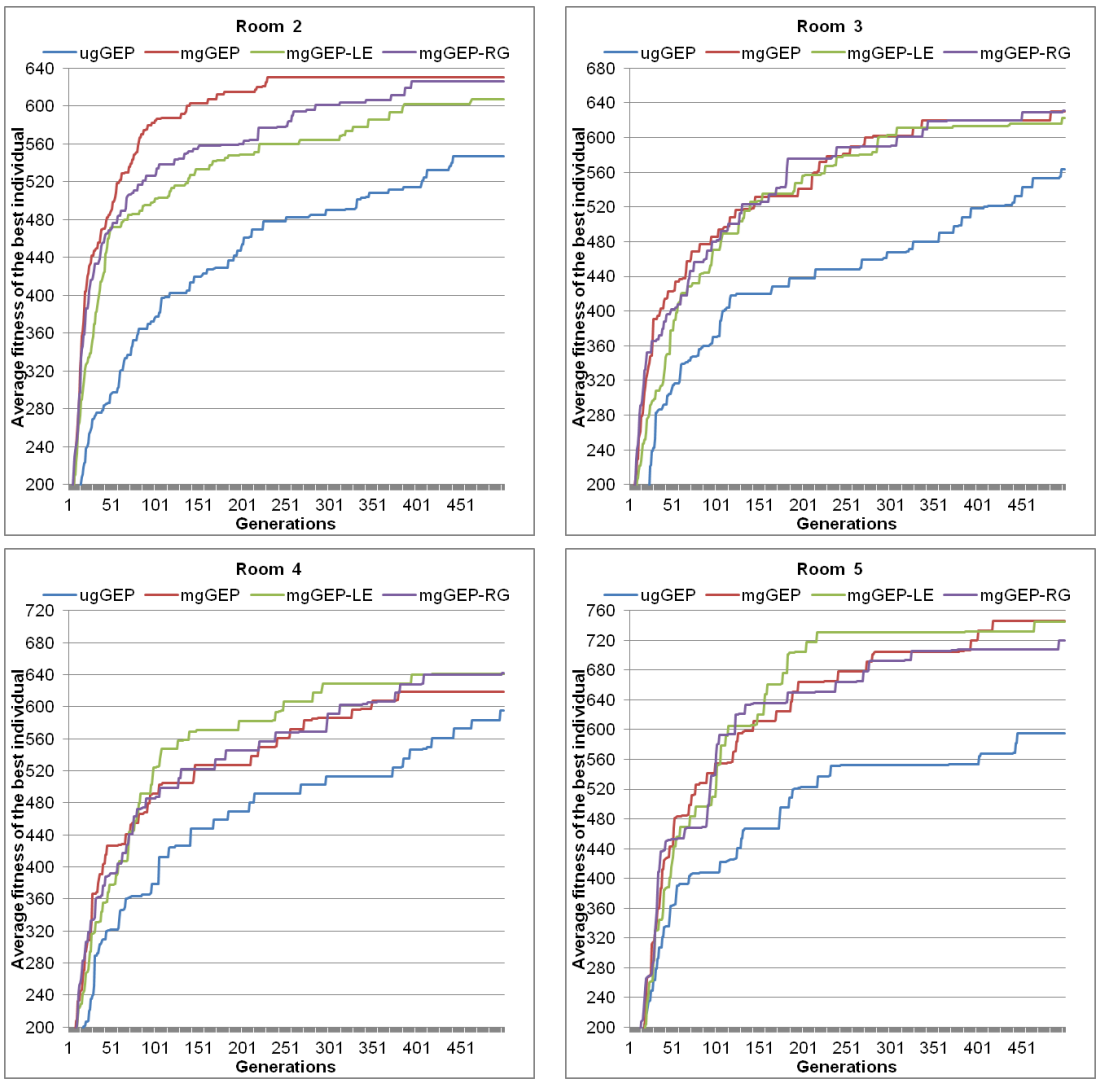


Figure 5.9: Progression of the mean fitness of the best individual in the population as achieved through ugGEP, standard mgGEP, mgGEP-LE and mgGEP-RG in the different room types. The mean fitness has been computed from the best individual fitness achieved in each generation in all the 50 randomly seeded algorithms. In each evolutionary run, a population of 500 organisms was evolved for 500 generations and the best individual fitness recorded in each generation. The rest of the parameters are as shown on Table 5.3.

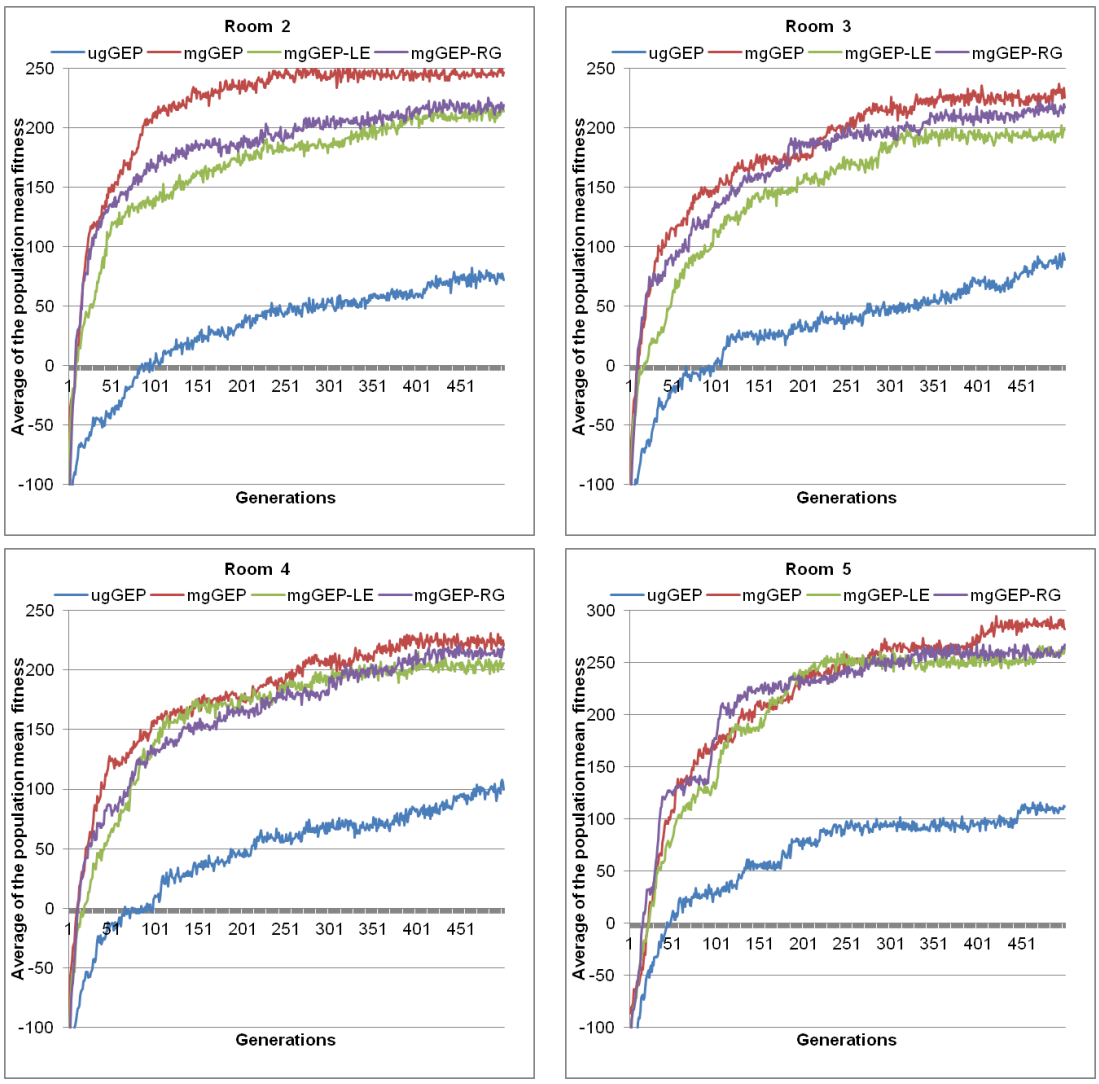


Figure 5.10: Progression of the mean of the population mean fitness as achieved through ugGEP, mgGEP, mgGEP-LE and mgGEP-RG in the different room types. The mean of the population mean fitness was computed from the population mean fitness achieved in each generation in all the 50 randomly seeded algorithms. The rest of the parameters are as shown on Table 5.3.

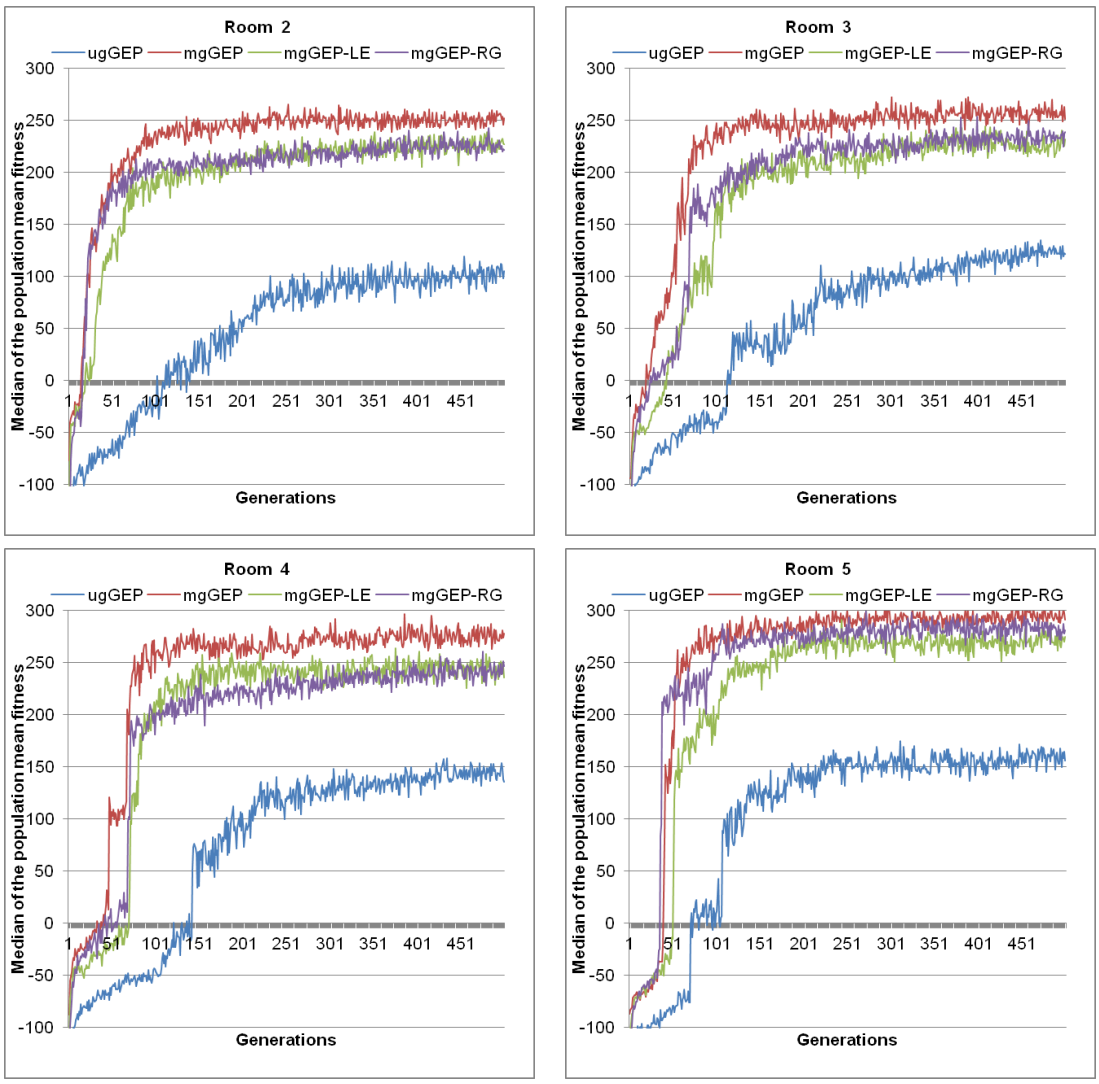


Figure 5.11: Progression of the median of the population mean fitness as achieved through ugGEP, mgGEP, mgGEP-LE and mgGEP-RG in the different room types. The median of the population mean fitness was computed from the population mean fitness achieved in each generation in all the 50 randomly seeded algorithms. The rest of the parameters are as shown on Table 5.3.

Figures 5.8, 5.9, 5.10 and 5.11 show the results of the ugGEP, mgGEP, mgGEP-LE and mgGEP-RG algorithms with an increased number of generations. As shown by the results, increasing the number of generations leads to an improvement in performance of the ugGEP, mgGEP, mgGEP-LE and mgGEP-RG results. Additionally the results suggest that mgGEP-LE and mgGEP-RG algorithms are able to approximate the performance of mgGEP where as previously mentioned, human expertise has been utilised. The success rates shown by figure 5.8 suggests that mgGEP, mgGEP-LE and mgGEP-RG algorithms achieved almost equal success rates in solving the wall following problem as described here. However, the mgGEP-RG appears to perform slightly better followed by standard mgGEP and then mgGEP-LE. These results suggest that given longer durations to evolve a solution, the mgGEP-RG and mgGEP-LE are able to approximate human designed modular architectures such as mgGEP.

Similar to the mean population mean fitness shown by Figure 5.10, the medians of the

population means shown by Figure 5.11 suggest that the mgGEP had a an overall better performance across all the rooms. The results also show that both mgGEP-LE and mgGEP-RG performed as well as the mgGEP in most of the rooms. These results suggest that these two new algorithms are able to approximate the mgGEP, where human expertise was utilised. The ugGEP did not perform as well as the modular controllers. The low ugGEP performance is likely to be as a result of lack of specialisation in the chromosome. Across all the rooms, the median population mean fitness values (Figure 5.11) are larger than the average population mean fitness values. The larger median values suggest that the bulk of the achieved mean fitness values lie to the right of the mean and have a negative skew. This suggests that in all the presented algorithms, the GEP performance generates a good result.

5.3.2 Statistical significance

Results shown in the previous sections suggested that there was an improved performance when modular algorithms were used to solve the wall following problem. In this section a Mann-Whitney U statistical test is carried out to determine whether there is any statistical significance between the different algorithms. Since the overall objective of an evolutionary algorithm is to find an optimal solution when an algorithm terminates, in order to carry out a Mann-Whitney U test, the average best individual fitness in an entire run was used. Thus, in each algorithm there were 50 independent observations resulting from the 50 randomly seeded runs carried out in the experiment. The null hypothesis H_0 was: "No difference in test algorithms" while the alternative hypothesis H_A was: "There is a difference in the test algorithms". Tables 5.4, 5.5 and 5.6 show the result of Mann-Whitney U test performed using ugGEP algorithm and each of the modular GEP algorithms. Additionally, tables 5.7, 5.8 and 5.9 show the comparison of the different modular algorithms.

Table 5.4: mgGEP vs ugGEP

	z Value	P_2 Value	Significant at:
Room 2	4.71	< 0.0001	2%
Room 3	2.85	0.0044	2%
Room 4	2.08	0.0375	5%
Room 5	1.77	0.0767	10%

Table 5.4 shows that using a two tailed Mann-Whitney U test, there is statistical significance between the ugGEP algorithm and the mgGEP algorithm. The third column shows the level of significance. A one tailed test with the alternative hypothesis H_A that mgGEP is better than ugGEP, shows that there was statistical significance across all the rooms. These results strengthens the conclusion arrived in the previous chapter where it was shown that mgGEP outperforms ugGEP.

Table 5.5 shows that there is a statistical significance between ugGEP and mgGEP-LE algorithms. As discussed earlier, mgGEP-LE algorithm uses an additional function set, referred as a linking set, where the functions used to link the genes in a chromosomes

Table 5.5: mgGEP-LE vs ugGEP

	z Value	P_2 Value	Significant at:
Room 2	3.82	< 0.0001	2%
Room 3	2.93	0.0034	2%
Room 4	1.93	0.0536	10%
Room 5	1.95	0.0512	10%

are randomly picked at the beginning of the run. The linking functions are subjected to genetic operations and thus evolve together with the result of the genome. Thus, the algorithm evolves various mechanisms on how the genes should be linked depending on the selected linking function. The results shown here suggest that this new and dynamic linking technique, reported for the first time in this chapter, is a better mechanism to evolve solutions than the monolithic algorithm.

Table 5.6: mgGEP-RG vs ugGEP

	z Value	P_2 Value	Significant at:
Room 2	2.95	0.0032	2%
Room 3	2.23	0.0257	5%
Room 4	2.06	0.0394	5%
Room 5	1.75	0.0801	10%

Table 5.4 shows that using a two tailed Mann-Whitney U test, there is statistical significance between mgGEP-RG and the ugGEP algorithms. Unlike standard mgGEP and mgGEP-LE, the mgGEP-RG and ugGEP algorithms starts with no particular strategy on how the problem should be solved. The two algorithms are thus similar at the beginning as they have to evolve a strategy on how the problem should be solved. Thus, the statistical significance shown by table 5.6 strengthens the conclusion that modular mechanisms to solving a problem outperform monolithic ones. Additionally, the results suggest that mgGEP-RG algorithm, reported in this chapter for the first time, is a competitive technique that can be utilised in solving robotic problems.

Table 5.7: mgGEP vs mgGEP-LE

	z Value	P_2 Value	Significant at:
Room 2	1.93	0.0536	10%
Room 3	0.84	0.4009	H_0
Room 4	0.16	0.8729	H_0
Room 5	0.14	0.8887	H_0

Table 5.8: mgGEP vs mgGEP-RG

	z Value	P_2 Value	Significant at:
Room 2	0.23	0.8181	H_0
Room 3	0.22	0.8259	H_0
Room 4	0.07	0.9442	H_0
Room 5	0.27	0.7872	H_0

Tables 5.7, 5.8 and 5.9 show that, overall, there is no statistical significance between the three described modular approaches. In this case, the null hypothesis (H_0) was accepted.

Table 5.9: mgGEP-RG vs mgGEP-LE

	z Value	P_2 Value	Significant at:
Room 2	1.44	0.1499	H_0
Room 3	0.78	0.4354	H_0
Room 4	0.0	1	H_0
Room 5	0.37	0.7114	H_0

Results in table 5.7 shows that the there was some level of significance in the performances of the standard mgGEP and mgGEP-LE in room 2. This statistical significance, though only at 10% level, can potentially be explained by the fact that the implementation of AND and OR linking functions could have lowered the overall algorithm performance, as describe in section 5.1.2. The lack of statistical significance in the rest of the rooms is to be expected since mgGEP-LE shares many characteristics with the standard mgGEP

Results in table 5.8 shows that the mgGEP-RG is not significantly different from standard mgGEP. Since mgGEP-RG has to evolve the entire mechanism to sub-divide behaviour, these results are of great significance as it shows that when presented with a problem requiring a modular approach, a developer does not necessarily need to think of a strategy to sub-divide the problem. The results shows that the algorithm is able to evolve suitable techniques that match techniques used by a human designed behaviour sub-division strategy. Table 5.9 also strengthens the claim that with no particular suggestions on how to solve a problem, a developer can use an algorithm such as mgGEP-RG to solve the problem with equally good success as mechanisms requiring human input.

