Behavior Evolution of Autonomous Mobile Robot Using Genetic Programming Based on Evolvable Hardware

Dong-Wook Lee, Chang-Bong Ban, Kwee-Bo Sim
School of Electrical and Electronic Engineering
Chung-Ang University
Seoul 156-756, Korea
kbsim@cau.ac.kr

Ho-Sik Seok, Kwang-Ju Lee, Byoung-Tak Zhang
Department of Computer Engineering
Seoul National University
Seoul 151-742, Korea
btzhang@scai.snu.ac.kr

Abstract

This paper presents a genetic programming based evolutionary strategy for on-line adaptive learnable evolvable hardware. Genetic programming can be useful control method for evolvable hardware for its unique tree structured chromosome. However it is difficult to represent tree structured chromosome on hardware, and it is difficult to use crossover operator on hardware. Therefore, genetic programming is not so popular as genetic algorithms in evolvable hardware community in spite of its possible strength. We propose a chromosome representation methods and a hardware implementation method that can be helpful to this situation. Our method uses context switchable identical block structure to implement genetic tree on evolvable hardware. We composed an evolutionary strategy for evolvable hardware by combining proposed method with other's striking research results. Proposed method is applied to the autonomous mobile robots cooperation problem to verify its usefulness.

1 Introduction

This paper present a method for evolving genetic programs on evolvable hardware. Evolvable hardware is a hardware designed to adapt to change in task requirements or changes in the environment, through its ability to reconfigure its own hardware structure dynamically and automatically[1]. Evolvable hardware uses evolutionary methods such as genetic algorithms or genetic programming for dynamic reconfiguration[2]. We select genetic programming for our evolutionary method. Genetic programming is an automatic programming method that finds the most fit computer programs by means of natural selection and genetic[3]. It is possible to understand that each subtree of a genetic tree represents a state of whole process. Therefore, it may be reasonable choice to use genetic programming. But, genetic programming has a major disadvantage as a learning rule for evolvable hardware. It is difficult to represent tree-structured chromosome in hardware. To overcome this problem, we make a context switchable identical block structure. In our structure, identical blocks are distributed on hardware. In run time, these blocks are converted to one of predetermined function blocks. By connecting these blocks, a genetic tree is represented on hardware. If the wanted genetic tree is too large to be represented in the evolvable hardware at hand, one subtree of whole tree is located in the hardware. We select autonomous mobile robot control problem to experiment our method in real world.

This paper is organized as follows. Section 2 reviews some studies of other researchers about evolvable hardware. Section 3 explains our method in the view of genetic programming. Section 4 explained autonomous robot control problem. In section 5, implementation details are explained and some experimental results are shown. Section 6 summarizes the result and points out some future work.

2 Evolvable Hardware

Evolvable hardware is reconfigurable hardware whose configuration is under the control of an evolutionary algorithm[4]. Usually, genetic algorithms and genetic programming are used as evolutionary algorithms for evolvable hardware. Due to its run-time reconfigurability, evolvable hardware is suitable for adaptive systems, fault-tolerant systems, and design automation. In this case, adaptive systems mean hardware systems able to reconfigure themselves through learning so as to adapt to environmental change or job change.

Many researchers prefer genetic algorithms. Why do researchers prefer genetic algorithms? The reasons are following. First, genetic algorithms can use configuration bit stream of a base architecture as chromosome. Second, while genetic algorithms is simple and practically blind mechanism of Nature, it can be easily realizable in hardware[5]. But, there are some researchers who attempted to use genetic programming as an evolutionary mechanism. Koza used genetic programming to evolve sorting network[6]. The

0-7803-6583-6/00/$10.00 © 2000 IEEE

advantage of genetic programming opposite to genetic algorithms is that genetic programming can use its function node as a function block of a circuit design. Therefore genetic programming based search can avoid unnecessary search step of genetic algorithms: evolution of function block from randomly distributed configuration bit stream. But it is very difficult to implement tree-structured chromosome on hardware. Therefore there is a need to research a method that can overcome this problem.

