Evolving Robot Sub-behaviour Modules using Gene Expression Programming

Jonathan Mwaura · Ed Keedwell

Abstract 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 paper, the performances of the unigenic gene expression programming (ugGEP) and multigenic GEP (mgGEP) in evolving robot controllers for a wall following robot is analysed. Additionally, the paper introduces Regulatory Multigenic Gene Expression Programming (RMGEP), a new evolutionary technique that can be utilised to automatically evolve modularity in robot behaviour. The proposed technique extends the mgGEP algorithm, by incorporating 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. In the initial experiments, the proposed algorithm is implemented for a robot wall following problem and the results compared to that of ugGEP and mgGEP. 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.

Keywords Gene expression programming · subsumption architecture · Layered learning · Evolutionary Robotics · Robot Behaviour Coordination

J. Mwaura
Department of Computer Science
University of Pretoria
Pretoria, South Africa
E-mail: jtmwaura@gmail.com

Ed. Keedwell
College of Engineering, Mathematics and Physical Sciences
University of Exeter
Exeter, United Kingdom
E-mail: e.c.keedwell@ex.ac.uk

1 Introduction

Intelligent autonomous systems (IAS) are designed "to do the right thing at the right time"[1] 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, space exploration, 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 [27, 40, 49]. The research is normally two fold: (a) development of robot morphology or body [38] and (b) development of the robot controller [48, 49]. 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 is time consuming and cannot be guaranteed to always generate a robust control system.

To counter the difficulties experienced in manual design and programming autonomous robots, evolutionary robotics (ER) has been used to generate robot controllers and occasionally robot morphology [9, 38]. These controllers are either neural-based with optimization achieved through use of an evolutionary algorithm (EA) or symbolic programs evolved using evolutionary approach [26, 50]. ER has been successful in evolving viable controllers, giving rise to increased research in this field [14–17, 20–22, 26, 28–31, 35, 43, 45, 48, 49, 57]. Programming or evolving one program requires time and careful planning in order to ensure all interlinking function or task work correctly with a view to generating the correct behaviour. Approaches to divide the tasks such that the main behaviours emerge as a collection of small tasks has been envisioned through the evolution of subsumption like architectures e.g. [2, 28, 31, 35, 54] as well as the incremental evolutionary approach [1, 17, 56] and more recently the layered learning approach [51, 55]. All these techniques involve the subdivision of tasks into various modules to solve the problem and the revision of the fitness function.

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. Thus, while the evolution of subsumption architecture, incremental evolution and layered learning approaches have a major advantage over the monolithic ER approaches, their success comes with additional overheads. Subsequently, there is a need to investigate ER mechanisms where robot behaviour sub-division and sub-behaviour coordination, can be achieved automatically. The automatic behaviour sub-division and coordination is likely to lower development time as well as lead to more robust robot controllers. The work reported here attempts to solve the problem of behaviour sub-division by firstly extending our previous work [44], where Unigenic Gene Expression Programming (ugGEP) and Multigenic Gene Expression Programming (mgGEP) algorithms were used to evolve robotic controllers for a wall following robot. Secondly, the paper aims to introduce Regulatory Multigenic Gene Expression Programming (RMGEP), a new evolutionary technique that can be utilised to automatically evolve behaviour modularity. The new technique extends mgGEP algorithm by incorporating a regulatory gene as part of the Gene Expression Programming (GEP) chromosome. In the reported

[1] http://www.ias.uwe.ac.uk/

work, the proposed algorithm is implemented for a robot wall following problem and the results compared to that of ugGEP and mgGEP. Additionally, results achieved using these three GEP variants in a robot foraging behaviour, shows that the position of a specific sub-behaviour in the sub-behaviour model is of importance to how the robot solves a task.

The rest of the paper is organised as follows: Firstly, a discussion of modular architectures in ER is presented, followed by a brief introduction to the GEP algorithm. Secondly, the RMGEP is introduced and implemented for the wall following problem in a 2D environment. The results of this new algorithm is compared to that of ugGEP and mgGEP and a discussion follows. Thirdly, a robot foraging behaviour is presented and results discussed. Finally a conclusion on the suitability of these methods in development of modular controllers is drawn.

2 Modular architectures in ER

The basic evolutionary methodology uses one module to evolve the required behaviour in its entirety. This is known as monolithic evolution. 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 the randomness of the initial (seed) population [39, 46, 47, 49]. 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. Among these mechanisms, is the process of 'divide and conquer' where the required behaviour is divided into simpler tasks. The following sub-sections briefly examines techniques employed in ER to the problem of behaviour sub-division.

2.1 Evolution of subsumption architecture

In the subsumption architecture, 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 the lower layers. Lower layers are thus used for simpler or default behaviours and the complex behaviours left for the higher layers [3, 55]. Mataric [34] has used subsumption architecture to program a controller that helps a robot to follow walls of an irregular room. In the work, a wall following behaviour is divided into four tasks, stroll (wander), avoid (avoiding obstacles), align (stay on straight path) and correct (if going out of boundary). Since the subsumption architecture is normally programmed by a human designer, then the issue of specifying a fitness function does not arise, the human programmer would just make certain control mechanism that determines when a particular tasks need to be executed. Through the integration of these simple behaviours, an emergent behaviour is achieved. In [27], Genetic Programming (GP) has been used to evolve a subsumption type controller. The evolved controller is able to perform all the required tasks and generate an emergent wall following behaviour.

2.2 Incremental evolution

Incremental evolution[1, 15, 17, 56, 58] 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.

2.3 Layered learning

Layered learning [5, 19, 20, 51, 55] approach is closely related to the subsumption architecture [34, 40]. In this approach, the targeted behaviour is divided into various tasks, then the controller is evolved in different modules sequentially (see Figure 1 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).

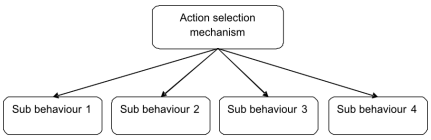


Fig. 1 A behaviour modularity model

The above approaches have been shown to generate better controllers than the canonical ER mechanism [17, 49, 55, 58]. However, there are various shortcomings with these 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 [1]. Although layered learning and subsumption architecture 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: Since multiple fitness functions require multiple simulations, this can lead to an increase in computational overhead compared to algorithms that use a single fitness

function. As a result, there is need to develop techniques that can formulate robot behaviour sub-division and coordination in a more intuitive manner. Our previous work [44] shows that GEP algorithm can be utilised to formulate sub-behaviour division resulting in more robust behaviours. The following section discuss GEP briefly.

3 Gene Expression Programming

Gene Expression Programming is a genotype/phenotype evolutionary algorithm developed in 2001 [12]. It follows genetic algorithms (GA) and GP in mimicking the Darwinian theory of evolution to solve complex problems in mathematics, computer science and related fields. GEP starts by forming linear string chromosomes representing the solutions, akin to a standard GA, and is referred as the genotype. The linear representation is formed in two domains, i.e. head and tail; the head contains both terminals and functions while the tail contains only terminals. The length of the head is normally provided as a user defined variable during a run while the length of the tail is a function of the head as shown in Equation 1:

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

Where t is the tail length, h is the head size and n is the maximum arity within the function set

After forming the linear representation the chromosome undergoes translation where the chromosome is decoded to form a coding region, (i.e. the part of the chromosome representing the region that will be used to solve the problem) and a non-coding part. The coding region is further translated into a tree-like structure similar in nature to that in GP, this is the phenotype and is expressed as a structure known as the Expression Tree (ET). The phenotype can also be expressed as a linear structure known as an Open Reading Frame (ORF), and is similar to this structure in natural biology, covering only the coding region of the genome. The concept of these coding and non-coding regions provides GEP with the capability to form correct ETs every time the genotype undergoes a genetic operation. The genotype is the subject of genetic operations while the phenotype undergoes selection.