Figure 5.12 shows a comparison of **P** values calculated using a Mann-Whitney U test on best individual fitness in each generation. The values that have been used to calculate the Mann-Whitney U test were generated from the 50 identical runs on each algorithm. Since the major interest is to show how the algorithms compared to each other statistically, only results from room 2 have been used. However, as shown in tables 5.4, 5.5, 5.6, 5.7, 5.8 and 5.9 the results correlate in all the rooms. The smaller the **p** value the less probable the experimental results is due to chance. Smaller **p** values also mean that there is a statistical significance between the tested algorithms. As shown by Figure 5.12 there was statistical significance between ugGEP and mgGEP in every generation. Similarly there was statistical signicance in the results achieved using ugGEP and mgGEP-LE algorithm across all generations, However the p values are not as small as with standard mgGEP. This is likely because mgGEP-LE uses some functions in the linking set which may not have scaled to the problem leading to lower performance than standard mgGEP. The comparison of statistical significance between ugGEP and mgGEP-RG algorithm shows that up to about the tenth generation, the null hypothesis (H_0) had to be accepted. This effect is because at the beginning the results are still largely random, however as the evolution progresses the mgGEP-RG utilises its modular structure to develop a better strategy than ugGEP and hence the statistical significance changes. The rest of the graph shows that there was statistical significance in the early generations between mgGEP and mgGEP-LE algorithms as well as mgGEP and mgGEP-RG algorithm. These results suggest that at the beginning the more superior standard mgGEP outperforms the new modular mecha-

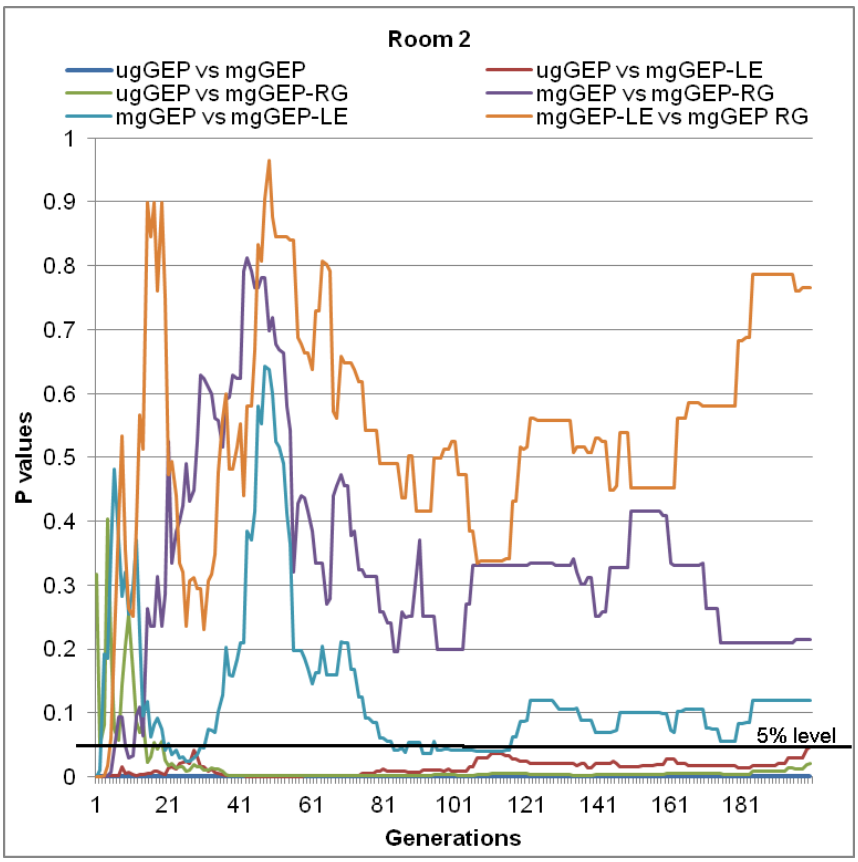


Figure 5.12: P values generated using mann whitney two tailed test. The values are generated from 50 observations in each generation. Each curve shows the comparison with a different algorithms. A summary of all the parameters used in this experiment is shown on Table 5.3.

nisms. However as the evolution continues the performance of the standard mgGEP is not significantly different from that of the new algorithms. The overall results suggest that modular algorithms perform better than monolithic algorithm particularly in this problem domain. In addition, the proposed "self-organising" modular algorithms, mgGEP-LE and mgGEP-RG, are suitable mechanisms that can be utilised to evolve modular robot behaviours.

5.3.3 Solution convergence

In this section, the 25th (Q1) and 75th (Q3) percentile fitness values were calculated from the fitness of the best individuals evolved by the different GEP algorithms. These values were then plotted alongside the median fitness in each generation. The median, Q1 and Q3 values were calculated from the best individual fitness values achieved by each of the 50 randomly seeded run in each generation. The aim of this analysis is to determine how quickly the controllers evolved by the different algorithms converge to the target fitness and to measure the fitness variability.

In all the figures shown below, the top dotted line represents Q3 values while bottom dotted lines represent Q1 values. Q2 (Median) values are represented by the solid line.

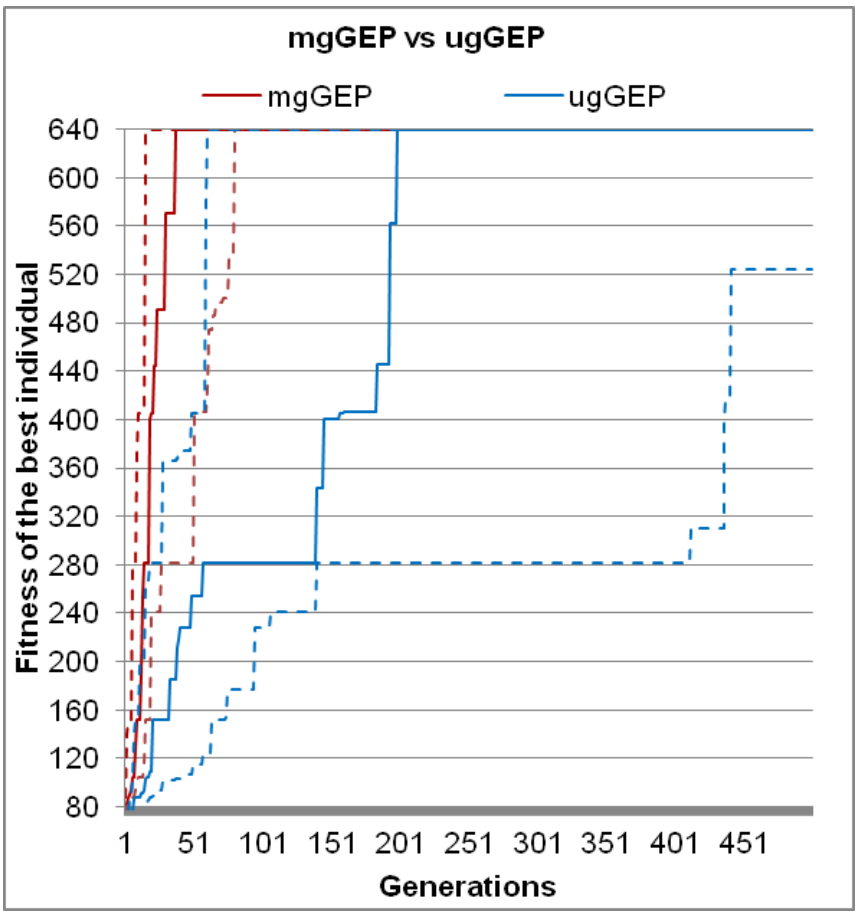


Figure 5.13: Q1, Q2 and Q3 values of the best individual in the population as achieved by ugGEP and mgGEP algorithms, plotted every generation. These values relate to experiment carried out in room 2 using the ugGEP and mgGEP algorithm parameters shown on Table 5.3. In both algorithms, the top dotted line represents Q3 values while bottom dotted lines represent Q1 values. Q2 (Median) values are represented by the solid line.

Figure 5.13 shows that within $15-20$ generations at least 25% of the mgGEP algorithm runs had evolved a controller that achieved the fitness set for room 2. Also, within $70-80$ generations the interquartile range was zero. Also, the results show that a median fitness corresponding to the target fitness was achieved within 40 generations. In the ugGEP algorithm, a median fitness corresponding to target fitness was achieved after 200 generations. Additionally, the interquartile range was still very large at the 500th generation. Thus, the obtained results show that mgGEP algorithm converged quickly while there was still high diversity in ugGEP controllers at the end of the run. Since all the individuals are of the same length, their performance can possibly be attributed to the chromosome structure. In the ugGEP case, there is only one gene forming the chromosome. The evolved controller, referred to as a monolithic controller, solves the entire problem. Additionally, the low convergence in ugGEP algorithm could potentially be caused by genetic operations such as mutation. As shown in section 5.3.5, destructive mutation effects are likely to affect the monolithic controllers more than the modular controllers.

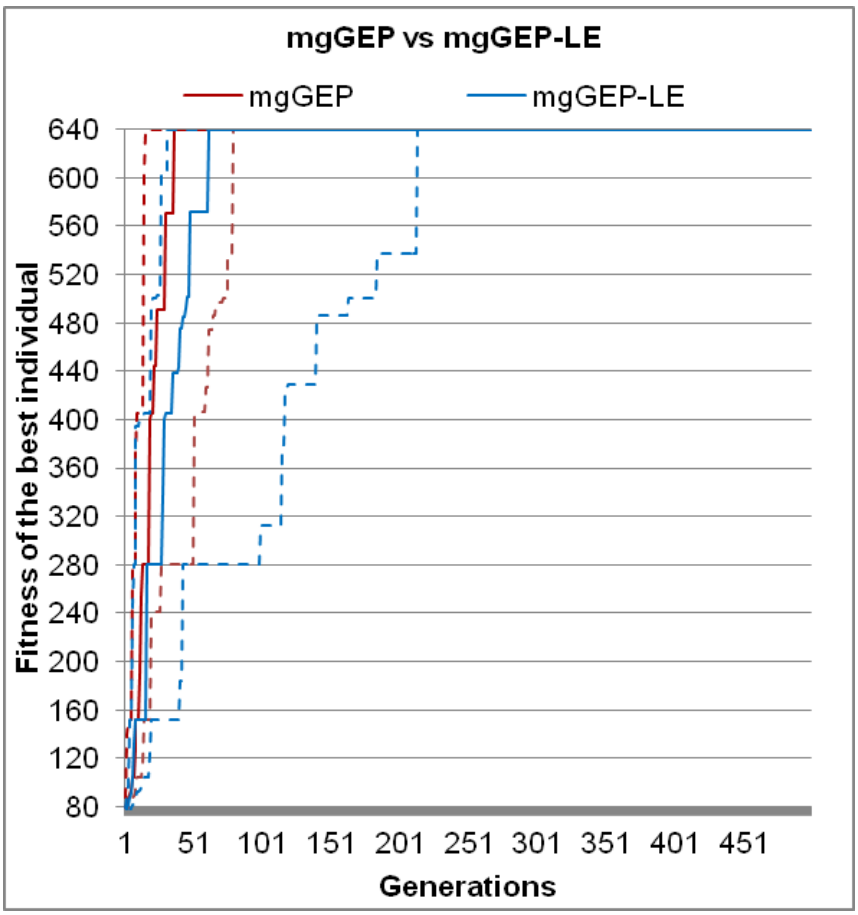


Figure 5.14: Q1, Q2 and Q3 values of the best individual in the population as achieved by mgGEP and mgGEP-LE algorithms, plotted every generation. These values relate to experiment carried out in room 2 using the mgGEP and mgGEP-LE algorithm parameters shown on Table 5.3. In both algorithms, the top dotted line represents Q3 values while bottom dotted lines represent Q1 values. Q2 (Median) values are represented by the solid line.

Figure 5.14 shows that a median fitness corresponding to the target fitness was achieved within $50-60$ generations using the mgGEP-LE algorithm. Additionally, within 35 generations at least 25% of the algorithm runs had evolved a controller that achieved the fitness set for room 2. Also, within 200 generations the interquartile range was reduced to zero. Thus the obtained results show that the mgGEP-LE controllers are able to approximate the performance of the mgGEP albeit using longer generation runs. This results can be explained by the fact that mgGEP-LE is similar in many ways to the standard mgGEP.

Similar to the standard mgGEP, the results shown by Figure 5.15 shows that within $15-20$ generations, 25% of the mgGEP-RG algorithm runs had evolved a controller that achieved the fitness set for room 2. Additionally, within 30 generations, a median fitness corresponding to the target fitness was achieved. These results suggest that mgGEP-RG approximated and outperformed the performance of the standard mgGEP algorithm. However, as discussed in the previous section, Figure 5.15 shows that mgGEP-RG required longer generations run in order to converge. Thus, the interquartile range was reduced to zero within 150 generations. The slow convergence rate in mgGEP-RG and mgGEP-LE can be attributed to the increase in search space attributed to genetic operations in the

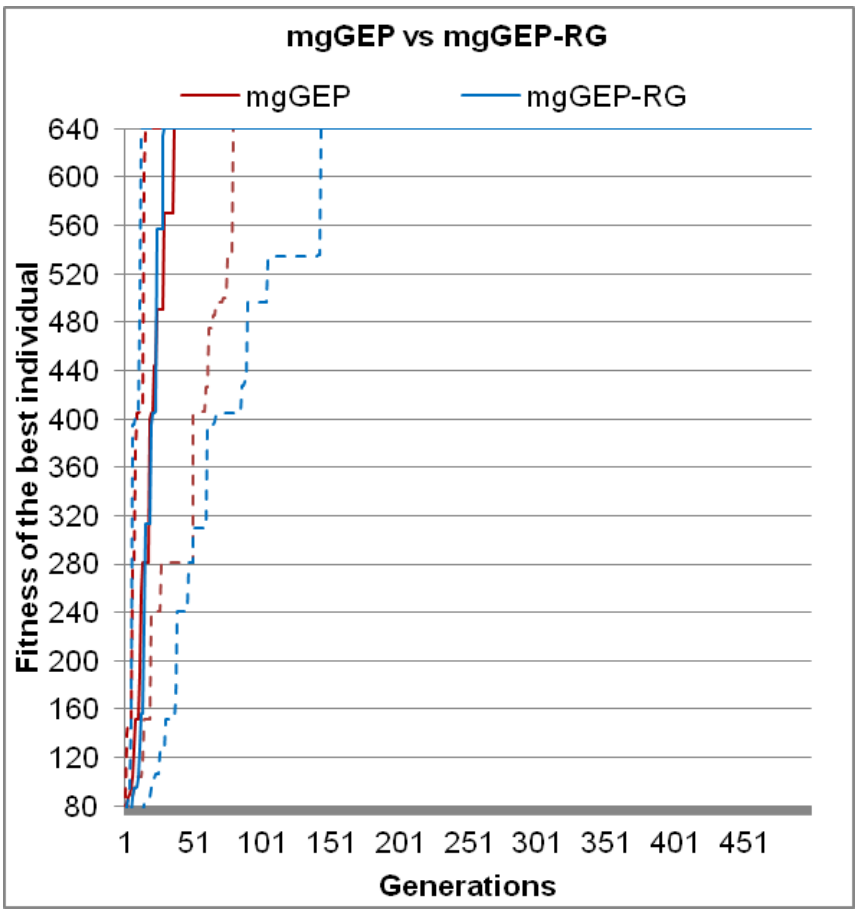


Figure 5.15: Q1, Q2 and Q3 values of the best individual in the population as achieved by mgGEP and mgGEP-RG algorithms, plotted every generation. These values relate to experiment carried out in room 2 using the mgGEP and mgGEP-RG algorithm parameters shown on Table 5.3. In both algorithms, the top dotted line represents Q3 values while bottom dotted lines represent Q1 values. Q2 (Median) values are represented by the solid line.

linking set for mgGEP-LE and the addition of a regulatory gene in case of mgGEP-RG.

5.3.4 Controller performance

This section shows various controllers evolved through mgGEP, mgGEP-LE and mgGEP-RG algorithms. The controllers were evolved in the 2D discrete environment (figure 4.3) with controllers as defined in section 4.3.4. All the controllers presented here solved the wall following problem in room 5 completely; thus, they all achieved the maximum fitness for room 5 (see Table 4.1 for success predicate). The functions; AND is represented by **A** while NOT function is represented by **N**. The translation of the controllers to expression trees (ET) is completed by starting from the controller root node, and moving from left to right, top to bottom.

In all the graphs, the top part shows the coding region of the genes which forms the controller while the expression tree (ET), shows the decoded controller. The linker forms the root of the controller and the bottom part shows how the overall controller solved the problem in room 5. As discussed previously, in mgGEP and mgGEP-LE, the linker

decides which sub controller effects motor controller based on the conditions supplied to the linking functions. For the mgGEP-RG, the regulatory gene provides mechanisms to activate or inhibit either of the two genes.

