Embodied Evolution with a new Genetic Programming Variation Algorithm

Anderson Luiz Fernandes Perez
Department of Automation and Systems
Federal University of Santa Catarina - UFSC
Florianópolis, SC, Brazil
anderson@das.ufsc.br

Guilherme Bittencourt
Department of Automation and Systems
Federal University of Santa Catarina - UFSC
Florianópolis, SC, Brazil
gb@das.ufsc.br

Mauro Roisenberg
Department of Computer Science
Federal University of Santa Catarina - UFSC
Florianópolis, SC, Brazil
mauro@inf.ufsc.br

Abstract

Embodied Evolution is a research area in Evolutionary Robotics in which the evolutionary algorithm is entirely decentralized among a population of robots. Evaluation, selection and reproduction are carried out by and between the robots, without any need for human intervention. This paper describes a new Evolutionary Control System (ECS) able to control a population of mobile robots. The ECS is based on a Genetic Programming algorithm and has two main modules. The first one, called EMSS (Execution, Management and Supervision System), is the system responsible for managing all the evolutionary process in each robot. The second module, called DGP (Distributed Genetic Programming), is an extension of classical Genetic Programming algorithm to support the robot control system evolution. To test the DGP's performance a simulation experiment, with the collision-free navigation task, was accomplished and its results are presented.

1. Introduction

Evolutionary Robotics (ER) [15] [16] aims at the development of adaptable control systems, based on Evolutionary Computation (EC) [2] techniques [3]. In ER the selection and genetic operator execution is performed in a centralized manner.

Embodied Evolution (EE) was introduced by Watson[12] and uses a population of robots that autonomously reproduce with one another while situated in their task environment. In EE the evolutionary process – evaluation, selection and reproduction – is carried out by and between the robots, without any need for human intervention.

One of the evolutionary algorithms used in ER is Genetic Programming (GP) [7]. GP is a technique that aims at automatically generating computer programs. The main goal of GP is teaching computers to program themselves, i.e., from specified primary behaviors, the computer must be able to generate a program that satisfies some conditions that aim at the solution of some task or problem.

Control systems for mobile robots are considered as hierarchical structures of basic behavior. For this reason, using GP to develop control systems for mobile robots represents one advantage. In GP the handled structures, functions and terminals, are high level and its generated programs are hierarchical associations between them.

Most works in EE, such as those described in [5], [14] and [18], are based on the use of Genetic Algorithm to optimaze the weights of a Neural Network controller that selects the low-level motor actions of the robot. There are some works that propose the use of GP to evolve the mobile robot control system, e.g. [1] and [11], but these works are not applied in EE.

The main contribution of the work described in this paper is an Evolutionary Control System (ECS) entirely based on GP to be applied on EE. The ECS have two main modules. The first one, called EMSS (Execution, Management and Supervision System), is responsible for executing and managing the second module, called DGP (Distributed Genetic Programming). The DGP is an extension of the classical GP algorithm [7].

To validate the DGP we accomplished one experiment with a population of five mobile robots using the Khepera

0-7695-3093-1/08 $25.00 © 2008 IEEE
DOI 10.1109/ICAS.2008.31

Simulator. The objective of each robot was to navigate through an unknown environment avoiding obstacles.

This paper is organized in the following way: Section 2 describes the Evolutionary Control System and the Distributed Genetic Programming algorithm, respectively. Section 3 describes the experiment setup. Results with DGP's performance are presented in Section 4. Section 5 presents conclusions and future works.

2. Evolutionary Control System

The ECS has two main modules. The first one, called EMSS (Execution, Management and Supervision System) is responsible for executing and managing the algorithm responsible for all evolutionary process. The second module, called DGP (Distributed Genetic Programming) is the algorithm responsible for all the evolutionary process of the robots control system.

Figure 1 illustrates, generically, the functioning of the ECS.

