Distributed Genetic Algorithm: Learning by Direct Exchange of Chromosomes

Aleš Kubík

Institute of Computer Science, Silesian University,
Bezručovo nám. 13, 746 01 Opava, Czech Republic
ales.kubik@fpf.slu.cz

Abstract. Genetic algorithms is a technique widely used to evolve controllers of agents or robots in dynamic environments. In this paper we describe a modification to a single-robot-based evolution of a controller - a distributed parallel genetic algorithm where the pool of chromosomes is dispersed over a multi-robot society. Robots share their experience in solving the task by direct exchange of individually evolved successful strategies coded by chromosomes.

1 Introduction

Evolutionary and genetic algorithms are suitable tool to evolve controllers from initially pseudo-randomly designed population of controllers evaluated by fitness function [4], [3].

Either the robot on its own runs a genetic algorithm to produce a controller with desired behavior (see e.g. [2])[1], or the evolution shapes individual robot behavior but that is influenced by couplings with other robots in a multiagent scenario (described in [9], [10], [8]).

In this paper we describe the experiments with the evolution of controllers using chromosomes dispersed over the whole society of robots. Robots create random sets of chromosomes and exchange successful population members among each other. In this case we can go along without a central process but the robots have to communicate with each other and when more robots are present even coordinate the exchange of chromosomes. As a testbed of our ideas we have chosen collision avoidance learning representing almost benchmark behavior in the experimental robotics.

In this place it should be cited the work of [11] where the authors implement a distributed genetic algorithm in a collective robotics scenario. As for the design and methodology our approach is nevertheless closer to standard genetic algorithm solution with combined individual and parallel evolution, selection, reproduction, fitness evaluation of genome etc.

[1] A good overview of evolutionary robotics focusing on single robot scenarios represents the work of [7].

W. Banzhaf et al. (Eds.): ECAL 2003, LNAI 2801, pp. 346–356, 2003.

2 Experimental Robots

In our experiments we used two Khepera robots ([6]) that are well suited for educational as well as experimental purposes (see [5]). The robot body has about 5cm in diameter and is equipped with two servomotors driving two wheels and eight infrared sensors (six in the front and two at the rear side of the robot - see figure 1) that serve to avoid obstacles.

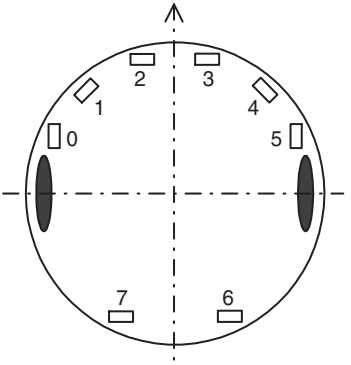

Fig. 1. Schema of a Khepera robot with two wheels and eight IR proximity sensors. The arrow indicates the front movement of a robot.

Programs written in C language run on the on-board processors of the robots. Robots thus conserve autonomy in relation to other robots and human intervention. It is possible that the program runs on the host computer and with the robot communicates through the serial communication line. Such type of a communication was used only for the purpose of collecting data from the experiments. Communication routines as well as program execution manages Motorola 68331 controller with 256 KB RAM and 512 KB ROM.

We used an additional component - a radio extension turret that is mounted on top of a robot in a plug and play fashion. It enables a peer-to-peer communication between robots. Robots communicate via sending and reading buffers each of which holds 16 bytes of data. Messages are sent directly to a robot with a specified identification number. There is a built-in mechanism of checking the correctness of messages. The sender considers a message to be sent correctly only if it receives the acknowledgement from the receiver. It repeats the sending process after timeout when it receives no acknowledgement. The message is considered lost after 10 unsuccessful trials.

3 The Learning Task

Robots learn to avoid dynamic and static obstacles. The behavior is coded by a very simple neural network resembling the perceptron. It controls the movements of a robot in a straightforward fashion. If the sensors of either side (left or right) of a robot perceive any kind of an obstacle, the motor on this side will speed up the wheel it drives. The algorithm is inspired by the work done in [1]. Genetic algorithm is used to evolve weight vector of input-output neuron connections. These are coded as chromosomes. Each robot evolves its own population of controllers. One of them sends (this will be called sender) the successful candidates to the second robot (denoted as receiver). The receiver uses both its own population of chromosomes as well as received chromosomes from the sender. In this way the receiver learns also from the experience of a sender. We expect that the receiver will learn better and faster than the sender.

