Applying Genetic Programming to Evolve Behavior Primitives and Arbitrators for Mobile Robots

Wei-Po Lee John Hallam Henrik Hautop Lund
Department of Artificial Intelligence
University of Edinburgh
Edinburgh, U. K.
{weipol, john, henrikl}@dai.ed.ac.uk

Abstract— **Behavior-based approach has been successfully applied to design the control system of a robot. This paper presents our approach, based on evolutionary algorithms, to program behavior-based robots automatically. Instead of handcoding all the behavior controllers or evolving an entire control system for an overall task, we suggest our approach at the intermediate level: it includes evolving behavior primitives and behavior arbitrators for a mobile robot to achieve the specified tasks. To examine the developed approach, we evolve a control system for a moderate complicated box-pushing task as an example. We first evolved the controllers in simulation and then transferred them to the Khepera miniature robot. Experimental results show the promise of our approach and the evolved controllers are transferred to the real robot without loss of performance.**

I. Introduction

The behavior-based design has become a main alternative to traditional robot design nowadays. A number of behavior-based robots have demonstrated the corresponding performance [1; 2; 14]. In general, a behavior-based system can be considered as a behavior network consisting of some behavior modules. To design such a system is to design individual behavior modules first and then design some kind of coordination mechanism to manage the interaction between them. This approach has been proven successful in that its robots can deliver real-time performance in a dynamic world, by creating tight coupling between perceptions and actions for each behavior. However, increasing robot and task complexity makes the design difficult. Consequently, automatic design process using artificial evolution is advocated to synthesize the controllers.

Some researchers have used evolutionary algorithms to evolve robot controllers in the forms of LISP-like programs, neural networks, or classifier systems (see [11] and [4] for a brief survey). Unlike the behavior-based architectures, most of their work has no explicit predefined modular structures for the control systems, and they expect to create a comprehensive evolutionary system to evolve a single overall control system for a robot. However, the application tasks achieved in their work, such as obstacle avoidance, exploration, light seeking, are mostly too simple to promise that their work can be scaled up to achieve other more difficult tasks.

In this paper, we present our approach at the intermediate level between handcoding a behavior-based system and evolving an overall control system. It takes the behavior-based architecture as the control scheme and evolves the individual components, including *behavior primitives* and *behavior arbitrators*, by employing an evolutionary approach. This will take the advantages of both but mediate their disadvantages. In our work, the controllers are defined at the logic level: a controller is much like a boolean network, which uses *conditional structures* (constructed on sensor information) to select appropriate motor commands or activate a behavior primitive. A genetic programming system is implemented to evolve this kind of boolean controller.

To prove our approach, we describe how we employ it to undertake an application task, in which a robot is required to push a box toward a specific position indicating by a light source. This task is fairly complex compared to the ones in previous Evolutionary Robotics experiments. Due to the time-consuming characteristic of on-line evolution, we use the technique of developing our approach and evolving control systems in simulation and then transferring to a real robot. Experimental and testing results show that the robot can achieve the specified task reliably.

II. Real and Simulated Robots

The robot we used to test the evolved controllers is the miniature mobile robot *Khepera* [13] (Fig.1). The Khepera robot has a diameter of 55 mm, a height of 30 mm, and a weight of 70 g. Its two wheels are driven by two DC motors with incremental encoder and they can revolve forward and backward. The robot is equipped with eight infra-red proximity sensors (which can serve as ambient light sensors as well): six sensors are positioned on the front and the other two on the back. The sensors are numbered from 0 to 7 and the layout of the arrangement of the sensors is illustrated in Fig.1. The robot can be controlled by the host computer via a cable or run the control program on its own control chips. In the later experiments, the controllers evolved from simulation are downloaded to the robot to test the performance without the cable connection.

As mentioned previously, we use the approach of evolving in simulation and testing on real robot. Some research work has shown that the gap between simulated and real world can be bridged by sampling the real world through the sensor and motors of the robot itself to build the simulator, and the simulation time can be reduced significantly by using the look-up tables constructed from the sampled date of sensors and motors [12; 8]. In this work, we use this approach to build the tables to record the responses of infra-red sensors to a box, the responses of the ambient light sensors to the 25 Watt light source, and the motion of the robot at different commanded speeds. These tables are then used in simulation to evolve the control systems.

