Robotic Control Using Hierarchical Genetic Programming

Marcin L. Pilat[1] and Franz Oppacher[2]

[1] Department of Computer Science, University of Calgary,
2500 University Drive N.W., Calgary, AB, T2N 1N4, Canada
pilat@cpsc.ucalgary.ca

[2] School of Computer Science, Carleton University,
1125 Colonel By Drive, Ottawa, ON, K1S 5B6, Canada
oppacher@scs.carleton.ca

Abstract. In this paper, we compare the performance of hierarchical GP methods (Automatically Defined Functions, Module Acquisition, Adaptive Representation through Learning) with the canonical GP Implementation and with a linear genome GP system in the domain of evolving robotic controllers for a simulated Khepera miniature robot. We successfully evolve robotic controllers to accomplish obstacle avoidance, wall following, and light avoidance tasks.

1 Introduction

Genetic programming (GP) is a powerful evolutionary search algorithm to evolve computer programs. However, its main drawback is that it tries to solve the entire problem at once. While canonical GP is suitable for smaller problems, it is often not powerful enough to solve difficult real world problems. Hierarchical genetic programming (HGP) tries to overcome the weakness of GP by introducing a divide-and-conquer approach to problems. Instead of solving a complex problem entirely, the method breaks the problem into smaller parts, generates solutions to the parts, and integrates the solutions into the solution to the overall problem. This modularization approach is based on what humans do to solve complex tasks.

Several hierarchical genetic programming methods have been proposed in the literature, each with its own advantages and disadvantages. In our research, we examine the methods of Automatically Defined Functions (ADFs) [4], Module Acquisition (MA) [2], and Adaptive Representation through Learning (ARL) [8].

We study the application of genetic programming techniques to the evolution of control programs for an autonomous miniature robot. We are interested in the evolution of robotic controllers for the obstacle avoidance, wall following, and light avoidance tasks. Nordin and Banzhaf [6] have studied the use of linear genome GP [5] for real time control of a miniature robot. Our research extends the analysis of [6] by carrying out experiments with the ADF, MA, and ARL hierarchical GP methods, the canonical tree-based GP implementation [3], and a variation of the linear genome GP system. Our results enable us to compare the performance of the studied method in the domain of robotic control.

2 Robotic Controllers

Nordin and Banzhaf [6] have experimented with a simulated and real Khepera miniature robot to evolve control programs with genetic programming. They used the Compiling Genetic Programming System (CGPS) [5] which worked with a variable length linear genome composed of machine code instructions. The system evolved machine code that was directly run on the robot without the need of an interpreter.

The initial experiments of Nordin and Banzhaf were based on a memory-less genetic programming system with reactive control of the robot. The system performed a type of function regression to evolve a control program that would provide the robot with 2 motor (actuator) values from an input of 8 (or more) sensor values. GP successfully evolved control programs for simple control tasks of obstacle avoidance, wall following, and light-seeking. The work was extended [6,7] to include memory of previous actions and a two-fold system architecture composed of a planning process and a learning process. Speed improvements over the memory-less system were observed in the memory-based system and the robots exhibited more complex behaviours [6].

The GP system in the robotic controller evolves control programs that best approximate a desired solution to a pre-defined problem. This procedure of inducing a symbolic function to fit a specified set of data is called Symbolic Regression [6]. The goal of the system is to approximate the function: f(s0, s1, s2, s3, s4, s5, s6, s7) = {m1, m2} where the input is comprised of 8 robotic sensors (s0-s7) and the output is the speed of two motors controlling the motion of the robot (m1-m2). The control program code of each individual constitutes the body of the function. The results are compared with a behaviour-based fitness function that measures the accuracy of the approximation by the deviation from desired behavior of the robot.

In our research, we evolve a population of control programs for the Khepera robot. The GP variants studied here are steady-state tournament selection systems with tournament size of 7. Experiments are performed with populations of size 50 to 200. The algorithm executes until desired behaviour is learned or until 300 generations elapse. We use our simulator - Khepera GP Simulator for Microsoft Windows® - for all our experiments. The simulator contains a user friendly interface and allows run-time modification of all simulation parameters, fitness functions, and robotic controllers. The simulator and its C++ source code are available free-of-charge for academic research purposes [1].