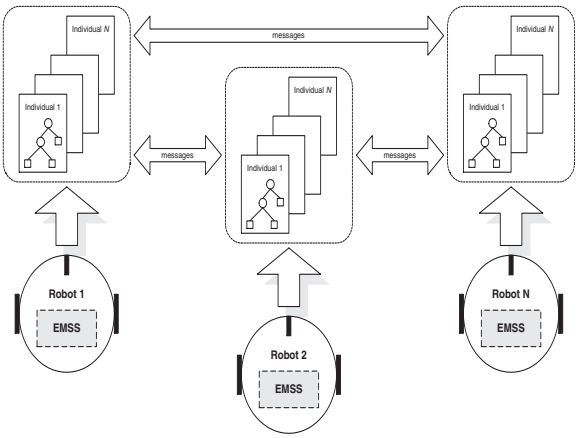

Figure 1. Schematic Representation of ECS.

In each robot, the EMSS is executed. Each robot has a local population, a set of programs that may solve a problem (a task that the robot should execute), that interacts with the other robots local population. The goal is to build, by using only GP, a control system for a population of mobile robots in which the robots interact among them to execute some task.

Sections 2.1 and 2.2 describe in details how EMSS and DGP work, respectively.

2.1. Execution, Management and Supervision System - EMSS

The EMSS (Execution, Management and Supervision System) is the system responsible for managing the evolutionary process that takes place in an embedded fashion in each robot. Figure 2 illustrates the EMSS components and the relationships among them.

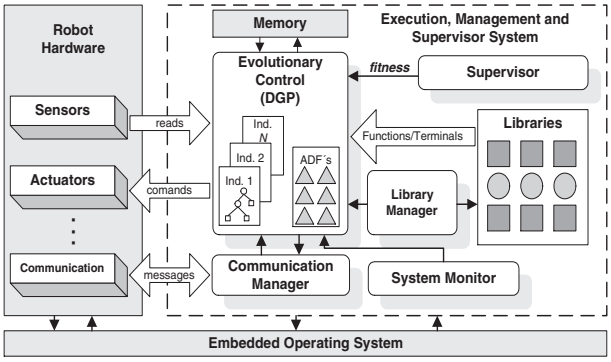

Figure 2. Basic Structure of EMSS.

Below is the complete description of each component of the EMSS and the connection among them (when there is one).

Evolutionary Control (EC): it is the main component in the EMSS. The EC is responsible for creating, randomly, a local population of individual programs using the functions and terminals library. In each new generation, the generated individuals are tested and executed by the EC. The ADF's (Automatically Defined Functions) are also managed by this component.

Memory: the memory stores the representation of the most adapted individuals in each generation. These individuals can be used in a reinitialization process, in the case the EC fails, or to optimize tasks that involve more than one competence, such as avoiding obstacles and moving an object from one place to another. In this case, the memory works as a kind of system snapshot, which can be reused to speed up the learning process through experiences taken place in the past.

Communication Manager (CM): it is the component responsible for sending and receiving the messages exchanged by the robots. In the message sending process, the EC sends to CM the sequence of functions and terminals and the fitness value that will be sent to the other robots. The CM creates a message containing this information and sends it to the other robots. In receiving, the CM receives the messages sent by the other robots and sends their contents to the EC.

Library Manager (LM): this component manages the sets of terminals and functions in each robot. In this component all the functions and terminals are identified through an unique identifier, an *even* number for the functions and an *odd* number for the terminals, so they can be sent to other robots through the CM. At any time, it is possible to add or

remove functions or terminals without having the need to redefine the control system.

Supervisor: it is responsible for evaluating and attributing a fitness value to each individual, using a *punishment* and *reward* method. For this method to work properly, for each task is assigned to the robot, one must define how and when reward and punishment take place.

System Monitor: it is the component responsible for evaluating the system execution. In the case the system is inactive, i.e., the robot stays still for a long period of time, the monitor activates an EC reinitialization process. When the EC is reinitialized, the images of the best individuals, which are stored in memory, are retrieved and the evolutionary process initiates from these individuals, that is, the local population is completed through the use of parts of these individual programs, instead of starting from a random population, as it happens in the original GP algorithm.