3.1 Basic Processes

In this section we describe the algorithms of employed processes of both robots.

The program body of a sender is composed of two main processes that are running interchangeably. One is responsible for running evolution of weight vectors and testing the resulting neural controllers. The second process is sending 1/4 of most successful population chromosomes to the receiver after each generation finishes its learning. Pseudocode of the sender's processes can be expressed as follows:

process 1:

```
1. create weight vectors with rand. values;
2. for each generation do {
3.  for each weight vector do {
     run the neural controller for 800 cycles;
     compute fitness value of the controller;
     change speed randomly for 100 cycles;
    }
4.  sort the population according to fitness;
5.  transform the chromosomes into an array
      of buffers of 16 bytes each;
6.  suspend this process;
7.  wait for a signal from the process 2;
8.  choose the best 1/2 of a population
     and place it in the next generation;
9.  crossover the parent members to create
     offspring weight vectors;
10. mutate the new population;
11. go to step 3;
   }
```

process 2:

```
1. in endless loop do {
2.   wait for a signal from the process 1;
3.   for each message buffer in the array do {
      send the buffer to the receiver;
      suspend the task for 1000 ms;
     }
4.   send a signal to the process 1;
   }
```

After each generation first process transforms the best-valued weight vectors into an array of buffers size of which depends on the size of population. The size of each message is 16 bytes. Then it gives control to the second process waiting for response in the background. This is done via its suspension that switches the control to other processes. After the process 2 flushes all of the buffers from the array it sends a signal to the first process and awaits a signal too. Processes of a sender don't run in a parallel manner because of problems with interference between the radio transmission and communication with host computer that these processes maintain.

The receiver consists of two main processes described by following pseudocode:

process 1:

```
1. create weight vectors with rand. values;
2. for each generation do {
3.  for each weight vector do {
     run the neural controller for 800 cycles;
     compute fitness value of the controller;
     change speed randomly for 100 cycles;
    }
4.  sort the population according to fitness;
5.  replace the second 1/4 of a population
      members with received weight vectors;
6.  choose the best 1/2 of a population
     and place it in the next generation;
7.  crossover the parent members to create
     offspring weight vectors;
8.  mutate the new population;
9.  go to step 3;
   }
```

process 2:

```
1. in endless loop do {
2.   receive an array of message buffers
3.   if the new array differs
       from a previous one do {
         for each message buffer in the array
         do {
```

```
            transform the message to
            a part of a new weight vector;
            make a copy of a message array;
          }
        }
    }
```

The receiver continually checks for new messages from the sender. The buffers are not flushed instantly which means that if no new message arrives the received buffer still holds old data. That is why the robot must remember the old array of buffers which it always compares with newly received ones. If new array of buffers arrives, the receiver rewrites the received weight vectors and uses it in its own genetic algorithm. Processes of the receiver run concurrently and there is no need of synchronization.

3.2 Neuro-controller

A neural controller with primitive architecture (see figure 2) controls the robot collision avoidance.[2] There are 8 input neurons fully connected with two output neurons that compute the speed of two motors. These connections are labeled by integer-valued weights. Each of the input neurons has the information about the value read by one of the proximity sensors.

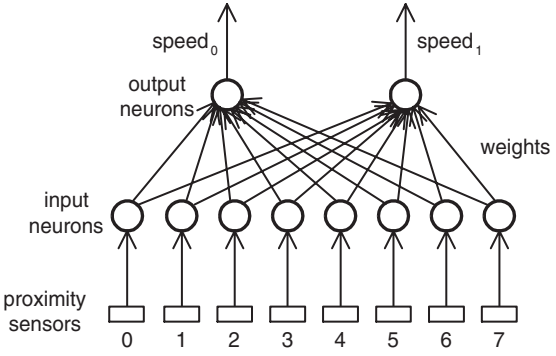

Fig. 2. Schematic view of a neural network architecture used in the experiments.

The speed of each motor is computed according to the Eq. (1),

$$speed_i = potential_i/400 + 10;\ i = \{0, 1\}, \tag{1}$$

where

$$potential_i = \sum_{j=0}^{7} (weight_{ij} * irvalue_j), \tag{2}$$

[2] It was a purpose to design an artificial neural network as simple as possible.

where $weight_{ij},\ i = \{0,1\},\ j = \{0,1,\ldots 7\}$ is a vector of weights represented as a two-dimensional array of integer values from the open interval $(-30, 30)$, and $irvalue_j,\ j = \{0,1,\ldots 7\}$ is a vector of values read by proximity sensors. Speed of motors must be normalized and set to some non-negative value if the value of $potential_i$ is close to 0.