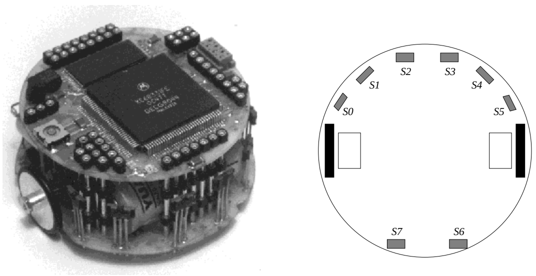

Fig. 1. The Khepera robot and its sensor arrangement. In the right figure, a sensor S_i can be an infra-red or an ambient light sensor.

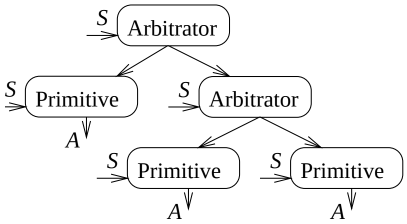

Fig. 2. The general architecture of a control system. S and A represent the sensors and actuators needed for a certain control work.

III. Evolving Control Systems

A. Control Architecture

Like the general behavior-based system, the architecture of our control system includes a set of behavior primitives and the behavior arbitrators to coordinate them. However, unlike the subsumption architecture [2], for instance, we do not hardwire a priority network in advance. A behavior arbitrator in our control system is treated as a reactive controller: it has the same structure as the primitives and the only difference is that the output of a primitive is used to control the motors and the output of an arbitrator is used to activate a behavior primitive. Thus, similar to a reactive planner [3], the arbitrators allow the binding between environment conditions and activations of lower level behaviors happening at the run time. This offers adaptiveness not only at the lower level sensor-motor control but also at the behavior level. Fig.2 illustrates the general architecture of our control systems.

B. The Application Task

To prove the developed approach, we use it to achieve a moderately difficult box-pushing task, which is to push a box toward a goal position indicated by a light source. Different versions of box-pushing tasks are accomplished by reinforcement learning [10] and genetic programming [6]. However, those tasks are simpler than ours because their robots are only required to push a box to any wall, which can be done without any deliberate strategy.

This task is difficult for several reasons. The robot only contacts the box at a point while pushing it, thus the box tends to slide and rotate unpredictably when the pushing force exerted by the robot is not directed straight through the center of the box. The robot has to adjust its position occasionally in order to keep pushing the box forward. Furthermore, as there is no restriction on the initial relative positions of the robot and box, the robot needs to move to a proper position deliberately to push a box, to satisfy the final goal. In learning such a task by a reactive controller, some failures could happen often: for instance, the robot keeps pushing a box but away from the light, or the robot itself moves toward the light without pushing the box.

To accomplish this task, we decomposed it into two other subtasks, *box-pushing* and *box-side-following*. The goal of *box-pushing* is to keep the robot pushing a box forward, while the goal of *box-side-following* is to require the robot moving along the side of a box. Each of them is controlled by a separate behavior primitive. In addition, a behavior arbitrator is involved to arrange the executing sequence of the behavior primitives. A genetic programming system is implemented to evolve both behavior primitives and arbitrators.

C. The Genetic System

The main evolutionary flow in this work is similar to the typical evolutionary approach. Given an environment and a goal formulated as a fitness function, an initial population is generated at random. After evaluating each individual, a certain selection scheme is used to choose parent individuals and some genetic operators are applied on them to create children individuals. In our implementation, the *tournament* selection scheme is used.

Three genetic operators, *reproduction*, *crossover*, and *mutation*, are applied to create a new generation. Because there are some constrained syntactic structures defined in this work, the crossover and mutation operations must be constrained, as explained in [7], to preserve the required structures.

In order to maintain the diversity and reduce the computation cost, an *island model* ([15]) GP system is implemented. Instead of using a single large population, it involves many small subpopulations, with an occasional exchange of useful information among the subpopulations. The migration policy used in this work is to copy a certain percentage of the best individuals in each subpopulation to substitute the same number of worst individuals in its neighbor subpopulations after a specified migration interval.

C.1 Representation

In our genetic system, the controller is reactive and is considered as a combinational logic system in which the output is determined completely by the current input states at each time step. In general, three types of non-terminals, the dummy root node, the logic components and the comparator, are defined for evolving the controllers. For the convenience of manipulating a GP system, a dummy root node is defined to connect some subtrees; the logic components are defined to constitute the main frame of the subtrees; and the comparator is defined to construct *sensor conditionals* which are described in the section below. The output of these subtrees are interpreted as actuator commands if a controller represents a behavior primitive, or used to activate other controllers if it is a behavior arbitrator.