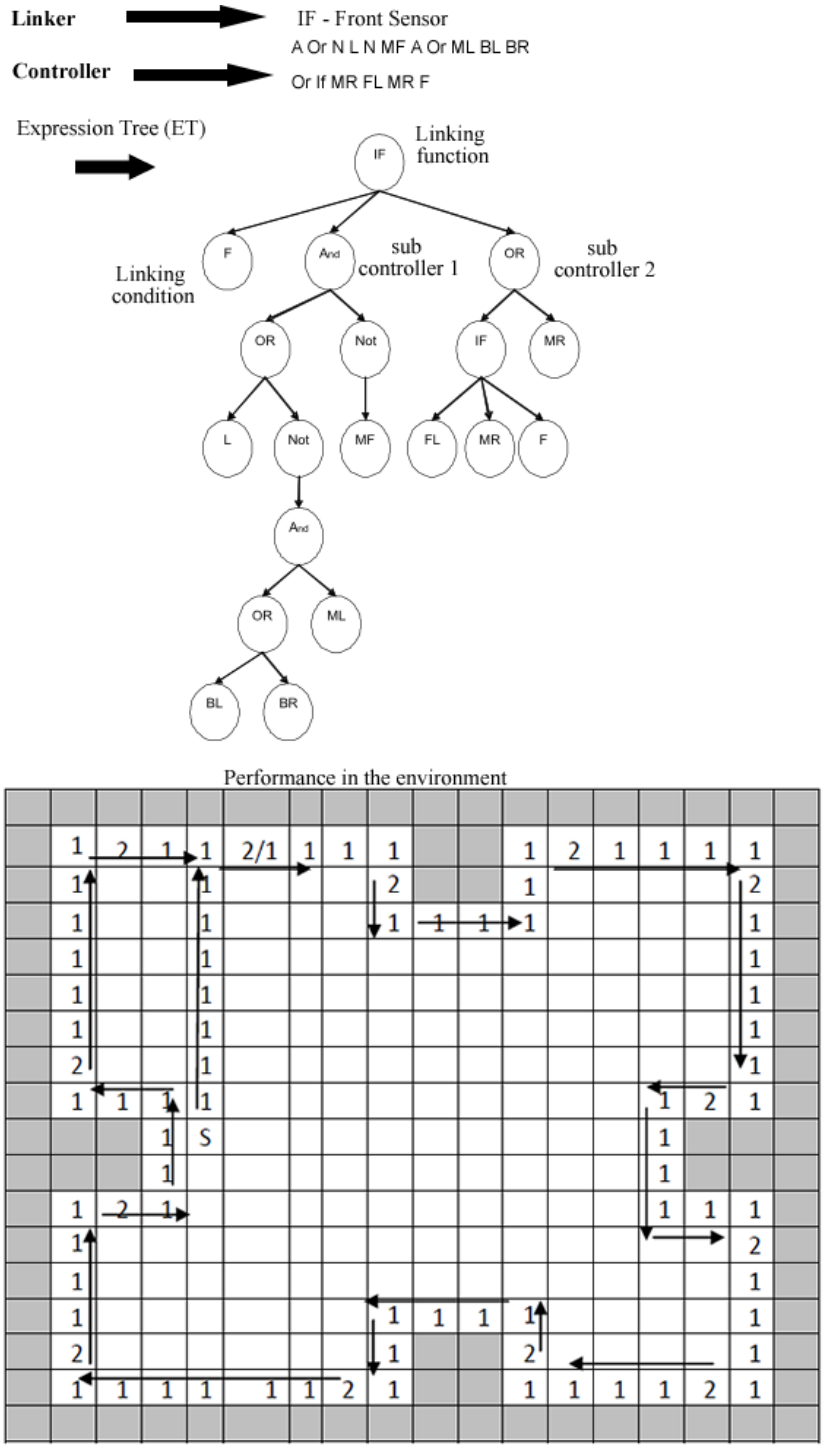


Figure 5.16: An example of a multi-controller evolved using standard mgGEP. In this example, sub controller (gene 2) is used for obstacle avoidance and gene 1 is used for exploration and straight line navigation with wall proximity.

Figure 5.16 shows an example of a controller evolved using mgGEP algorithm to solve the problem. As shown in the previous chapter, the algorithm had the problem manually subdivided and therefore one gene evolved obstacle avoidance behaviour whereas the other

gene evolved an exploration with wall proximity.

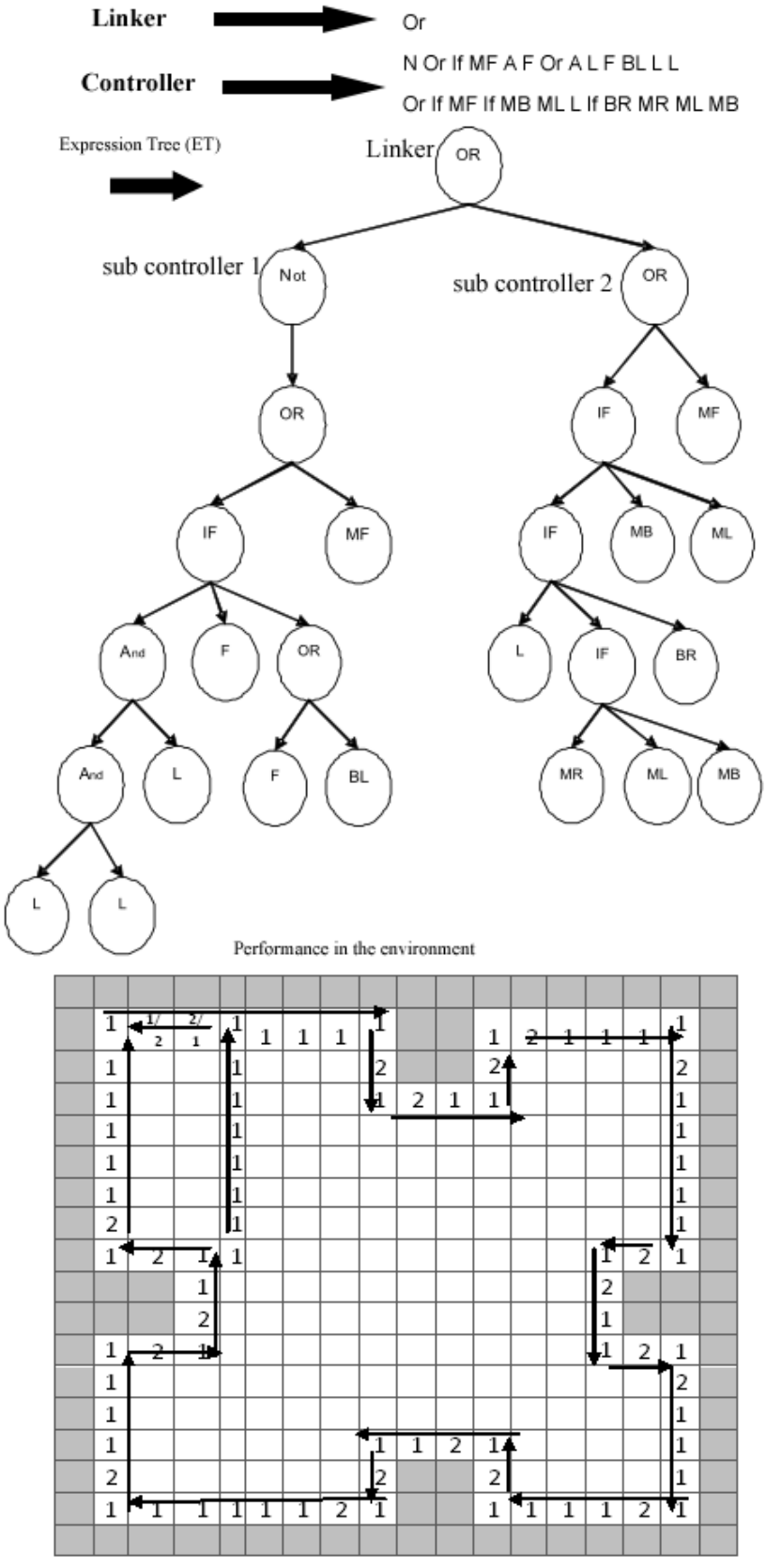


Figure 5.17: An example of a multi-controller evolved using mgGEP-LE. In this example, sub controller (gene 2) is used for obstacle avoidance and gene 1 is used for exploration and straight line navigation with wall proximity. This controller utilises the left sensor to enable the robot navigate around the the extrusions and also uses the front sensor to evade obstacles. This is a novel technique than the mgGEP example shown on Figure 5.16.

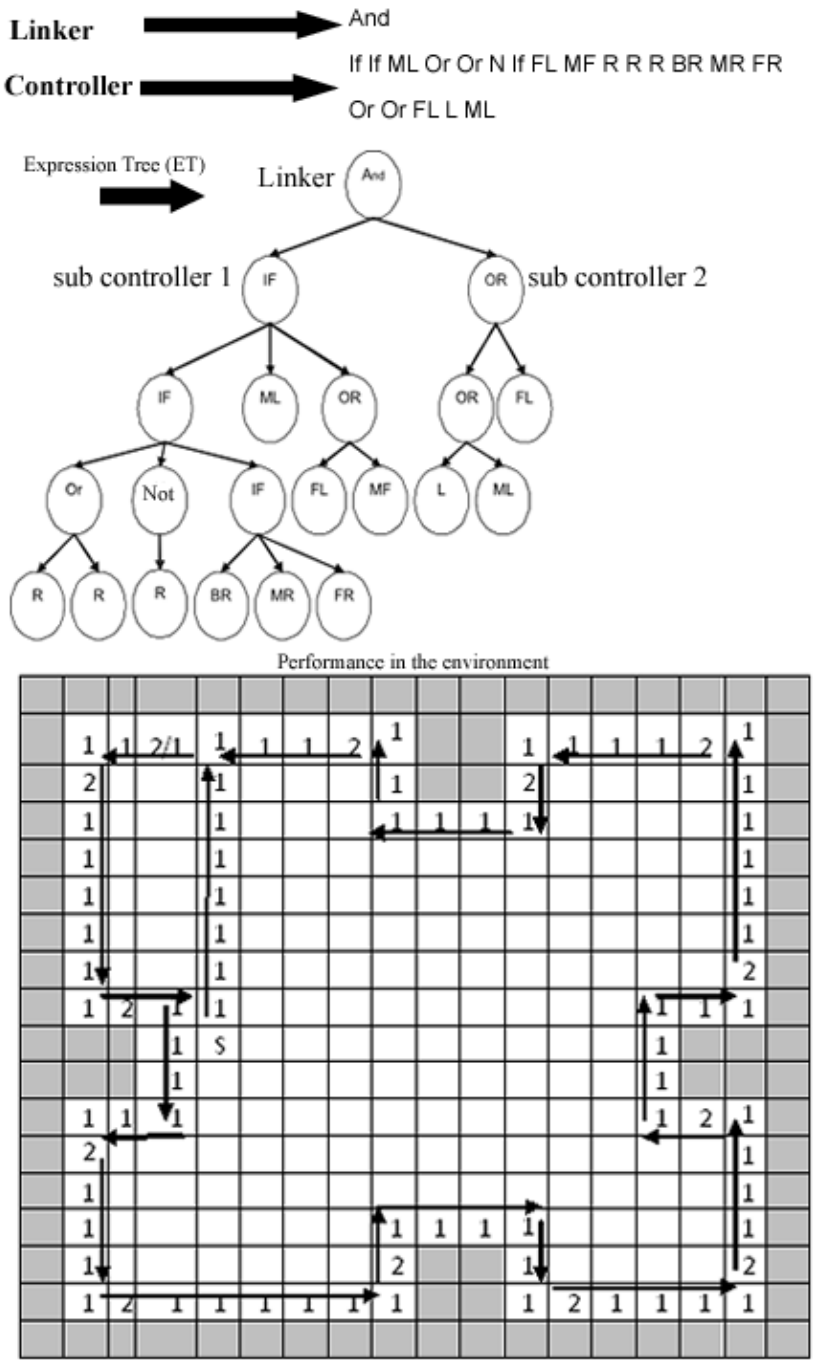


Figure 5.18: A multi-controller evolved using mgGEP-LE. In this example, sub controller (gene 2) is used for obstacle avoidance and gene 1 is used for exploration and straight line navigation with wall proximity. This approximates the standard mgGEP as shown on Figure 5.16, by moving in opposite direction.

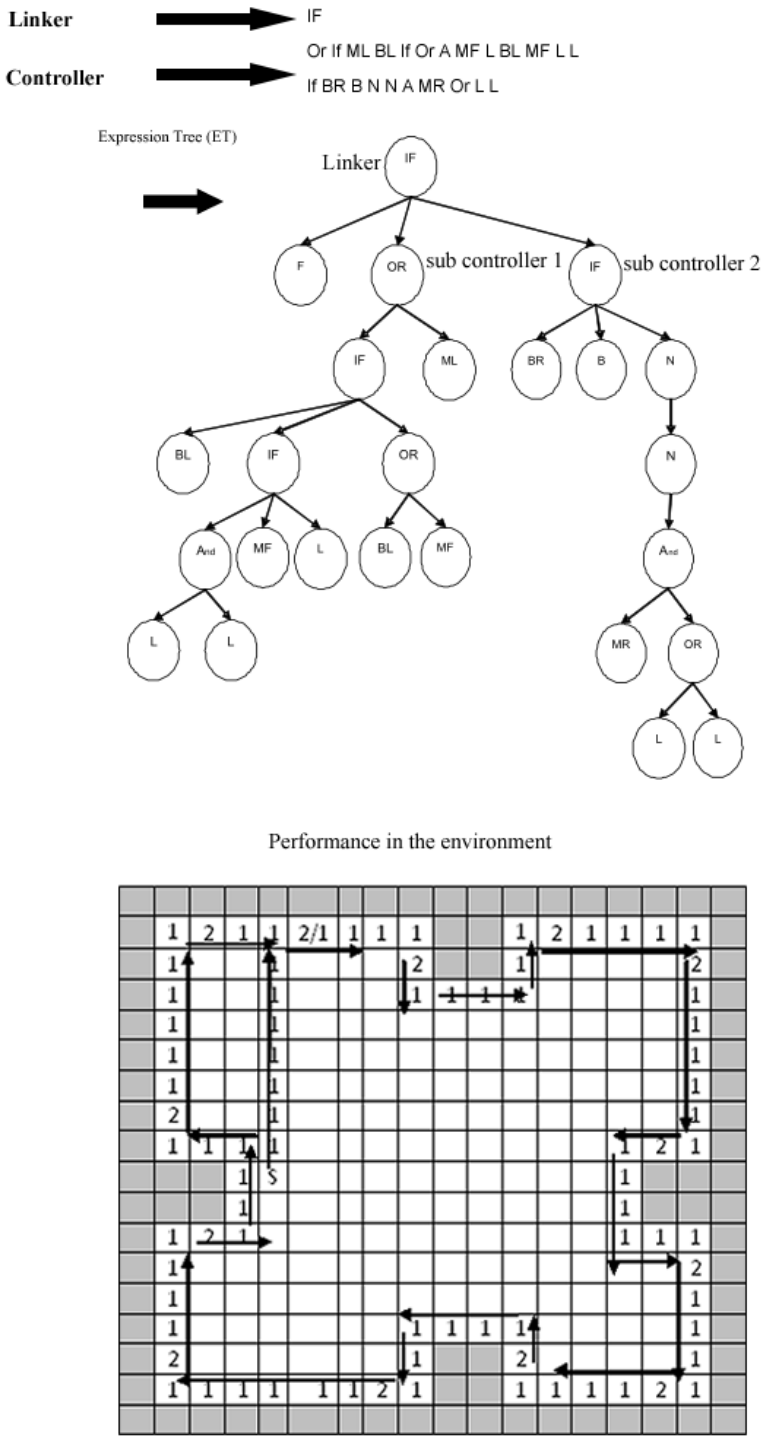


Figure 5.19: An evolved mgGEP-LE controller that replicates the behaviour of a controller evolved using mgGEP. In this example, sub controller (gene 2) is used for obstacle avoidance and gene 1 is used for exploration and straight line navigation with wall proximity. This approximates the standard mgGEP as shown on Figure 5.16.

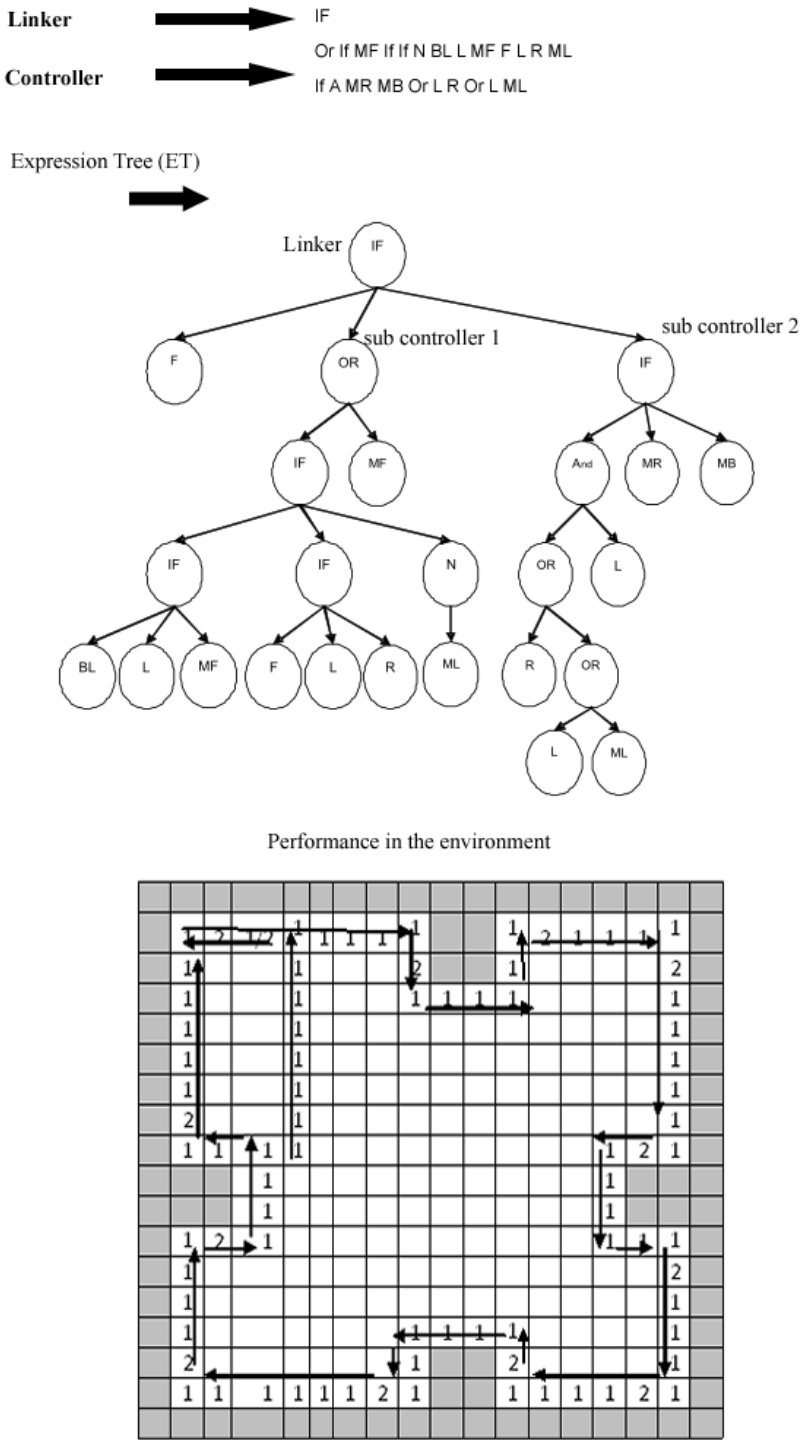


Figure 5.20: Example of a multi-controller evolved using mgGEP-LE. In this example, sub controller (gene 2) is used for obstacle avoidance and gene 1 is used for exploration and straight line navigation with wall proximity. This controller is different to controllers in Figures 5.16 and 5.19 but generates a similar behaviour.

Figures 5.17, 5.18, 5.19 and 5.20 shows the various strategies that mgGEP-LE used to solve the problem. From the evolved controllers it is evident that mgGEP-LE was able to evolve mechanisms that accurately approximated the human designed strategy developed using the standard mgGEP.

Table 5.10: Analysing the linking set

Linker:	IF	OR	AND
Room 2	84%	6%	10%
Room 3	86.8%	5.3%	7.9%
Room 4	80%	15%	5%
Room 5	74.5%	10.6%	14.9%

Table 5.10 shows the percentage of the organism that used a particular linking function in solving the problem. From the results, the IF linker outperformed both OR and AND linkers. The IF linker, as discussed, checks the front sensor and then activates either of the two sub-controllers. The results suggest that this particular mechanism to divide behaviour is the best for this problem. The introduction of the other two linkers, OR and AND, lowers the number of plausible controllers in the solution set and may lead to an overall low success rate in the mgGEP-LE algorithm. These results further suggest that care should be taken in selecting the linking functions, so as to choose controllers that contribute positively to the algorithm.

