On Using Gene Expression Programming to Evolve Multiple Output Robot Controllers

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

Ed Keedwell
Department of Computer Science
CEMPS,University of Exeter
Exeter, UK
Email: e.c.keedwell@exeter.ac.uk

***Abstract*—Most evolutionary algorithms (EAs) represents a potential solution to a problem as a single-gene chromosome encoding, where the chromosome gives only one output to the problem. However, where more than one output to a problem is required such as in classification and robotic problems, these EAs have to be either modified in order to deal with a multiple output problem or are rendered incapable of dealing with such problems. This paper investigates the parallelisation of genes as independent chromosome entities as described in the Gene Expression Programming (GEP) algorithm. The aim is to investigate the capabilities of a multiple output GEP (moGEP) technique and compare its performance to that of a single-gene GEP chromosome (ugGEP). In the described work, the two GEP approaches are utilised to evolve controllers for a robotic obstacle avoidance and exploration behaviour. The obtained results shows that moGEP is a robust technique for the investigated problem class as well as for utilisation in evolutionary robotics.**

Keywords*—Gene Expression Programming, Robot Control, Evolutionary robotics.

I. Introduction

An autonomous robot should be able to explore its environment without colliding with obstacles and causing harm to itself or people. There are many studies that have been conducted on developing obstacle avoidance behaviours using genetic algorithms (GAs) and genetic programming (GP) [10], [17], [18], [21]. 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 [13]. Evolutionary Robotics (ER), however, using a population of control programs has been shown to generate behaviours that are more robust than human coded ones [5]–[8], [11], [17]. Furthermore, these evolved behaviours have been shown to adapt successfully in new environments [17].

The main objective of the work reported in this paper is to investigate the capabilities of Gene Expression Programming (GEP) to evolve multiple output controllers for a robot. In order to distinguish this mechanism from the GEP utilising a linking function (described in [3], [4], [16]), this work refers to this mechanism as multiple output Gene Expression Programming (moGEP). In the described work, the moGEP is utilised to evolve a controller for a navigation and obstacle avoidance problem. The evolved controllers are tested for adaptation into new environments and their performance is evaluated and compared to the fitness in the training environments. The obtained results are compared to those of the unigenic Gene Expression Programming (ugGEP) approach.

The rest of the paper is divided as follows: A brief introduction of GEP, including a segment describing the moGEP approach, is presented in section II. The description of the experiment set up including the robot environments and algorithms implementations is presented in Section III. In Section IV, the results of the experiments are presented and a discussion follows. Section V provides a conclusion for the paper.

II. Gene Expression Programming

Gene Expression Programming (GEP) is an evolutionary algorithm with one main distinct characteristic: it is formed using fixed-length genes (genotypes) which are later translated to variable length expression tress (ET) or phenotypes. The fixed-length string genes are similar to the GAs linear encoding while the ramified tree structures are similar to GP encoding of solutions.

GEP genes are composed of functions (logical operators, mathematical functions and conditional operators) and terminals. In addition, each GEP gene is divided into a head and tail sections. The head section contains functions and terminals while the tail contain only terminals. The length of the head is an independent user input while the tail length is given by

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

where h is the length of the head and n is the maximum arity of the functional symbols.

The translation process for the GEP chromosome is carried out using breadth-first parsing scheme. The resulting expression is known as an open reading frame (ORF) and is similar in nature to a similar structure in systems biology. The ORF is ordinarily expressed as a parse tree referred as expression tree (ET). The GEP chromosome, thus, undergoes the genetic operations while the ET is the actual solution and is subjected to function fitness evaluations.

Similar to most evolutionary algorithms, GEP utilises both sexual and asexual genetic operators. The sexual genetic operators are: one point recombination, two point recombination and gene recombination. One point and two point recombination are identical to similar operators in GAs. However, gene

978-1-4799-4479-8/14/$31.00 ©2014 IEEE

recombination is only utilised when the GEP chromosome is constructed using more than one gene. The process involves selecting one gene within the parent chromosome and swapping with another gene from the other parent chromosome. The asexual genetic operators are: mutation which is similar to the GA mutation implementation, insertion sequence (IS) transposition, root insertion sequence (RIS) transposition and gene transposition. The IS operator selects an allele (transposon) within the gene and inserts it at the head section (except the root position), without affecting the tail section. The RIS operator is similar to the IS except that the transposon must belong to the function set. For the gene transposition, an entire gene is selected as the transposon and inserted as the first gene of the chromosome. In all the cases, the replaced allele or gene is deleted at the place of origin and other elements pushed downstream.