According to our program design, structured sensor-conditionals involve comparing the values from different sensors or comparing the sensor values to the numerical values as thresholds. Sensors and thresholds, which are between 0 and 1 inclusive, are defined as terminals in our GP system. Depending on the characteristics of the specific tasks, different kinds of sensors are required and defined for evolving different behavioral modules. In general, a sensor terminal is defined to be associated with a value between 0 and 1 which indicates the angle between the direction in which the sensor is pointing and the robot's heading. Thus, whenever a sensor terminal is evaluated, the normalized sensor information, in the direction indicated by the value associated with it, is returned. For instance, a sensor terminal with a value 0.5 will return the converted sensor information in the direction 0.5 revolution (180 degrees) relative the robot's heading. In this way, the sensor positions and directions can be co-evolved with the controllers. The details are described in our previous research work [7]. In this work, we intend to use a fixed structure robot to verify our approach, so the sensors are fixed and named as S*0* to S*7*. The sensor arrangement is shown in Fig.1.

Thus, a structured sensor-conditional in this work is defined as a *constrained syntactic structure* X >= Y, where X, Y can be any terminal. The logic components then play the roles of mapping the structured sensor information into appropriate output. The aspect of a typical controller is illustrated in Fig.3.

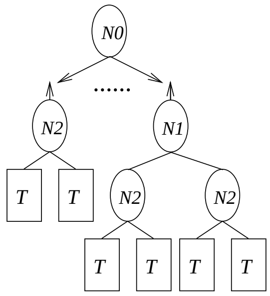

Fig. 3. The aspect of a typical controller. In this figure, $N1$, $N2$, $N3$ represent the three types of non-terminals, root-node, logic components, and comparator, respectively. The terminal T can be a normalized sensor response or a threshold between 0 and 1 inclusive.

C.2 Terminals and Non-terminals

As described earlier, there are three controllers, two behavior primitive and one arbitrator, involved in the application task. To apply a GP system to evolve these controllers, we need to define terminals and non-terminals for each of them. For both primitives, we defined infra-red sensor IR as the sensor terminal to provide distal information for the robot to locate the position of the box. In addition, numerical values are defined as the other kind of terminals to be the thresholds. For the behavior arbitrator here, both infra-red sensor IR and ambient light sensor LDR are defined as terminals to provide the perception clues of box and the target position. As in other controllers, numerical values are defined as the terminals to be the thresholds.

Although different control work needs different kinds of sensor terminals, they use the same non-terminals, except the number of arguments of a root node: it depends on how the output of the subtrees of a controller is interpreted. We do not distinguish them here and defined the root as PROG in general. Thus the non-terminals for evolving all the primitives and arbitrators are defined to include the root node PROG, the logic components AND, OR, NOT, XOR, and the comparator >=.

IV. Experiments and Results

In order to evolve a robust controller, we train a controller in multiple trials. For each task, we defined a training set including some training cases which are sampled randomly for training the controllers during the run. The best individual appearing in the last generation is designated as the final solution.

To evaluate a controller is to execute this controller for a given number of time steps and to measure its performance by certain criteria (fitness function). The accumulated fitness of a controller during the given time steps represents its performance. In the following experiments, a fitness function is defined as a penalizing function which needs to be minimized. The rest of this section describes the evaluation criteria for each control work and presents the results.

A. Evolving Primitive Box-Pushing

Box-pushing is to keep a robot pushing a box as straight as possible. It can be described as to keep the activation value of its front IR sensor high; the robot moving forward; and the speed difference of two motors low. The pressure from keeping the front distal sensor with high activation value will reinforce the robot to approach and face a box, and the pressure from keeping robot moving forward with low speed difference is to encourage the robot move straight and prevent it from getting stuck in front of a box. The combination of the these can lead a pushing forward behavior. The fitness function for a controller C is defined as:

$$f(C) = \sum_{t=1}^{T} \alpha * (1 - s(t)) + \beta * (1 - v(t)) + \gamma * w(t)$$

in which $s(t)$ is the average of normalized sensor activations of the front sensors IR*2* and IR*3*; $v(t)$ is the normalized forward speed; and the $w(t)$ is the normalized speed difference of two motors at each time step t.