3.1 Multigenic GEP

As with natural biology a collection of genes forms a chromosome, similarly, in GEP a chromosome/genome with more than one gene can be formed. The genes combine to form a chromosome using a specified function known as a linker. For instance a logical IF could be used to select the gene that determines the collective output or the genes could be added together to appear as one long concatenated sequence. This is called a multigenic GEP (mgGEP) chromosome and GEP derives most of its power over the other adaptive techniques through this. A chromosome with only one gene is referred as unigenic GEP (ugGEP) chromosome and is reasonably close to genetic programming in its capability [13, 41, 43].

In standard GEP, when a multigenic chromosome is used in problem solving, the number of genes and the head size is chosen. A linking function (linker) is also

chosen prior to the run. The various genes are translated to various expression trees (ET) and the genome is held together by the linker. Consider the example below:

Head size = 4, Function set (-, +, *, /), Terminal set (x,y,2,3), number of genes = 3, linking function: +. Using Equation 1 above, 3 genes could be formed as shown below.

+*xy|xy32y
-/x*|yy23x
*-x3|xxy23

The following chromosome can be formed:

+*xy|xy32y -/x*|yy23x *-x3|xxy23 *The | indicates the beginning of the head and tail portions of the genes.*

The above genes, forms individual sub-ETs and ORFs. If the addition function (+) is used as a linking function, then the ET shown by Figure 2 is formed. The translation of a gene to ET is completed by starting from the gene root node, and moving from left to right, top to bottom.

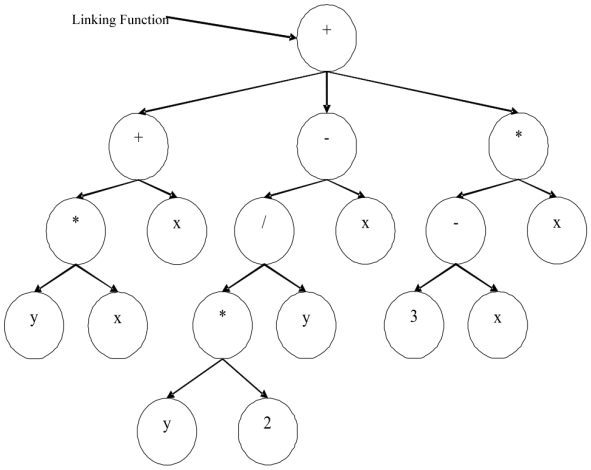


Fig. 2 Expression Tree (ET) for a multigenic GEP chromosome. The genes are joined using an addition function linker.

Using this mgGEP, behaviours can be evolved separately by each gene; the linker can then be used to decide which gene controls the motor output at any given time. This linking function defines how the genes interact with each other to evolve the targeted behaviour [12, 42]. 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). In essence, as shown by our previous work [41, 44], mgGEP has the ability to evolve modularity akin to subsumption or layered architecture. Nevertheless, 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. Thus, there is a need to develop mechanism where behaviour modularity and action selection strategies can be developed automatically.

The work presented in this paper introduces a new mechanism that can be used to evolve behaviour modularity as well as behaviour coordination. The next section introduces the new technique and its relation to mgGEP algorithm.

4 Regulatory Multigenic Gene Expression Programming

In natural organisms, the regulation of gene expression is achieved through genetic regulatory systems structured by networks between the genes, Ribonucleic Acid (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 messenger RNA (mRNA) [4, 18]. 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 [8, 33, 52]. In natural organisms, once the genes are transcribed into mRNA, the mRNA is then translated into a protein or set of proteins [6]. These proteins can either be structural and hence utilised to give particular structural properties to a cell, or 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 [52, 53]. Those proteins whose primarily role is regulation of gene activities are referred to as transcription factors [6, 52].

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, are usually interlinked using a linking function. 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 by controlling the expression of the genes within the mgGEP chromosome. The linking function and the conditions for gene interactions are provided by the designer, before the start of an evolutionary run. Whereas this method is better than monolithic approach, the main drawback to mgGEP approach is that the linking function and conditions require problem specific knowledge. In problems with multiple sub-tasks, determining the conditions for gene regulation or behaviour coordination is not easy to achieve [58]. Therefore, there is a need to develop mechanism where conditions for behaviour coordination can be achieved automatically.

The Regulatory Multigenic Gene Expression Programming (RMGEP) 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 conditions under which a particular gene or group of genes should be activated or repressed is evolved during optimisation. The regulatory gene is therefore a part of the chromosome but it is not involved in the direct control of the robot actions. Figure 3 shows an example of the proposed model.

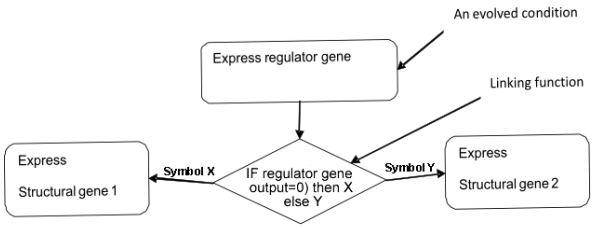


Fig. 3 RMGEP implementation model. The output of the regulatory gene is a symbol that is then used for decision making

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.

The proposed RMGEP algorithm closely resembles the mixture of experts (ME) architectures proposed by Jacobs et al. [25][24]. However, the modules of RMGEP can be specialised to solve a particular task or one module could generalise to solve the entire task. As such a gene in the RMGEP chromosome does not always translate to a task-solving module, the redundant or pseudogenes created by RMGEP are important for evolution of more fit and better individuals in the population. In addition to this, unlike ME architectures, RMGEP does not require separate gating networks, the regulatory gene evolves the arbitration mechanism required to evolve modularity. Also, in the described ME architectures of [25], the number of "experts" are defined *a priori*. Whereas this is similar to specifying a number of *structural genes* in the RMGEP, the RMGEP genes are not specified for any particular task. Additionally, when no knowledge exists about a particular task, a large number of genes could be specified in the RMGEP and the algorithm will create the modules automatically.

Other related task decomposition approaches in the literature include analog genetic encoding (AGE) [10, 11, 36, 37]. Nevertheless, unlike RMGEP, AGE uses alphabetic labels or tokens which separate coding from non-coding parts of the genome to evolve parts of a system. Tokens in the AGE architecture are predefined and each neuron in the neural network is associated with a token. As such, RMGEP is different from AGE architectures where neurons (defining certain devices/tasks)

are predefined. Additionally, RMGEP is an evolutionary algorithm whereas AGE utilises evolutionary technique (a GA) to evolve neural controllers.

Previous work by Darwen and Yao [7], Liu et al. [32] utilises speciation in evolutionary algorithms as modularising mechanisms. In [7], GA individuals in the last generation are grouped using a gating algorithm to create species/modules while in [32] a *k-means* neighbourhood algorithm is utilised to form clusters/niches which are used as modules. In comparison to these techniques, the RMGEP individuals do not need to be grouped into niches as one RMGEP individual has the capability to create modules to solve different sub-tasks. Additionally, the use of negative correlation learning [32] encourages different networks within the ANN to learn different parts or aspects of training data, as such if training data is not available as is the case in robotics, this system would not yield the required results.

5 Evolution of a wall following behaviour

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

5.1 Program implementation

In the work reported in this section, a 2D Java simulator was implemented to simulate the robot and its environment. The experimentation replicates that of Lazarus and Hu [29] where the capabilities of GP in evolving wall following behaviours in environments of increasing complexity was investigated. The experiments implemented here involves a robot moving in four room types where each room is a $16m \times 16m$ cell map (see Figure 4). The room types progress from 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 into any empty cell and detected any obstacle adjacent to its cell location.