2.2. Distributed Genetic Programming - DGP

The DGP is the algorithm responsible for all the evolutionary process of the robots control system. The DGP is an extension of the classical GP algorithm to support the control system evolution for the robots that are part of the mobile robots population.

The DGP is based on Microbial GA [6], a GA variation. Its functioning is similar to the genetic combination (infection) that takes place in bacteria, where DNA segments are transferred between two members of the population.

In DGP two population sets are considered. The first, called local set or $P_{local_{R_i}}$, refers to the local population of each robot R_i, i.e., the set of individuals or solutions that are embedded in the robot. Each $x \in P_{local_{R_i}}$ represents a candidate solution to a problem. The second set, called total set or P_{total}, is made of the union of all the local populations in each robot, where: $P_{total} = P_{local_{R_1}} \cup P_{local_{R_2}} \cup \cdots \cup P_{local_{R_n}}$. The evolutionary process takes place always considering the total population, that is, parts of a local individual of a certain robot may be considered in another robot's local population evolutionary process.

Differently from other approaches in EE, such as [5], [8], and [13], in DGP the evolutionary process is asynchronous, that is, it is not necessary that two robots are synchronized to reproduce.

In DGP, parts of a more adapted individual are sent to all the other robots. The step sequence in DGP is the following:

1. Randomly create a program population;
2. Iteratively execute the following steps until some halt criterion is satisfied:
   (a) Evaluate each population program through a heuristic function (fitness) that expresses its fitness, that is, how well the program solves the robot task;
   (b) Receive[1] parts of a remote[2] individual sent by another robot;
   (c) Select the t best individuals in the local population, using the tournament selection method;
   (d) Randomly select a part of the best (more adapted) local individual and broadcast it to the other robots;
   (e) Compare whether the worst locally selected individual fitness is lower than the remote individual fitness. If so, execute the mutation operator, replacing parts of the individual by the parts received from a remote individual;
   (f) Execute the crossover and mutation operators;
3. Return the best program found.

The selection method used in the DGP is tournament selection with the parent preservation after the crossing. The remotely received parts are added to the worst individual structure, from the t selected best, following equation 1:

$$M(A) = \left\{ \begin{array}{ll} Muta(A) & \text{if } \textit{FitnessR} > \textit{FitnessL} \\ A & \text{if } \textit{FitnessR} \leq \textit{FitnessL} \end{array} \right\} \quad (1)$$

where ***M(A)*** is the remote mutation operator that happens all time when a part of remote individual is received. ***FitnessR*** is the value of the remote individual's fitness, which sent a part of its structure. ***FitnessL*** is the value of the local individual's fitness. A is the local individual's structure, and ***Muta(A)*** is the mutation function, where a part of the A structure is randomly chosen to be replaced by the received part of the remote individual program.

The messages passing between the robots must contain the fitness and a part of the best local population individual program. For such, all the functions and terminals receive an unique numerical identification, an *even* number for each function and an *odd* number to each terminal.

It is important to emphasize that for a correct functioning of DGP all robots must contain the same sets of functions and terminals, or at least they must use the same functions and terminals for a particular problem.

DGP assures the continuity of the control system of the evolutionary process, even when there is a problem with the

[1] In each execution cycle the DGP considers only one received message. Each new message, containing parts of a remote individual, is stored in the local buffer. The buffer is overwritten every time a new message is received.

[2] A remote individual is a program that is part of the local population of another robot.

other robots that are part of the robot population. This is possible because each robot has a local population of programs, that guarantees the continuity of the evolutionary process.

3. Experimental Setup

With the purpose of evaluating the DGP algorithm we defined a simulation environment in which a set of five mobile robots must navigate along an unknown environment avoiding obstacles. The simulation was accomplished in the Khepera Simulator version 2.0 [9]. This version of the Khepera Simulator can operate with a group of mobile robots instead of a single robot.

Figure 3 illustrates the main interface of the Khepera Simulator. In the figure there are five robots dispersed in random positions along the environment and a set of rectangles that represents obstacles.

