Using Non-Determinism to Improve the Robustness of Robot Programs Generated by Genetic Programming

Thanut Sukkarnjanonth[1] and Prabhas Chongstitvatana[2]
Department of Computer Engineering
Chulalongkorn University
Bangkok, Thailand
Email: g41tsr@cp.eng.chula.ac.th[1], prabhas@chula.ac.th[2]

Abstract: Robustness is essential for programs generated by Genetic Programming (GP). This paper presents a method to improve the robustness. The method employs non-determinism in two ways: one is to evolve robot programs in noisy environments and another is to use probabilistic branch in the function set. The experiment is carried out on robot navigation problems. The result of the experiment shows that the robustness of robot programs has been improved. The analysis shows that the robustness is caused by the acquired "experience" and the amount of reuse of this experience while performing the task.
Key words: Robustness, Robot programs, Non-determinism, Evolutionary Computation, Genetic Programming

1. Introduction

Artificial Intelligence (AI) has been widely used to automatically generate programs. Genetic Programming (GP) [1] is one of methods that becomes popular in automatic generating programs.

Our experiment use GP to generate programs for controlling a robot. GP is performed in a simulation because of the speed limit of real robots. The robot programs work successfully in the environment that they are evolved on, but when they are transferred to the real environment some programs may not work properly. This failure comes from many factors. For example, they fail to work when the environment is changed. The slight changing between the real environment and the simulated environment can lead to failure. Therefore, programs generated by GP are brittle or lack of robustness.

The goal of this paper is to improve the robustness. We present an approach that combines two methods, both of which can be regarded as using non-determinism. The first method presented in [3], [5] uses perturbation to create multiple environments that are used to evolve robot programs. The second method presented in [4] uses probabilistic branch in the function set to create "variety" of robot actions when facing an unknown environment.

This paper is structured as follows: a short introduction to GP and review of related works is given in the Section 2. The robot navigation problems and GP set-up are described in Section 3. Section 4 explains how to improve the robustness. The result of this experiment is presented in Section 5. Section 6 analyses the robustness. Section 7 is the conclusion of our work and discusses the future work.

2. Background

First subsection explains the overview of Genetic Programming. Second subsection discusses other related works.

2.1. Genetic Programming

Genetic Programming is an extension of the Genetic Algorithm [2]. GP is a technique for automatic generation of computer programs to solve a specified problem without explicit programming. It does this by performing a search and using operations that are inspired by natural evolution. There are three main phases in the GP technique. First, GP constructs an initial population (programs). Second, evaluating each programs in the population for fitness values. Third, reproducing better programs, using genetic crossover and mutation to create new programs. Phase 2 and 3 are iterated until the solution is found or the terminating criteria are met.

2.2. Related Works

In [3], GP was used to generate robot programs that control the robot in a navigation problem. Perturbation has been used to improve the robustness by inject it into the simulation during the evolutionary process, resulting robot programs are

more robust as they have evolved to tolerate change in the environments. The result of the experiment shows that the robustness depends on the size of experience that the robot program acquired during evolutionary process.

In [4], adding a special probabilistic function, 2-way branch, to improve the robustness. The result indicates that robustness is improved by probabilistic function. The analysis shows that robot programs that express variety of behaviors in the different environment have more robustness.

There are many other attempts to enhance robustness of programs. Reynolds [6] produced programs to control a robot moving in a narrow corridor by adding noise in robot operations. The result shows programs are not robust since there are no programs that control robot moving along corridor without crash. They concluded that the problem is very difficult or is caused by the improper use of GP parameters. Ito, Iba, and Kimura [7] used special branching functions to enhance robustness in the box moving problem. In order to test robustness of generated programs, they change the initial condition at each generation and added noise to robot's sensors. They concluded that robot behave robustly due to the redundancy of the programs. McNutt [8] used co-evolution to create robot programs. The result indicates 98% of robot programs can navigate the course successfully.

3. The Experiment

This section describes robot navigation problems, the details of GP run for this problem without any method to improve robustness, and how to evaluate robustness.

3.1. Robot Navigation Problems

The task is to control a mobile robot from a starting point to a target point in an environment full of obstacles. The simulation is used. The size of the simulated environment is 550 x 750 units surrounded by the wall. There are many obstacles. The obstacles have four geometrical shapes: right-angled triangle, rectangle, circle, and hexagon. Each shape has average size 20 x 20 units. The total obstacle area is 20 percent of the whole area. The mobile robot is a circle with a 5 units radius.

3.2. Implementation details of GP

To solve this problem, GP is used to generate robot controlling program. The implementation details of GP are described as follows. The terminal set that use in this problem is

$$T = \{\texttt{left}, \texttt{right}, \texttt{forward}, \texttt{isnear}\}$$

`left` rotates a robot anticlockwise and `right` rotates a robot clockwise. Both terminals change robot direction by 22.5^o degree. `forward` moves a robot in forward direction by 1 unit. `isnear` outputs 1 if the robot is moving nearer to the target compared to the previous move and 0 otherwise.