3 Genetic Programming

Genetic programming is a stochastic search method suitable for addressing inductive learning tasks. Inspired by evolutionary process in natural living organisms, the genetic programming system maintains a population of programs to accomplish robust search. Genetic programming uses tree-structured chromosome inspired by the functional programming of LISP[7][8][9]. An illustrative example of the genetic program is shown in Fig. 1. Tree-structured form represents the interpretation flow of input data. Genetic programming searches wanted solution using following procedure. First, a population of chromosomes is randomly created to represent a pool of candidate solution. Second, they are assigned fitness value based on how close they come to the wanted function. Third, chromosomes are selected based on their fitness value and they are modified using genetic operators such as crossover and mutation. Fourth selected chromosomes comprise a new generation and are assigned new fitness values. This step is repeated until termination condition is satisfied. More detailed explanation about each step of genetic programming is followed. Explanation is associated with our application - autonomous robot control problem.

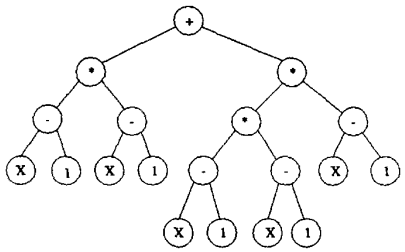

Fig. 1. An Example genetic tree. This genetic tree means $(x-1)^2+(x-1)^3$. As known from fig. 1, genetic programming uses tree structured chromosome which corresponds to certain process of a program.

Preparatory steps for genetic programming runs begin with the definition of function set and terminal set.

- Function Set

Function set is responsible for interpreting sensory data.

- ***IF-OBJ*** is a two-output conditional branching function that executes its first branch if robot find object, but otherwise executes its second branch.
- ***IF-GOAL*** is a two-output conditional branching function that executes its first branch if robot find goal identified as light source.
- ***IF-FORWARD*** is a two-output conditional branching function that executes its first branch if robot find goal and if object is located between goal and front of the robot.
- ***IF-OBS 1~4*** is a two-output conditional branching function that executes its first branch if one of four sets of sensor indicates existence of obstacle.

- Terminal Set

Terminal set corresponds to robot actions.

- ***MOVE FORWARD*** moves the robot forward that the robot is currently facing. (Facing means the direction of robot's linear vision sensor)
- ***TURN LEFT and MOVE FORWARD*** changes the direction of the robot by 90 degrees counter-clockwise and moves forward.
- ***TURN RIGHT and MOVE FORWARD*** changes the direction of the robot by 90 degrees clockwise and moves forward.
- ***MOVE BACKWARD*** moves the robot backward.
- ***TURN LEFT*** changes the direction of the robot by 90 degrees counter-clockwise.
- ***TURN RIGHT*** changes the direction of the robot by 90 degrees clock-wise.

- Crossover and Mutation

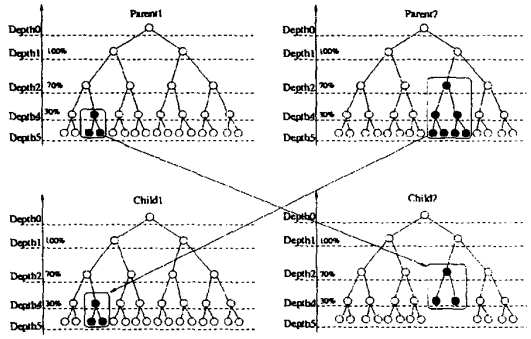

Fig. 2. This shows simple depth-dependant crossover. Depth-dependant crossover assigns crossover probability to each depth. Crossover point is determined according to crossover probability. This figure is modified figure of Fig. 16.1 of [11]

Crossover is a major operator of evolutionary methods. The traditional view is that crossover is primarily responsible for improvements in fitness[10]. Fig. 2 shows illustrative example of crossover. Crossover modifies chromosome by exchanging a subtree of one parents tree with a subtree of another parents tree. Crossover is essential for generating a solution program in genetic programming. But, it has a severe problem. The normal crossover operator select a crossover point randomly regardless of its position within the tree structure. Thus crossover has the possibility of destroying building blocks[11]. To overcome this problem, there is a need to improve random selection property. For this purpose, we use depth-dependant crossover in which a crossover point is determined by a depth selection probability[11]. Ito proposed fine-tuning depth-dependant crossover but we use predefined selection probability. In our simple depth-dependant crossover, crossover probability is assigned before execution and remains unchanged during experiment execution. As one can know from Fig. 2, selection sbutree can overrun the depth limit(subtree of parents 2 has overrunned the depth limit, so the overruned part is erased). In this case, the overrunned part should be erased to keep the depth limit. At present research, we select one subtree of the overrunned subtree and exchange the selected tree with the overruned tree. Subtree selection is performed randomly.