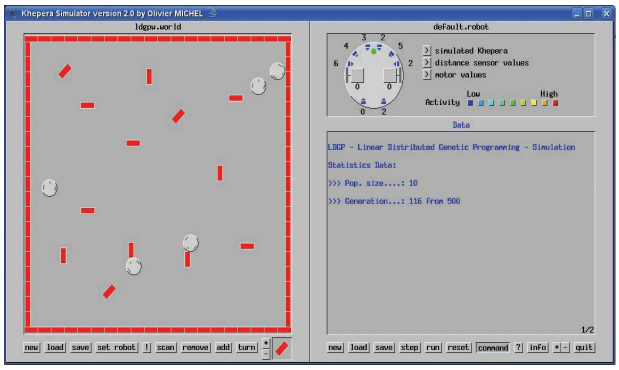

Figure 3. Khepera Simulator Screenshot.

Each Khepera Robot has eight (8) infrared sensors for proximity measurements, eight (8) light sensors for ambient light measurement and two motor-drive wheels for movement. Each infrared sensor returns a value ranging [0 - 1023]. Where 0 means that no object is perceived and 1023 means that an object is very close to the sensor. The light sensors can measure the level of ambient light around the sensor and return a value ranging [0 - 525]. Where 0 means a maximum brightness and 525 a maximum darkness.

The goal of each robot is to navigate along the environment avoiding obstacles, walls and other robots. This task is called in the robot literature collision-free navigation. According to [5] this task is trivial for tradicional robots techniques but provides a non-trivial search space for an evolutionary system.

Table 1 describes the function and terminal sets used in the experiment.

In Table 1 the *Id.* column represents the identification of functions and terminals according to the identification rule of DGP, an *even* number for each function and an *odd* number for each terminal.

Table 1. Functions and Terminals sets.

Functions			
Name	**Arity**	**Id.**	**Definition**
Prog1	1	2	Perform one branch of the tree.
Prog2	2	4	Perform two branches of the tree.
Prog3	3	6	Perform three branches of the tree.
Terminals			
TurnRight	0	1	Turn right (15 degrees).
TurnLeft	0	3	Turn left (15 degrees).
GoForward	0	5	Go forward (300 ms).
Return	0	7	Return (300ms).

The parameters used in the experiment are described below:

- **Number of robots**: 5
- **Number of generations**: 500
- **Population size**: 10 individuals
- **Vector size**: 50 positions
- **Crossover probability**: 60%
- **Mutation probability**: 10%
- **Selection method**: tournament selection with tournament size of 6
- **Number of executions**: 10
- **Fitness function**: punishment and reward method. `if BUMPED; fitness -= 3; else fitness += 2`

The fitness function is very simple. Basically when a robot hit in an obstacle, the punishment method is activated and its fitness value is decrease in **3** points. Otherwise, when a robot is navigating in the environment without hit any object, the reward method is activated and its fitness value is increase in **2** points.

It is important to emphasize that for this experiment no limit was imposed on the fitness value. That is, the fitness could be any negative or positive value. The highest the value corresponds to the best individual in the population.

The DGP algorithm was developed in C with a linear representation method [10]. Each population individual (program) is represented by one fixed size vector. Each vector position only contains the identification number of a function or a terminal.

4. Experimental Results

At each execution time, the robots are dispersed in random positions of the simulation environment. The graphs in figures 4 and 5 illustrate, respectively, the average fitness value and the fitness value of the best individual of the population at each generation for each robot during 500 generations.

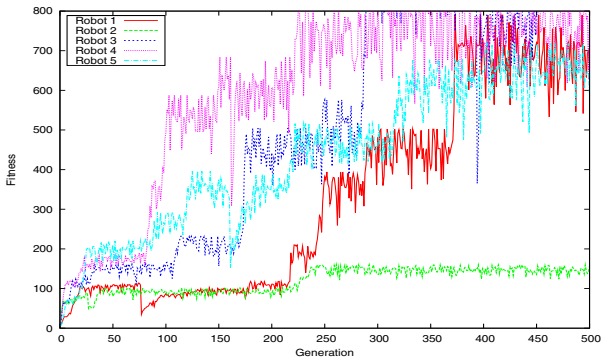

Figure 4. Average population fitness at each generation for each robot.