2.1 Linear Genome GP

The linear genome GP system with binary machine code was introduced in [5] as Compiling Genetic Programming System (CGPS). The method was used to evolve a robotic controller for Khepera robots [6]. The structure of our linear genome GP controller closely resembles the controller used by Nordin and Banzhaf.

[1] Khepera GP Simulator home page is available at http://www.pilat.org/khepgpsim.

The controller by Nordin and Banzhaf [6] used variable length strings of 32 bit instructions for a register machine performing arithmetic operations on a small set of registers. We represent each instruction as a text string and process it by a Genome Interpreter prior to evaluation. A loss in performance is noticed since processing of the string based instructions is more time intensive than for numerical representations. However, the performance of the text-based representation is sufficient for the purpose of the research and provides greater readability.

Each individual is composed of a series of instructions (genes). The instructions are of the form: $resvar = var1\ op\ \{var2|const\}$ where $resvar$ is the result variable and op is a binary operator working on either two variables ($var1$ and $var2$) or a variable and a constant ($var1$ and $const$). Values of instruction parts are selected randomly from a set of primitive values. Valid variable values are 8 robotic infrared proximity sensors ($s0 - s7$), 8 ambient light sensors ($l0 - l7$) and intermediate variables ($a - f$). The operator set consists of: $add, subtract, multiply, leftshift, rightshift, XOR, OR, AND$. Constants are randomly chosen from the range $0 - 8191$.

The linear genome GP method employs three genetic operators: reproduction, crossover and mutation. Variable length 2-point crossover is applied to the two fittest individuals of a tournament, according to a given probability. Genes are treated as atomic units by the crossover operator and are not internally modified. If crossover is not selected, reproduction occurs and copies of fittest individuals are inserted into the new population. Simulated bit-wise mutation modifies the contents of a gene. We use crossover probability of 0.9 and mutation probability of 0.05.

2.2 Tree-Based GP

The tree-based GP system works with the canonical tree representation of chromosomes. Program trees for the tree-based GP method and all HGP methods are created randomly with the "full" method [3] to generate the initial random population. Maximum tree height at creation is 6 and maximum overall tree height is 10. The standard function set is composed of functions: *Add*, *Sub*, *Mul*, *Div*, *AND*, *OR*, *XOR*, $<<$, $>>$, *IFLTE*. The terminal set consists of variables denoting the robotic sensors ($s0 - s7$ and $l0 - l7$) and constants in range $0 - 8192$.

Reproduction and crossover are the genetic operators for this method. Single subtree switching crossover is applied to the two fittest individuals in a tournament, with a given probability. For the tree-based GP method and all HGP methods, we use crossover probability of 0.9 in our experiments and no mutation.

2.3 Automatically Defined Functions HGP

The Automatically Defined Function (ADF) method in our research is based on the method proposed by Koza [4]. The method automatically evolves function definitions while evolving the main GP program.

The result producing branch is built with the standard terminal set and the standard function set augmented with the ADFs contained in the same chromosome. Separate terminal and function sets are used by the function defining branches to define the ADFs. These secondary sets contain dummy variables to denote the arguments to function calls. All ADFs defined in an individual are only available locally to the program tree of the same individual. We study ADF program trees with one, two, and three ADF definitions and two ADF arguments.

As in the tree-based method, reproduction and crossover operators are used in the ADF chromosomes. To preserve the structure of valid ADF chromosomes, the crossover operator is only allowed to swap non-invariant nodes and nodes of similar type using branch typing.

2.4 Module Acquisition HGP

The Module Acquisition (MA) method of Angeline and Pollack [2] uses two extra operators of compression and expansion to modularize the program code into subroutines. The program tree structure is identical to the structure of the tree-based representation. The function set of each chromosome is extended by parameterized subroutines called modules. Modules are local to a chromosome and are only propagated through the population by reproduction and crossover.

The Module Acquisition method employs four genetic operators: reproduction, crossover, and two mutation operators of compression and expansion. The reproduction and crossover operators are similar to those used by the canonical tree-based representation.

The compression operator creates a new subroutine from a randomly selected subtree of an individual in the population using depth compression [1] of maximum depth range $2 - 5$. A subroutine call replaces the chosen subtree in the program tree and the new module extends the function set of the chromosome. If the chosen subtree is beyond the maximum depth value, a parameterized subroutine is created. The expansion operator is complementary to the compression operator and restores a single module call to its original subtree. We use compression probability of 0.1 and expansion probability of 0.01.