Mutation modifies chromosome by changing one of node to another kind of nodes. Koza has argued that mutation is in fact useless in genetic programming because of the position-independence of genetic programming subtrees, and because of the large number of chromosome positions in typical genetic programming populations[3]. But mutation is an necessary operator for finding hiding nodes, and mutation is often better for small populations, depending on the domain[10].

- Fitness Function

Fitness is the measure used by genetic programming during simulated evolution of how a program has learned to predict the outputs from the inputs - that is, the fitness of the learning domain[12]. Each chromosome has a corresponding fitness value and is selected for next generation using its fitness value. But, some problems such as robot control require at least two kind of fitness function. Existing genetic programming methods are practical enough to find an optimal solution in this domain. To evolve this kind of complex behavior we use a method called fitness switching[13]. Fitness switching is a method for evolving complex behaviors with genetic programming. It is based on the incremental learning procedure. In

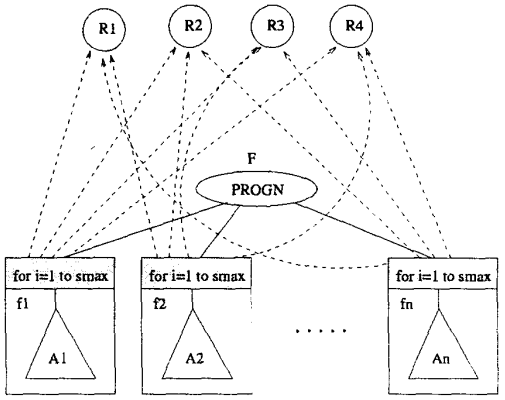

Fig. 3. Schematic diagram for genetic programming with fitness switching. For multiprocess control, each subtree A_i represents a control program for a micro-behavior of each process R_i, The fitness of subtree A_i is assigned by executing it Smax times and measuring the goodness of its behavior by fitness function f_i (This figure is from fig. 18.2 of [13])

this method, there is a set of predefined micro-functions that constitute the original problem solving behavior. And there is a set of fitness functions for each micro-behavior. Sequentially, one element of fitness function set is used to evolve corresponding micro-behavior. After all micro-behaviors are evolved, evolved trees constitute a large tree. This large tree is the wanted solution[13]. For our autonomous robot control problem, we need to evolve two kind of micro-behavior: herding and homing. Herding means finding an object that will be transformed to goal by robots. For herding, we use equation 1 as fitness function. Homing means transforming object to goal. For homing, we use equation 2 a fitness function.

$$f_1: F_{+w} = F_{old} + w_1 \times N_{collisions} + w_2 \times N_{steps} \quad (1)$$

$$f_2: F_{+w} = F_{old} + w_1 \times N_{miss} + w_2 \times N_{steps} + V \times w_3 \quad (2)$$

- Selection Method

After the fitness of each chromosome has been determined by fitness function, it is determined whether to apply genetic operators to chromosome and whether to keep it in the population or allow it to be replaced[12]. The selector operator is used for this task. There are various selection operators. Fitness proportional selection specifies probabilities for chromosomes to be given a chance to pass offspring into the next generation. A chromosome i is given a probability of equation 3 for being able to pass on traits[12].

$$P_i = \frac{f_i}{\sum_i f_i} \quad (3)$$

Ranking selection is based on the fitness order, into which the individuals can be sorted. The selection probability is assigned to individuals as a function of their rank in the population. Tournament selection is not based on competition within the full generation but in a subset of the population. A number of individuals, called the tournament size, is selected randomly, and a selective competition takes place. The traits of the better individuals in the tournament are then allowed to replace those of the worse individuals[12]. For our problem, we select fitness proportional selection operator.