The graph (Figure 4) shows that the average population fitness, with exception of Robot number 2, increase for all robots. The effect of the remote mutation, caused by DGP functioning when a message from the other robots is received and accepted, produce a genetic variation in each robot local population.

Differently from the other robots, robot number 2 had not a good performance, because it stayed away for a long period of time in a corner of the environment.

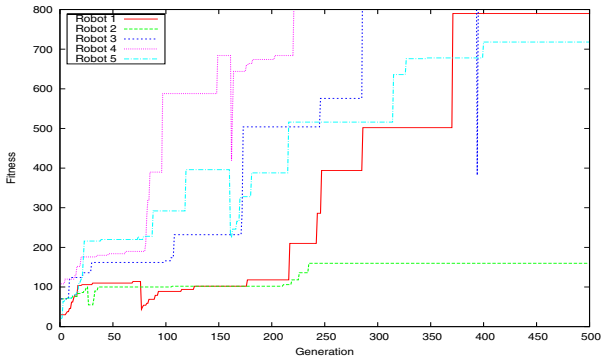

Figure 5. Fitness value of the best individual at each generation for each robot.

The performance graph of the best individual at each generation for each robot is illustrated in Figure 5. The fitness value of the best individual increases faster because a genetic variation occurs any time a message from other robots is received and accepted to mutate one individual (from the t selected best) in the local population, according to DGP rules.

```
44631563314544657163572652616566773651566735731 77
```

Figure 6. The best individual structure at the last generation of the robot 4.

Figure 6 is an example of how DGP represents each individual in the population. This Figure represents the best individual structure at the last generation (500) of the robot number 4. The numbers represent a function or a terminal identification, according to *Id.* column in Table 1.

The advantage of using a linear representation is that the genetic operators are applied to a vector containing only numbers, instead of to tree structure as is usual in traditional GP algorithm applications [7]. This characteristic helps to improve the DGP's performance.

The experimental results show that it is possible to build a control system to be applied on EE using only a GP algorithm. After a few generations the group of five robots learned how to navigate along an unknown environment avoiding obstacles.

A disadvantage of GP, common to other EC [3] techniques, is that the definition process of the terminals and especially of the functions requires more attention and experience from the programmer. Functions and terminals developed for a particular problem may not apply to other kinds of problems.

The set of functions and terminals must be designed to be as comprehensive as possible, that is, with few modifications or even none, in such a way that it is possible to use the same functions and terminals in many kinds of problems, only redefining the evaluation function.

5. Conclusions

In this paper was described an Evolutionary Control System (ECS) able to control a population of mobile robots. The ECS has two main modules: the first one, called EMSS (Execution, Management and Supervision System), is the system responsible for managing all the evolutionary process that takes place in an embedded fashion in each robot. The second module, called DGP (Distributed Genetic Programming), is an extension of classical Genetic Programming algorithm to support the control system evolution for the robots that are part of the mobile robots population.

To test DGP algorithm one experiment using the Khepera Simulator was accomplished. The experiment was a collision-free navigation task, in which a group of five mobile robots must navigate through an unknown environment avoiding obstacles, walls and other robots. The results show that the remote mutation, caused by program transfer from robot to robot, helps to increase the variability of the local population at each robot. All robots learned how to navigate along the environment avoiding obstacles.

In the experiment described in this paper, a population of mobile robots autonomously reproduce with one another while situated in their task environment. Its results show that is possible to build a control system to be applied on EE using only a GP algorithm.

Future works include experiments with ECS in other problems in mobile robotics, such that, box pushing, and predator and prey. With these experiments we intend to evaluate how easy it is to change the task of the robots without reprogramming the whole control system. These experiments will be accomplished using a simulator and real robots. The Eyebot [17] robots will be used for the experiments. Eyebot is a kind of mobile robot developed to be used in robot soccer.