3.3 Parameters of a Genetic Algorithm

The genetic algorithm was used to evolve the weight vector $weight_{ij},\ i = \{0,1\}$, $j = \{0,1,\ldots 7\}$. In each run we evaluated the population of neural controllers with the fitness function consisting of three components taken from [2] (see Eq. (3)).

$$\Phi = V(1 - \sqrt{|speed_0 - speed_1|})(1 - i), \tag{3}$$

where V is an average speed of two wheels, and i is a sensory value recorded by a proximity sensor with highest activation. These three components of a fitness function measure an average speed of a robot (it should be maximized), speed difference of rotating wheels (it should be minimized), and the highest activation of proximity sensors (it should be minimized).

The sender learns as follows. From each population it picks 1/2 of chromosomes with the highest fitness values and places it in the next generation. Then pairs of parental chromosomes are taken sequentially and the crossover at the random spot is performed to produce offspring chromosomes. Mutation of chromosomes was done with 5% probability.

The receiver robot combines 1/4 of its own chromosomes with the best 1/4 of the sender's best population members to produce the offspring generation as explained in the section 3.1. We experimented with various size of population - 12, 20, and 40 chromosomes. The smallest size of population was chosen so that it almost always finds "acceptable" solution for at least one of the robots - a controller that performs well judged not by the fitness value but by looking at the moving robot performance. We were mainly interested in the best controller, not the average performance that can be determined only computationally.

4 Results

In this section we present results of our experiments. Each experiment consists of 5 independent runs that were evaluated for the highest and average values. Each run consisted of 20 generation of evolving neural controllers.

4.1 Experiment 1

In the first experiment we used 12 chromosomes in one population. The performance of the best controller of a receiver is much better and it also learns faster in comparison with a sender robot (see figure 3a)[3]. Due to a small population

[3] Capital S and R in subsequent figures denote sender and receiver, resp.

size the learning peaks in the 14^{th} generation. Its performance is even better on average even if not so markedly than the performance of best controllers. To measure how sharing the chromosome pool improved the performance of a

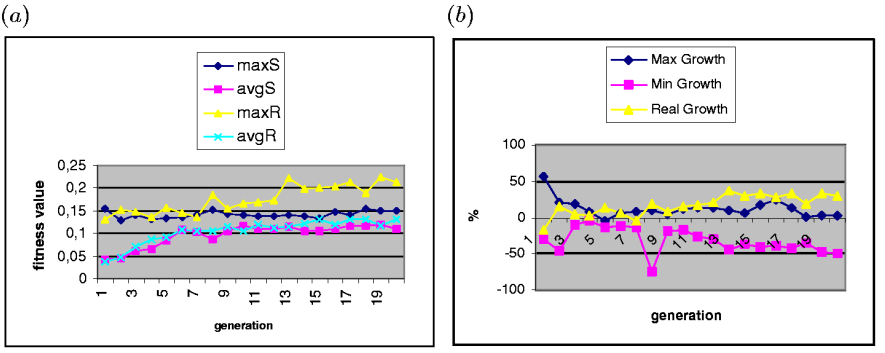

Fig. 3. Experiment with the population of 12 chromosomes. (a) Max. and avg. fitness values for both robots. (b) Expected vs. real fitness growth in receiver's chromosomes.

receiver we compared in each run best evolved controllers of the sender and the receiver. The difference (that could be called maximum expected fitness growth) measured in percentage shows how big the improvement could be in case of best-performing receiver's controllers. Figure 3b shows also the lowest expected fitness growth[4] along with the real improvement. In the first experiment real growth in fitness is higher than we would expect.

4.2 Experiment 2

In this experiment the population of controllers consisted of 20 members. Again the performance of a receiver was better in highest fitness values but approximately the same on average (see figure 4). In this experiment the receiver learned even faster than in previous case (the highest fitness value in the 5^{th} generation), but from this point on the learning started slowly to degrade.

The expected versus real growth in fitness of receiver's best controllers can be seen in figure 5a.