In [13], a single-gene GEP chromosome is used to evolve a controller for a robotic obstacle avoidance and exploration behaviour. The work shows that the evolved controllers are capable of adapting to new environments. In addition, single-gene GEP chromosomes have been used to solve different problems ranging from sequence induction [4], [19] to automatic rule identification for agent-based crowds [25]. All the described works show that single-gene GEP chromosomes are powerful tools for solving problems. Nevertheless, it has been shown that GEP chromosomes can be formed using more than one gene resulting in a more robust and complex programs. The next section discusses how to utilise multiple gene GEP chromosomes.

A. Multigenic Gene Expression Programming

A major difference between evolutionary algorithms such as GA and GP with the GEP is that GEP chromosomes can be constructed using more than one gene of equal length. In essence, it is this capability that gives GEP its superior performance to GP [4]. A GEP chromosome composed of more than one gene is referred as multigenic GEP chromosome (mgGEP) while a chromosome with only one gene is referred as unigenic GEP (ugGEP) chromosome. When multiple genes are used in problem solving, the number of genes as well as the length of the head is chosen a prior. Each of the genes forms a sub-ET and the sub-ETs interact with each other to solve the given problem.

When a GEP chromosome is composed of multiple genes, they can be combined using a special function, known as a linker, to form longer concatenated sequences. Alternatively, the genes may individually be utilised to solve a part of the problem. 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 this work, the concatenated GEP chromosomes are referred as mgGEP chromosome while the multiple output chromosome is referred as moGEP. However, it is important to note that the main difference between these algorithms is that the former utilises a linking function to combine the genes leading to only one output, while the latter leads to more than one output.

The capabilities of the concatenated multigenic GEP chromosomes have been investigated for various problems. For instance, this mechanism was shown to be more robust when solving a sequence induction problem as well as evolving cellular automata rules for the density-classification problems [3], [15] and evolving classification rules [26]. In addition, recent work has showed the capabilities of mgGEP to evolve robotic behaviours [13], [16]. Modifications of the mgGEP algorithm to form varied length chromosomes has been shown by [1], [15].

The idea of using the multiple genes of the mgGEP as single independent entities, to partially solve a problem, is not new. This idea stems from the initial paper describing the GEP algorithm [3]. However, to the best of the authors' knowledge, there has not been any single work carried out utilising this method in general and in ER in particular. As such, the work reported here is of a preliminary nature and aims at investigating the capabilities of utilising moGEP in ER.

The moGEP approach is similar in nature to the multi-chromosome approach used with Cartesian GP (CGP), referred as MC-CGP [22] or MCE [23]. The main difference with the two techniques is that in MC-CGP each gene (or chromosome in CGP) is subjected to a function fitness evaluations independently. This means that the number of function fitness evaluations increases with the number of genes. Contrary, in moGEP the entire chromosome is evaluated once for the task at hand. It is, however, possible to equate the two approaches by modifying either of the algorithm to suit function fitness evaluations needs. However, CGP represents programs as directed graphs while GEP evolves tree-like phenotypes. In addition, CGP uses a list of integers for its chromosome whiles moGEP uses string symbols. As such, these techniques are different in implementations and problem solving.

The moGEP is also similar to Parisian GP [2] since both approaches uses sub-solutions to form a more complex solution. However, in the Parisian GP, an entire chromosome represents a partial solution with the whole solution being formed by a set of chromosomes in the population. In contrast, the moGEP chromosome encodes the entire potential solution. Further, moGEP is different in structure to the multiple output GP (Modi) described by [24]. The Modi program is still a one tree GP which gives a multiple output after the modification of the GP's output structure. Finally, moGEP is similar in nature to the Multi Expression Programming (MEP) algorithm [20]. However, whereas only the fittest statement provides the program output in MEP, in the moGEP approach all statements (genes) provides important output for the intended application.

For a detailed discussion on GEP, please see [4], [16].

III. Experimental set up

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 [21].

This section describes the experiment set up in sufficient detail.

A. Robot and environment implementation

In this experiment, the *simbad* simulator[1] [9] was used to simulate the robot and its environment. The simulator was used in background mode to avoid rendering the environments during the course of the run.

In all experiments where the simbad simulator was utilised, the simulated robot used eight infra-red sensors placed 0.25 radians apart along the perimeter of the top of the robot. The robot movement was achieved using a simulated pair of wheels (left and right). The simbad simulator offers two methods 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 reported implementation, the robot used 8 sensors placed on specific angles as shown in Table I. The robot sensors returned a value between $0.0m$ and $2.0m$. A value of $0.0m$ meant that the robot was adjacent to the wall, whereas a value of $2.0m$ meant that the robot was a minimum of $2.0m$ away from the obstacle. The aim was thus to evolve a mechanism to map various sensor readings to specific motor functions.