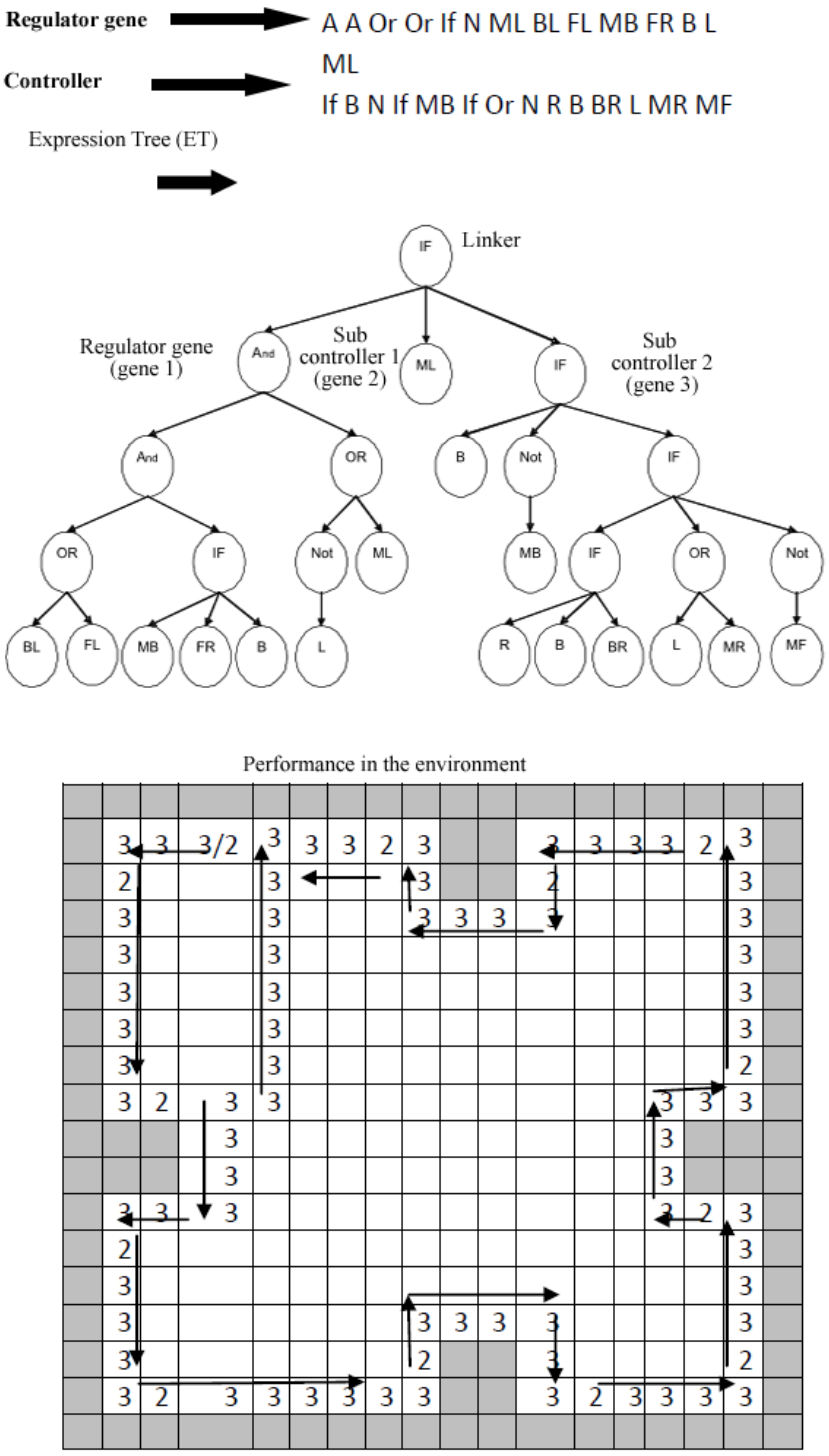


Figure 5.21: An example of a multi-controller evolved using mgGEP-RG. This controller utilises gene 2 for obstacle avoidance and uses gene 3 for exploration and straight line navigation with wall proximity. Its behaviour replicates the behaviour of the controller evolved using mgGEP-LE as shown on Figure 5.18.

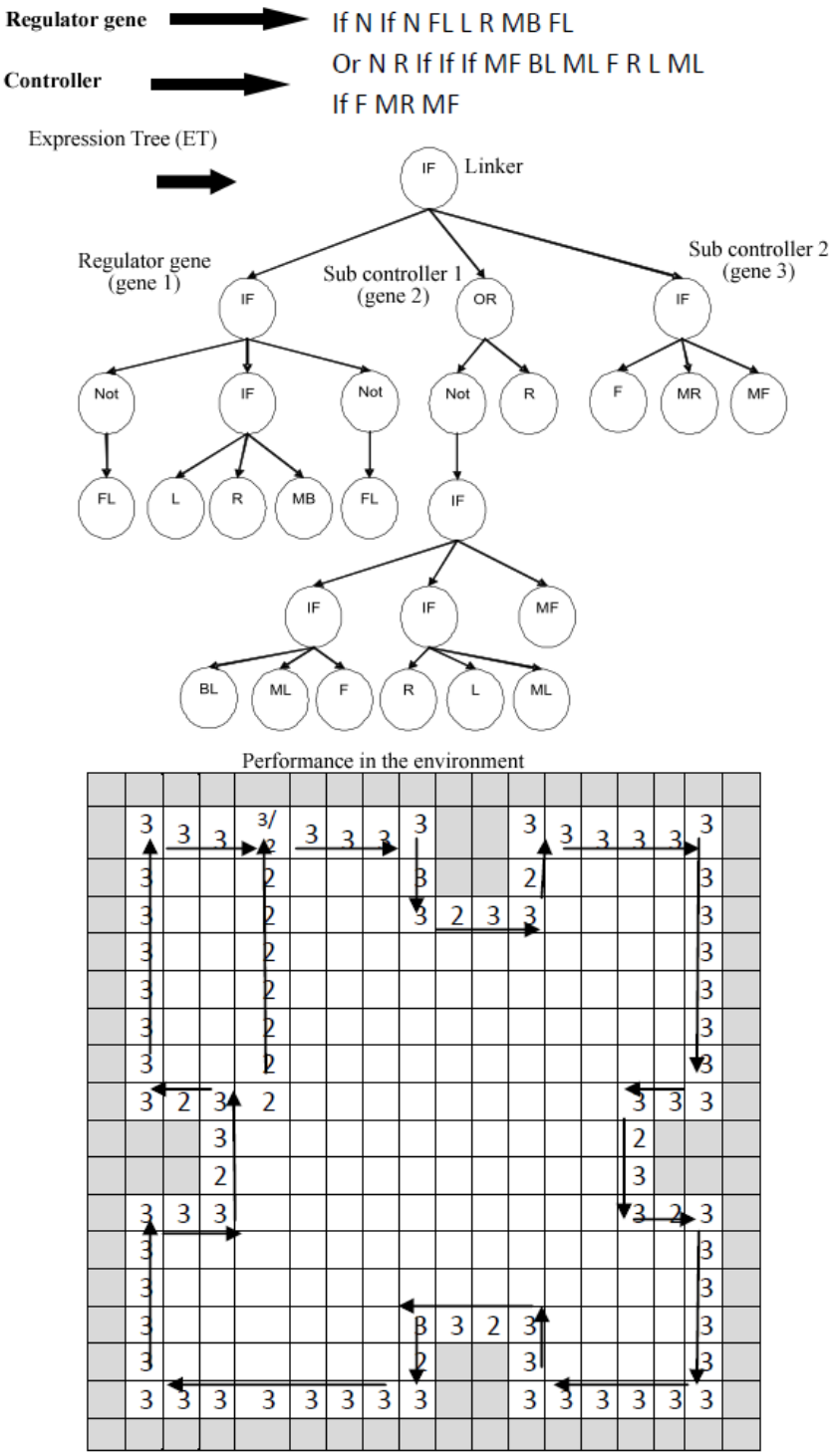


Figure 5.22: Example of a novel multi-controller evolved using mgGEP-RG. This controller utilises gene 2 for exploration in open areas and for navigation around the extrusions. The controller utilises gene 3 for navigation with wall proximity and evading obstacles detected by the front sensor.

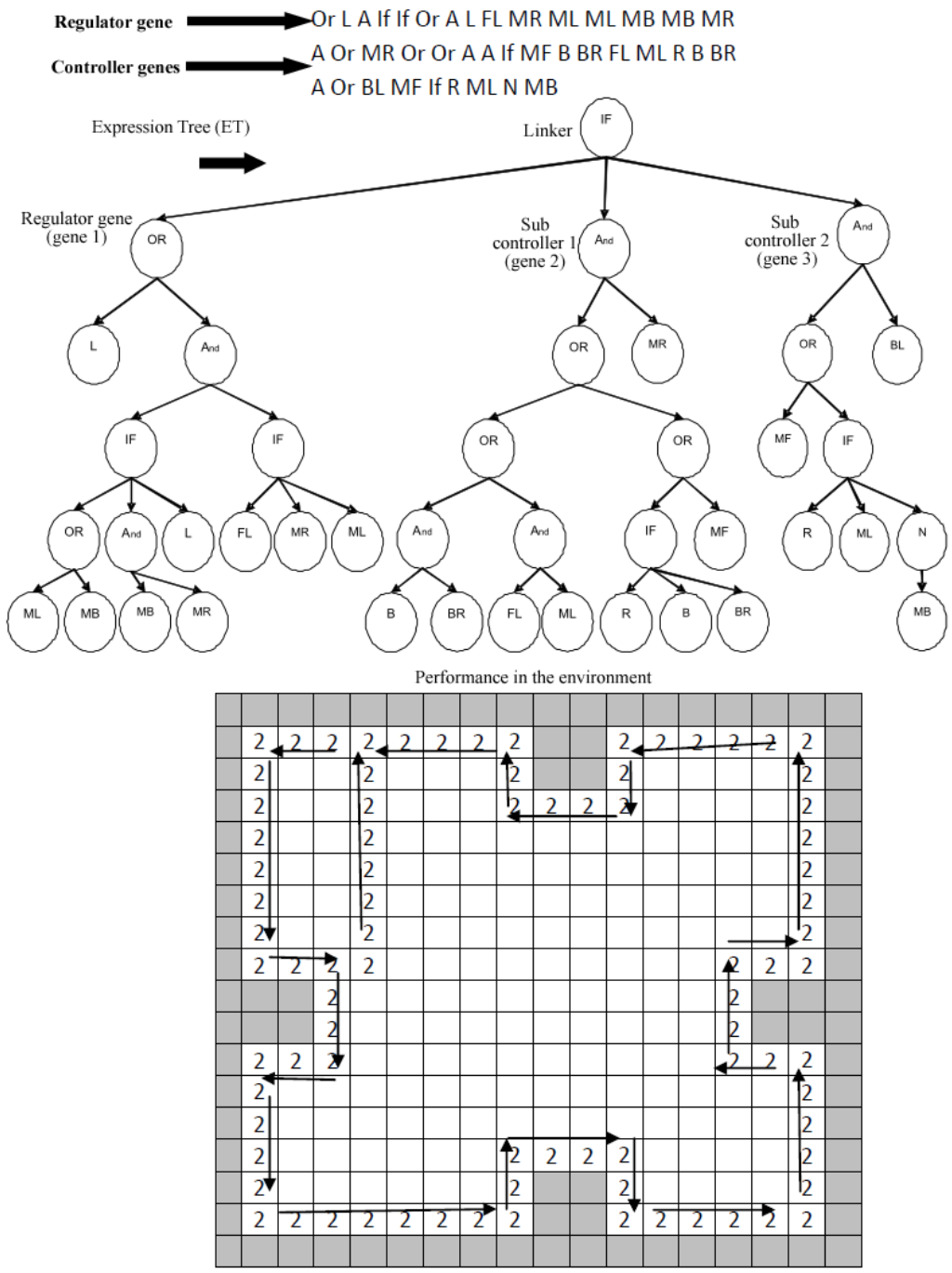


Figure 5.23: A strategy used by mgGEP-RG to solve wall following problem in room 5. In this example, the controller utilised gene 2 to perform the whole task of wall following. The rest of the genome was utilised as pseudo gene and created redundancy in the genome.

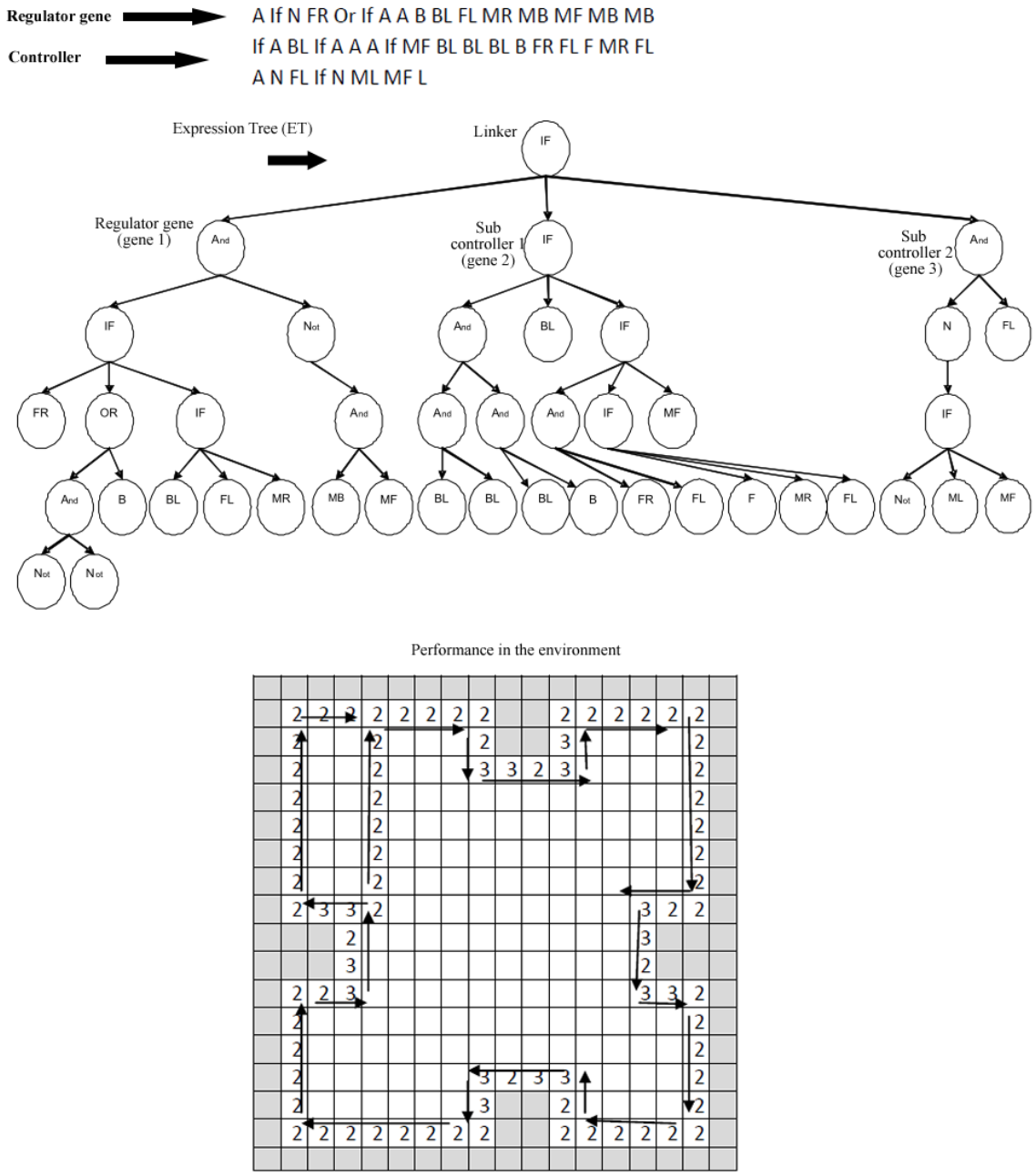


Figure 5.24: A novel mgGEP-RG controller evolved to solve wall following problem in room 5. In this example the second gene is used for exploration and obstacle avoidance and the third gene used to navigate around obstacles.

The mgGEP-RG used various strategies to solve the problem. In figure 5.21, the second gene was used for obstacle avoidance and the third gene used for the exploration with wall proximity. This controller is an approximation of controllers evolved using standard mgGEP. The second example, Figure 5.22, the controller used gene 3 for obstacle avoidance and exploration with wall proximity by concentrating on the use of front sensor. For exploration in the open areas and around the extrusions the controller used gene 2. In Figure 5.24, the controller utilised the second gene for exploration and obstacle avoidance and the third gene was used to navigate around obstacles. From these examples, it is evident that mgGEP-RG techniques used new and novel methods to solve the problem which the human designer would possibly not have thought of or would be too complicated

to implement. In another example, shown by Figure 5.23, the algorithm used the second gene to solve the entire problem; in this case the third gene is a pseudo gene and provides redundancy to the chromosome. The redundancy built into an algorithm like the mgGEP-RG is important for the evolution of more fit individuals.

5.3.5 Mutation effect

Figures 5.8, 5.9 and 5.10 show that the ugGEP performs poorly in comparison to the mgGEP methods on this task, even with an extended period of evolution. One theory posited for this is the potentially destructive effect mutation will have on a monolithic chromosome. This section focusses on how mutation operations affect the performance of the presented algorithms.

To conduct the experiment, a population of 500 organisms were evolved for 200 generations. Similar to previous experiments, the mgGEP-RG was implemented using 3 genes while the mgGEP and mgGEP-LE were evolved using 2 genes. The rest of the parameters, including ugGEP algorithm settings, are as reported on Table 5.3. The probability of mutation, P_m, was set to at least two mutations per chromosome. Thus, for each algorithm the probability was given as :

$$P_m = 2/Chromosome_{length} \tag{5.1}$$

Similar to the experimentations in section 4.3, each controller in the population was tested from 10 starting points and the fitness evaluated using Equation 4.1. The experiment was carried out using the wall following room 2 (see Figure 4.3 for room types). The maximum fitness for each room is shown in Table 4.1. In all the experiments, 10 randomly seeded algorithm runs were conducted.

During the evolution, the fitness of an individual was recorded before and after mutation operation. No fitness was recorded if a mutation operation did not occur. The recorded fitness values were then used to calculate the percentage of controllers where the mutation had a positive, negative or neutral effect. The obtained results are shown in Table 5.11 and Figure 5.25.