The learning of a receiver is much more unstable with big differences between the highest and the smallest fitness values in each generation illustrated in figure 5b. There are mainly three reasons to this fact:

– Controllers acquired from the sender with worse performance in comparison with the receiver's own population pool can perturb learning and destabilize

[4] This denotes to what degree the received controllers could negatively influence the performance of a receiver.

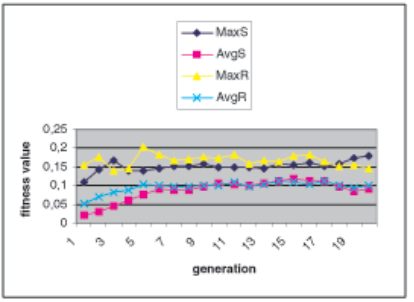

Fig. 4. Experiment with the population of 20 chromosomes. Max. and avg. fitness values for both robots.

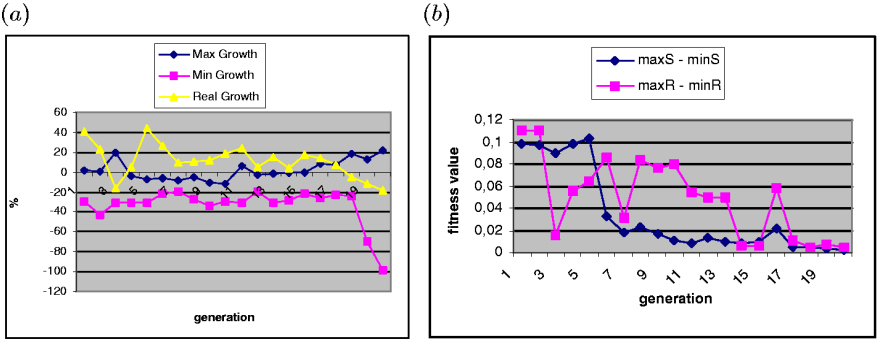

Fig. 5. Experiment with the population of 20 chromosomes. (a) Expected vs. real fitness growth in receiver's chromosomes. (b) Max. - min. fitness values taken from the run with the smallest difference.

it. Even if it should not influence the best chromosomes it has an influence on creating suboptimal solutions.

– Communication errors. The communication among Khepera robots is not very reliable. There appears loss of data during the transmission which we roughly estimate to be 5 to 15 %. If the robot fails to correctly determine when new array of message buffers arrives it uses old values instead which deteriorates the performance of a given generation. The bigger the population of chromosomes is, the more fatal the consequences appear to be. Firstly, the robots must exchange number of messages that equals half of a number of chromosomes in the population.[5] The more the messages robots exchange, the more probable is the occurrence of transmission errors.

[5] Radio turrets of Khepera robots use a protocol that enables to transfer only messages of size 16 bytes (more precisely unsigned bytes), it means only small positive integers. One weight vector that contains 16 weights must therefore be split into 2 messages. Each message holds 1/2 of the weight vector values plus the indication of a sign of a particular weight value. E.g. message containing data {... 1 7 1 3 0 19 0 21 0 19 0 2 1 16 1 15} is a half of a chromosome with values {7 3 -19 -21 -19 -2 16 15}.

– Time delay in sending and testing chromosomes. Because of non-concurrent nature of sender's processes it takes more time for the sender to run all the generations when compared to the receiver because the sender stops the genetic algorithm while sending data to the other robot. That is when there can occur transmission lags in that the sender can send the receiver chromosomes from generations that are older than the currently executed generations of the receiver. The learning of the receiver can suffer from this fact.

The consequences of the facts explained above will be more visible in the last experiment.

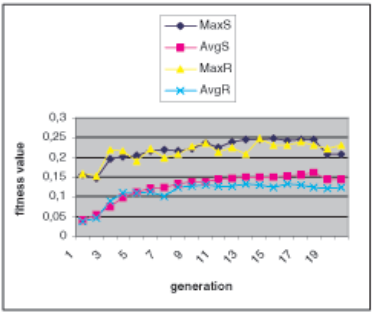

Fig. 6. Experiment with the population of 40 chromosomes. Max. and avg. fitness values for both robots.

4.3 Experiment 3

Even if both robots achieved the highest fitness values in this experiment (with 40 chromosomes in population), the receiver robot didn't perform better than the sender. Its best controllers were better in comparison with best-performing sender's controllers only in a couple of generations and it completely failed when we take average performance into account (see figure 6). In the last experiment the improvement in receiver's fitness got hardly positive values as depicted in figure 7a.