2.5 Adaptive Representation HGP

The Adaptive Representation through Learning (ARL) method by Rosca and Ballard [8] extends the canonical tree-based GP system by introducing parameterized blocks (i.e. functions). Unlike in the ADF approach, the functions are discovered automatically and without human-imposed structure. The method differs from the MA approach by the algorithms for function discovery and the management of function libraries.

The structure of the ARL chromosome program trees is identical to the tree structure of the tree-based GP method. The ARL functions are stored in a global library and are accessible to all the chromosomes in the population.

The function library is dynamically grown by the execution of the evolutionary algorithm. Standard tree reproduction and crossover operators are used.

The implementation relies on differential fitness [8] for function discovery. We do not use the concept of block activation [8] because of the large performance overhead on the system. We employ a simple measure of subroutine utility [8] by assigning to each subroutine an integer utility value. This value is decremented each generation the subroutine is not used until the subroutine is removed from the library.

Population entropy provides a measure of the state of a dynamic system represented by the population and is calculated by using Shannon's information entropy formula [9] on groupings of individuals based on their phenotype. Our entropy calculation is not based on raw fitness values because of the dynamic nature of our fitness calculation. We implement a standardized classification measure to calculate a value for individual classification. The value is calculated using an average of motor output values obtained from three test cases with fixed input sensor values. Individuals are partitioned into 20 categories based on the value of their standardized classification measures. Population entropy value is then calculated by Shannon's formula applied to the categorized chromosomes.

The measure of population entropy is important since it correlates to the state of diversity in the population during a GP run [9]. We use a static entropy threshold value of 1.5 to decide on the initiation of a new epoch and discovery of new subroutines. Candidate blocks of height 3 are selected from most promising individuals. Discovered functions are automatically inserted into the function set. To counteract the drops in population entropy, the ARL method generates random individuals using the new function set. The new individuals are placed into the existing population. The number of new individuals to create is specified by the replacement fraction parameter which is set to 0.2 in our experiments.

3 Results

3.1 Obstacle Avoidance

The task of obstacle avoidance is very important for many real-world robotic applications. Any robotic exploratory behavior requires some degree of obstacle avoidance to detect and manoeuver around obstacles in the environment. We define obstacle avoidance as the behavior to steer the robot away from obstacles. For the Khepera robot, this task is equivalent to minimizing the values of the proximity sensors while moving around the environment.

Our fitness function for obstacle avoidance is based on the work of Nordin and Banzhaf [6]. The pleasure part of the fitness function is computed from motor values and encourages the robot to move around the environment using straight motion. The pain part is composed of sensor values and punishes the robot for object proximity. The fitness function can be expressed as the equation: $fitness = \alpha\ (|m_1|+|m_2|-|m_1 - m_2|)$ - $\beta \sum_{i=0}^{7} s_i$ where m_x are motor values (in range -10 to 10) and s_x are proximity sensor values (in range 0-1023). In our experiments, scale parameters are set to $\alpha = 10$ and $\beta = 1$.

Various robotic behaviours are observed while learning the obstacle avoidance task and are summarized in Figure 1. We subdivide the learned behaviours into groups based on the complexity and success rate of each behavior. The simplest Type 1 behaviours (straight, curved) are solely based on the simple movement of the robot with no sensory information. The second level of behaviour, Type 2, is composed of circling, bouncing, and forward-backup behaviours. The highest level of behaviour, Type 3, is called sniffing and it involves processing sensor data to detect and avoid obstacles. Avoidance is learned either by circling away from an obstacle or by backup behaviour. The perfect sniffing behaviour produces smooth obstacle avoidance motion around the entire testing environment.

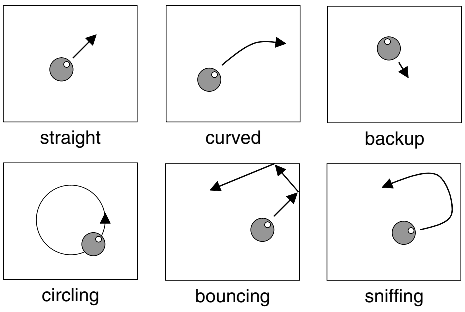


Fig. 1. Summary of learned behaviours.