TABLE I. TERMINAL SET AND SENSOR POSITIONS

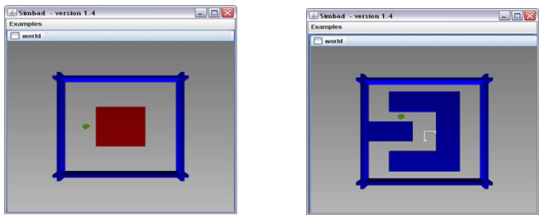
Sensor	Terminal Representations	Sensor positions
Front Sensor	F	0
Back Sensor	B	π
Left Sensor	L	$\frac{\pi}{2}$
Right Sensor	R	$\frac{3\pi}{2}$
Front Right Sensor	FR	$\frac{7\pi}{4}$
Back Right Sensor	BR	$\frac{5\pi}{4}$
Front Left Sensor	FL	$\frac{1\pi}{4}$
Back Left Sensor	BL	$\frac{3\pi}{4}$

To perform the required experiments, three types of environments were used as shown by Figure 1. Environment 1 (Figure 1a) was set up using a $10m \times 10m$ box with one square obstacle within it. Environment 2 was a $15m \times 15m$ box with a U shaped obstacle covering most of the space (see Figure 1b). Environment 3 was a $20m \times 20m$ box with a more complex array of obstacles as shown in Figure 1c.

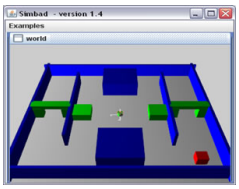
B. Algorithm parameters

The implemented GEP algorithm variants used the following attributes to describe the terminal and function sets.

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



(a) Environment 1 (b) Environment 2



(c) Environment 3

Fig. 1. Robot environments. These images have been reproduced from authors earlier work on [13] where the ugGEP algorithm was initially utilised to evolve robot obstacle avoidance and exploration behaviour. Since the presented work builds on from [13], the entire work can be viewed to be of incrementally nature and as such, the use of images does not require pre-authorisation from the IEEE.

1) 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 arguments (less than or equal to ($\leq$)) and then gives the output of the third argument if the comparison is true, or the fourth argument if the comparison is false.

2) Terminal set: Table I shows the terminal representation used in the experiment. The terminals represent different meaning depending on which part of the IFLTE function they are positioned in. The left part of the IFLTE statement is the premise and the right hand side is the consequent. As shown on Figure 2, when terminals are positioned on the right hand side (output) they are utilised for motor activity while the left hand side (input) are for sensory activity. Thus, the algorithms were set up such that similar alphabets were used for the sensory and motor functions.

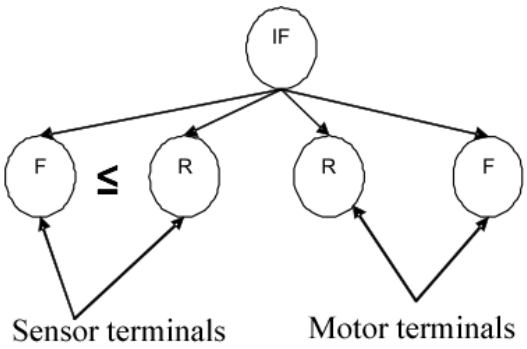


Fig. 2. Example of a potential robot avoidance controller using terminals and functions as reported on Table I.

C. Evolving controllers using ugGEP

Two implementations of the ugGEP algorithm was carried out. In both implementations, the robot movements was controlled using the default kinematic drive. This section outlines how the ugGEP implementations were carried out.

1) standard ugGEP: For the standard ugGEP implementation, the robots translation velocity was set manually to $2.0m/s$ for forward movement, i.e, when **F** terminal symbol was returned. The reverse translation velocity was set to $-2.0m/s$ when **B** terminal symbol was returned. When any of the other terminal symbols was returned, the robot rotated on it axis based on the angle associated with the returned terminals sensor position (See Table I for sensor positions).

2) ugGEP with sensor based velocity control: In the standard ugGEP implementation, the set velocity 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. As such, a second ugGEP implementation was carried out. This second implementation utilised sensor readings to control velocity. The controller is thus similar to controllers developed using neural networks [2]; that is, the controller takes the sensors readings as inputs and translates them to a motor output. However, the model is simpler and contains only one output that effects robot motor controls. In this second implementation, the output of the evolved GEP controller was used to set the robots translational or rotational velocity. The new implementation was referred as ugGEP-sensor.