The robot was awarded a fitness point for moving in the inner cells adjacent to the walls. 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, mgGEP and RMGEP algorithms were allowed to evolve a strategy to solve the problem and only terminated when the total number of allowed steps was reached. 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 [29], each robot controller was tested by starting the robot at 10 different starting points. The total fitness achieved was thus the

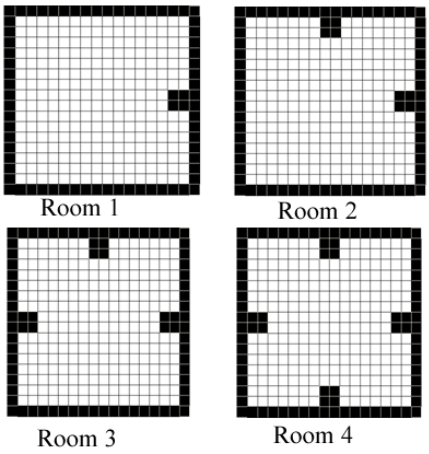


Fig. 4 Four different room types used in the experiment. These room types have been adapted from Lazarus and Hu [29].

sum of fitness achieved in each of the 10 start points. Table 1 shows the maximum fitness for each room over 10 random starts.

Table 1 Maximum fitness awarded to a controller for visiting each cell adjust to the walls from 10 starting points

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

5.2 Algorithm primitives

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

Robot sensors: 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 below:

– 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

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.

5.3 Algorithm set up

In this experiment, 3 genes each with a head size, $h = 8$, were used to form the RMGEP 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. Algorithm 1 shows how the RMGEP was implemented for this problem.

Algorithm 1 using RMGEP 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

As shown on Table 2, the mgGEP was implemented using 2 genes each with a head size = 8 while the monolithic ugGEP had the head size = 16. This meant that the chromosomes were of equal length and differed only in the number of genes used. For the mgGEP, 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 motor 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. Thus, the implementation of the mgGEP effectively incorporates human, problem specific expertise into the algorithm.

Table 2 RMGEP: Algorithm parameter settings

Parameters	ugGEP	mgGEP	RMGEP
Maximum generations	200	200	200
Population	500	500	500
No. of Genes	1	2	3
Head size	16	8	8
Parent organisms	2	2	2
Mutation probability	0.041	0.041	0.027
1-Point Recombination	0.7	0.7	0.7
2-Point Recombination	0.2	0.2	0.2
Gene Recombination	0.0	0.1	0.1
IS Transposition	0.1	0.1	0.1
RIS Transposition	0.1	0.1	0.1
Gene Transposition	0.0	0.1	0.1
Function set	4	4	4
Terminal set	12	12	12
Total starting points	10	10	10
Randomly seeded runs	50	50	50

Table 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 algorithms.

5.4 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. Equation 2 shows the fitness function employed.

$$\begin{aligned} f_j &= (\sum_{i=1}^{m} p_i(x_i, y_i)) - C - Wp \\ fitness &= \sum_{j=1}^{n} f_j \end{aligned} \tag{2}$$

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. 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.

5.5 Results and Discussion

Table 3 shows that standard mgGEP was more successful in room 1 than both ugGEP and RMGEP algorithms. In room 2 and 3, the RMGEP had a better success rate than standard mgGEP while ugGEP had the lowest success rate. The two modular algorithms, mgGEP and RMGEP, had an equal success rates in room 4. Thus in general, these two modular algorithms appear to have solved the problem with almost equal success with RMGEP appearing to be slightly better.

Table 3 Comparison of success rates achieved in the four different room types using standard ugGEP, mgGEP and RMGEP algorithms. The success rate refers to the percentage of the total number of runs where the target fitness was achieved. The best performance is shown in bold.

	ugGEP	mgGEP	RMGEP
Room 1	52	**92**	78
Room 2	52	70	**80**
Room 3	56	64	**70**
Room 4	66	**84**	**84**

This performance suggests that the new algorithm, RMGEP, evolved controllers that approximated the performance of mgGEP algorithm which used 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 RMGEP. This offers better capability for the robot as well as less work for the designer. The monolithic ugGEP algorithm was least successful in solving the wall following problem. The likely explanation to this low performance is that since the algorithm tries to solve the entire problem in one module, it may have encountered local minima during the evolutionary run.

The results shown by Figure 5 suggests that the performance of RMGEP was slightly better than standard mgGEP in rooms 2 and 3, slightly lower in room 4 and was outperformed in room 1. There could be various contributing factor to the slight success in rooms 2 and 3: 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 RMGEP, 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 RMGEP is longer in length than the mgGEP the number of generations used in the experiment may not be sufficient to enable the RMGEP to converge, this could potentially explain the RMGEP performance in rooms 1 and 4. Additionally, in all the rooms the fitness of the best individual does not seem to converge near the optimal fitness. This means that all the 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,

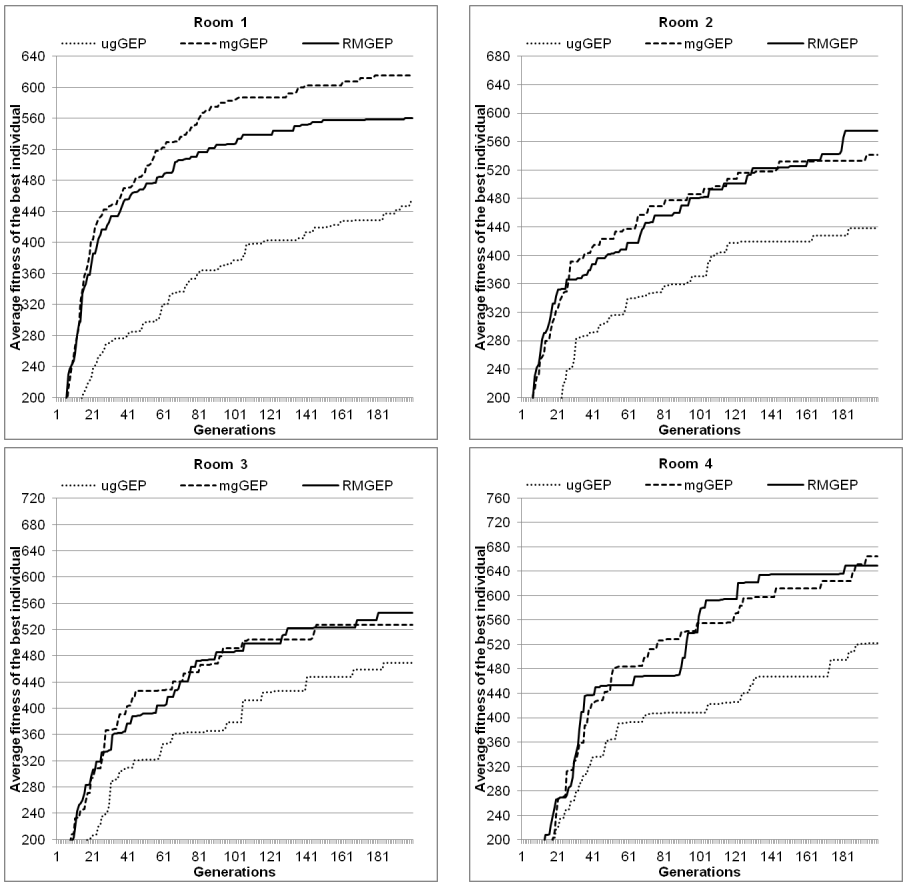


Fig. 5 Comparison of best individual performances in the population as achieved through ugGEP, standard mgGEP and RMGEP in the different room types.