We used the measures of entropy stability and average chromosome size stability to compare the studied methods. We define stability by a gradual change of measured values and lack of large abrupt changes. For the obstacle avoidance task, the ARL method provides the most stable entropy values. The MA and tree-based methods provide the worst stability with large drops of entropy values. Best chromosome size stability is seen with the linear genome method and worst with the MA method. Among the HGP methods, the most stable method is the ARL method.

The results with Type 2 and 3 behaviours are processed to calculate average generation values of first occurrence of the stated behaviour. Summary of the results of our behaviour calculation can be found in Figure 2. The method with best initial behaviour occurrence values is the ARL HGP method and with the worst is the linear genome GP method. Overall, the HGP methods perform comparable with the tree-based GP method. A sample experimental trace of evolved perfect obstacle avoidance behaviour can be seen in Figure 3.

3.2 Wall Following

The task of wall following allows the robot to perform more difficult and interesting behaviours such as maze navigation. The purpose of the task is to teach

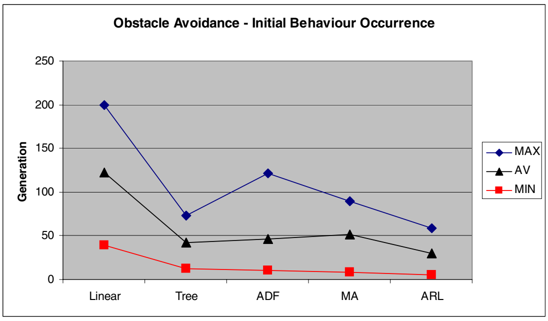
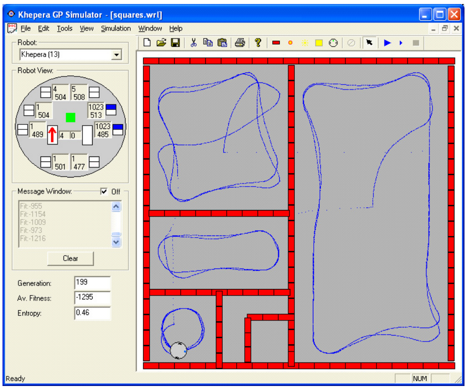


Fig. 2. Graphs of minimum, maximum, and average generations of first detection of Type 2 and 3 obstacle avoidance behaviour for each chromosome representation method.



Khepera (13)

4 504 | 5 508 | 1 504 | 1023 513 | 1 489 | 4 | 0 | 1023 485 | 1 501 | 1 477

Fit:-955
Fit:-1154
Fit:-1009
Fit:-973
Fit:-1216

Generation: 199
Av. Fitness: -1295
Entropy: 0.46

Fig. 3. Trace runs of perfect evolved obstacle avoidance behaviour in various testing environments.

the robot to move around the environment with a certain desirable distance away from obstacles. The learned task should include some obstacle avoidance behaviour; however, that is not the main requirement of the experiments.

The wall following fitness function is composed of sensor and motor parts. The sensor part is computed from a subset of the robotic sensor values and the motor part is defined as the absolute motor value sum minus the absolute value of the difference. The fitness function can be summarized as: Fitness = $\alpha \cdot$ $MotorPart$ + $\beta \cdot$ $SensorPart$. The calculated sensor part value acts as either pleasure or pain depending on the values of the sensors. Thus, the robot is punished when it is either too far away from an obstacle or too close to it.

Summary of the behaviours obtained is provided in Figure 1. We categorize the behaviours based on their relative performance and success. The first Type 1 category is of poor wall following behaviour and consists of simple wall-bouncing and circling behaviours. The Type 2 category of good wall following behaviour consists of wall-sniffing and some maze following. The best behaviour category, Type 3, consists of perfect maze following without wall touching and usually taking both sensor sides of the robot into consideration.

The most stable entropy behaviour is noticed in experiments using the ARL method whereas the least stable is observed using the ADF method. Most stability of average size values is seen in the linear genome and ARL methods. The largest drops in average chromosome size are noticed with the MA method.