Since this controller is a primitive, the output is grouped as two sets and converted to motor commands to drive two motors: the first three and the second three subtrees are decoded as motor commands for left and right motor, respectively. In our experiments, each motor command lasted 200 ms for both simulated and real robots.

For a certain run, two populations of 50 individuals were used and each individual was evaluated for 150 time steps. The evolution process lasted 50 generations and the best individual appearing in the last generation is:

```
(PROG
(OR (>= 0.13 IR0)(>= 0.13 IR0))
(>= IR1 IR1)
(XOR (>= IR3 IR7)(OR (>= 0.13 IR0)(>= 0.13 IR0)))
(>= IR1 0.36)
(>= IR1 IR1)
(XOR (>= 0.13 IR0)(>= IR2 0.13)))
```

After evolving in simulation, we transferred the above evolved controller to the Khepera robot. Fig.4 shows the typical behaviors of the simulated (left) and real (right) robots. This controller was tested many times on the real robot and each time it started from an arbitrary position and heading, in which it can sense the box. It always generated the consistent behavior which is to turn to face the box and then to approach and push it. Although the robot did not contact and push the box exactly at the point of front center, it did achieve the goal reliably: continuously pushing the box forward.

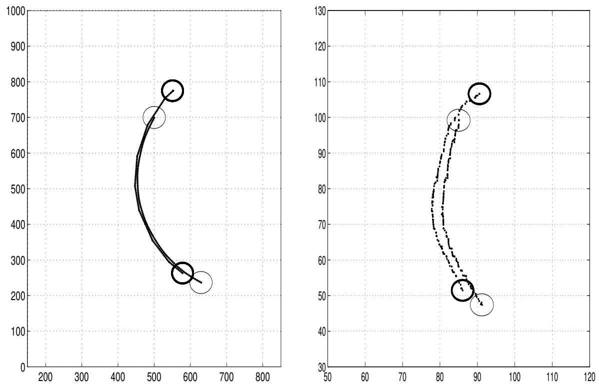

Fig. 4. The trajectories of simulated (left) and real (right) robots in pushing a box (the darker circles represent the robots). The figure for real robot is conducted by setting LEDs on the tops of the robot and box and using a video tracking system [9] to record their trajectories.

B. Evolving Primitive Box-Side-Circling

Box-side-circling is to keep a robot moving forward and circling along the sides of a box. To prevent itself from touching the box too often, the robot is encouraged to keep a certain distance away from the box while circling the box. The evaluating criteria are described as to keep the robot's speed positive and the sensor IR*1* with a certain activation value. The fitness function is then defined as:

$$f(C) = \sum_{t=1}^{T} \alpha * abs(s(t) - k) + \beta * (1 - v(t))$$

where *abs* is a function which gives the absolute value of it argument; $s(t)$ is a normalized activation value of the specific sensor; k is a pre-defined constant indicating the distance between a robot and the box, in terms of the normalized sensor range; and v is the forward speed of a robot.

As in the *box-pushing* control work, this controller is a primitive and the output is interpreted exactly the same as the *box-pushing* primitive. In this experiment, two populations of 50 individuals were used. Each individual was evaluated 150 time steps and the number of generations was 50. The resulting controller is:

```
(PROG
(NOT (>= IR6 IR1))
(>= IR1 0.78)
(OR (>= IR5 IR1)(>= IR3 IR3))
(OR (>= 0.32 IR3)(>= IR5 IR4))
(>= IR4 IR3)
(OR (>= IR5 IR1)(>= IR6 IR3)))
```

Fig.5 shows the typical behaviors of the simulated and real robots. We tested this evolved controller several times by putting the real robot around the box with an arbitrary heading each time. In all the tests, the robot showed the similar behavior: it performed turning until the specific sensor IR*1* sensed the box and then moved along the side of the box. From the testing results, we can see that the robot has been able to achieve the specified goal and generate reliable behavior.

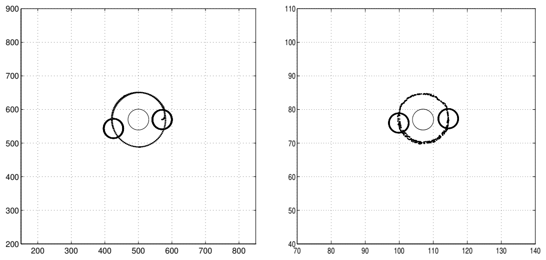

Fig. 5. The behaviors of simulated (left) and real (right) robots.