Table 5.11: Mutation effect on GEP algorithms

Effect	ugGEP	mgGEP	mgGEP-LE	mgGEP-RG
Positive effect(%)	2.15	1.0	0.89	1.03
Neutral effect (%)	67.58	76.38	77.47	81.61
Negative effect(%)	30.27	22.62	21.64	17.36
Mean effect	−170.62	−159.28	−151.83	−113.52

Figure 5.25 and table 5.11 shows the effect of mutation when applied to ugGEP, mgGEP, mgGEP-LE and mgGEP-RG algorithms. The results suggest that mechanisms utilising more than one gene have a higher probability of having non-destructive mutation effect. The largely neutral effect is brought by mutation occurring at non-coding regions of the

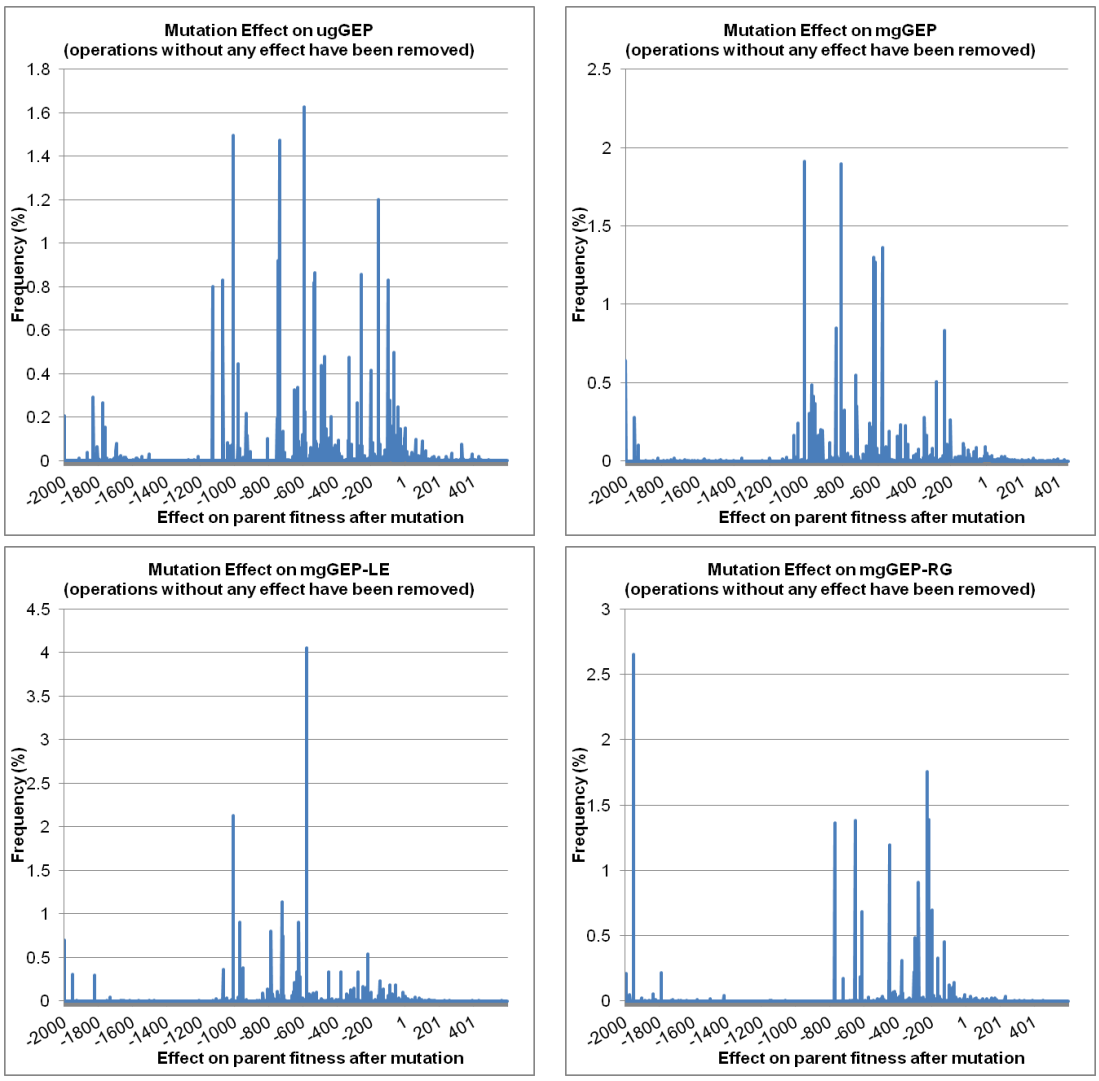


Figure 5.25: Mutation effect on ugGEP, mgGEP, mgGEP-LE and mgGEP-RG individuals. In each experiment, a population of 500 organisms was evolved for 200 generations. During the evolution, the fitness of an individual was recorded before and after a mutation operation. No fitness was recorded if a mutation operation did not occur. All the experiments were replicated over 10 randomly seeded runs. The rest of the parameters are as shown on Table 5.3.

gene. Additionally, neutral effects could be as a result of mutations at the coding region that do not change the phenotype of the organism: for instance, when a terminal is mutated to a different terminal or even when functions are mutated causing no effect in the overall phenotype [39, 68]. These neutral effects could contribute to the overall better performance shown by the modular GEP algorithms. A detailed discussion on neutrality in evolutionary algorithms can be found in [52]. The mutation analysis also indicates that there is a high probability of a mutation to be either destructive or not have any effect. The high destructive effect in ugGEP can be attributed to the monolithic nature of the organism. The probability of a mutation occurring in the head region is twice in ugGEP than in standard mgGEP and mgGEP-LE, and is thrice the probability of mutation occurring in mgGEP-RG. If a destructive mutation operation occurs in a monolithic chromosome it would have a huge negative impact on the chromosome performance than it would have in modular chromosomes. Similarly, if the effect is positive it is likely to have an impact

on the ugGEP rather than the modular chromosomes. This can potentially explain the slightly higher positive mutation effect in ugGEP. Since the mgGEP-RG uses 3 genes, there is a high probability of neutral effect as well as low probability for positive and negative effect. In an algorithm such as GA where every allele in a chromosome is a part of the overall solution, mutation effects could have a huge effect on the solution. However, as shown here, since GEP chromosomes have coding and non-coding regions, mutations occurring in the non coding regions do not have any effect on the solution.

5.3.6 Discussion

In section 5.3.1, the obtained results show that the performance of the best individual evolved using mgGEP-LE outperformed the standard mgGEP in room 4 and performed as well as mgGEP in rooms 3 and 5. However, its performance was slightly lower than that of mgGEP in room 2. As mentioned earlier, the performance of the mgGEP-LE in this problem can be explained from the linking functions in the linking set. As shown by Table 5.10 and the standard mgGEP performance, the IF linker is the most effective linker. The AND and OR linker thus lowered the mgGEP-LE overall performance compared to the mgGEP where all the chromosomes used the IF linker. As mentioned earlier, the AND and OR linkers use the first gene as a regulatory gene. Thus, the first gene is not specialised for either of the available tasks or for evolving a condition. Therefore, it is likely the use of the first gene as both structural and regulatory gene reduced the capability of the mgGEP-LE to solve the problem.

The performance of the best individual evolved by mgGEP-RG outperforms mgGEP in room 4 and performs equally as well as mgGEP in rooms 2 and 3. However it is slightly outperformed in room 5. The overall average fitness of the population shows that mgGEP-RG approximated the performance of mgGEP across rooms 3, 4 and 5 but was outperformed in room 2. The success rates, however, show that mgGEP-RG outperformed mgGEP in rooms 3 and 4 and was outperformed in rooms 2 and 5. Therefore, the obtained results show that these two algorithms almost had an equal performance. This means that given a problem that requires a modular controller, a mgGEP-RG technique is capable of generating controllers that perform equally as well as those evolved using a pre-determined linking conditions. Since most autonomous robots are used in unstructured environments, the developer may not always know the prevailing conditions that may require certain motor control. Additionally, even in situations where the conditions are known, implementing the controllers and conditions manually is not only tedious but also prolong the development time. The results shown here suggests that an algorithm such as mgGEP-RG can be utilised to develop the conditions and behaviour sub-division automatically using evolutionary process.

The results of statistical significance test shown in section 5.3.2 shows that all three modular controllers are significantly different from the monolithic ugGEP controller. Additionally, it is shown that these modular controllers performed equally the same in this task. These results thus strengthen the conclusion that modular controllers are better than monolithic ones. Additionally, the results show that dynamic approach to behaviour

sub-division, as described by mgGEP-RG, is a plausible alternative approach to human determined behaviour sub-division.

The results shown in sections 5.3.1, 5.3.3 shows that mgGEP-LE and mgGEP-RG techniques require longer evolutionary runs in order to converge to the optimal fitness. This is to be expected since for mgGEP-LE, the algorithm require to evolve the best linking function as well as generate the best solution. Thus, in comparison to standard mgGEP, mgGEP-LE is likely to require a longer evolutionary run. However, in problems where the best linking function is unknown or where a number of candidate controllers exist then using the mgGEP-LE algorithm would be potentially a better technique than mgGEP. For mgGEP-RG, using a regulatory gene increases the search space, the algorithm thus requires longer generation run to evolve a suitable regulating mechanism as well as generate a solution for the problem. In addition, most real-life situations where an autonomous robot may be employed require different behaviour coordination strategy that a human designer may not have anticipated. In such situations, mgGEP-RG technique can easily be employed with satisfactory results.

Section 5.3.4 shows that both mgGEP-LE and mgGEP-RG evolved controllers which solved the problem in similar way to those evolved using mgGEP. Additionally, Figures 5.22, 5.23 and 5.24 shows that mgGEP-RG is capable of evolving novel controllers that than those evolved using mgGEP. The strategies devised by the evolved controllers are both unique and complex and could not be achieved using the mgGEP or mgGEP-LE technique. Due to the limited nature in which the conditions for the IF linker is formulated, the genes in mgGEP and mgGEP-LE can only check the front sensor before effecting motor control. However, this limitation is non existent in the mgGEP-RG as it relies on an evolutionary process to generate the condition. Thus, the algorithm is likely to evolve unique and innovative techniques to solve a problem.

The mutation effects described in section 5.3.5 shows that modular controllers are less likely to be affected by destructive mutation effects. Additionally, the results show that a large percentage of mutation operations does not alter the phenotype of the controllers. As mentioned earlier, neutral mutation is beneficial for the evolution and may have contributed to overall best performance by the modular algorithms. Also, similar to the rest of the genes, the regulatory gene undergoes genetic operations such as mutations. Thus any effect of a genetic operation to the chromosome is distributed across the genome. It is therefore likely that neutral mutation was larger in mgGEP-RG since it uses a longer chromosome than the rest of the controllers.

5.4 Evolving in 3D environments

The results reported in the previous sections suggest that mgGEP-RG and mgGEP-LE are suitable algorithms that can be used to generate robot controllers automatically. The results show that these two new algorithms replicated and in some environments performed better that the human behaviour sub-divided mgGEP. Following this improved performance, the experiments were extended to 3D environments. The experimentations

in this section replicates similar experiments reported in section 4.4. However, in the following experiments, the mgGEP-LE and mgGEP-RG algorithms were used to evolve the controllers. Comparisons were made with the results achieved using the standard mgGEP algorithm.

5.4.1 Experimental set up

As reported in section 5.1, the mgGEP-LE algorithm was provided with a linking set comprising of IF, OR and AND functions. The IF linker is set up using similar condition as described in section 4.4. Similar to the experiments in section 5.1, OR and AND linkers use the output of the expression of the first gene to regulate which of the two genes to effect motor control. In the implementation reported here, the regulation is based on whether the output of the expression of the first gene is less or equal to 0.5.

Algorithm 2 Sub-controller selection using mgGEP-RG in the wall following problem

Require: regulatory gene ⇒ gene1
Require: Structural genes ⇒ gene2, gene3
determineController← Translate (gene1) {*The variable "determineController" is the string symbol output from the translation of the regulatory gene*}
if ('F' ⊂ determineController) ‖ (determineController = 'L') ‖ (determineController = '0.1') ‖ (determineController = '0.2') ‖ (determineController = '0.3') **then**
motorEffectControl ← Execute (gene2)
{ *The variable "motorEffectControl" gets the output of the execution of a structural gene and convert it to a robotic action*}
else
motorEffectControl ← Execute (gene3)
end if

Algorithm 2 shows how the structural genes were selected to effect motor control. As shown, the mgGEP-RG uses the first gene (regulatory gene) to determine which of the other two genes (structural genes) effects motor control on the robot. The current implementation is symbol based, such that the result of translation of the regulatory gene is a string symbol from the terminal set.

In the implementation reported here, all the variant GEP algorithms used the parameters listed in section 4.4. However, three constants $0.4, 0.5, 0.6$ were added to the terminal set to ensure that the algorithms had sufficient capability to evolve suitable controllers. Note that the robotic sensors returns values between $0.0m$ and $1.5m$ and hence the algorithms require sufficient parameters to develop obstacle avoidance capability.

The GEP algorithms were run using an initial population of 500 random controllers with each run lasting for 200 generations. To ensure that results were not due to random effects, 20 randomly seeded algorithm runs were conducted for each algorithm. Also, as earlier reported, each controller was tested from 10 different starting points. Equation 4.2 was used to evaluate the controllers and maximum fitness was set as reported on Table 4.1. Table 5.12 shows a summary of the algorithm parameters used.

Table 5.12: Algorithm parameter settings

Parameters	mgGEP	mgGEP-LE	mgGEP-RG
Maximum generations	500	500	500
Population	500	500	500
No. of Genes	2	2	3
Head size	4	4	4
Parent organisms	2	2	2
Mutation probability	0.041	0.041	0.027
1-Point Recombination probability	0.7	0.7	0.7
2-Point Recombination probability	0.2	0.2	0.2
Gene Recombination probability	0.1	0.1	0.1
IS Transposition probability	0.1	0.1	0.1
RIS Transposition probability	0.1	0.1	0.1
Gene Transposition probability	0.1	0.1	0.1
Selection range	5%	5%	5%
Functions(IFTLE)	1	1	1
Linking set(If, And, Or)	–	3	–
Terminals(8 Sensor/Motors, 6 constants)	14	14	14
No. of starting points	10	10	10
No. of randomly seeded runs	20	20	20

5.4.2 Results and Discussion

Figure 5.26 shows average performance of the best individual in the population across rooms 2-5. From these results, the standard mgGEP outperformed both mgGEP-LE and mgGEP-RG. As previously mentioned, the behaviour sub-division used in the standard mgGEP was defined *a priori* and can be described to approximate human intelligence. The performance of mgGEP-LE on the other hand has been lowered by the use of some functions in the linking set that did not scale to the presented problem. As shown by Figure 5.27, when mgGEP-LE is run using IF & AND linkers, **mgGEP-LE(IF & AND)**, the algorithm performance is lowered. Similarly, results obtained using the three linkers together, **mgGEP-LE**, suggests that the combination of the linkers did not provide sufficient capabilities to solve the problem. Therefore, these results show that careful consideration has to be undertaken before placing functions in the linking set as this has potential to affect the performance of the algorithm.

Figure 5.28 shows that the standard mgGEP had a better performance across the four rooms than mgGEP-LE and mgGEP-RG. As mentioned earlier, the good performance can be attributed to the already predefined behaviour sub-division. The performance of the mgGEP-LE can be attributed to the lack of sufficient capability to solve the problem due to the provided linking functions as shown by figure 5.27. The mgGEP-RG had lower performance than mgGEP and mgGEP-LE in rooms 1 and 2, outperformed mgGEP-LE in room 4 and performed equally as well as mgGEP-LE in room 5. Since the mgGEP

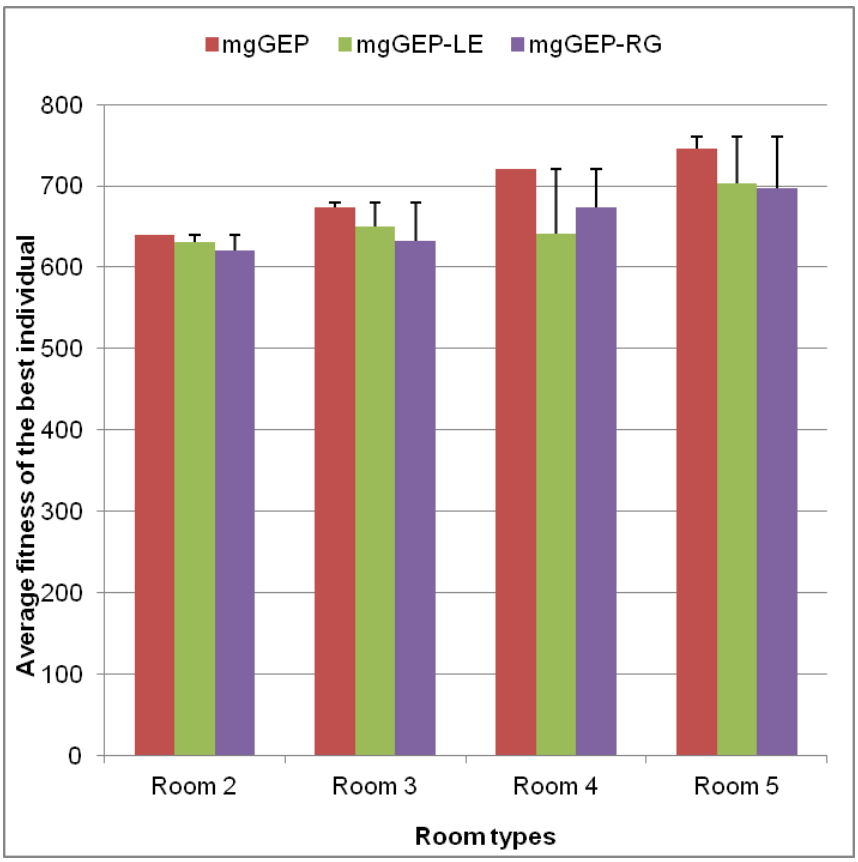


Figure 5.26: Average performance of the best individual in the population across rooms 2-5 evolved using mgGEP, mgGEP-LE and mgGEP-RG. The top of the error bars shows the maximum fitness that can be achieved. The average fitness has been computed from the best individual fitness achieved in each generation in all the 20 randomly seeded algorithms. In each evolutionary run, a population of 500 organisms was evolved for 500 generations and the best individual fitness recorded in each generation. The rest of the parameters are as shown on Table 5.12.

achieved 100% success rates in room 2 and room 4 and more than 95% in rooms 3 and 5, the progression of best individual fitness for mgGEP-RG appears to be lower in comparison. However, as shown by figure 5.26 the overall average best fitness of the best individual evolved using mgGEP-RG was only marginally lower than the expected fitness. Moreover, the progression of best individual fitness for mgGEP-RG as shown by figure 5.28 appears to be on a continuous upward curve. This upward projection suggests that the algorithm may not have converged. The lack of convergence, as discussed in section 5.3.3, implies that longer generational run may have been required in order to solve the problem fully.

In addition to the mgGEP, mgGEP-LE and mgGEP-RG, a different implementation of the standard mgGEP was used. In this new implementation, the IF linker was implemented to use the output of the left sensor to determine which of the two genes effected robot motor control. That is, if the left sensor value is less than 0.5 then gene 1 effects motor control otherwise the second gene effects robot motor control. Apart from the obvious difference from the standard mgGEP, this new implementation ensured that the robot used gene 1 when moving along side the walls and developed new capability for obstacle avoidance. For description purposes, this new implementation is referred as **mgGEP(L)**.