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 RMGEP on the other hand is required to evolve the behaviour sub-division mechanism during the evolutionary run. As previously described, the RMGEP 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 RMGEP 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 6 shows an example of a controller evolved using RMGEP. Although the shown controller was evolved using room 1 (Figure 4), it adapts successfully in all the other rooms. Across the entire range of controllers evolved successfully,

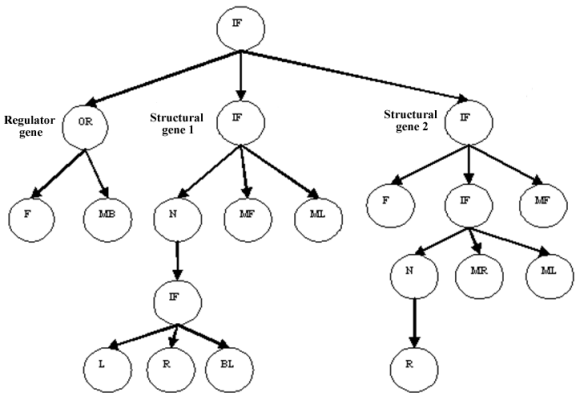


Fig. 6 an example of a controller evolved using RMGEP

it was observed that the first gene (regulatory gene) evolved a structure that selects between a sensor and a movement action. The RMGEP genome (Figure 6) 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 around 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 2). 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 [1, 49, 55, 58]. 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.

6 Algorithm performance analysis

Results presented on Table 3 and Figure 5 suggest that the RMGEP algorithm performs equally as well as the mgGEP. These results suggest that this mgGEP variant is 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 statistical 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. Finally, a description of controllers evolved using the mgGEP and RMGEP is presented.

6.1 Experiment set up

The results presented in the previous sections suggest that the RMGEP may have required longer generation runs in order to solve the wall following problem fully. This is due to the fact that there is more genetic material to evolve. 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, the rest of the parameters were left as detailed on Table 2.

6.2 Results and Discussion

The results achieved in these experimentations were used to analyse the performance of the new algorithm in solving the robot wall following problem.

6.2.1 Performance Comparison

Table 4 and Figures 7 and 8 show the results of the ugGEP, mgGEP and RMGEP algorithms with an increased number of generations. As shown by the results, increasing the number of generations leads to an improvement in performance of all the algorithms. The success rates shown by Table 4, shows that RMGEP outperformed mgGEP in rooms 2 and 3 and was outperformed in rooms 1 and 4.

Table 4 Comparison of success rates achieved in the four different room types using standard ugGEP, mgGEP and RMGEP algorithms. The success rate refers to the percentage of the total number of runs where the target fitness was achieved. The best performance is shown in bold.

	ugGEP	mgGEP	RMGEP
Room 1	74	**98**	94
Room 2	76	88	**90**
Room 3	78	78	**86**
Room 4	76	**96**	94

Figure 7 shows that the performance of the best individual evolved using RMGEP outperformed mgGEP in room 3 and performed equally as well as mgGEP in rooms 1 and 2. However, it is slightly outperformed in room 4. The overall average fitness of the population, Figure 8 shows that RMGEP approximated the performance of mgGEP across rooms 2, 3 and 4 but was outperformed in room 1. The obtained results suggest that RMGEP algorithm is able to approximate the performance of mgGEP where as previously mentioned, human expertise has been utilised. The results shown here suggests that the RMGEP algorithm can be utilised to automatically develop mechanism for behaviour sub-division and behaviour coordination. In all the test cases, the ugGEP did not perform as well as the modular controllers. The low ugGEP performance is likely to be as a result of local minima due to lack of specialisation in the chromosome.

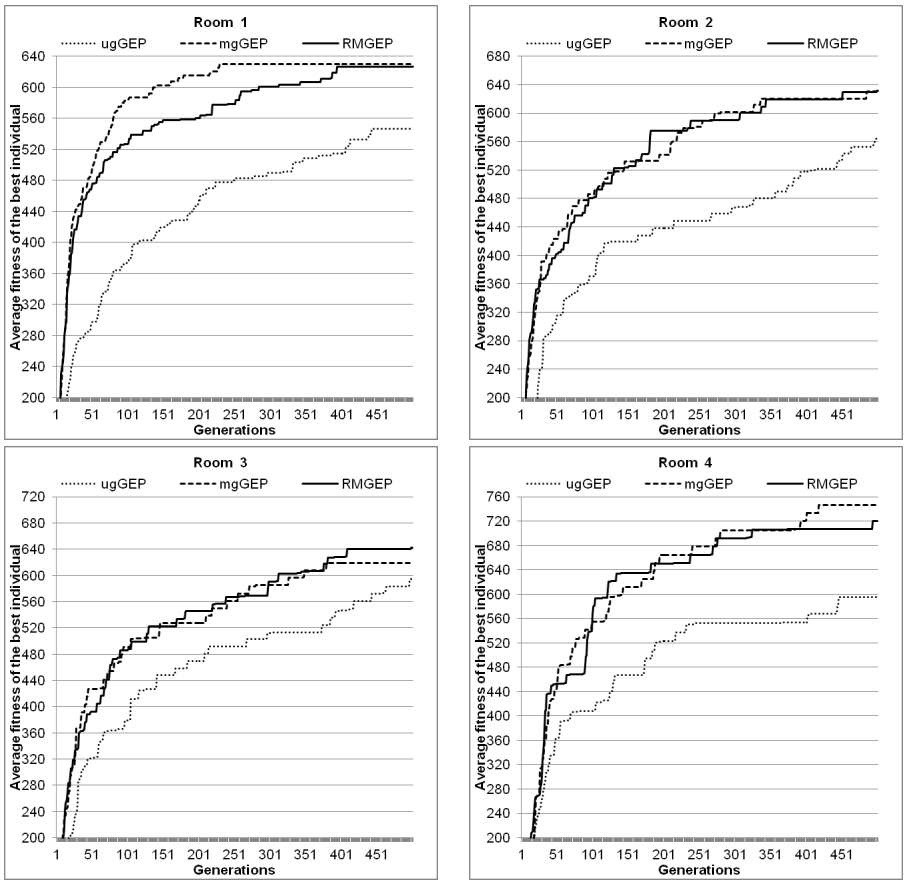


Fig. 7 Progression of the mean fitness of the best individual in the population as achieved through ugGEP, standard mgGEP and RMGEP across the different room types.

The negative average fitness depicted on Figure 8 is as a result of some of the individuals in the population colliding with obstacles or evolving a wandering