C. Evolving the Behavior Arbitrator

Having evolved the first two controllers *box-push* and *box-side-circling*, we then evolve the third controller serving as the decision-making mechanism to manage the first two controllers to achieve the task of pushing a box toward a light source.

To achieve this goal, the robot needs both infra-red sensors and ambient light sensors to detect the box and the light, and the latter must be higher than the box to be able to receive the light in the case that the box is between the robot and the light. However, in the original design of the Khepera robot, there is no physical distinction between infra-red and ambient light sensors: the eight sensors equipped on the robot serve as both. We are currently duplicating the eight sensors on the top of our Khepera robot to test the evolved arbitrator. The results presented here are from the simulation. In the simulation, the look-up table of the light response is constructed from the existing sensors and we assume that, in addition, there are equivalent sensors on top of the simulated robot to respond to the light.

In this experiment, the box was placed 22 cm away from the light, which was masked by wrapping a piece of paper around it and placed 11 cm above the area. The robot is expected to push the box as close as possible to the center of the area brightened by the light. The fitness function is defined as:

$$f(C) = \sum_{t=1}^{T} D_{b,l}(t)$$

in which $D_{b,l}(t)$ represents the distance between the box and the light source at each time step t.

This controller is a behavior arbitrator and its single output is used to activate one of the above evolved primitives at each time step, according to the environment conditions. For a single run, 4 populations of 40 individuals were used and each individual was trained to act 600 time steps. The evolved controller is:

(PROG
(OR (OR (>= 0.62 LDR*5*))(OR (OR (>= IR*6* LDR*3*)(>= 0.62 LDR*7*)) (AND (>= LDR*6* LDR*7*)(>= LDR*3* LDR*7*)))) (AND (>= LDR*3* LDR*7*)(OR (OR (OR (>= LDR*5* LDR*3*)(>= LDR*3* 0.62)) (>= IR*6* LDR*7*))(AND (NOT (>= IR*6* LDR*4*))(>= LDR*3* LDR*7*))))))

Fig.6 illustrates the typical behavior of the simulated robot. The arbitrator first activated the primitive *box-side-circling* to move along the side of a box to an appropriate position in which the box was between the light and the robot itself. It then switched to the other primitive, *box-pushing*, to push the box forward. The *box-side-circling* behavior was activated again if the robot's path was biased. From Fig.6, we can see that the box was pushed to almost the center of the bright area and the robot continuously performed *box-side-circling* primitive to prevent pushing the box away from the box, after the box was pushed to the target position.

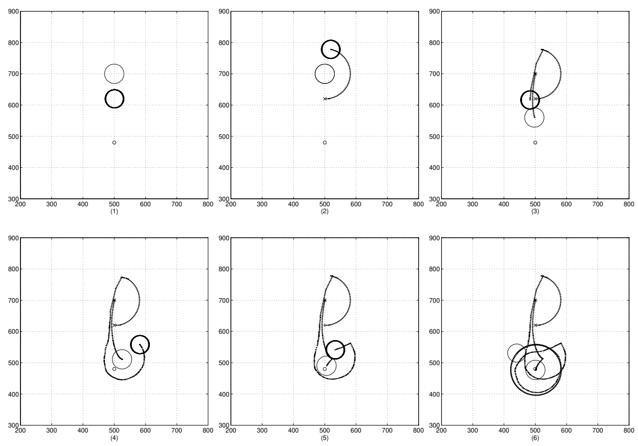

Fig. 6. The behavior sequence of the robot: (1) The initial positions of the robot, the box and the light; (2) the robot moved along the side of the box; (3) pushing the box forward; (4) circling again to an appropriate position; (5) pushing the box again to the goal position; (6) continuously circling after the box has been pushed to the goal position.

V. Conclusion and Future Work

In this paper, we have presented our approach in applying evolutionary techniques to evolve the control system for a simulated robot and then transferred evolved controllers to a real robot. Instead of hand-coding all of the individual components needed in a general behavior system or evolving the whole control system for an overall task, we propose to take the control architecture of a behavior-based system and construct the behavior primitives and arbitrators by the use of an evolutionary approach. To assess the performance of the developed approach, we have evolved a control system to achieve an application task of box-pushing as an example. Experimental results show the promise and efficiency of the presented approach.