The function set comprises of

$$F = \{\texttt{if-and}, \texttt{if-or}, \texttt{if-not}\}$$

There are three standard functions with the arity 4, 4, and 3 respectively. The semantic of these three functions are as same as basic control flow statement in programming language. Figure 1 shows a part of generated program.

```
...(if-not   forward   left   (if-and
forward isnear right forward))...
```

Figure 1: A part of robot program

The fitness function is given by:

$$f = (10{,}000 \times D) + T \qquad (1)$$

where D is Euclidean distance from robot position to target position after the program stops and T is number of symbols (terminal and function) that is executed.

For parameter setting, we use 1,500 programs for the population. The reproduction has been applied at 10% rate, the crossover rate is 90 %, the average initial program size is 180 symbols, and the mutation operator is not used. The maximum number of generation is 100. The criteria to stop execution are when the number of executed symbols exceeds 10,000 symbols or the robot reaches the target point. Figure 2 shows a simulated environment and the robot motions.

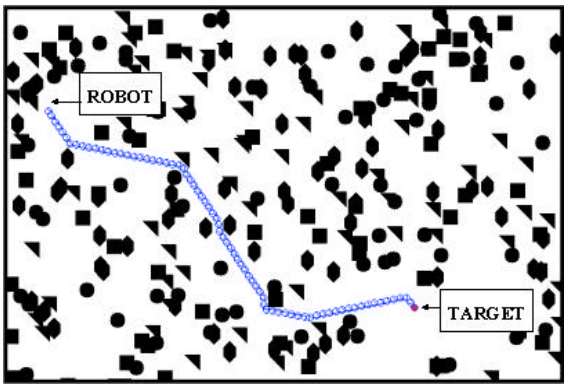

Figure 2: The simulated environment and the robot motions

3.3. Robustness Testing

To test the robustness of programs, we create the new environment perturbed from the original environment by randomly changing obstacle positions. The displacement is eight units. We define the robustness as an ability of a program to control the robot to reach the target in these new environments. Robustness value can be calculated from:

$$Robustness = \frac{Numbers\,of\,successful\,run}{Total\,number\,of\,environment} ? 100 \quad (2)$$

10,000 different environments are used to test the robustness of robots program. The new environments are classified into 10 categories, determined by the percent of disturbance from 10% to 100%, which each category has 1,000 environments. The percent of disturbance is the number of obstacles that have been moved.

4. Improving Robustness

In this section we describe the use of non-determinism to improve robustness. The method employs non-determinism in two ways: one is to evolve robot programs in noisy environments and another is to use probabilistic branch in the function set.

In ordinary evolutionary process, GP is evolved in a static environment. To construct a dynamic environment, perturbation is injected into an environment then use this noisy environment as multiple environments to evolve solutions. After analysing the data from [5], this work uses 30 environments with 30% disturbance. The fitness is computed by averaging fitness value from every environment.

A probabilistic branch promotes variation in robot actions. The `eio2` function branches to one or another path based on a flip of an unbiased coin. When running a program which has this function, trajectory of the robot varies from run to run. Consequently, we evaluate and compute average fitness value over six runs in each environment.

The above two forms of non-determinism can be combined as they are of different types. The first one has a "deterministic" program that has been evolved to tolerate the changing environment. The second one has a "non-deterministic" program that can act differently to anticipate the change in an environment.

Combining the two can be achieved by evolving robot programs in multiple environments and use `eio2` in the function set. One program evolves in 30 environments with 30% disturbance. In each environment the evaluation is repeated six times. Totally one program runs 180 times for fitness evaluation. Fitness value will be calculated by averaging these fitness values.

5. Results

In order to confirm the effectiveness of the combined method. We run the experiment to compare the robustness of four methods: (1) Normal, GP with no improving method, (2) Perturbation, GP using 30 environments with 30% disturbance (3) EIO, GP using special function `eio2` and, (4) P+EIO, GP using both Perturbation and `eio2`.

We run each method 20 times with the different initial population. The robustness of each method is computed by using the average of robustness from every run, categorized by percent of disturbance. The robustness curve of this experiment is shown in Figure 3.

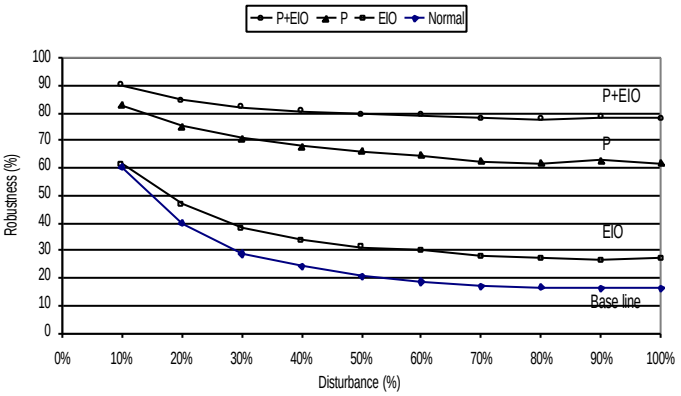

Figure 3: The robustness curve

As shown in Figure 3, the lowest curve represents Normal method. We use this curve as a base line to compare other methods. EIO is about 10% better than the base line. Perturbation is about 40% better than the base line. P+EIO is the best with 60% improvement over the base line. Hence, the combined method is very robust. Note that the more

disturbance, the less is robustness and the more gap between each method.