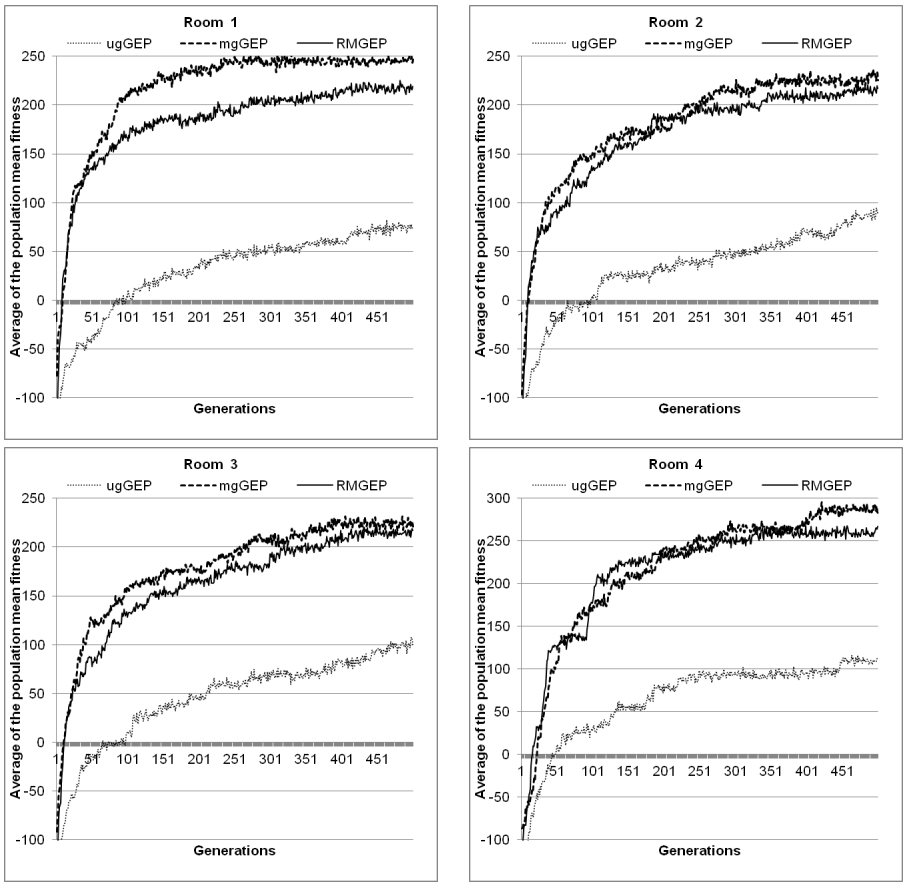


Fig. 8 Progression of the mean of the population mean fitness as achieved through ugGEP, mgGEP and RMGEP across the different room types.

behaviour. As shown by the fitness function, Equation 2, collision and wandering behaviours attract a penalty. In the earlier generations, it is to be expected that most of the individuals would collide with the walls or wander in the environment hence attracting penalties which lead to negative overall fitness. Nevertheless, as the evolutionary epoch continues better individuals are evolved contributing to the overall positive fitness. The continuous progression of the average of the population mean fitness also showcase that the GEP variants used here did not get stuck into a bootstrap problem.

6.2.2 Controller performance

This section shows various controllers evolved through mgGEP and RMGEP algorithms. The controllers were evolved in the 2D discrete environment (Figure 4) with controllers as defined in section 5.2. All the controllers presented here solved the wall following problem in room 4 completely; thus, they all achieved the maximum fitness for room 4 (see Table 1 for optimal fitness in each room).

The functions; AND is represented by **A** while NOT function is represented by **N**. 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 4.

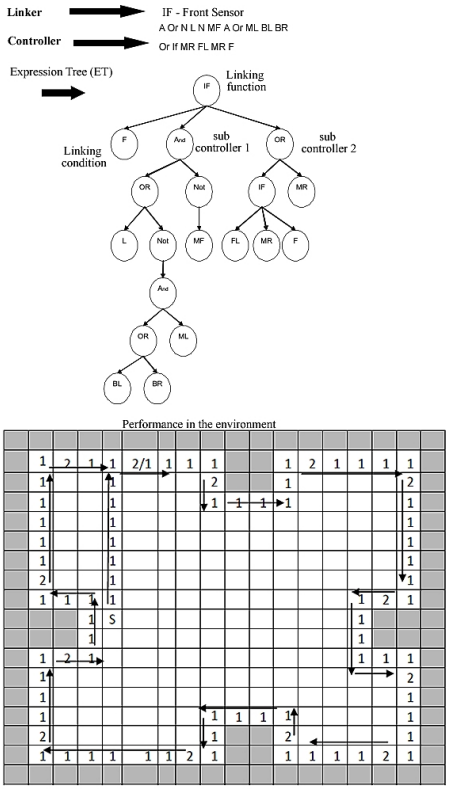


Fig. 9 An example of a multi-controller evolved using standard mgGEP. In this example, sub controller (gene 2) is used for obstacle avoidance and gene 1 is used for exploration and straight line navigation with wall proximity.

Figure 9 shows an example of a controller evolved using mgGEP algorithm to solve the problem. As discussed previously, this algorithm had the problem manually subdivided and therefore one gene evolved obstacle avoidance behaviour whereas the other gene evolved an exploration with wall proximity. For the RMGEP, the regulatory gene provides mechanisms to activate or inhibit either of the two genes. The RMGEP used various strategies to solve the problem. On Figure 10, 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 11, 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

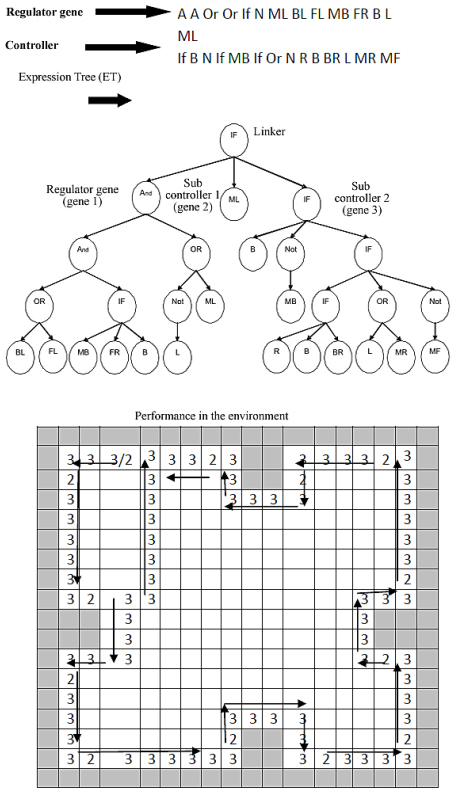


Fig. 10 An example of a multi-controller evolved using RMGEP. This controller utilises gene 2 for obstacle avoidance and uses gene 3 for exploration and straight line navigation with wall proximity. Its behaviour replicates the behaviour of the controller evolved using mgGEP as shown on Figure 9.

used gene 2. In another example, not shown here, 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 RMGEP is important for the evolution of more fit individuals. From these examples, it is evident that RMGEP 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.

6.2.3 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

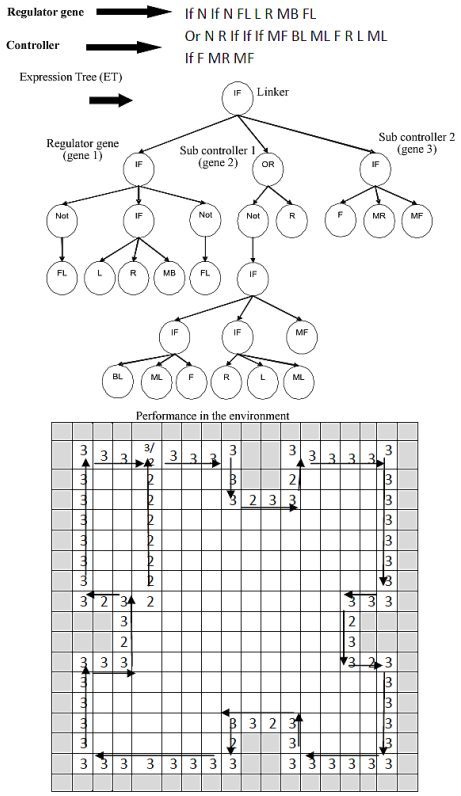


Fig. 11 Example of a novel multi-controller evolved using RMGEP. This controller utilises gene 2 for exploration in open areas and for navigation around the extrusions. The controller utilises gene 3 for navigation with wall proximity and evading obstacles detected by the front sensor.

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".

Table 5 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 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 hy-

pothesis H_A that mgGEP is better than ugGEP, shows that there was a statistical significant difference between these algorithms across all the rooms.

Table 6 RMGEP 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 6 shows that using a two tailed Mann-Whitney U test, there is statistical significance between RMGEP and the ugGEP algorithms. Unlike standard mgGEP, the RMGEP 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. The statistical significance shown by tables 5 and 6 strengthens the conclusion that modular controllers outperform monolithic controllers [44, 49, 55]

Table 7 mgGEP vs RMGEP

	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

Table 7 shows that, overall, there is no statistical significance between mgGEP and RMGEP approaches. In this case, the null hypothesis (H_0) was accepted. Since RMGEP 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 7 also strengthens the claim that with no particular suggestions on how to solve a problem, a developer can use an algorithm such as RMGEP to solve the problem with equally good success as mechanisms requiring human input.