By employing the evolutionary approach, our work shows a way to aid the design of a behavior-based robot, including automatically programming all the individual behavior controllers and dealing with the selection problem among these behavior controllers. Since the control architecture in our work is arranged to be similar to that of behavior-based robots, the resulting control system therefore inherits the characteristic of the behavior-based robots: it delivers real-time performance.

As far as applying an evolutionary approach is concerned, our approach also has several potential advantages. First of all, owing to the separate design of behavior primitives and arbitrators, the goal of each task is relatively simple. It means that designing a fitness function for such a task is not as difficult as generally thought. Secondly, since each individual controller involves fewer input and outputs, one can apply a simple evolutionary system to evolve a robust controller without too much effort. For example, in the application task achieved in this paper, the evolutionary system can converge to a stable and sufficient solution within only 30 generations. Thirdly, our controllers only involve logical operators, such as AND OR NOT, that are very simple to evaluate. This means that our approach is computationally cheaper; especially, in the cases that the first argument of a AND is false, or the first argument of a OR is true, there is no need to evaluate the other argument. Finally, because our controllers are constituted of very basic logic components, they can be easily compiled to custom hardware such as FPGAs to speed up the evaluation in controlling a robot.

Some further research work is currently in progress. The first is the construction of new sensors on the real robot to test the behavior arbitrator. Another is to investigate whether this approach can be applied to evolve control systems for even more complicated tasks.

References

[1] R. C. Arkin. Motor Schema Based Navigation for a Mobile Robot: An Approach to Programming Behavior. In *Proceedings of IEEE International Conference on Robotics and Automation*, p264-271, 1987.

[2] R. A. Brooks. A Robust Layered Control System for a Mobile Robot. In *IEEE Journal of Robots and Automation*, vol RA-2(1), p14-23, 1986.

[3] R. J. Firby. Task Networks for Controlling Continuous Processes. In *Proceedings of the Second International Conference on AI Planning Systems*, p49-54, 1994.

[4] T. Gomi, A. Griffith. Evolutionary Robotics – An Overview. In *Proceedings of IEEE International Conference on Evolutionary Computation*, 1996.

[5] J. R. Koza. *Genetic Programming: on the Programming of Computers by Means of Natural Selection*, MIT Press, 1992.

[6] J. R. Koza, J. P. Rice. Automatic Programming of Robots using Genetic Programming. In *Proceedings of AAAI-92*, p194-201, 1992.

[7] W.-P. Lee, J. Hallam, H. H. Lund. A Hybrid GP/GA Approach for Co-evolving Controllers and Robot Bodies to Achieve Fitness-Specified Tasks. In *Proceedings of IEEE International Conference on Evolutionary Computation*, 1996.

[8] H. H. Lund, J. Hallam. Sufficient Neurocontrollers can be Surprisely Simple. Research Paper 824, Department of Artificial Intelligence, University of Edinburgh, 1996.

[9] H. H. Lund, E. d. Ves, J. Hallam. Mobile Robot Tracking. Research Paper, Department of Artificial Intelligence, University of Edinburgh, 1996.

[10] S. Mahadevan, J. Connell. Automatic Programming of Behavior Based Robots Using Reinforcement Learning. In *Proceedings of AAAI-91*, p768-773, 1991.

[11] M. Mataric, D. Cliff. Challenges in Evolving Controllers for Physical Robots. Technical Report CS-95-184, Department of Computer Science, Brandeis University.

[12] O. Miglino, H. H. Lund, S. Nolfi. Evolving Mobile Robots in Simulated and Real Environments. In *Artificial Life*, 2(4), 1996.

[13] F. Mondada, E. Franzi, P. Ienne. Mobile Robot Miniaturation: A Tool for Investigation in Control Algorithms. In *Proceedings of the Third International Symposium on Experimental Robotics*, 1993.

[14] L. Steels. Building Agents out of Autonomous Behavior Systems. In *The Artificial Life Route to Artificial Intelligence.* L. Steels and R. Brooks (eds), 1993.

[15] R. Tanese, Distributed Genetic Algorithms. In *Proceedings of the Third International Conference on Genetic Algorithms*, 1987, p434-439.