When the ugGEP controller is executed the output is one terminal symbol. In the ugGEP-sensor implementation, 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 I 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. Since the robot sensors were set to return a value between $0.0m$ and $2.0m$, the maximum speed that the robot could move at was $2.0m/s$. In this experiment, therefore, the robot learns the mapping between specific sensors to various translational and rotational velocities.

D. Evolving controllers using moGEP

In the experiments reported here, the moGEP approach was utilised to evolve two sub-ETs that work together to evolve an obstacle avoidance controller. The output of each sub-ET is the sensor reading of that particular terminal symbol as shown in Table I. Similar to ugGEP with sensor velocity, the returned sensor readings are utilised to provide motor activity but only in m/s. For instance, if the returned output is **F** with a *Front* sensor reading of $1.5m$, then this is translated as $1.5m/s$. All sensor readings are therefore utilised to provide a translation velocity and no angular or rotational velocity is provided.

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. In the conducted experiments, the output of the first sub-ET controlled the velocity of the left wheel while the output of the second sub-ET controlled the right wheel. The difference of the two outputs in terms of sensor readings produced an angular velocity while an equal output gave a linear velocity. The robot, thus, had to establish a link between linear and angular velocity in order to accomplish exploration and obstacle avoidance tasks.

The two genes were part of the whole moGEP chromosome and were thus evolved as such. Note that there was no linking function provided for the genes and that every gene had an independent contribution to the overall robot behaviour. The fitness achieved was for the collective performance. In this case the robot was required to evolve a strategy to control its velocity and to avoid obstacles.

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.

E. Genetic operators and algorithm parameter values

All the GEP operators were used in the experiment with the probabilities set as per Table II

TABLE II. PARAMETER SETTINGS

Parameters	**ugGEP**	**ugGEP-Sensor**	**moGEP**
Maximum generations	100	100	100
Population size	200	200	200
Head size	8	8	4
No. of genes	1	1	2
Functions(IFLTE)	1	1	1
Terminals(see Table I)	8	8	8
Mutation probability	$\leftarrow$	2/chromosome size	$\rightarrow$
1-Point Recombination Probability	0.7	0.7	0.7
2-Point Recombination Probability	0.2	0.2	0.2
IS Transposition probability	0.1	0.1	0.1
RIS Transposition probability	0.1	0.1	0.1
Gene transposition probability	0.0	0.0	0.1
Gene recombination probability	0.0	0.0	0.1

As described in section II, GEP uses seven genetic operators. The values of genetic operators probabilities, shown on Table II, are derived from work carried out by the author in [12]. 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.

The implemented algorithms were generational and used replication to pass the best organism in the previous generation. Roulette wheel selection was used to select parent organisms.

The experiments were carried out using a population of 200 organisms and run for 100 generations. For fair algorithm

[2]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 [17])

comparisons, the head size h was set to 4 for the moGEP and 8 for the ugGEP variants. This ensured that both moGEP and ugGEP chromosomes were of equal lengths. The algorithms were evaluated over 20 randomly seeded runs. Each controller in the population was allowed to control the robot for 50 virtual seconds. The rest of the parameters were set as defined in Table II.

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

$$fitness = (\sum_{i=1}^{n} p_i(x_i, z_i)) - C - S \quad (2)$$

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

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

IV. Results and Discussion

This section provides the results of the experiments described above. The main aim is to investigate how the moGEP can be utilised to evolve monolithic robot controllers that produce multiple output. The results are compared with those of the ugGEP with sensor based velocity control and an ugGEP not utilising sensor controls. The described work does not carry out any comparisons with GP, GA or any other complimentary methods. The reasons for this is because such comparisons for ugGEP and GP has been shown on [14] as well as work described in [4].

2) Experiment 1: Evolving controllers in test environments: The aim of the initial experiment was to test the viability of the moGEP to evolve robot controllers. In the first experiment, Environment 3 (Figure 1c) was used. The obtained results are as shown in Figures 3, 4, 5 and 6

As observed in this experiment, the moGEP performs better that standard ugGEP and ugGEP with sensor based velocity control. Figure 4 shows that in all the different chromosome lengths tried the moGEP 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 and 4 shows that there is a greater probability of evolving successful controllers using the moGEP technique.