6. Analysis of Robustness

In this section, we define 'trace' of programs and use it to explain 'experience' which is the cause of robustness.

6.1. Trace

The programs that created from GP are trees. A trace is the executed path on a tree from root to leaf. A program may have many traces, depending on what path is executed. Figure 4 illustrates a structure of a robot program.

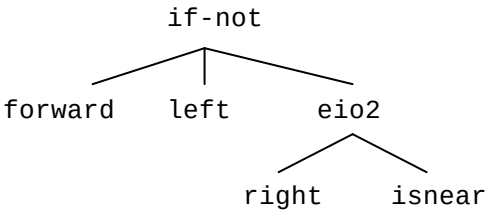

Figure 4: A robot program

The robot program in Figure 4 has possible 3 traces: `{if-not forward left}`, `{if-not forward eio2 right}`, `{if-not forward eio2 isnear}`. When running a program in an environment we will acquire a particular set of traces.

6.2. Experience

The experience is the reuse of the trace acquired during evolution (in the final generation) while navigating in the testing environment, depicted as *S* in Figure 5 shown below.

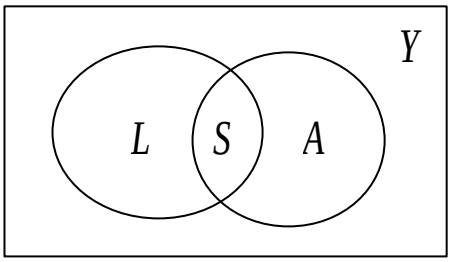

Figure 5: The Experience

Set *Y* denotes to all possible traces, which a program has. Set *L* denotes the set of trace acquired during evolution. Set *A* denotes the trace in testing environments. The intersection, *S* is the experience. The experience is calculated from the following formula.

$$Experience = \frac{|L \cap A|}{|A|} \times 100\ \% \qquad (3)$$

Figure 6 depicts the experience of each method grouped by percent of disturbance. From Figure 6, the method P+Eio has highest experience followed by Perturbation, Eio and Normal respectively. This result corresponds to the robustness curve. Therefore we found the relation of the experience and the robustness. The correlation coefficient between experience and robustness is 0.99537.

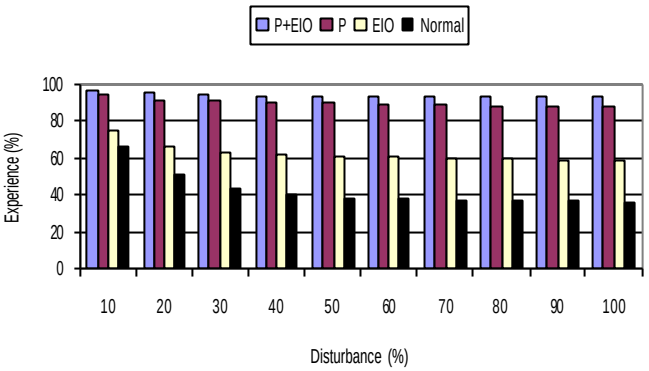

Figure 6: Comparing experiences of the four experiments

7. Conclusion and Future Work

This paper presented the use of non-determinism to improve robustness. The technique is the combination of using perturbation and probabilistic branch. The result shows the merit of this combination over using each technique alone. The analysis shows that the robustness can be attributed to the use of experience acquired during the evolution. Our future research will be concentrated on testing this concept on a real robot.

8. References

[1] Koza, J. R. Genetic Programming: On the programming of computers by means of natural selection, MIT Press, 1992.

[2] Holland, J. H. Adaptation in Natural and Artificial Systems, University of Michigan Press, 1975.

[3] Chongstitvatana, P. Using Perturbation to Improve Robustness of Solutions Generated by Genetic Programming for Robot Learning, Journal of Circuits, Systems, and Computers, Vol. 9: pp. 133-143, 1999.

[4] Prateeptongkum, M. and Chongstitvatana, P. Improving the robustness of a genetic programming learning method by function set tuning, In Proc. of Third Annual National Symposium on Computational Science and Engineering (ANSCSE99), pp. 301-305, 1999.

[5] Nopsuwanchai, R. and Chongstitvatana, P. Analysis of robustness of robot programs generated by genetic programming, In Proc. of

First Asian Symposium on Industrial Automation and Robotics, 1999.

[6] Reynolds, C. W. Evolution of obstacle avoidance behavior using noise to promote robust solution, Advance in Genetic Programming, pp.221-241, MIT Press, 1994.

[7] Ito, T. Iba, H. and Kimura, M. Robustness of robot programs generated by genetic programming, In Proc. Conf. Genetic Programming 96, MIT Press, 1996.

[8] McNutt, G. Using Co-Evolution to Produce Robust Robot Control, In Proc. Conf. Decision and Control 36th, 1997.