4 Autonomous Robot Control Problem

Robotics has until developed systems able to automate simple and repetitive tasks. These robots are mostly programmed in a very explicit way and the environment of the robot and all the operations are described previously. But for the mobile applications, the environment is perceived via sensor system of robot. These data is far from well-defined environmental data of usual robots. Thus the mobile robot must have ability to decide in a complex and dynamic environment in order to its autonomy. Building autonomous robot is a good example of AI and Artificial Life research whose aim is to build such an autonomous system which can adapt to the world it is embedded in, this world being changing and often unpredictable[14].

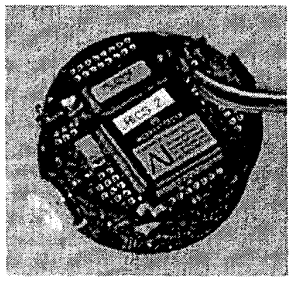

Fig. 4. The experimental robot: Khepera. This robot has one line vision sensor to recognize the goal and eight IR sensors to detect obstacles and eight light sensors to detecting lighting object.

Lee made a box-pushing robot using genetic programming. The robot pushed a object to a goal in non-obstacle environment[15]. Our problem is similar to Lee's problem. But our work is more difficult that Lee's work, because our robot wanders in environment with obstacles and must evolve cooperative strategy to push an object to goal with another robot.

Fig. 5 shows a control structure of an autonomous mobile robot using genetic tree. As described in Section 3, genetic tree is composed of function nodes and terminal nodes. Terminal nodes represent movement of a robot and function nodes represent interpretation of sensor pattern. A genetic tree must have sufficient depth to interpret all possible input pattern. But without depth limit, a tree can grow to unacceptable size by addition of redundant node - this problem is known as the intron problem[16]. In the case of hardware implemented genetic programming, the intron problem can bea severe drawback. With introns, the size of genetic tree can easily exceed the precious hardware resource limit. Of course, this drawback is easily overcomed with context switching ability.

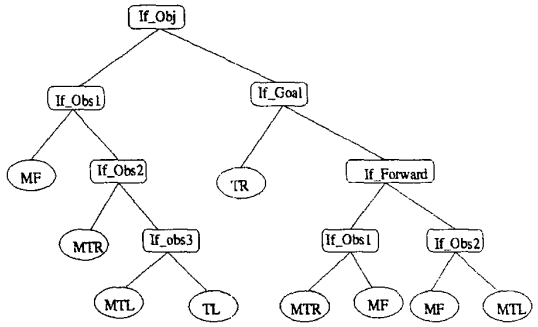

Fig. 5. This shows one control structure. At robot, robot determines it researches the object that is pushed to the goal. If it finds the object, control authority is over to the right subtree, else left subtree. At one of two subtree, robot determines it confronts an obstacle. According to result, robot chooses one acceptable action

5 Implementation and Experiments

Fig. 6 shows context switching procedure. Dashed square box in sub procedure (a) represents given hardware like case (a). At this step, hardware implemented genetic programming can be viewed as empty idea. But if the job does not require intensified parallel computation ability, subprocedures are executed sequentially. Context switching idea is based on this phenomenon. To make context switching possible, the whole procedure is divided into subprocedures whose size is representable on the given hardware. Procedure dividing step does not dive into atomic process. If the remaining process is representable on the given hardware, procedure division stops at the step. For procedure division, genetic programming can be ideal candidate due to its tree structured chromosome. By its

embryogical feature, genetic tree can be easily divided into subtrees. Therefore it is easy to pick up a subprocess for being represented on hardware in genetic programming. After all division analysis are done, whole process has form like figure (b). Subprocesses of (b) are located after their sequential computation order. Subprocess is selected as their computation order and converted to hardware representable form. The selected subprocess is converted to hardware logic and outputs computation result according to given data. This step corresponds to(c). After subprocesses are computed, the midresult are combined to produce final result like (d). To make easy context switching, all function block has identical structure like Fig. 7. There are two kind of identical blocks on hardware - one for function node and the other for terminal node. Fig. 7. is the block for function node. When the selected subprocess is located, the * printed gates is changed to represent one of predefined functions. By conversion blocks and routing these blocks, the selected subprocess is converted to tree structured hardware logic.