Figure 5 shows that moGEP algorithm used approximately $5 - 10$ generations to evolve an individual that solved the

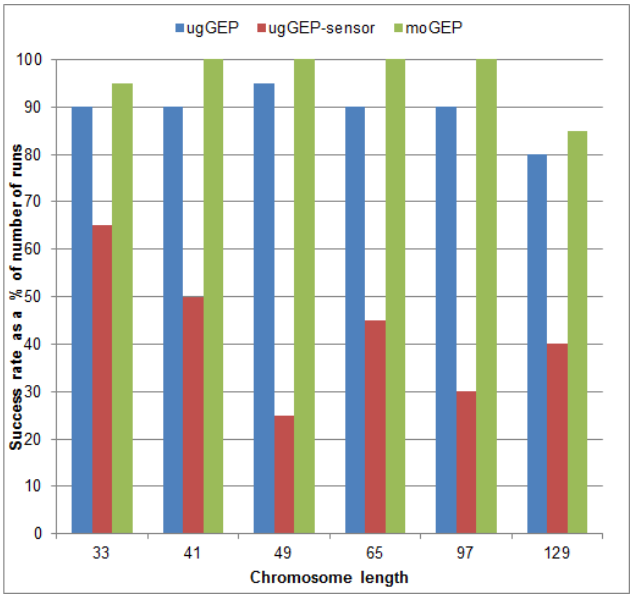


Fig. 3. Comparison of success rates achieved using ugGEP, ugGEP with sensor based velocity control and with moGEP.

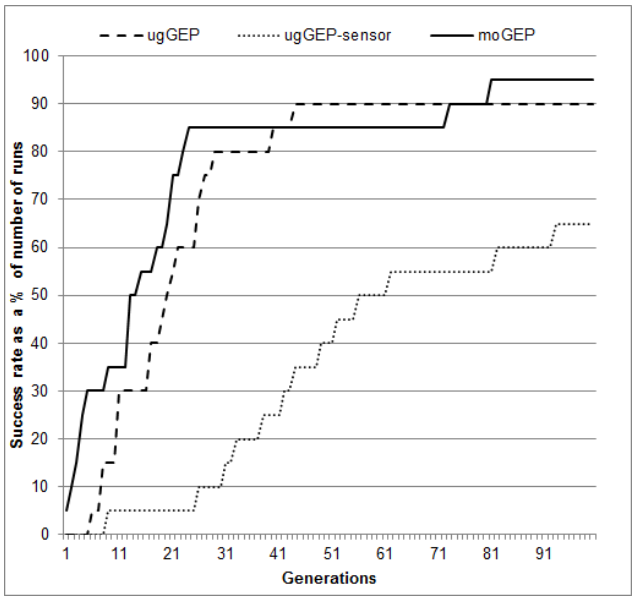


Fig. 4. Comparison of success rate over generations as achieved using ugGEP, ugGEP with sensor based velocity control and with moGEP.

problem. Additionally, Figure 6 shows that the moGEP 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 Environment 3 (Figure 1c)), this suggests that the moGEP 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

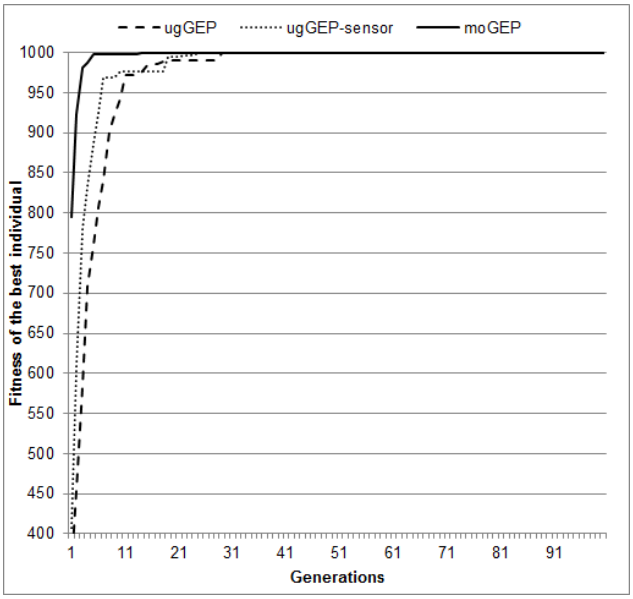


Fig. 5. Progression of best individual in the population as achieved using ugGEP, ugGEP with sensor based velocity control and with moGEP in Environment 3 (Figure 1c).