Figure 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 1 have been used. However, as shown in tables 5, 6 and 7 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 12 there was statistical significance between ugGEP and mgGEP

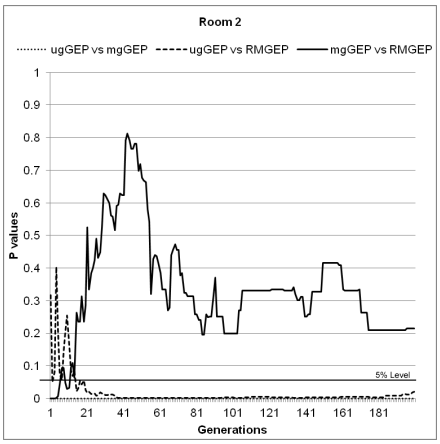


Fig. 12 P values generated using Mann-Whitney U two tailed test. The values are generated from 50 observations in each generation. Each curve shows the comparison with a different algorithms.

in every generation. The comparison of statistical significance between ugGEP and RMGEP 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 RMGEP 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 RMGEP algorithm. These results suggest that at the beginning, the influence of the human division of behaviour means that mgGEP outperforms the new modular mechanism. 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 algorithm, RMGEP, is a suitable mechanism that can be utilised to evolve modular robot behaviours.

7 Effect of behaviour coordination mechanism

The mgGEP implementation, discussed in the previous section, 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 mgGEP and RMGEP are better than monolithic evolution and they have 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 the action selection mechanism or behaviour coordination strategy in the performance of a controller. To provide an answer to this question, a robot foraging behaviour was evolved using ugGEP, mgGEP and RMGEP algorithms.

The robot was required to navigate around the environment looking for "energy sources" placed in various locations within the robot world (see figure 13).

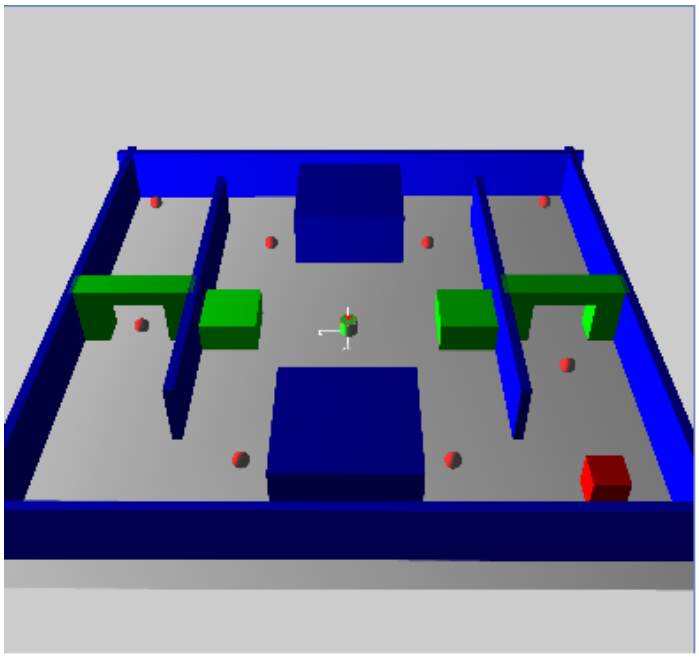


Fig. 13 Robot world used in the foraging experiments. The round objects represents "energy sources" that the robot picks to improve its energy while navigating.

7.1 Robot and environment implementation

In the experiments reported here, the robot moves within the environment looking for energy sources placed in various locations in the environment. The *simbad* simulator[2][23] simulator was used to simulate the robot and its environment. 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. The robot is equipped with eight infra-red sensors and two wheels (left and right). The robot perceives the environment using the sensors and interacts with the environment by performing eight wheel motor actions. Table 8 shows the terminal set used and the sensor positions. 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 wheels translation velocity was set manually to $2.0m/s$ for forward movement and $-2.0m/s$ for reverse movement. 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 on Table 8.

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

Table 8 Terminal set and sensor positions

Symbol	**Represents (Terminal)**	**Represents (Motor)**	**Sensor position (Radians)**
F	Front Sensor	Move Forward	0
B	Back Sensor	Move Back	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	$\frac{3\pi}{4}$

In addition to the reported sensors and motor actions, 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 "energy detector" sensor to enable the robot detect the energy sources. The energy detector sensor was set to be always "ON"; this means that the robot was always active to pick up any energy source once detected. Once the robot detected an energy source, the robot's energy was increased and the energy 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 an energy 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 energy 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 energy 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 [23]).

7.2 Algorithm settings

Due to the use of a continous environment (3D environment) 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. In addition to the reported terminals, representing sensors/Motor as shown on Table 8, 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. An 'If Less Than or Equal to' (IFLTE), function was used as the sole conditional function in the function set. As shown in [43], 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.

In all the described experiments, the mgGEP algorithm was implemented using three genes. The gene interaction and behaviour coordination mechanism utilised is as reported on the experiment set up. Algorithm 2 shows how the RMGEP was implemented for this problem.

Algorithm 2 RMGEP in the foraging problem

Require: regulatory gene ⇒ gene1
Require: Structural genes ⇒ gene2, gene3, gene4
Require: Forward Terminals⇒ F, FR, FL
Require: Back Terminals⇒B, BR, BL
determineController← Translate (gene1)
{*The variable "determineController" is the string symbol output from the translation of the regulatory gene*}
if determineController ⊂ Forward 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
if determineController ⊂ Back Terminals **then**
motorEffectControl ← Execute (gene3)
end if
else
motorEffectControl ← Execute (gene4)
end if

In both mgGEP and RMGEP algorithms, each gene had the head size, h, set to 4. Thus, using Equation 1 the total gene length for mgGEP/RMGEP gene was 17 alleles and therefore the mgGEP chromosome length was 51 alleles. Similarly, the structural genes for the RMGEP had a total length of 51 alleles. Since the mgGEP and RMGEP 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 and RMGEP, 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 9 provides a summary of the utilised parameters.

The fitness function shown by Equation 3 was utilised in the experiments reported here.

$$fitness = (\sum_{i=1}^{n} p_i(x_i, z_i)) - C - S \tag{3}$$

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 collision penalty, C, is set to 25 while the stationary penalty, S, is set to 50 and is only applied if the robot does not move from initial starting point within the first 5 virtual seconds. The number of robot steps, n, is initially set to 1000 but depending on the energy sources visited, this can

[3] Please see Table 8 on how the sensors and motor terminals are implemented. E terminal refers to the robot battery.

Table 9 Robot foraging behaviour: Algorithm parameter settings

Parameters	ugGEP	mgGEP	RMGEP
Maximum generations	200	200	200
Population	100	100	100
No. of genes	1	3	4
Head size	12	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	0.0	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	0.0	0.1	0.1
Selection range	5%	5%	5%
Functions(IFLTE)	1	1	1
Terminals(R, L, F, B, FR, FL, BR, BL, E^3)	9	9	9
No. of randomly seeded runs	20	20	20

go up to 3000. Thus, the maximum fitness was set to 3000 corresponding to the maximum life span of the robot.

7.3 Experimental results

The gene interaction and behaviour coordination for the mgGEP algorithm was reconfigured in three experiments to determine the effect of a particular sub-controller position to the overall performance of the whole controller. The achieved results were then compared to the performances of ugGEP and RMGEP algorithms.

7.3.1 Experiment I

In the first experiment, the mgGEP algorithm was implemented using sub-behaviour model shown by Figure 14. 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.