Figure 7b shows instability of the receiver's evolving populations that is on average higher than in previous experiment.

5 Conclusions

In this paper we proposed distributed genetic algorithm in a multi-robot society. We described the experiments with learning to avoid obstacles and compared the performance of a robot that learns only by individual evolution of controllers with the one that learns also by the experience of another robot through direct

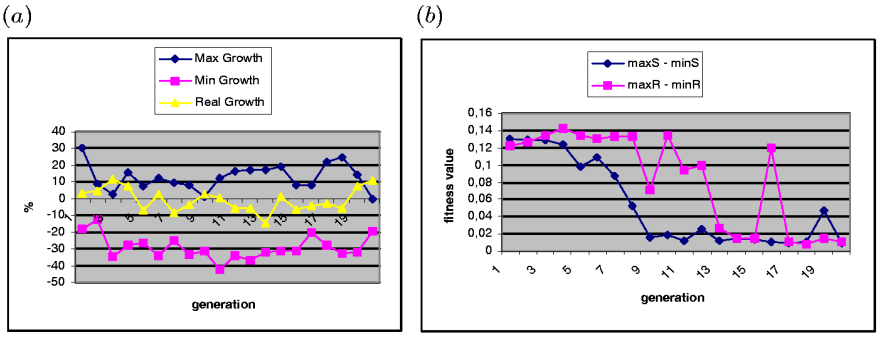

Fig. 7. Experiment with the population of 40 chromosomes. (a) Expected vs. real fitness growth in receiver's chromosomes. (b) Max. – min. fitness values taken from the run with the smallest difference.

exchange of chromosomes coding weight vectors of neural controllers. We discussed stability and the speed of learning in both cases as well as measured expected vs. observed growth in fitness values of chromosomes in the population of a second robot.

References

1. V. Braitenberg, *Vehicles. Experiments in Synthetic Psychology*, MIT Press, Cambridge, MA, 1984.
2. D. Floreano, and F. Mondada, Automatic Creation of an Autonomous Agent: Genetic Evolution of a Neural Network Driven Robot, From Animals to Animats III: Proceedings of the Third International Conference on Simulation of Adaptive Behavior (D. Cliff, P. Husbands, J. Meyer, and S. Wilson, eds.), MIT Press-Bradford Books, Cambridge, MA, pp. 402–410, 1994.
3. D. E. Goldberg, *Genetic algorithms in search, optimization and machine learning*, Addison-Wesley, Redwood City, CA, 1989.
4. J. H. Holland, *Adaptation in natural and artificial systems*, The University of Michigan Press, Ann Arbor, 1975.
5. J. Kelemen, and A. Kubík, RADIUS: Looking for Robot's Help in Computer Science Research and Education, *ERCIM News* (51), pp. 48–49, 2002.
6. F. Mondada, E. Franzi, and P. Ienne, Mobile Robot Miniaturization: A Tool for Investigation in Control Algorithms, *Proceedings of the Third International Symposium on Experimental Robotics*, Kyoto, Japan, 1993.
7. S. Nolfi, and D. Floreano, *Evolutionary Robotics: The Biology, Intelligence and Technology of Self-Organizing Machines*, MIT Press, Cambridge, MA, 2000.
8. S. Nolfi, and D. Floreano, Co-evolving Predator and Prey Robots: Do 'Arm Races' Arise in Artificial Evolution?, *Artificial Life*, Vol. 4(4), pp. 311–335, 1998.
9. M. Quinn, Evolving Communication Without Dedicated Communication Channels, *Advances in Artificial Life: Proceedings of the.* 6^{th} *European Conference on Artificial Life* (J. Kelemen, and P. Sosík, eds.), Springer Verlag, Berlin, pp. 357–366, 2001.

10. M. Quinn, L. Smith, G. Mayley, and P. Husbands, Evolving Teamwork and Role-Allocation with Real Robots, *Proceedings of the Eighth International Conference on Artificial Life* (R. K. Standish, M. A. Bedau, and H. A. Abbass, eds.), MIT Press, Cambridge, MA, pp. 302–311, 2002.
11. Watson, Richard A., Ficici, Sevan G. and Pollack, Jordan B., Embodied Evolution: Embodying an Evolutionary Algorithm in a Population of Robots. *1999 Congress on Evolutionary Computation* (Angeline, Michalewicz, Schoenauer, Yao, and Zalzala, eds.), IEEE Press, pp. 335–342, 1999.