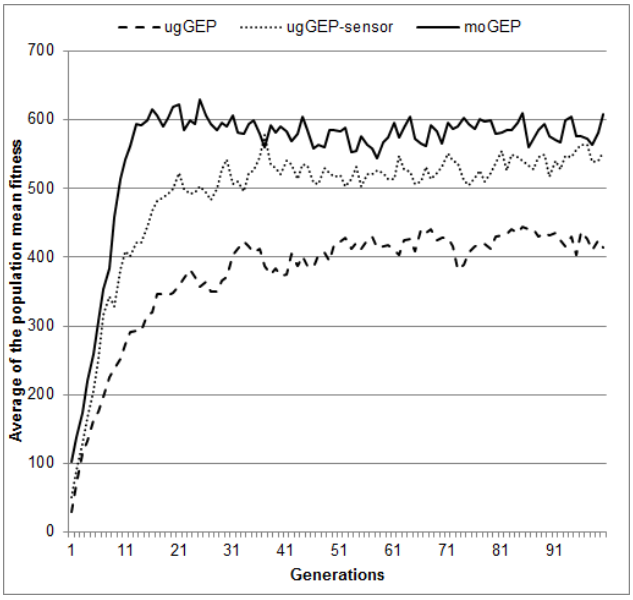


Fig. 6. Progression of the mean population mean fitness as achieved using ugGEP, ugGEP with sensor based velocity control and mgGEP with multiple output gene in Test environment 2 (Figure 1c).

control.

3) 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 IV. Environment 1 (see Figure 1a) was used for the training phase while Environment 2 (Figure 1b) was used in the testing phase. The results for this experiment are shown by Figures 7, 8, 9 and Table III

Figure 7 shows that on average, moGEP and the standard ugGEP evolved the best individual in less than 10 generations.

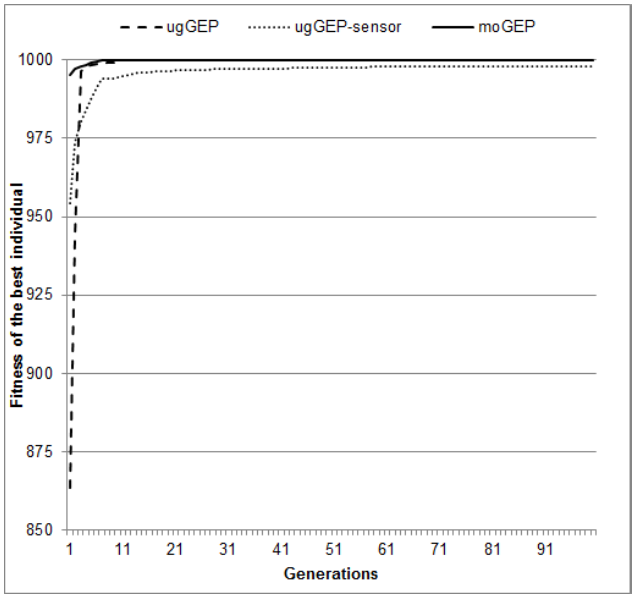


Fig. 7. Progression of best individual in the population as achieved using ugGEP and mgGEP with multiple output. Each individual curve represents the best performing evolutionary run out of the 20 randomly seeded runs.

In addition, moGEP 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 moGEP 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 moGEP shows that GEP has a great potential of evolving self organising mechanism needed for robot control.

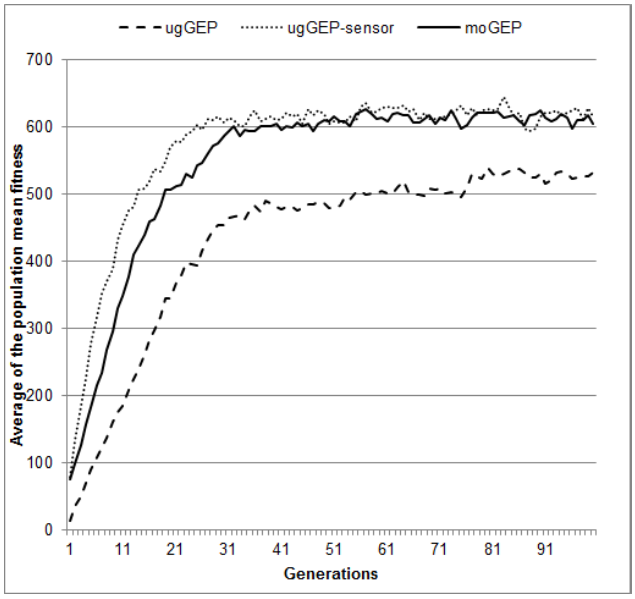


Fig. 8. Mean of the population mean fitness in the population as achieved using ugGEP, ugGEP with sensor based velocity control and moGEP.

The progression of average fitness over generations (Figure

8) shows that the moGEP controllers performed equally as well as the ugGEP approach.