Type 2 and 3 behaviour category results are processed to calculate average generation values of first occurrence of the stated behaviour. Summary of the results can be found in Figure 4. The ARL method produces the best overall results with the smallest deviation. The worst performance and largest deviation is seen with the MA method. A sample experimental trace of evolved perfect maze following behaviour can be seen in Figure 5.

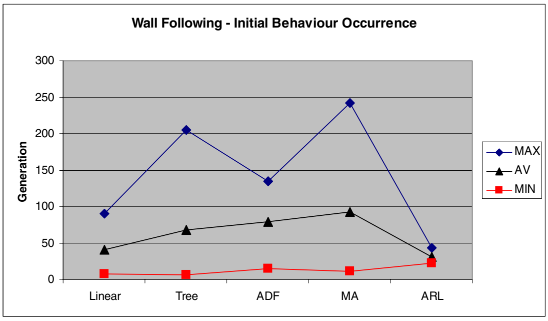
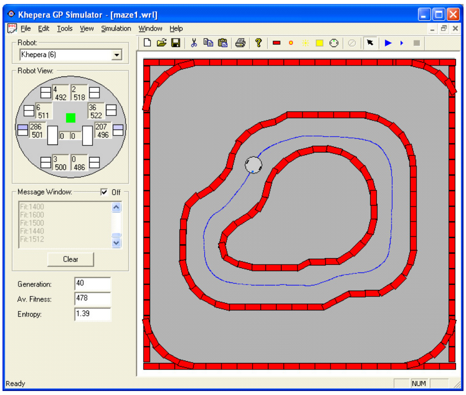


Fig. 4. Graphs of minimum, maximum, and average generations of first detection of Type 2 and 3 wall following behaviour for each chromosome representation method.


Khepera GP Simulator - [maze1.wrl]
File Edit Tools View Simulation Window Help
Robot View:
Message Window:
Clear
Ready
NUM

Robot: Khepera (6)

4 492 | 2 518 | 6 511 | 36 522 | 286 501 | 207 496 | 0 | 0 | 3 500 | 0 486

Message Window: ✓ Off

Fit:1400
Fit:1600
Fit:1500
Fit:1440
Fit:1512

Generation: 40
Av. Fitness: 478
Entropy: 1.39

Fig. 5. Trace run of perfect evolved maze-following behaviour.

3.3 Light Avoidance

The light avoidance task is similar to the obstacle avoidance task but uses the ambient light sensors of the robot instead of the proximity sensors. Light sources in the training and testing environments are overhead lamps that cannot be touched by the robot. The robot must learn to stay inside an unlit section of the world environment while moving as much as possible.

The fitness function for light avoidance is similar to the fitness function used for obstacle avoidance. The function contains a pleasure part computed from the motor values of the robot and a pain part computed from the light sensors. The function can be expressed as: Fitness $= \alpha$ ($|m_1|$+$|m_2|$-$|m_1 - m_2|$) - β (4000 - $\sum_{i=0}^{7} l_i$) where m_x are motor values and l_x are ambient light sensor values (in range 0-500). In our experiments, scale parameters are set to $\alpha = 10$ and $\beta = 1$. The constant 4000 denotes the highest possible sensor sum and transforms the fitness function to behave similar to the fitness function of the obstacle avoidance task.

We subdivide the learned behaviours into two categories. The Type 2 category consists of behaviours with some degree of light detection and avoidance. The Type 3 behaviour category consists of behaviours with definite light detection and avoidance. Perfect behaviour usually consists of movement around the boundary of the dark area in the testing environment.

The most stable entropy behaviour is noticed with the linear genome method and worst stable with the ADF method. Most stable average size values are observed using the ARL and tree-based methods. The worst average size behaviour is seen with the MA method.

Type 2 and 3 categories are processed to calculate average generation values of first occurrence of the stated behaviour. Summary of the results can be found in Figure 6. The best results come from experiments with the ARL method and the worst from linear genome experiments. The HGP methods perform comparable to the tree-based method. A sample experimental trace of perfect evolved light avoidance behaviour can be seen in Figure 7.