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 15 shows a comparison of the performance of mgGEP (OA priority) and the ugGEP algorithm in evolving suitable controllers for a foraging robot.

7.3.2 Experiment II

The second experiment employed the sub-division model shown by Figure 16. In this context, the robot's energy was set as the priority for the robot; that is, the robot checked the energy before it decided which actions (in terms of genes) to

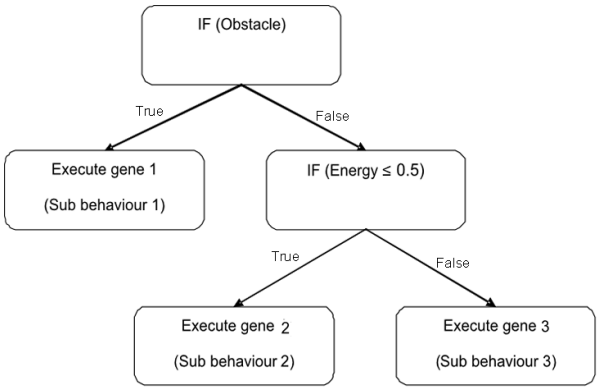
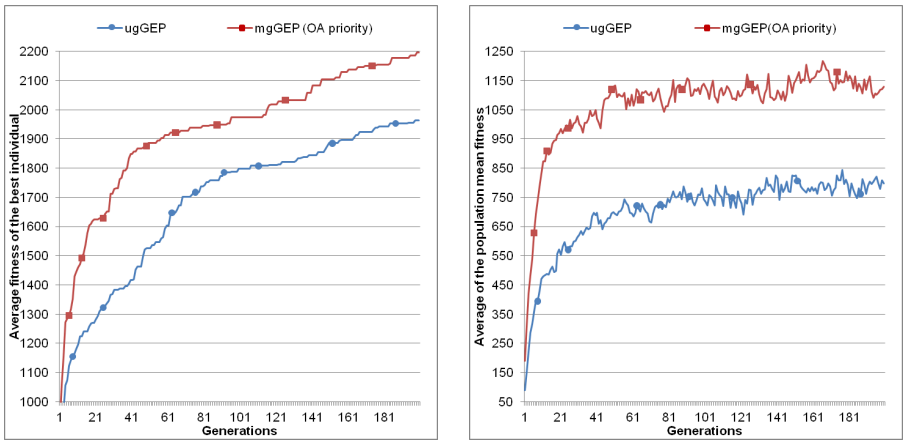


Fig. 14 A behaviour coordination model that relies on robot proximity to obstacles in order to choose a robot task.



Best individual performance

Population average performance

Fig. 15 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.

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 Figure 18.

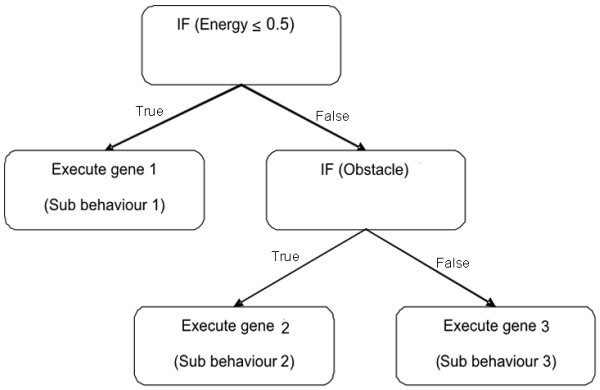


Fig. 16 A behaviour coordination model that relies on robots energy level in order to select a robot task.

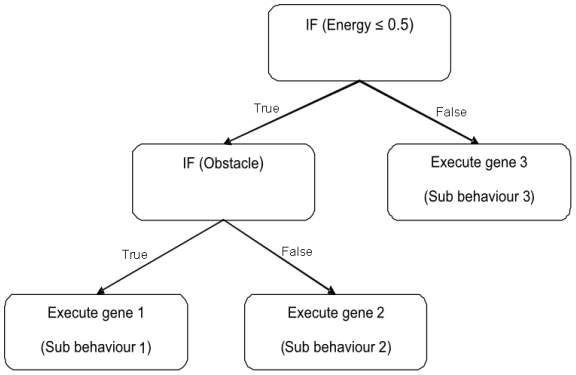
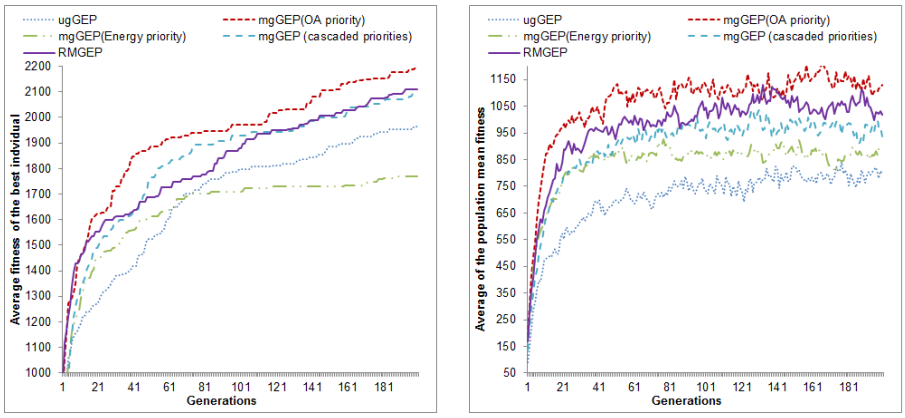


Fig. 17 A behaviour coordination model that uses robots energy to determine behaviour selection. In this case sub-behaviour selection is determined via cascading priorities.

7.3.3 Experiment III

The third experiment, implemented the behaviour sub-division model shown by Figure 17. 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 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 18.



Best individual performance Population average performance

Fig. 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 RMGEP algorithms 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 runs. Similarly, the mean of the best individual fitness was derived from the 20 randomly seeded runs.

7.4 Discussion

Similar to the results obtained in sections 5, Figure 15 shows that the mgGEP mechanism outperformed the ugGEP algorithm in solving the foraging behaviour problem. The performance of the mgGEP algorithm has 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 in any direction as shown in Table 8. 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.

Figure 18 shows 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 fact, the results of mgGEP (OA priority) are better than those of the self-organising RMGEP mechanism. In the implementation reported here, obstacle avoidance is an important behaviour as the robot incurs a high penalty (**25**) for 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 the absence of obstacles the robot then checks the energy before making a decision whether to look for energy 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. The likely explanation to the performance of the RMGEP is that the algorithm may have required longer generation run in order to converge. However, as can be seen from both Figure 18 (average best indvidual fitness and mean population fitness), the RMGEP is as competitive as the mgGEP (OA priority) and performs better than the other action selection mechanisms.

Results shown by Figure 18 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 energy 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 18 shows that the average performance of 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. 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 it is outperformed by mgGEP (OA priority) and RMGEP. 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 modu-

lar controller. As such, self-organising mechanisms such as the proposed RMGEP algorithm could be easily and effectively be utilised in this type of problems.

8 Conclusion

In this paper the performance of GEP in evolving mechanisms to sub-divide and coordinate sub-behaviours was investigated. Additionally a new technique, RMGEP has been proposed as a more biologically plausible alternative to the standard mgGEP as discussed in [12, 44]. Two behaviours, wall following and foraging, were implemented and results discussed. The obtained results show that ugGEP, mgGEP and RMGEP algorithms are able to evolve the two behaviours. In addition to this, the modular techniques, mgGEP and RMGEP, performed better than the ugGEP algorithm in the wall following problem. Similarly, the mgGEP and RMGEP algorithms outperformed ugGEP algorithm in the food foraging problem. The conclusions made from both experiments shows that mgGEP and RMGEP chromosomes have an advantage over the single chromosome used in ugGEP. The mgGEP and RMGEP mechanisms are shown to evolve specialised modules while ugGEP evolves a more general purpose controller.