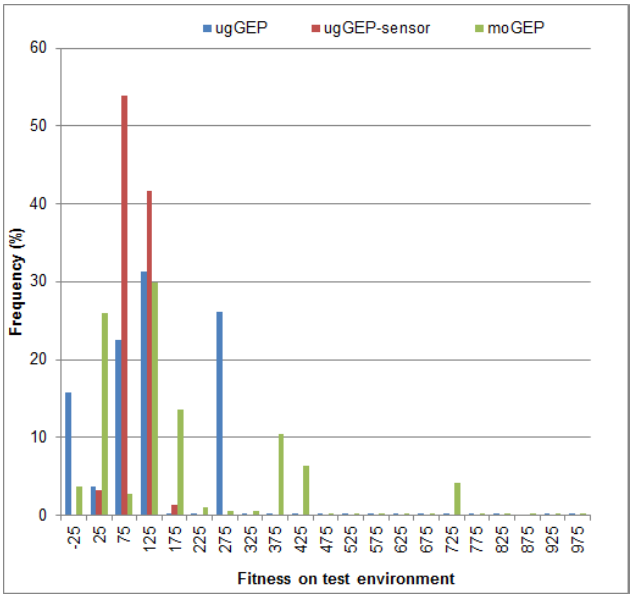


Fig. 9. Comparison of performance of best individuals in new environments.

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 important to note that the robot solves an obstacle avoidance problem in the training environment (Figure 1a, Environment 1) while it is tested to perform a wall following task when introduced to the new environment (Figure 1b, Environment 2). Results shown by Figure 9 suggests that controllers evolved using moGEP 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 moGEP 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 moGEP 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 moGEP and ugGEP with sensor based velocity control, the robot 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 10 shows the results achieved.

Figure 10 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 moGEP. Controllers using the latter two approaches had the

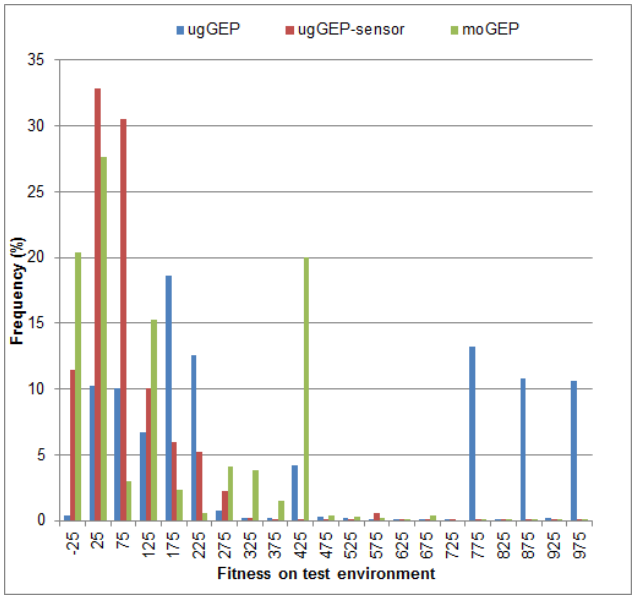


Fig. 10. Comparison of performance of best individuals in new environments.

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 moGEP that can set speed automatically might be better positioned in evolving more robust controllers.

4) Statistical comparison: Table III shows the results of a Mann-Whitney U test between the moGEP 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.

TABLE III. 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%

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 moGEP and the ugGEP approaches" while the alternative hypothesis for the one tailed test (P_1) is that "moGEP 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, moGEP is better than the ugGEP approaches.

V. Conclusion

This paper has introduced techniques of evolving robotic behaviours using GEP. The main aim was to investigate the performance of the more complex multiple-output GEP (moGEP) in evolutionary robotics problems. The performances of both ugGEP and moGEP algorithms in training and test environments were 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.

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.

As shown in the experimentation above, GEP can be used to evolve multigenic chromosomes that can be used to solve problems requiring multiple output. The results show that moGEP was more successful in solving the problem and was also more robust in the new environment. The moGEP technique involves evolving independent modules for each robotic motor functions. As such, increasing robotic capabilities will mean evolving more genes for each motor function. For instance a robot with two wheels and a robotic arm may need three separate genes in the moGEP algorithm. Thus, future work will entail testing the moGEP technique with more complex robotic problems.

References

[1] E. Bautu, A. Bautu, and H. Luchian, "AdaGEP-An Adaptive Gene Expression Programming Algorithm," in *Proceedings of Ninth International Symposium on Symbolic and Numeric Algorithms for Scientific Computing*, 2008, pp. 403–406.