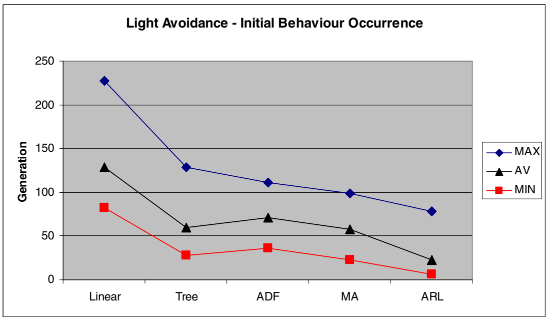
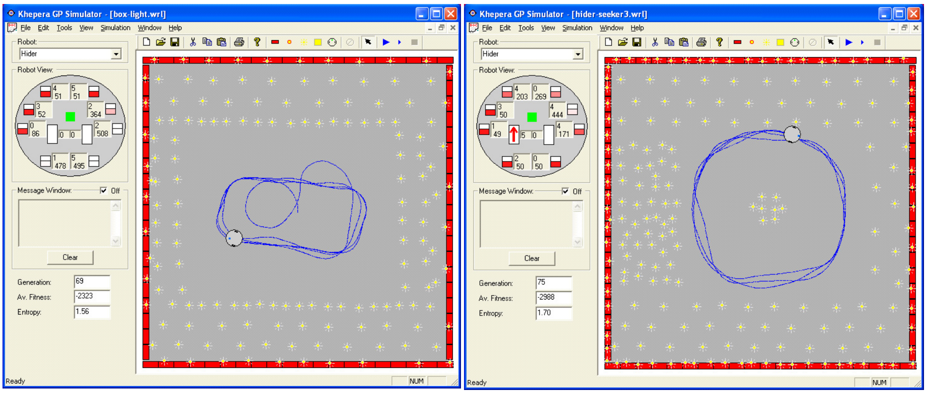


Fig. 6. Graphs of minimum, maximum, and average generations of first detection of Type 2 and 3 light avoidance behaviour for each chromosome representation method.


Khepera GP Simulator - [box-light.wrl]
File Edit Tools View Simulation Window Help
Robot View:
Message Window:
Off
Clear
Ready
NUM
Khepera GP Simulator - [hider-seeker3.wrl]
File Edit Tools View Simulation Window Help
Robot View:
Message Window:
Off
Clear
Ready
NUM

Robot: Hider

Generation: 69

Av. Fitness: -2323

Entropy: 1.56

Robot: Hider

Generation: 75

Av. Fitness: -2988

Entropy: 1.70

Fig. 7. Trace runs of perfect evolved light avoidance behaviour in different testing environments.

4 Conclusion

Our research is concerned with the evolution of robotic controllers for the Khepera robot to perform tasks. The reactive robotic control problem provides a challenge to the genetic programming paradigm. With the lack of test cases for fitness function evaluation, the fitness of an individual can differ greatly depending on the immediate neighbourhood of the robot. From tuning experiments, we notice that the definition of the fitness function can greatly influence the population contents and thus the resulting behaviours. Because of the constantly changing local environment, even good performing behaviour can eventually be replaced by worse behaviours. We feel that more testing of fitness functions and their parameters should be done to identify the optimal fitness function definition for each learning task.

We have noticed that the robotic controllers often over-adapt to the training environment. This problem of overfitting is a common problem in genetic programming. We notice that sharp corners of the environment form an area of difficulty for the robotic controller. This difficulty is probably due to the corner fitting between the fields of view of the proximity sensors.

The entropy value is an important indicator of population diversity in our experiments. We notice that best behaviours are observed in populations with relatively high entropy value (above 0.6). Low entropy value signifies convergence in the population which usually accompanies a convergence to a low average chromosome size. Populations of suboptimal individuals with low chromosome size do not contain enough information to successfully search for a good solution.

We study three HGP learning methods: Automatically Defined Functions, Module Acquisition, and Adaptive Representation through Learning. Robotic controllers using each method are able to evolve some degree of proper behaviour for each learning task. Summary of method performance is available in Table 1.

The ADF method uses a predefined, constant function set containing one or more ADFs. The only method of function call acquisition is through crossover with another individual of the population. The ADFs inside individuals showing nice behaviour are usually quite large and complex with no noticeable patterns. It is possible that in our experiments, the ADFs have not much purpose other than to provide few extra tree levels of instructions. In our experiments, the ADF method performance was usually below that of the tree-based method. We notice that the ADF trees do not shrink enough before the population prematurely converges. We think that it would be best to specify a smaller initial and maximum size of the ADF trees so that the functions require less time to find optimal configurations.