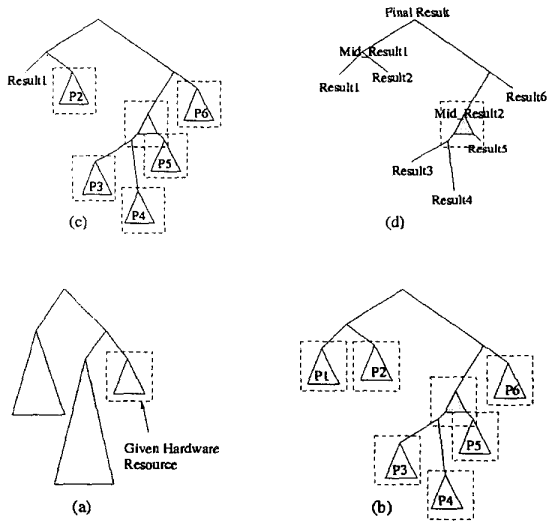

Fig. 6. This shows context switching procedure. Dashed square box represents given hardware resource. In context switching, the tree is divided into subtrees whose size is representable on hardware.

For experiment, we use XC6216 FPGA of XILINX. It is mounted on H.O.T. board of V.C.C. ∞. The board is put in P.C.I slot of host PC and communicates with robot through serial communication. Fig. 8 shows experimental environment. there are two Khepera robot, one object, and some obstacles. Our problem is to find an evolutionary strategy that makes the robots cooperate to push the object to goal while avoiding collisions with obstacles. At our problem the subprocesses that constitute the whole process are

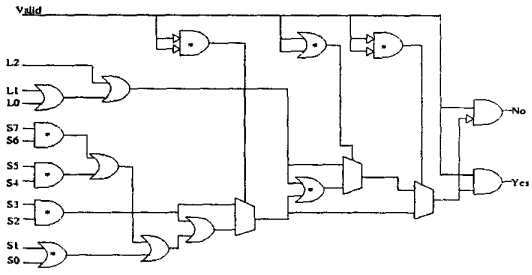

Fig. 7. This block is for function node. By changing (*) printed gate, this block is converted to one of predefined function

herding and homing and they can be located on hardware altogether. So we do not need to build context switchable structure. By experiment, we found the control structure by 50 generation.

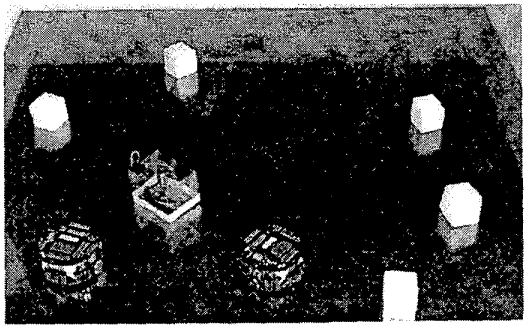

Fig. 8. This figure shows the experimental environment. There are two Khepera robots, one object and some obstacles. Our problem is to find an evolutionary strategy that makes the robots cooperate to push the object while avoiding collision with obstacles.

6 Conclusion

We propose an evolutionary strategy for on-line adaptive learnable evolvable hardware. Evolvable hardware is a rising alternative viewed as having potential for conquering the limit of general processor and ASIC. Our method is based on genetic programming. The unique tree structured chromosome of genetic programming has possibility for being control architecture of complex behavior by combining micro behavior. Despite of its possibility, genetic programming is not so popular as genetic algorithms in evolvable hardware community. Due to the difficulty of hardware implementation and the difficulty of using

crossover operator, genetic programming has not been primary dynamic reconfiguration algorithm for evolvable hardware. We propose context switchable identical block structure to overcome these drawbacks. We applied proposed method to autonomous mobile robot cooperation problem and affirmed the usefulness of our method. In autonomous mobile robot cooperation problem, the problem space is relatively small so there is no need for making context switchable chromosome. In the future research, we must choose a problem that has large search space. We should proof usefulness of the proposed method using more realistic problem

Acknowledgements

*This research was supported by Brain Science and Engineering Research Program sponsored by Korea Ministry of Science and Technology
**This research was supported by the Korea Science and Engineering Foundation (KOSEF) under grant 981-0920-107-2

References

[1] Higuchi T., Iwata M., Keymeulen D., Sakanashi H., Murakawa M. Kajitani I., Takahashi E. Toda K., Salami M., Kajihara N., and Otsu N., "Real-world Applications of Analog and digital Evolvable Hardware," *IEEE Transactions on Evolutionary Computation*, vol. 3, no. 3, pp. 220-235, September 1999.