[2] P. Collet, E. Lutton, F. Raynal, and M. Schoenauer, "Polar IFS+Parisian Genetic Programming=Efficient IFS Inverse Problem Solving," *Genetic Programming and Evolvable Machines*, vol. 1, no. 4, pp. 339–361, 2000.

[3] C. Ferreira, "Gene Expression Programming: A New Adaptive Algorithm for Solving Problems," *J. Complex Systems*, vol. 13, no. 2, pp. 87–129, 2001.

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

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

[6] 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.

[7] I. Harvey, P. Husbands, and D. Cliff, "Issues in Evolutionary Robotics," in *Proceedings of the 3^{rd} International Conference on Simulation of Adaptive Behavior*, J.-A. Meyer, H. Roitblat, and S. Wilson, Eds., vol. 2, 1993, pp. 73–110.

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

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

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

[11] ——, "Evolution of Subsumption Using Genetic Programming," in *Proceedings of the First European Conference on Artificial Life: Towards a Practice of Autonomous Systems*, P. B. F. J. Varela, Ed., 1993, pp. 110–119.

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

[13] ——, "Evolution of Robotic Behaviours Using Gene Expression Programming," in *Proceedings of the IEEE Congress on Evolutionary Computation (CEC2010)*, Barcelona, Spain, 2010, pp. 1–8.

[14] ——, "Evolving Modularity in Robot Behaviour Using Gene Expression Programming," in *Towards Autonomous Robotic Systems - 12th Annual Conference, (TAROS 2011), August 31 - September 2*, ser. Lecture Notes in Computer Science, R. Gross, L. Alboul, C. Melhuish, M. Witkowski, T. J. Prescott, and J. Penders, Eds., vol. 6856. Sheffield, UK: Springer, 2011, pp. 392–393.

[15] J. Mwaura, E. Keedwell, and A. Engelbrecht, "Evolved Linker Gene Expression Programming: A new technique for Smbolic Regression," in *Proceedings of the 1st BRICS conference on computational Intelligence*, Porto de Galinhas, Brazil, 2013, pp. 1–8.

[16] J. Mwaura and E. Keedwell, "Evolving robot sub-behaviour modules using Gene Expression Programming," *Genetic Programming and Evolvable Machines*, pp. 1–37, 2014. [Online]. Available: http://dx.doi.org/10.1007/s10710-014-9229-x

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

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

[19] M. Oltean and C. Grosan, "A Comparison of Several Linear Genetic Programming Techniques," *J. Complex Systems*, vol. 14, pp. 285–313, 2003.

[20] M. Oltean, C. Grosan, and M. Oltean, "Encoding Multiple Solutions in a Linear Genetic Programming Chromosome," in *Computational Science - ICCS 2004*, ser. Lecture Notes in Computer Science, M. Bubak, G. van Albada, P. Sloot, and J. Dongarra, Eds. Springer Berlin Heidelberg, 2004, vol. 3038, pp. 1281–1288.

[21] M. L. Pilat and F. Oppacher, "Robotic Control Using Hierarchical Genetic Programming," in *GECCO (2)*, ser. Lecture Notes in Computer Science, vol. 3103. Springer, 2004, pp. 642–653.

[22] J. Walker, J. Miller, and R. Cavill, "A Multi-Chromosome Approach to Standard and Embedded Cartesian Genetic Programming," in *Proceedings of the GECCO Conference (GECCO'06)*. ACM, 2006, pp. 1–8.

[23] J. Walker, K. Vlk, S. Smith, and J. Miller, "Parallel Evolution using Multi-chromosome Cartesian Genetic Programming," *Genetic Programming and Evolvable Machines*, vol. 10, no. 4, pp. 417–445, 2009.

[24] Y. Zhang and M. Zhang, "A Multiple-Output Structure in Genetic Programming," in *Proceedings of the 2004 Asia-Pacific Workshop on Genetic Programming*, R. McKay, Ed., 2004, pp. 1–12.

[25] J. Zhong, L. Luo, W. Cai, and M. Lees, "Automatic Rule Identification for Agent-Based Crowd Models Through Gene Expression Programming," in *Proceedings of the 13^{th} International Conference on Autonomous Agents and Multiagent Systems (AAMAS2014)*, A. Lomuscio, P. Scerri, A. Bazzan, and M. Huhns, Eds., 2014, pp. 1125–1132.

[26] C. Zhou, W. Xiao, T. Tirpak, and P. Nelson, "Evolving Classification Rules with Gene Expression Programming," *IEEE Transanctions on Evolutionary Computation*, vol. 7, 2003.