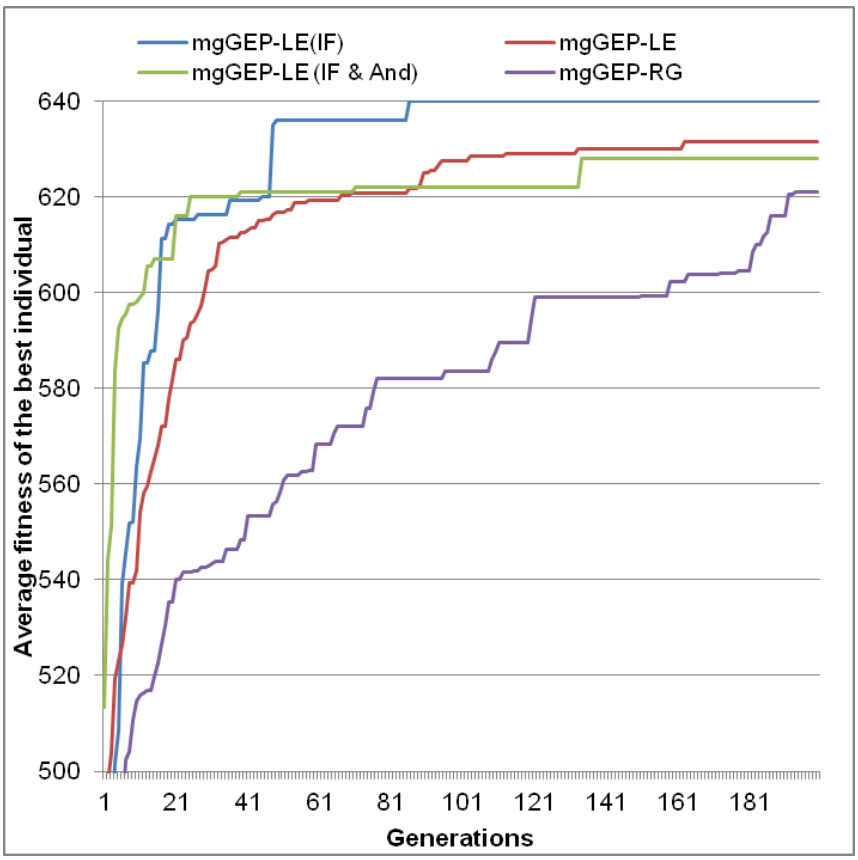


Figure 5.27: Average performance of the best individual in the population evolved using mgGEP-RG and different variations of mgGEP-LE algorithm. These experiments were conducted in room 2 using the parameters shown on Table 5.12. The average fitness has been computed from the best individual fitness achieved in each generation in all the 20 randomly seeded algorithms. In each evolutionary run, a population of 500 organisms was evolved for 500 generations and the best individual fitness recorded in each generation.

The performance of the new implementation was compared to the performance of the standard mgGEP and mgGEP-RG. Additionally, the three mgGEP variants were run for 500 generations in order to test whether mgGEP-RG required longer generation runs in order to converge. Figures 5.29, 5.30 shows the overall performance of the algorithms.

As shown by figure 5.29, the performance of mgGEP-RG improved significantly when the number of generations in the run were increased. This performance strengthens the observations made in figure 5.28, where the low performance of mgGEP-RG was attributed to small number of generations. The mgGEP-RG performance is also shown to outperform mgGEP(L), this shows that if the designer does not know which of the sensors or conditions to use for behaviour selection mechanism, then there is a high likelihood of developing controllers that do not scale to the problem. The use of mgGEP-RG to evolve conditions and develop modular controllers is thus a better technique than manual trial and error. The results shown by Figure 5.29 suggests that the mgGEP-RG requires a longer generation run in order to converge to the target solution. Although longer evolutionary runs increases computational and time overheads, the benefits of evolving modular architecture outweighs the costs. For instance, if a particular new robot controller is required, it would likely require a huge amount of time to develop using trial and error techniques such as

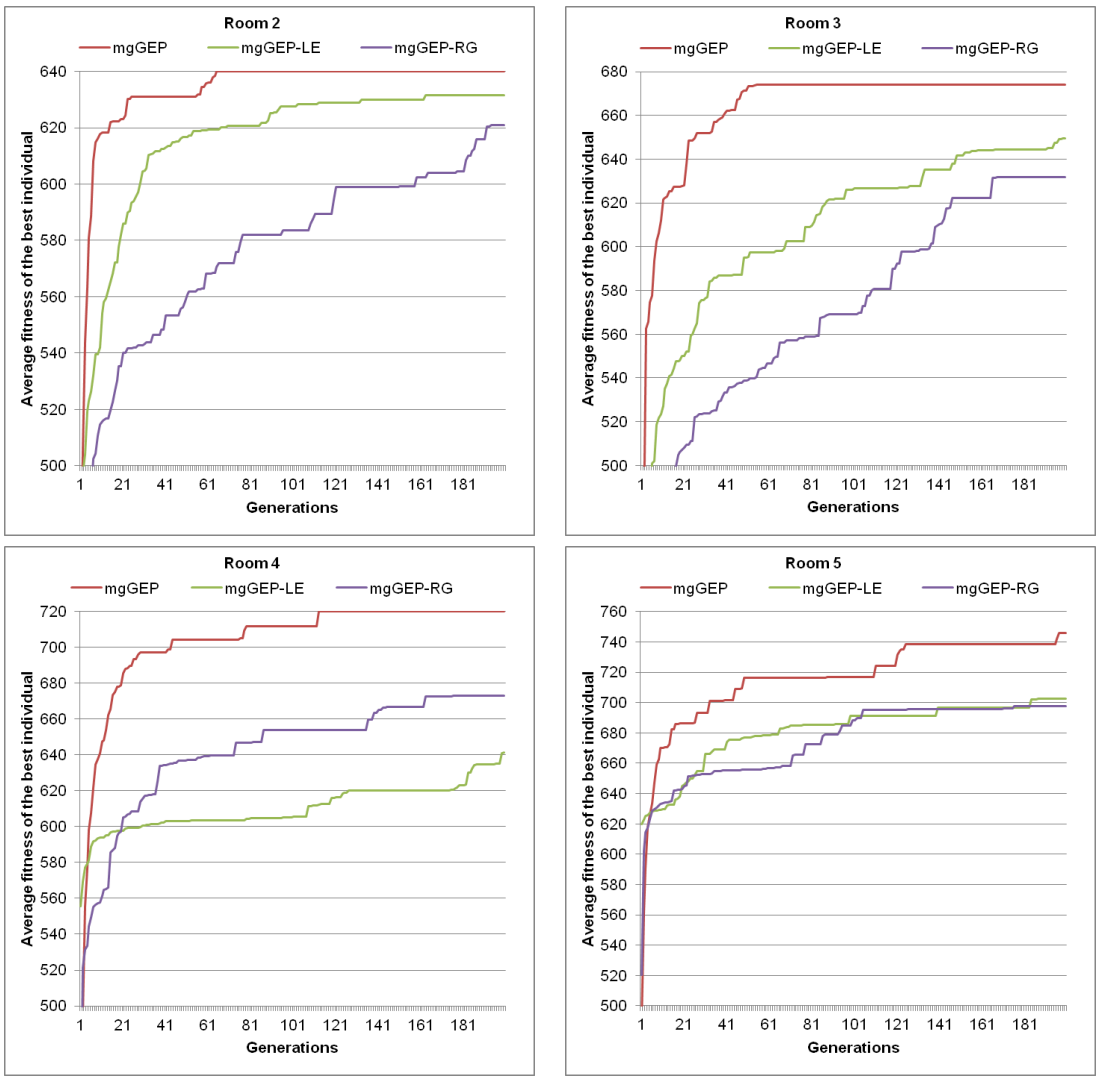


Figure 5.28: Comparison of progression of the fitness of the best individual fitness in the population as achieved through mgGEP, mgGEP-RG and mgGEP-LE in the different room types. The average fitness has been computed from the best individual fitness achieved in each generation in all the 20 randomly seeded algorithms. In each evolutionary run, a population of 500 organisms was evolved for 500 generations and the best individual fitness recorded in each generation. The rest of the parameters are as shown on Table 5.12.

subsumption architecture. Moreover, divide and conquer techniques such as layered learning and incremental evolution may lead to very good sub-behaviours but fail to develop the target global behaviour. Additionally, even when viable sub-behaviours are developed, the developer has to design a suitable behaviour coordination mechanisms. The mgGEP-RG performance as shown in this example may require longer generations but it is capable of developing viable controllers with near-equal success rates to human designed behaviour sub-division strategies (see Figure 5.30 for success rates).

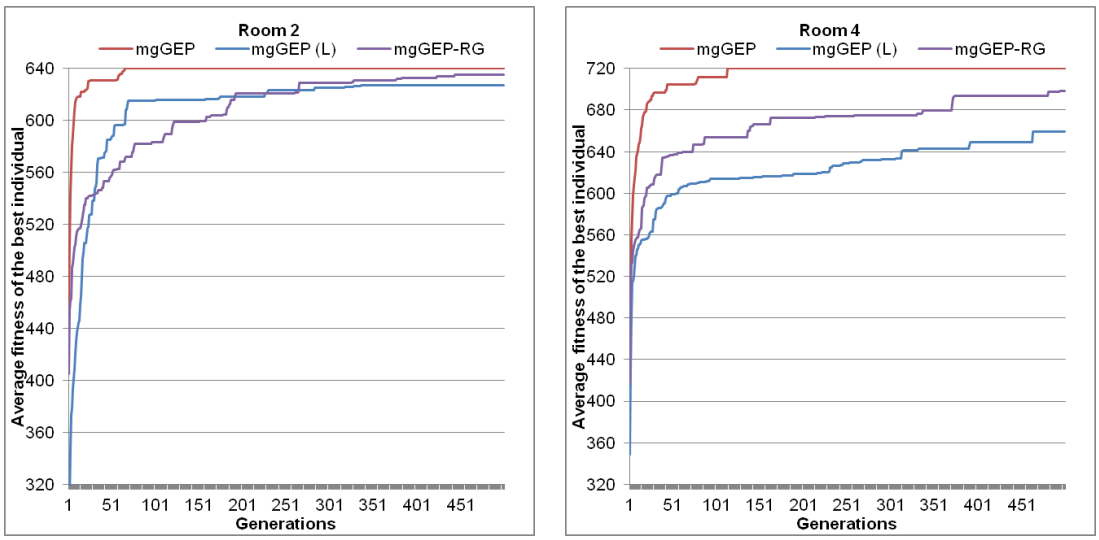


Figure 5.29: Progression of the best individual in the population as achieved through mgGEP, mgGEP(L) and mgGEP-RG in room 2 and 4. The average fitness has been computed from the best individual fitness achieved in each generation in all the 20 randomly seeded algorithms. In each evolutionary run, a population of 500 organisms was evolved for 500 generations and the best individual fitness recorded in each generation. The rest of the parameters are as shown on Table 5.12.

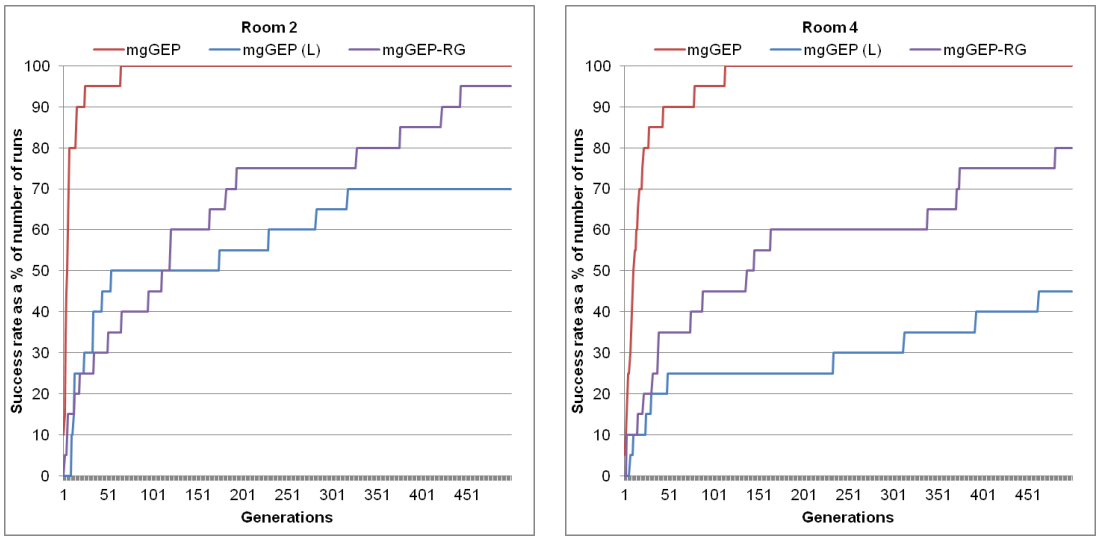


Figure 5.30: Progression of success rates with number of generations using mgGEP, mgGEP(L) and mgGEP-RG. The success rate refers to the percentage of the total number of runs where the target fitness was achieved. Each evolutionary run had a population of 500 organisms and lasted for 500 generations. In all the experiments, 20 randomly seeded algorithm runs were conducted. The rest of the algorithm parameters are as shown on Table 5.12.

5.5 Conclusion

This chapter introduces two new algorithms, as more biologically plausible alternatives to the standard mgGEP as discussed in [38, 95]. The two algorithms, mgGEP-LE and mgGEP-RG, have been implemented and used to solve a wall following problem in discrete

2D environments and then introduced into 3D environments where a robot wall following was implemented. The experimental results suggests that GEP can efficiently evolve modularity by incorporating either a regulatory gene or by evolving the linker that determines which gene to express.

The experimental results suggests that mgGEP-LE is a suitable technique to use in this problem domain. However, as shown by the results, the performance of the algorithm is affected by the type of linking functions in the linking set. If the available linking functions do not scale to the problem, then the mgGEP-LE performance is affected greatly. As such, care has to be taken before deciding which functions to include in the linking set. Nevertheless, as shown by the experimental results, the new technique has a huge potential in solving problems where various behaviour/action selection strategies are present but the designer does not know which of the available strategies is the best for a particular problem. Different linking functions, just like any function in the function set, have different arities. The use of linking functions with different arities can be utilised to evolve chromosomes with variable lengths. Further research could be carried out using this technique to not only evolve robotic controllers but also solve mathematical problems such as symbolic regression.

The mgGEP-RG technique has been shown to perform equally as well as the human designed mgGEP. Additionally, this technique was shown to evolve controllers that the human designer could possibly not have thought of, or that would be hard to implement. The mgGEP-RG uses a more biologically plausible technique not only to solve the presented problems but also to evolve the conditions that determine how sub-behaviour division should occur. This technique removes the need for the designer to specify behaviour modularity as well as design a specific behaviour/action selection strategy. However, the results show that mgGEP-RG requires longer generational runs in order to converge to the required solution. Given the amount of work that a designer may need to carry out in using a trial and error technique to develop a robot controller, the experimental results shown in this chapter suggest that the potential offered by mgGEP-RG algorithm outweighs the time overhead. Further work could be carried out with longer runs, such as 1000 generations, to determine the performance of mgGEP-RG.

6 Conclusions and Further Work

The research presented in this thesis investigates whether the capabilities of gene expression programming (GEP), as described in [38], can be extended to the field of evolutionary robotics (ER). The conducted research utilises well known robot problems to illustrate the capabilities of GEP in the evolutionary robotics domain. Additionally, two new evolutionary techniques based on multiple genes GEP (mgGEP) are introduced and implemented for robotic tasks.

This chapter summarises the main results of the presented research and how they contribute to answer the research questions posed in the introductory chapter.

6.1 GEP for automatic development of robot controllers

The research presented in Chapter 3, introduces gene expression programming (GEP) to the exciting field of evolutionary robotics. The presented work utilises a monolithic evolution approach to investigate whether GEP is a feasible algorithm for ER problems. In the described work, GEP is implemented to evolve obstacle avoidance behaviour in simulated 3D environments. The obtained results show that the controllers evolved using GEP were able to solve the problem with satisfactory results. Further experiments showed that the evolved controllers performed satisfactorily when introduced to new environments. Additionally it was shown that controllers with shorter ORFs/phenotypes adapted better in the environment than controllers with longer phenotypes. The explanation with regard to the performance of shorter phenotypes is that they are less likely to overfit when introduced into a new environment.

In addition to the unigenic GEP approach, work presented in Chapter 3 show that GEP can be implemented to evolve monolithic robot controllers using multiple genes. The obtained results show that this novel technique of evolving two genes in parallel to control a robot, generates more successful robot controllers than the standard monolithic approach. The good performance shown by the multiple-output controllers is likely to have been as a result of a "divide and conquer" approach where each gene was specialised for the control of one robot motor. Moreover, utilising one gene for a particular motor lowered the search space leading to quicker evolution speeds. In comparison to the unigenic approach, the results obtained from the multiple-output GEP algorithm showed that the mgGEP-multiple output controllers were more likely to perform well in new environment than the unigenic GEP controllers. This, as mentioned earlier, is likely to be as a result of specialisation within the sub-controllers/genes.

The capabilities of GEP as outlined by [38, 43, 93, 95] goes beyond the use of one gene (ugGEP) to code for GEP programs. This means that GEP can be extended from a monolithic approach towards modular evolution. Subsequently, the mechanism to use multiple genes (mgGEP) to encode GEP programs has been implemented to evolve robot wall following and foraging behaviours. The approach, presented in Chapter 4, is that of manual division of the genes functionality using a linking function specifically designed for the task. The results obtained using this approach show that modular systems evolved better performing controllers than the monolithic ones. The results further suggest that the capability of GEP to code for multiple genes, offers a simple yet effective approach to develop modular robot controllers. The good performance shown by the mgGEP is likely to be as a result of reduction in search space and the implemented "divide and conquer" approach.

Results obtained using robot foraging behaviour, show that behaviour coordination mechanisms can affect the ability of a modular controller to solve a problem successfully. This finding is in line with most research in this area [4, 18, 92, 117, 142]. Following this, two new mechanisms; multigenic GEP with Linker Evolution (mgGEP-LE) and multigenic GEP with Regulatory Gene (mgGEP-RG), were implemented and used for evolution of robot controllers. The results obtained from these two approaches are shown to approximate results from the human guided mgGEP. These results suggests that GEP not only offers capabilities to evolve modular controllers but also a more biologically plausible approach to implement action selection mechanisms.

In summary, the results obtained in Chapter 3 and 4 suggests that GEP algorithm is a suitable and effective technique that can be utilised in evolutionary robotics. Additionally, the experiments shows that GEP can be implemented either as a unigenic organism or using multiple genes. In the presented robot problems, the obtained results show that modular GEP controllers perform better than unigenic controllers. The modular controllers offer more biologically plausible capabilities in problem solving as well as reducing the search space through behaviour sub-division. Thus, research in Chapter 3 helps to answer the first two research questions posed in section 1.1.1 as well as support the first claim in section 1.1.3. Similarly, research in Chapter 4 contributes an answer to the third research question as well as a support for the second research claim.