[2] Kajitani I., Hoshino T., Nishikawa D., Yokoi H., Nakaya S., Yamauchi T., Inuo T., Kajihara N., Iwata M., Keymeulen D., and Higuchi T., "A gate-level EHW chip: Implementing GA Operations and Reconfigurable Hardware on a single LSI," *Second International Conference on Evolvable Systems 1998*, pp. 1-12, 1998.

[3] Koza J. R., *Genetic Programming: On the Programming of Computers by Natural Selection*, Cambridge, MA, USA: MIT Press, 1992.

[4] Stoica A., Keymeuler D., Tawel R., Salazar-Lazaro C., and Li W., "Evolutionary Experiments with a Fine-Grained Reconfigurable Architecture for Analog and Digital CMOS Circuits," *Proceedings of the first NASA/DoD workshop on Evolvable Hardware*, pp. 76-84, 1999.

[5] Perkowski M., Chebotarev A., and Mishchenko A., "Evolvable Hardware or Learning Hardware? Induction of State Machines from Temporal Logic Constraints," *Proceedings of the First NASA/DoD Workshop on Evolvable Hardware 1999*, pp.129-138, 1999.

[6] Koza J. R., Bennett III, Forrest H, Hutchings, Jeffrey L., Bade, Stephen L., Keane, Martin A., and Andre, David, "Evolving Computer Programs using Rapidly Reconfigurable Field-Programmable Gate Arrays and Genetic Programming," *Proceedings of the ACM Sixth International Symposium on Field Programmable Gate Arrays*, pp. 209-219, 1998.

[7] Nikolaev N. I., Iba H., and Slavov V., "Inductive Genetic Programming with Immune Network Dynamics," *in Advances in Genetic Programming 3*, MIT Press, pp. 355-376, 1999.

[8] Andre D., Teller A., "A Study in Program Response and the Negative Effects of Introns in Genetic Programming," *Proceedings of Genetic Programming 1996*, pp.12-20, 1996.

[9] Koza J. R, and Bennett III F. H., "Automatic Synthesis, Placement, and Routing of Electrical Circuits," *in Advances in Genetic Programming 3*, MIT Press, pp. 105-134, 1999.

[10] Luke S., and Spector L., "A Comparison of Crossover and Mutation in Genetic Programming," *Proceedings of Genetic Programming 1997*, pp. 240-248, 1997.

[11] Ito T., Iba H., and Sato S., "A Self-Tuning Mechanism for Depth-Dependent Crossover," *in Advances in Genetic Programming 3*, MIT Press, pp. 377-399, 1999.

[12] Banzhaf W., Nordin P., Keller R. E., and Francone F. D., *in Genetic Programming an Introduction*, Morgan Kaufmann Publishers, Inc, 1998.

[13] Byoung-Tak Zhang, and Dong-Yeon Cho, "Fitness Switching: Evolving Complex Group Behaviors Usign Genetic Programming," *Proceedings of Genetic Programming 1998*, pp. 431-438, 1998.

[14] Pattie M., "Behavior-based Artificial Intelligence," *From Animals to Animats 2: Proceedings of the Second International Conference on Simulation of Adaptive Behavior*, pp. 2-10, 1993.

[15] Lee W. P., Hallam J., and Lund H. H., "Applying Genetic Programming to Evolve Behavio Primitives and Arbitrators for Mobile Robots, *IEEE International Conference on Evolutionar Computation*, pp. 501-506, 1997.

[16] Andre D., and Teller A., "A Study in Prograr Response and the Negative Effects of Introns i Genetic Programming," *Proceedings of Geneti Programming 1996*, pp. 12-20, 1996.