Table 1. Summary of results from our experiments for each of the studied methods. Behavioural performance was based on first occurrence of good behaviour. Performance of each method is scored relative to performance of other methods.

Method	Entropy Stability	Size Stability	Behavioural Performance
Linear GP	excellent	excellent	poor
Tree GP	average	average	average
ADF HGP	poor	average	average
MA HGP	average	poor	average
ARL HGP	excellent	excellent	excellent

The slowest function set growth is observed with the MA method. Most of the individuals in the populations with good behaviour do not use any of the functions in the module set. The creation of functions produces program size loss which in turn often lowers the entropy of the population. The behavioural performance of the MA method is usually worse than that of the tree-based method. Since no strong mechanism exists to counteract the loss of program size and accompanying loss of entropy, the population often converges prematurely to suboptimal solutions. Probability-based compression and expansion operator invocation might be replaced by need-based operator invocation similar to those found in the ARL method. This new operator invocation should lead to better behaviour through adjustments of operator frequencies based on population needs.

The ARL method displays the most stable entropy and average chromosome size behaviour in most experiments. This behaviour is observed only when function creation occurs, thus we think that the function creation and new individual creation processes are responsible for the stability. The method also achieves the best time and smallest deviation to reach good behaviour in most experiments. The size of the function set of the ARL method is very dynamic through the runs of the algorithm. Many of the functions from populations with good behaviour seem to calculate some form of ratio of the function parameters, thus, we think that some of the evolved functions were of benefit to the individuals. Influx of random individuals to the population during evolution can lead to problems since too many random individuals would destabilize good solutions present in individuals of the previous population. We think that a low replacement fraction used with elitism of best individuals should produce the optimal evolutionary balance. Elitist individuals would always be copied into the population and would ensure the fittest individuals are not lost between generations.

Our results indicate that reactive memoryless controllers can be trained to exhibit some level of proper behaviour for the studied tasks. An extension to this research would be to study memory-based robotic controllers that can store previous actions and use them to decide future behaviour. Such controllers using the linear genome method have been shown in [6] to successfully and quickly evolve more complex behaviours than a memoryless controller.

We would also like to use a real Khepera robot to verify our results. Physical robots train in a noisy and sometimes unpredictable environment and would provide a real world test case for our research.

References

1. Angeline, P.J.: Genetic Programming and Emergent Intelligence. In K.E. Kinnear, Jr. (ed.): Advances in Genetic Programming, chapter 4. MIT Press (1994) 75–98
2. Angeline, P.J., Pollack, J.B.: The Evolutionary Induction of Subroutines. In: Proceedings of the Fourteenth Annual Conference of the Cognitive Science Society. Lawrence Erlbaum, Bloomington, Indiana, USA (1992)
3. Koza, J.: Genetic Programming: On the Programming of Computers by Means of Natural Selection. MIT Press, Cambridge, MA (1992)
4. Koza, J.: Genetic Programming II: Automatic Discovery of Reusable Programs. MIT Press, Cambridge, MA (1994)
5. Nordin, P.: A Compiling Genetic Programming System that Directly Manipulates the Machine-Code. In K.E. Kinnear, Jr. (ed.): Advances in Genetic Programming, chapter 14. MIT Press, Cambridge, MA (1994) 311–331
6. Nordin, P., Banzhaf, W.: Real Time Control of a Khepera Robot using Genetic Programming. Cybernetics and Control. **26**(3) (1997) 533–561
7. Nordin, P., Banzhaf, W., Brameier, M.: Evolution of a world model for a miniature robot using genetic programming. Robotics and Autonomous Systems. **25**(1-2) (1998) 105–116
8. Rosca, J.P., Ballard, D.H.: Discovery of Subroutines in Genetic Programming. In P.J. Angeline and K.E. Kinnear, Jr. (eds.): Advances in Genetic Programming 2, chapter 9. MIT Press", Cambridge, MA, USA (1996) 177–202
9. Rosca, J.P.: Entropy-Driven Adaptive Representation. In J.P. Rosca (ed.): Proceedings of the Workshop on Genetic Programming: From Theory to Real-World Applications. Tahoe City, California, USA (1995) 23–32