6.2 GEP for automatic development of modular robot controllers

The subsumption architecture provides a mechanism for the designers of robotic controllers to code for different behaviours within a single robot. This enables the robot to undertake a range of behaviours from simple self-preservation (obstacle avoidance, battery charging) through to goal directed behaviours (exploration and goal finding). Traditionally, the delineation between behaviours has been hand-coded by human designers. As discussed previously, work in ER utilising subsumption architecture does not precisely delineate sub-behaviours into different modules, instead an overall emergent behaviour is evolved that

is made up of various tasks. Incremental learning on the other hand, evolves behaviours incrementally with the controllers having performed well in one task, evolved further to perform a different task. As show in [5, 55, 56], for the successful evolution of a particular tasks, the different fitness functions used should be closely related; with fitness functions used in lower layers being a function of the higher layers fitness functions. Again, when the controller is analysed, the behaviour is more of an emergent behaviour rather than a sum of separate modules. Thus, the evaluated ER implementation use only one layer and does not model subsumption architecture in its intended form.

As reported in section 2.5.2, layered learning approaches [117, 136] involves "freezing" the learning of the lower layers once a set fitness has been achieved. This means that the robot stops learning any more about self-preserving behaviours and concentrates on higher layers such as goal finding. Since an autonomous agent works in an unstructured environment, life-time learning is important in order to continue performing efficiently. There is thus a need for algorithms that offer concurrent multi-layer learning. To solve this problem, the mgGEP-LE and mgGEP-RG algorithms were introduced in Chapter 5. These two algorithms offers the capability to determine the division between behaviours automatically, allowing the controller to self-organise according to the problem at hand. Analysing the evolved robot controllers suggests that both mgGEP-LE and mgGEP-RG implements a behaviour sub-division which would appear logical from a human perspective, e.g. division into obstacle avoidance and exploratory behaviours. The various advantages offered by layered learning are carried over to mgGEP-LE and mgGEP-RG algorithms as discussed below:

- Task decomposition: Monolithic evolution (one layer with one fitness function), the widely used evolutionary mechanism in evolutionary robotics, offers an automatic method to generate robotic controllers. However, with a difficult task, a better way of solving the problem would be to decompose the problem into simpler tasks and solve them with increasing complexity. This is what incremental evolution, layered learning and the proposed modularised evolution offers.

- Search dimensionality: By dividing the problem into different tasks, each in its own layer, the algorithm only has to search a small space for each particular task; unlike in incremental evolution where genome size has to be maintained as there is no particular difference within layers, i.e. only one layer is available and fitness function in higher layers has to include the fitness function in a lower layer. This decrease in search dimensionality is shared with layered evolution. As shown in Chapter 5, controllers evolved using mgGEP were much shorter than those evolved using ugGEP.

- Automatic behaviour sub-division: The approach proposed through mgGEP-RG offers a technique where behaviour sub-division can be accomplished automatically. In incremental learning, layered evolution and the standard mgGEP, behaviour sub-division is accomplished manually by the designer specifying which module performs a particular task. However, in the proposed mgGEP-RG technique, a regulator gene determines how behaviour sub-division occurs. This is of great benefit to the devel-

opment of robot controllers: There is no time spent evaluating the best behaviour coordination mechanism, evolved controllers are not restricted to a particular rule and there is high likelihood of development of robust controllers. Additionally, a robot control program requiring a large number of modules is likely to be evolved with more ease using mgGEP-RG than mechanisms requiring manual behaviour sub-division.

- Symbiosis modelling: Togelius [136] argues that layered evolution can be interpreted as a symbiotic relationship between the different behaviours. This is also true for the mgGEP, mgGEP-LE and mgGEP-RG approaches. For the effective control of a robot, the various sub-controllers have to co-exist and cooperate in order to achieve the set objective, for instance the evolved avoidance behaviour has to co-exist with exploration with wall proximity each aiding the other to achieve the overall required behaviour.

- Biological plausibility: The GEP algorithm in general, offers a more biological plausible approach to evolution than both GAs and GP. This is shown by:

  Genotype to Phenotype mapping: Unlike GA and GP, GEP individuals have a distinctive delineation of genotypes and phenotypes. Similar to natural biology, the genotypes undergo genetic operations while the phenotypes are subject of selection. Also, during the course of evolution the phenotypes vary in size and shape while the genotypes remain as fixed length structures.

  Open reading frames (ORF): The GEP genes contains a coding region of the genome, known as ORF, and non-coding region. As mentioned in section 2.3.5, the only difference between the GEP ORFs and similar structures in natural biological is that GEP ORFs have non-coding regions running only downstream while in natural biology the ORFs have non-coding regions upstream and downstream. The non-coding regions allow GEP phenotypes to increase or reduce their size without affecting the length of the chromosome.

  Multiple gene chromosomes: Similar to chromosomes in natural biology, the GEP chromosomes can be formed using more than one gene. As the results show, these multiple genes chromosomes evolves better controllers than the monolithic ones.

  The mgGEP-LE and mgGEP-RG approaches offers concurrent evolution of sub-controllers as well as a behaviour organiser. This is more biologically plausible than a layered evolution where learning in lower behaviours is stopped when a set fitness is achieved. Also, mechanisms utilising mgGEP-RG replicates the working of a cell in a biological organism, albeit in a very simple way. The obtained results suggest that these mechanisms are as effective as mechanisms utilising human expertise.

The experimental results and the subsequent discussion in Chapter 5, together with the discussion in this section contributes an answer to the fourth research question (for research questions, please see section 1.1.1). The use of mgGEP-LE to evolve linking functions means that the algorithm has the capability to evolve the best behaviour coordination mechanism available in the linking set. This is more natural than providing only one

behaviour coordination as is common in standard mgGEP or layered learning techniques. However, as previously mentioned, care has to be taken to ensure that relevant linking functions are provided in the linking set. Similarly, mgGEP-RG provides more biologically plausible mechanisms to ER problems. By replicating the working of a biological cell, the algorithm offers a mechanism where a developer needs to only decide on number of sub-tasks and the algorithm generates not only the modular behaviour but also a coordination mechanism.

The results obtained in Chapter 4 showed that a linking function could be used to co-ordinate which of the two genes would effect motor control. Similarly, the results of mgGEP-LE in Chapter 5 showed that these linking functions could be grouped together in a linking set and evolved with the rest of the genome. In addition, the implementation of the mgGEP-RG and subsequent results in Chapter 5, shows that one gene in a multi-gene chromosome can be utilised to evolve the conditions under which gene regulation occurs. Therefore, the algorithm implementations and results obtained in Chapters 4 and 5 supports the third research claim as listed in section 1.1.3.

6.3 Further work

This sections highlights further research work that can be conducted using the techniques described in this thesis.

6.3.1 Simulator to on-board evolution

As mentioned above, research in this thesis was carried out in simple robot simulators. As a result, further work would be to carry out more investigations using a robotic simulator that offers more complex robotic capabilities such as vision, touch and communication as well as simulator to real robot transfer capabilities. A robotic simulator that offers more complex robot capabilities would act as a good test bed to evolve behaviours that require many modules. Also controllers developed in such a simulator can then be tested in a situated robot. The success of simulator to real robot transfer could be further extended to on-board development.

6.3.2 RoboCup

RoboCup [1] is an international robot competition where teams of soccer playing robots compete against each other. The goal is to develop a team of soccer playing humanoid robots that will be able to win a match against the winner of the most recent World Cup, following FIFA rules by the year 2050. Thus the research in RoboCup is multi-agent oriented. However, in a game of soccer an individual is required to perform numerous tasks. These include; ball kicking, ball searching, communication with team mates, identifying opponents, running, walking, obstacle avoidance, goal saving and many more. This means

[1] More information regarding Robocup can be found on `http://www.robocup.org/`

that a potential control program for a soccer playing robot has to be modular in nature. The task of developing a modular control program for a soccer playing robot is likely to be tedious and require the developer to understand soccer rules and anticipate all the scenarios that a robot may encounter.

The multigenic GEP with regulator gene (mgGEP-RG) algorithm, described in Chapter 5, offers the capabilities to evolve modular controllers without any human expertise. For a soccer playing behaviour, the mgGEP-RG algorithm could offer great benefits for a developer. Firstly, the developer does not need to program particular functionalities of a particular module, instead the developer needs to only supply sufficient number of genes to the algorithm and an additional gene that evolves the conditions for behaviour coordination. Secondly, only one fitness function needs to be formulated; this would reduce the time taken for behaviour evaluation. Finally, unlike other approaches where one module is developed and then tested before another module can be developed, the mgGEP-RG approach offers concurrent development. Thus, reducing the overall development time.

Further, the project can be extended to evolve multi-agents using co-evolution techniques.

6.3.3 Complex robots

Complex robots such as humanoids and autonomous ground vehicles require the use of numerous sensors and actuators in order to meet their objectives. Additionally, in order to display a certain behaviour a complex robot requires to perform multiple tasks. For instance, a walking behaviour in a humanoid may require the robot to; bend the knee, lift its leg, move the leg and then step on the ground. To accomplish any of the behaviours, complex robots need to use modular controllers. The proposed mgGEP-RG technique can be used in further work involving complex robots. As the results presented in this thesis show, the mgGEP-RG algorithm is likely to evolve robust controllers so long as sufficient number of genes are supplied and enough evolutionary duration.

Bibliography

[1] J. Abraham, N. Nedjah, and L. de Macedo Mourelle. Evolutionary Computation: From Genetic Algorithms to Genetic Programming. In A. Abraham, N. Nedjah, and L. de Macedo Mourelle, editors, *Genetic Systems Programming*, volume 13 of *Studies in Computational Intelligence*, pages 1–20. Springer Berlin / Heidelberg, 2006.

[2] I. Al-Taharwa, A. Sheta, and M. Al-Weshah. A Mobile Robot Path Planning Using Genetic Algorithm in Static Environment. *J. Computer Science*, 4:341–344, 2008.

[3] R. C. Arkin. Motor Schema-Based Mobile Robot Navigation. *International Journal of Robot Automation*, 8:92–112, 1989.

[4] R. C. Arkin. *Behavior-Based Robotics*. MIT Press, Cambridge, MA, USA, 1st edition, 1998.

[5] D. Bajaj and H. A. Marcelo. An Incremental Approach in Evolving Robot Behaviour. In *Proceedings of the Sixth International Conference on control, Automation, Robotics and Vision*. IEEE, 2000.

[6] A. Baker. Simplicity. In E. N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*. `http://plato.stanford.edu/archives/sum2011/entries/simplicity/`, summer 2011 edition, 2011.

[7] J. E. Baker. Reducing Bias and Inefficiency in The Selection Algorithm. In J. J Grefenstette, editor, *Proceedings of the Second International Conference on Genetic Algorithms and their Application*, pages 14–21, Hillsdale, New Jersey, 1987.

[8] D. Baldassarre, S. Nolfi, and D. Parisi. Evolving Mobile Robots Able to Display Collective Behaviours. *J. Artificial Life*, 9:255–267, 2002.

[9] A. G. Barto, R. S. Sutton, and C. J. C. H. Watkins. *Learning and Sequential Decision Making*, pages 497–537. MIT Press, 1990.

[10] S. M. Best and P. T. Cox. Programming an Autonomous Robot Controller by Demonstration Using Artificial Neural Networks. In *Proceedings of the IEEE Symposium on Visual Languages and Human Centric Computing*, pages 157–159, 2004.

[11] H-G. Beyer and H-P. Schwefel. Evolution Strategies: A Comprehensive Introduction. *J. Natural Computing*, 1:3–52, 2002.

[12] J. Billingsley, D. Oetomo, and J. Reid. Agricultural Robotics [TC Spotlight]. *J. Robotics Automation Magazine, IEEE*, 16(4):16–19, 2009.

[13] J. Blynel and D. Floreano. Exploring the T-Maze: Evolving Learning-Like Robot Behaviours Using CTRNNs. In *Proceedings of the 2003 international conference on Applications of evolutionary computing*, pages 593–604. Springer-Verlag, 2003.

[14] M. Botros. Evolving Complex Robotic Behaviors Using Genetic Programming. In A. Abraham, N. Nedjah, and L. de Macedo Mourelle, editors, *Genetic Systems Programming*, volume 13 of *Studies in Computational Intelligence*, pages 173–191. Springer Berlin / Heidelberg, 2006.

[15] M. Brameier. *On Linear Genetic Programming*. PhD thesis, University of Dortmund, 2004.

[16] C. Brauers, M. Dombrowski, H. Surmann, R. Worst, T. Linder, and J. Winzer. The RescueBot - A new Variant of the VolksBot. In *Proceedings of the International Conference on Simulation, Modeling and Programming for Autonomous Robots*, pages 296–303, 2010.

[17] R. A. Brooks. A Robust Layered Control System for a Mobile Robot. *IEEE Journal of Robotics and Automation*, 2(1):14–23, 1986.

[18] R. A. Brooks. Intelligence Without Representation. *J. Artificial Intelligence*, 47: 139–159, 1991.

[19] H. Burchardt and R. Salomon. Implementation of Path Planning Using Genetic Algorithms on Mobile Robots. In *Proceedings of the IEEE Congress on Evolutionary Computation*, pages 1831–1836, 2006.

[20] A. Cangelosi and J.L. Elman. Gene Regulation and Biological Development in Neural Networks: An Exploratory Model. Technical report, Institute of Psychology, CNR, viale Marx 15, Rome, 1995.

[21] A. Cangelosi, S. Nolfi, and D. Parisi. Artificial Life Models of Neural Development. In S. Kumar and P.J. Bentley, editors, *On Growth, Form and Computers*, pages 339–352. Elsevier Academic Press, 2003.

[22] G. Capi and K. Doya. Evolution of Neural Architecture Fitting Environmental Dynamics. *J. International Society for Adaptive Behaviour*, 13(1):53–66, 2005.

[23] E. G. Carrano, C. M. Fonseca, R. H. C. Takahashi, L. C. A. Pimenta, and O. M. Neto. A Preliminary Comparison of Tree Encoding Schemes for Evolutionary Algorithms. In *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics (SMC 2007)*, pages 1969–1974, 2007.

[24] D. J. Chalmers. The Evolution of Learning: An Experiment in Genetic Connectionism. In D. S. Touretzky, J. Elman, T.J. Sejnowski, and G.E. Hinton, editors, *Proceedings of the 1990 Summer School on Connectionist Models*, pages 81–90, San Francisco, CA, 1990.

[25] A. Cherubini, F. Giannone, and L. Iocchi. Layered Learning for a Soccer Legged Robot Helped with a 3d Simulator. In U. Visser, F. Ribeiro, T. Ohashi, and F. Dellaert, editors, *RoboCup 2007: Robot Soccer World Cup XI*, pages 385–392. Springer-Verlag, 2008.

[26] A. L. Christensen and D. Marco. Evolving an Integrated Phototaxis and Hole-Avoidance Behavior for a Swarm-bot. In *Proceedings of the 10th International Conference on the Simulation and Synthesis of Living Systems (Alife X)*, pages 248–254. MIT Press, 2006.

[27] S. Clancy and W. Brown. Translation:DNA to mRNA to Protein. *Nature Education*, 1(1), 2008.

[28] D. Cliff, P. Husbands, and I. Harvey. Explorations in Evolutionary Robotics. *J. Adaptive Behaviour*, 2:73–110, 1993.

[29] D. Cliff, P. Husbands, and I. Harvey. Seeing the Light: Artificial Evolution, Real Vision. *From Animals to Animats 3, Proceedings of the 3rd International Conference on Simulation of Adaptive Behavior*, pages 392–401, 1994.

[30] A. J. Critchlow. *Introduction to Robotics*. MacMillan Publishing Company, New York, 1985.

[31] R. Dawkins. *The Blind Watchmaker*. Penguin Books, 1986.

[32] R. Dawkins. *The Greatest Show on Earth: The Evidence of Evolution*. Bantam Press, 2009.

[33] P. Dayan and Y. Niv. Reinforcement Learning: The Good, The Bad and the Ugly. *J. Current Opinion in Neurobiology*, 18:185–196, 2008.

[34] H. De Jong. Modelling and Simulation of Genetic Regulatory Systems: A Literature Review. *J. Computational Biology*, 9(1):67–103, 2002.

[35] E. De Momi and G. Ferrigno. Robotic and Artificial Intelligence for Keyhole Neurosurgery: The ROBOCAST project, a multi-modal autonomous path planner. *J. Engineering in Medicine*, 224(5):715–727, 2010.

[36] L. Fausett. *Fundamentals of Neural Network*. Prentice-Hall, 1994.

[37] J. A. Fernandez-Leon, G. G. Acosta, and M. A. Mayosky. Behavioral Control Through Evolutionary Neurocontrollers for Autonomous Mobile Robot Navigation. *J. Robotics and Autonomous Systems*, 57:411–419, 2009.

[38] C. Ferreira. Gene Expression Programming: A New Adaptive Algorithm for Solving Problems. *J. Complex Systems*, 13(2):87–129, 2001.

[39] C. Ferreira. Genetic Representation and Genetic Neutrality in Gene Expression Programming. *J. Advances in Complex Systems*, 5(4):389–408, 2002.

[40] C. Ferreira. Analyzing the Founder Effect in Simulated Evolutionary Processes Using Gene Expression Programming. *J. Soft Computing Systems: Design, Management and Applications*, pages 153–162, 2003.

[41] C. Ferreira. Gene Expression Programming and the Evolution of Computer Programs. *J. Recent Developments in Biologically Inspired Computing*, pages 82–103, 2004.

[42] C. Ferreira. *Gene Expression Programming: Mathematical Modelling by an Artificial Intelligence (2nd edition)*. Springer, 2006.

[43] C. Ferreira. Automatically Defined Functions in Gene Expression Programming. In A. Abraham, N. Nedjah, and L. de Macedo Mourelle, editors, *Genetic Systems Programming*, volume 13 of *Studies in Computational Intelligence*, pages 21–56. Springer Berlin / Heidelberg, 2006.

[44] P. Fidelman and P. Stone. The Chin Pinch: A Case Study in Skill Learning on a Legged Robot. In *RoboCup-2006: Robot Soccer World Cup X*. Springer Verlag, 2007.

[45] D. Floreano and L. Keller. Evolution of Adaptive Behaviour in Robots by Means of Darwinian Selection. *J. PLoS Biology*, 8, 2010.

[46] D. Floreano and F. Mondada. Automatic Creation of an Autonomous Agent: Genetic Evolution of a Neural-Network Driven Robot. In *Proceedings of the Third International Conference on Simulation of Adaptive Behavior*. MIT Press-Bradford Books, Cambridge, MA, 1994.

[47] D. Floreano and F. Mondada. Evolution of Plastic Neurocontrollers for Situated Agents. In P. Maes, M. Mataric, J-A. Meyer, J. Pollack, and S. Wilson, editors, *From Animals to Animats 4, Proceedings of the 4th International Conference on Simulation of Adaptive Behavior (SAB'1996)*, pages 402–410. MA: MIT Press, 1996.

[48] D. Floreano and J. Urzelai. Evolutionary Robots with On-line Self-Organization and Behavioural Fitness. *J. Neural Networks*, 13:431–443, 2000.