6. Acknowledgments

The author, Anderson Luiz Fernandes Perez, acknowledge the CNPq (Conselho Nacional de Desenvolvimento Científico e Tecnológico), Brazil, for the scholarship.

References

[1] Byoung-Tak Zhang and Dong-Yeon Cho. Evolving Complex Group Behaviors Using Genetic Programming with Fitness Switching. *Artificial Life and Robotics* 4:2 (2000) 103–108.

[2] D.B. Fogel. What is Evolutionary Computation? *IEEE Spectrum* 37:2 (2000) 28–32.

[3] Darrell Whitley. An overview of evolutionary algorithms: practical issues and common pitfalls. *Information and Software Technology* 43:14 (2001) 817–831.

[4] Dario Floreano and Stefano Nolfi. Adaptive Behavior in Competing Co-Evolving Species. *Fourth European Conference on Artificial Life*. The MIT Press, Cambridge, MA, Phil Husbands and Inman Harvey (1997) 378–387.

[5] Eduardo D. V. Simões and Keith R. Dimond. Embedding a Distributed Evolutionary System into Population of Autonomous Mobile Robots. *Proceeding of The 2001 IEEE System, Man, and Cybernetics Conference* (2001).

[6] Inman Harvey. Artificial Evolution: A Continuing SAGA In Evolutionary Robotics: From Intelligent Robots to Artificial Life. *8th International Symposium on Evolutionary Robotics (ER2001)*. Springer-Verlag Lecture Notes in Computer Science LNCS 2217 (2001).

[7] John R. Koza. Genetic Programming: A Paradigm for Genetically Breeding Computer Population of Computer Programs to Solve Problems. MIT Press (1992).

[8] Jordan B. Pollack, Hod Lipson, Sevan Ficci, Pablo Funes, Greg Hornby, and Richard A. Watson. Evolutionary Techniques in Physical Robotics. *ICES* (2000) 175–186.

[9] O. Michel. *Khepera Simulator Package version 2.0: Freeware Mobile Robot Simulator written by Oliver Michel*. http://www.epfl.ch/lami/team/michel/khep-sim/ (1996).

[10] Markus Broodier and Wolfgang Banzhaf. Linear Genetic Programming. Springer (2006).

[11] Peter Nordin and Wolfgang Banzhaf. An on-line method to evolve behavior and to control a miniature robot in real time with genetic programming. *Adaptive Behaviour* 5:2 (1997) 107–140.

[12] R. Watson, S. Ficici and J. Pollack. Embodied Evolution: Distributing an Evolutionary Algorithm in a Population of Robots. *Robotics and Autonomous Systems* 39:1 (2002) 1–18.

[13] S. Ficici, R. Watson and J. Pollack. Embodied Evolution: A Response to Challenges in Evolutionary Robotics. J. L. Wyatt and J. Demiris, *Eighth European Workshop on Learning Robots* (1999) 14–22.

[14] Stefan Elfwing, Eiji Uchibe, Kenji Doya and Henrik I. Christensen. Biologically Inspired Embodied Evolution of Survival. *Proceedings of the 2005 IEEE Congress on Evolutionary Computation* (2005) 2210–2216.

[15] Stefano Nolfi and Dario Floreano. Evolutionary Robotics. The Biology, Intelligence, and Technology of Self-Organizing Machines. MIT Press (2001).

[16] Stefano Nolfi and Dario Floreano. Synthesis of autonomous robots through evolution. *Trends in Cognitive Science* 6:1 (2002) 31–36.

[17] Thomas Brunl. *EyeBot Online Documentation*. http://robotics.ee.uwa.edu.au/eyebot/ (2006).

[18] Yukiya Usui and Takaya Arita. Situated and Embodied Evolution in Collective Evolutionary Robotics. *Proceedings of the 8th International Symposium on Artificial Life and Robotics* (2003) 212–215.