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 mgGEP action selection mechanisms were more successful than others. As reported earlier, 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 [51, 58]. The manual mechanism implemented in this paper 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 somewhat 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. Consequently, techniques such as the proposed RMGEP algorithm need to be investigated for evolution of modular robotic behaviours.

The proposed RMGEP algorithm 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 RMGEP 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 paper suggest that the potential offered by RMGEP algorithm outweighs the time overhead. The RMGEP also provides a distinctive advantage over human-derived modular controllers in environments where such a division is not known or is difficult to determine. Although the human designer needs to determine the

number of genes to use, this can be over-specified and any extraneous genes will not be used by the algorithm.

There are various real life robotic problems where automatic behaviour subdivision is important. For instance, 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 example, 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 RMGEP technique can be used in further work involving complex robots. As the results presented in this paper show, the RMGEP algorithm is likely to evolve robust controllers so long as sufficient number of genes are supplied with enough evolutionary time.

References

1. 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.
2. 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.
3. R. A. Brooks. A Robust Layered Control System for a Mobile Robot. *IEEE Journal of Robotics and Automation*, 2(1):14–23, 1986.
4. 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.
5. 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.
6. S. Clancy and W. Brown. Translation:DNA to mRNA to Protein. *Nature Education*, 1(1):101, 2008.
7. P. J. Darwen and Xin Yao. Speciation as Automatic Categorical Modularization. *Trans. Evol. Comp*, 1(2):101–108, July 1997.
8. H. De Jong. Modelling and Simulation of Genetic Regulatory Systems: A Literature Review. *J. Computational Biology*, 9(1):67–103, 2002.
9. Stéphane. Doncieux, Jean-Baptiste. Mouret, and Nicolas. Bredeche. Exploring New Horizons in Evolutionary Design of Robots. In *Workshop on Exploring new horizons in Evolutionary Design of Robots at IROS 2009*, pages 5–12, Saint Louis, United States, 2009.
10. P. Dürr, C. Mattiussi, and D. Floreano. Neuroevolution with analog genetic encoding. In *Proceedings of the 9th International Conference on Parallel Problem Solving from Nature*, volume 6856 of *Lecture Notes in Computer Science*, pages 671–680, Berlin, Heidelberg, 2006. Springer-Verlag.
11. P. Dürr, F. Mattiussi, and D. Floreano. Genetic representation and evolvability of modular neural controllers. *IEEE Comp. Int. Mag.*, 5(3):10–19, 2010.

12. C. Ferreira. Gene Expression Programming: A New Adaptive Algorithm for Solving Problems. *J. Complex Systems*, 13(2):87–129, 2001.
13. C. Ferreira. *Gene Expression Programming: Mathematical Modelling by an Artificial Intelligence (2nd edition)*. Springer, 2006.
14. D. Floreano and L. Keller. Evolution of Adaptive Behaviour in Robots by Means of Darwinian Selection. *J. PLoS Biology*, 8(1):e1000292, 2010.
15. 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.
16. 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.
17. 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.
18. 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.
19. D. Gu and H. Hu. Evolving fuzzy logic controllers for sony legged robots. In A. Birk, S. Coradeschi, and S. Tadokoro, editors, *RoboCup 2001: Robot Soccer World Cup V*, volume 2377 of *Lecture Notes in Computer Science*, pages 356–361. Springer Berlin Heidelberg, 2002.
20. 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.
21. 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.
22. 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.
23. L. Hugues and N. Bredeche. A Quick Programming Guide for Simbad Simulator., August 2005. URL `http://simbad.sourceforge.net/guide.php`.
24. R.A. Jacobs and M.I. Jordan. Hierachical mixtures of experts and the EM Algorithm. *Neural Computation*, 6:181–214, 1994.
25. R.A. Jacobs, M.I. Jordan, and A.G. Barto. Task Decomposition Through Competition in a Modular Connectionist Architecture: The What and Where Vision Tasks. *Cognitive Science*, 15:219–250, 1991.
26. S. Kent. *Evolutionary Approaches to Robot Path Planning*. PhD thesis, Brunel University, 1999.
27. J. R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, 1992.
28. 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.

29. C. Lazarus and H. Hu. Using Genetic Programming to Evolve Robot Behaviours. In *Proceedings of the 3rd British Conference on Autonomous Mobile Robotics and Autonomous Systems*, 2001.
30. C. Lazarus and H. Hu. Evolving Goalkeeper Behaviours for Simulated Soccer Competition. In *Proceedings of the 3rd IASTED International Conference on Artificial Intelligence and Applications*, 2003.
31. W-P. Lee. Evolving Complex Robot Behaviors. *J. Information Sciences*, 121: 1–25, 1999.
32. Y. Liu, X. Yao, and T. Higuchi. Evolutionary Ensembles with Negative Correlation Learning. *IEEE Transactions on Evolutionary Computation*, 4(4): 380–387, 2000.
33. 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.
34. M. J. Mataric. A Distributed Model for Mobile Robot Environment-Learning and Navigation. Technical report, Massachusetts Institute of Technology, Cambridge, MA, USA, 1990.
35. Maja J. Mataric. Learning in behaviour-based multi-robot systems: policies, models, and other agents. *J Cognitive Systems and Research*, 2:81–93, 2001.
36. C. Mattiussi and D. Floreano. Evolution of Analog Networks using Local String Alignment on Highly Reorganizable Genomes. In *Proceedings of the 2004 NASA/DoD Conference on Evolution Hardware*, pages 30–37. IEEE Computer Society, 2004.
37. C. Mattiussi and D. Floreano. Analog Genetic Encoding for the Evolution of Circuits and Networks. *IEEE Trans. Evolutionary Computation*, 11(5): 596–607, 2007.
38. C. Mautner and R. K. Belew. Evolving Robot Morphology and Control. In *Proceedings of the Artificial Life and Robotics (AROB)*, 1999.
39. 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.
40. R. R. Murphy. *Introduction to AI Robotics*. MIT Press, Cambridge, MA, USA, 1st edition, 2000.
41. J. Mwaura. *Evolution of Robotic Behaviours Using Gene Expression Programming*. PhD thesis, University of Exeter, Exeter, UK, 2011.
42. J. Mwaura and E. Keedwell. Adaptive Gene Expression Programming Using a Simple Feedback Heuristic. In *Proceedings of the AISB*, Edinburgh, UK, 2009.
43. 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.
44. 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.
45. 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.
46. 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.
47. S. Nolfi. Evolutionary Robotics: Exploiting the full power of self organization. *J. Connection Science*, 10 (3–4):167–183, 1998.
48. 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.
49. S. Nolfi and D. Floreano. *Evolutionary Robotics. The Biology, Intelligence, and Technology of Self-organizing Machines.* MIT Press, 2000.
50. P. Nordin and W. Banzhaf. Real Time Control of a Khepera Robot Using Genetic Programming. *J. Cybernetics and Control*, 26:533–561, 1997.
51. T. J. Prescott, P. Redgrave, and K. Gurney. Layered Control Architectures in Robots and Vertebrates. *J. Adaptive Behavior*, 7(1):99–127, 1999.
52. 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.
53. 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.
54. 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.
55. J. Togelius. Evolution of a Subsumption Architecture Neurocontroller. *J. Intelligent Fuzzy System*, 15:15–20, 2004.
56. J. Urzelai and D. Floreano. Incremental Evolution with Minimal Resources. In *Proceedings of the International KHEPERA Workshop*, 1999.
57. J. Urzelai and D. Floreano. Evolution of Adaptive Synapses: Robots with Fast Adaptive Behaviour in New Environments. *J. Evolutionary Computation*, 9: 495–524, 2001.
58. 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.