[49] D. Floreano, J. Godjevac, A. Martinoli, F. Mondada, and J. D. Nicoud. Design, Control, and Applications of Autonomous Mobile Robots. In S. G. Tzafestas, editor, *Advances in Intelligent Autonomous Agents*. Kluwer Academic Publishers, 1998. Part 2, Chapter 8, p. 159–186.

[50] D. Floreano, P. Durr, and C. Mattiussi. Neuroevolution: From Architectures to Learning. *J. Evolutionary Intelligence*, 1(1):47–62, 2008.

[51] D. B. Fogel and K. Chellapilla. Revisiting Evolutionary Programming. In *Proceedings of SPIE Aerosense98*, Applications and Science of Computational Intelligence, pages 2–11, 1998.

[52] E. Galván-López, R. Poli, A. Kattan, M. O'Neill, and A. Brabazon. Neutrality in Evolutionary Algorithms...What do we know? *J. Evolving Systems*, 12:365–401, 2011.

[53] E. Gat. On Three-Layered Architectures. *J. Artificial Intelligence and Mobile Robots*, pages 195–210, 1998.

[54] D. E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning.* Addison-Wesley Longman Publishing Co., Inc., 1st edition, 1989.

[55] F. Gomez and R. Miikulainen. Incremental Evolution of Complex General Behaviour. Technical report, Technical Report AI96-248, Austin, TX: University of Texas at Austin, 1996.

[56] F. Gomez and R. Miikulainen. Solving Non-Markovian Control Tasks with Neuroevolution. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 1999.

[57] A. J. F. Griffiths, J. H. Miller, D. T. Suzuki, R. C. Lewontin, and W. M. Gelbart. *An Introduction to Genetic Analysis.* W. H. Freeman, New York, 7th edition, 2000.

[58] D. Gu and H. Hu. Evolving Fuzzy Logic Controllers for Sony Legged Robots. In A. Birk, S. Coradeschi, and S. Tadokoro, editors, *RoboCup 2001, LNAI 2377*, pages 356–361, 2001.

[59] D. Gu, H. Hu, J. Reynolds, and E. Tsang. GA-Based Learning in Behaviour Based Robotics. In *Proceedings of the IEEE International Symposium on Computational Intelligence in Robotics and Automation*, pages 16–20, 2003.

[60] I. Harvey, P. Husbands, and D. Cliff. Issues in Evolutionary Robotics. In J-A. Meyer, H. Roitblat, and S. Wilson, editors, *Proceedings of the 3rd International Conference on Simulation of Adaptive Behavior*, volume 2, pages 73–110, 1993.

[61] I. Harvey, P. Husbands, D. Cliff, A. Thompson, and N. Jakobi. Evolutionary Robotics: The Sussex Approach. *J. Robotics and Autonomous Systems*, 20:205–224, 1997.

[62] S. Haykin. *Neural Networks.* Prentice-Hall, 2 edition, 1999.

[63] A. Hosseinzadeh and H. Izadkhah. Evolutionary Approach for Mobile Robot Path Planning in Complex Environment. *International Journal of Computer Science*, 2010.

[64] C. Hsinghua, G. Premkumar, and C. Chao-Hsien. Genetic Algorithms for Communications Network Design - an empirical study of the factors that influence performance. *J. IEEE Transactions on Evolutionary Computation*, 5(3):236–249, 2001.

[65] L. Hugues and N. Bredeche. A Quick Programming Guide for Simbad Simulator., August 2005. URL `http://simbad.sourceforge.net/guide.php`.

[66] L. Hugues and N. Bredeche. Simbad: An Autonomous Robot Simulation Package for Education and Research. In *Simulation of Adaptive Behaviour*, 2007.

[67] P. Husbands, I. Harvey, D. Cliff, and G. Miller. Artificial Evolution: A New Path for Artificial Intelligence? *J. Brain and Cognition*, 34(1):130–159, 1997.

[68] C. Igel and M. Toussaint. Neutrality and Self-Adaptation. *J. Natural Computing*, 2:117–132, 2003.

[69] K. Ito. Simple Robots in a Complex World: Collaborative Exploration Behavior using Genetic Programming. In John R. Koza, editor, *Genetic Algorithms and Genetic Programming at Stanford 2003*, pages 91–99. Stanford Bookstore, 2003.

[70] Gurney K. *An Introduction to Neural Networks*. UCL Press, 1997.

[71] S. Kent. *Evolutionary Approaches to Robot Path Planning*. PhD thesis, Brunel University, 1999.

[72] H. L. Kian, K. L. Wee, and H. A. Marcelo. A Hybrid Mobile Robot Architecture with Integrated Planning and Control. In *Procedings of the Autonomous Agents and Multi-Agent Systems (AAMAS-02)*, pages 219–226. ACM Press, 2002.

[73] P. E. Kladitis. How small is too small? True Microrobots and Nanorobots for Millitary Applications in 2035. Technical report, Air commands and staff college, Air University, 2010.

[74] J. R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, 1992.

[75] J. R. Koza. Evolution of Subsumption Using Genetic Programming. In P. Bourgine F. J. Varela, editor, *Proceedings of the First European Conference on Artificial Life: Towards a Practice of Autonomous Systems*, pages 110–119, 1993.

[76] J. Dario Landa-Silva and Edmund K. Burke. *Applications of Multi-Objective Evolutionary Algorithms, Advances in Natural Computation*, volume 1, pages 727–751. World Scientific, 2004.

[77] D. K. Laszlo. Evolution of Intelligent Agents: A New Approach to Automatic Plan Design. In *Proceedings of the IFAC Workshop on Control Applications of Optimization*, pages 237–243. Elsevier, 2003.

[78] C. Lazarus and H. Hu. Using Genetic Programming to Evolve Robot Behaviours. In *Proceedings of the 3^{rd} British Conference on Autonomous Mobile Robotics and Autonomous Systems*, 2001.

[79] C. Lazarus and H. Hu. Evolving Goalkeeper Behaviours for Simulated Soccer Competition. In *Proceedings of the 3^{rd} IASTED International Conference on Artificial Intelligence and Applications*, 2003.

[80] W-P. Lee. Evolving Complex Robot Behaviors. *J. Information Sciences*, 121:1–25, 1999.

[81] S. Luke. Evolving Soccerbots: A Retrospective. In *Proceedings of the 12^{th} Annual Conference of the Japanese Society for Arti Intelligence (JSAI)*, 1998.

[82] L.T. MacNeil and A.J.M Walhout. Gene Regulatory Networks and the Role of Robustness and Stochasticity in the Control of Gene Expression. *Genome Research*, 21:645–657, 2011.

[83] M. J. Mataric. A Distributed Model for Mobile Robot Environment-Learning and Navigation. Technical report, Massachusetts Institute of Technology, Cambridge, MA, USA, 1990.

[84] C. Mautner and R. K. Belew. Evolving Robot Morphology and Control. In *Proceedings of the Artificial Life and Robotics (AROB)*, 1999.

[85] Z. Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs (2nd, extended ed.)*. Springer-Verlag New York, Inc., 1994.

[86] R. Miikkulainen. Neuroevolution. *J. Encyclopedia of Machine Learning*, pages 716–720, 2010.

[87] J. F. Miller and P. Thomson. Cartesian Genetic Programming. In *Proceedings of the Third European Conference on Genetic Programming (EuroGP2000)*, volume 1802 of *LNCS*, pages 121–132. Springer-Verlag, 2000.

[88] M. Minsky. Steps Toward Artificial Intelligence. In *Proceedings of the Institute of Radio Engineers*, pages 8–30. Institute of Radio Engineers, New York, 1961.

[89] M. Mitchell. *An Introduction to Genetic Algorithms*. MIT Press, 1998.

[90] Jean-Baptiste. Mouret and Stéphane. Doncieux. Overcoming the bootstrap problem in evolutionary robotics using behavioral diversity. In *Proceedings of the IEEE Congress on Evolutionary Computation'09*, pages 1161–1168, 2009.

[91] L. Moussi and M. K. Madrid. Simple Target Seek Based on Behavior. In *Proceedings of the 6^{th} WSEAS International Conference on Signal Processing, Robotics and Automation*, pages 133–139, 2007.

[92] R. R. Murphy. *Introduction to AI Robotics*. MIT Press, Cambridge, MA, USA, 1st edition, 2000.

[93] J. Mwaura and E. Keedwell. Adaptive Gene Expression Programming Using a Simple Feedback Heuristic. In *Proceedings of the AISB*, Edinburgh, UK, 2009.

[94] J. Mwaura and E. Keedwell. Evolution of Robotic Behaviours Using Gene Expression Programming. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC2010)*, pages 1–8, Barcelona, Spain, 2010.

[95] J. Mwaura and E. Keedwell. Evolving Modularity in Robot Behaviour Using Gene Expression Programming. In R. Gross, L. Alboul, C. Melhuish, M. Witkowski, T. J. Prescott, and J. Penders, editors, *Towards Autonomous Robotic Systems - 12th Annual Conference, (TAROS 2011), August 31 - September 2*, volume 6856 of *Lecture Notes in Computer Science*, pages 392–393, Sheffield, UK, 2011. Springer.

[96] G. Nagib and W. Gharieb. Path Planning for a Mobile Robot Using Genetic Algorithms. In *Proceedings of the International Conference on Electrical, Electronic and Computer Engineering*, pages 185–189, 2004.

[97] A. L. Nelson, E. Grant, G. J. Barlow, and T. C. Henderson. A Colony of Robots Using Vision Sensing and Evolved Neural Controllers. In *Proceedings of the IEEE/RSJ International Conference On Intelligent Robots And Systems*, volume 3, pages 2273–2278, 2003.

[98] A. L. Nelson, E. Grant, J. M. Galeotti, and S. Rhody. Maze Exploration Behaviours Using an Integrated Evolutionary Robotics Environment. *J. Robotics and Autonomous Systems*, 46(3):159–173, 2004.

[99] A. L. Nelson, G. J. Barlow, and L. Doitsidis. Fitness Functions in Evolutionary Robotics: A survey and analysis. *J. Robotics and Autonomous Systems*, 57:345–370, 2009.

[100] A.L. Nelson and E. Grant. Developmental analysis in evolutionary robotics. In *Proceedings of the 2006 IEEE SMC Mountain Workshop on Adaptive and Learning Systems (SMCals06)*, pages 201–206, 2006.

[101] R. Neruda and S. Slusny. Evolving Neural Network which Control a Robotic Agent. In *Proceedings of the IEEE Congress on Evolutionary Computationn*, 2007.

[102] N. J. Nilsson. Introduction to Machine Learning: An Early Draft of a Proposed Textbook., 1996.

[103] S. Nolfi. Evolutionary Robotics: Exploiting the full power of self organization. *J. Connection Science*, 10 (3–4):167–183, 1998.

[104] S. Nolfi. Behaviour as A Complex Adaptive System: On the role of self-organization in the development of individual and collective behaviour. *J. ComplexUs*, 2 (3–4): 195–203, 2006.

[105] S. Nolfi and D. Floreano. *Evolutionary Robotics. The Biology, Intelligence, and Technology of Self-organizing Machines.* MIT Press, 2000.

[106] P. Nordin and W. Banzhaf. Real Time Control of a Khepera Robot Using Genetic Programming. *J. Cybernetics and Control*, 26:533–561, 1997.

[107] M. Oltean. A Comparison of Several Linear Genetic Programming Techniques. *J. Complex-Systems*, 14(4):285–313, 2003.

[108] M. Oltean. Solving Even-Parity Problems Using Traceless Genetic Programming. In *Proceedings of the IEEE Congress on Evolutionary Computation*, pages 1813–1819, 2004.

[109] M. Oltean, C. Grosan, L. Diosan, and C. Mihăilă. Genetic Programming with Linear Representation, A survey. *International Journal on Artificial Intelligence Tools*, 8: 197–238, 2008.

[110] A. Oreback and H. I. Christensten. Evaluation of Architectures for Mobile Robotics. *J. Autonomous Robots*, 14:33–49, 2002.

[111] K. M. Passino and S. Yurkovich. *Fuzzy Control*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st edition, 1997.

[112] D.W. Patterson. *Artificial Neural Networks: Theory and Applications*. Prentice Hall PTR, 1st edition, 1998.

[113] V. M. Peri and D. Simon. Fuzzy Logic Control for an Autonomous Robot. In *North American Fuzzy Information Processing Society*, pages 337–342, 2005.

[114] G. M. Pierce II. Robotics: Military Applications for Special Operations Forces. Technical report, Air commands and staff college, Air University, 2000.

[115] M. L. Pilat and F. Oppacher. Robotic Control Using Hierarchical Genetic Programming. In *GECCO (2)*, volume 3103 of *Lecture Notes in Computer Science*, pages 642–653. Springer, 2004.

[116] R. Poli, W. B. Langdon, and N. F. McPhee. *A Field Guide to Genetic Programming*. Published via `http://lulu.com` and freely available at `http://www.gp-field-guide.org.uk`, 2008. With contributions by J. R. Koza.

[117] T. J. Prescott, P. Redgrave, and K. Gurney. Layered Control Architectures in Robots and Vertebrates. *J. Adaptive Behavior*, 7(1):99–127, 1999.

[118] Z. W. Pylyshyn. *The Robot's Dilemma: The Frame Problem in Artificial Intelligence*. Ablex Publishing Corporation, 1986.

[119] S. Qiao, C. Tang, J. Peng, J. Hu, and H. Zhang. BPGEP: Robot Path Planning Based on Backtracking Parallel-Chromosome GEP. In *Proceedings of the International Conference on Sensing, Computing and Automation*, 2006.

[120] A. Radi and R. Poli. Discovering Efficient Learning Rules for Feedforward Neural Networks Using Genetic Programming. In A. Abraham, L. Jain, and J. Kacprzyk, editors, *Recent Advances in Intelligent Paradigms and Applications*, Studies in Computational Intelligence, pages 133–159. Springer Berlin / Heidelberg, 2003.

[121] T. Reil. Dynamics of Gene Expression in an Artificial Genome - Implications for Biological and Artificial Ontogeny. In *Proceedings of the 5th European Conference on Advances in Artificial Life*, ECAL '99, pages 457–466. Springer-Verlag, 1999.

[122] T. Reil. Artificial Genomes as Models of Gene Regulation. In S. Kumar and P.J. Bentley, editors, *On Growth, Form and Computers*, pages 256–277. Elsevier Academic Press, 2003.

[123] A. Réka. Boolean modeling of genetic regulatory networks. *Lect. Notes Phys.*, 650: 459–481, 2004.

[124] S. J. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Pearson Education, 2 edition, 2003.

[125] J. A. Sauter and R. Matthews. Evolving Adaptive Pheromone Path Planning Mechanisms. In *Proceedings of the First International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS-02)*, pages 434–440, 2002.

[126] K. H. Sedighi, K. Ashenayi, T. W. Manikas, R. L. Wainwright, and H-W. Tain. Autonomous Local-Path Planning for a Mobile Robot Using a Genetic Algorithm. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC)*, 2004.

[127] Z. M. Skolicki. An Analysis of Island Models in Evolutionary Computation. In *Proceedings of the Genetic and Evolutionary Computation Conference - GECCO'05*, pages 386–389, 2005.

[128] W. D. Smart and L. P. Kaelbling. Effective Reinforcement Learning for Mobile Robots. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 3404–3410. IEEE, 2002.

[129] S. Srinivasa, D. Ferguson, C. Helfrich, D. Berenson, A. Collet, R. Diankov, G. Gallagher, G. Hollinger, J. Kuffner, and M. VandeWeghe. HERB: A Home Exploring Robotic Butler. *J. Autonomous Robots*, 2009.

[130] K. O. Stanley, B. D. Bryant, and R. Miikkulainen. Evolving Adaptive Neural Networks With and Without Adaptive Synapses. In *Proceedings of the 2003 Congress on Evolutionary Computation*, 2003.

[131] W-H. Steeb. *The Non-Linear Workbook (3rd Ed)*. World Scientific Publishing Co. Pte, 2005.

[132] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. MIT press, Cambridge, MA, 1998.

[133] E-G. Talbi. *Metaheuristics - From Design to Implementation*. Wiley, 2009.

[134] L. Teodorescu. Gene Expression Programming Approach to Event Selection in High Energy Physics. *J. IEEE Transactions of Nuclear Science*, 53(4), 2006.

[135] T. Thompson and J. Levine. Scaling-up Behaviours in EvoTanks: Applying Subsumption Principles to Artificial Neural Networks. In *Proceedings of the IEEE Symposium on Computational Intelligence and Games*, 2008.

[136] J. Togelius. Evolution of a Subsumption Architecture Neurocontroller. *J. Intelligent Fuzzy System*, 15:15–20, 2004.

[137] E. Tuci, M. Quinn, and I. Harvey. An Evolutionary Ecological Approach to The Study of Learning Behaviour Using a Robot-Based Model. *J. Adaptive Behaviour*, 10:201–221, 2002.

[138] J. Urzelai and D. Floreano. Incremental Evolution with Minimal Resources. In *Proceedings of the International KHEPERA Workshop*, 1999.

[139] J. Urzelai and D. Floreano. Evolution of Adaptive Synapses: Robots with Fast Adaptive Behaviour in New Environments. *J. Evolutionary Computation*, 9:495–524, 2001.

[140] A. Vaccarella, E. De Momi, and P. Cerveri. IGSTK-based Application for Neurosurgical Robotics. *J. Software Developer's Quarterly*, 2010.

[141] N. van Hoorn, J. Togelius, and J. Schmidhuber. Hierarchical Controller Learning in a First-Person Shooter. In *Proceedings of the IEEE Symposium on Computational Intelligence and Games*, pages 294–301, 2009.

[142] M. Wahde. Evolution Robotics: The Use of Artificial Evolution in Robotics, a tutorial. In *Proceedings of the IEEE/ESJ International Conference on Intelligent Robots and Systems*, 2004.

[143] L. Wang, C. K. Tan, and C. M. Chew. *Evolutionary Robotics: From Algorithms to Implemnetations*. World Scientific Pulishing Co.Pte. Ltd, 2006.

[144] B. Webb. What Does Robotics Offer Animal Behaviour? *J. Animal Behaviour*, 60: 545–558, 2000.

[145] E. Weiner. Could Robots Replace Humans in Mines? `http://www.npr.org/templates/story/story.php?storyId=12637032`, 2007. accessed on 8^{th} June 2011.

[146] D. Whitley. Genetic Algorithms and Neural Networks. *J. Genetic Algorithms in Engineering and Computer Science*, pages 191–201, 1995.

[147] H. Yamamoto. Robot Path Planning by Genetic Programming. *J. Artificial Life and Robotics*, pages 28–32, 1998.

[148] K. Yanai and H. Iba. Multi-Agent Robot Learning by Means of Genetic Programming: Solving an Escape Problem. In *Proceedings of the 4th International Conference on Evolvable Systems: From Biology to Hardware*, pages 192–203. Springer-Verlag, 2001.

[149] X. Yao. Evolving Artificial Neural Networks. *Proceedings of the IEEE*, 87(9):1423–1447, 1999.