Accepted for publication in *International Journal of Smart Engineering Systems Design*

Evolving and Optimizing Braitenberg Vehicles by Means of Evolution Strategies

Ralf Salomon
AI Lab, Department of Computer Science, University of Zurich
Winterthurerstrasse 190, 8057 Zurich, Switzerland
FAX: +41-1-363 00 35; Email: salomon@ifi.unizh.ch

Abstract

This paper presents a practical application of the evolution strategy to the evolution and optimization of Braitenberg vehicles. Braitenberg vehicles are a special class of autonomous agents. Autonomous agents are embodied systems that behave in the real world without any human control. One major goal of research on autonomous agents is to study intelligence as the result of a system environment interaction, rather than understanding intelligence on a computational level. Braitenberg vehicles are controlled by a number of parameters, which are mostly determined by hand in a trial and error process. This paper shows that a simple evolution strategy evolves Braitenberg vehicles very efficiently. Other research has used genetic algorithms for very similar tasks. A comparison of both approaches shows that the evolution strategy approach is much faster; the evolution strategy approach takes about one and a half hours, whereas the genetic algorithm approach takes more than two and a half days. Since autonomous agents are very important in the field of new AI, the results presented in this paper suggest that this research field should spend more attention to evolution strategies.

Keywords: Evolutionary Algorithms, Autonomous Agents, Embedded Systems, Artificial Intelligence, Optimization, Neural Control.

1 INTRODUCTION

Autonomous agents are autonomous, self-sufficient embodied systems (robots) [3] that have to behave in their environment. Autonomous means that the agent operates without any human control. Self sufficient means that the agent can maintain its internal energy level over a long time. The idea behind self sufficiency is the agent gets some reward after doing useful work, where "useful" depends on the context or the ecological niche. An example is a robot that collects empty soda cans. Autonomous agents are equipped with sensors and effectors, such as motors or grippers. An agent perceives its environment through sensors like infrared sensors and it behaves and manipulates its environment by using the effectors.

One major goal of research in autonomous agents is to study intelligence as the result of a system environment interaction, rather than understanding intelligence on a computational level. Thus, autonomous agents are very important in the field of new AI. A first example of behavior-based intelligence can be found in [15], and an overview of special issues involved in autonomous agent design can be found in [9, 16].

The *Khepera*^{TM} robot is an example of a small robot, which is widely used for research on autonomous agents. Section 2 describes Khepera in more detail. In order to achieve the goal of operating autonomously, an agent has to perform different tasks, such as exploring the environment, moving around, avoiding obstacles, and so forth. The agent's behavior is controlled by a control system, which is typically implemented as a neural network. Such a controller uses the sensor readings and its internal state to determine the agent's next action. Section 3 describes a simple neural control architecture called Braitenberg vehicle [2].

Determining the weights for a Braitenberg architecture is much different from training neural networks for standard applications. Rather than working with a fixed set of training patterns, which is selected prior to training, an autonomous agent determines the pattern's relevance at runtime by behaving in its environment. Furthermore, the sensory data is very noisy and the data dynamically changes due to the agent's interaction with the environment.

Due to the ubiquitous uncertainties and the problem of applying standard neural network training procedures [13], evolutionary algorithms are adequate candidates for the evolution and optimization of Braitenberg vehicles. Evolutionary algorithms are population-based, heuristic search procedures that incorporate random variation and selection. The three mainstreams of methods are evolution strategies (ESs) [11, 14], genetic algorithms (GAs) [7] and evolutionary programming [6]. Each of these algorithms maintain a population of promising candidate solution. In each generation, a set of offspring is generated by applying random variation operators, such as mutation and recombination. Each offspring is evaluated by assigning it a fitness value, and the best are typically selected as parents for the next generation. Among some other differences, genetic algorithms typically apply a mutation to only one parameter per offspring, whereas evolution strategies [11, 14] typically apply mutations to all parameters. In addition, evolution strategies control the amount of change by maintaining a step size for each individual. Prior to applying a mutation, the step size is randomly changed, and statistically those offspring survive that have the best adapted step size. For further details concerning the step size adaptation, the interest reader is referred to [1, 14].

The experimental setup is presented in Section 4. It consists of an arena and the fitness function. The arena is made from wood of size 60x45 cm with walls of 5 cm hight. The fitness function is designed such that the evolved agent favors straight movements while it avoids obstacles. Section 5 reports some experiments obtained by applying a (3,6)-ES, which generates 6 offspring and selects the 3 best of them as parents for the next generation, to the evolution of Braitenberg vehicles. The results show that such a simple evolution strategy evolves reasonable controllers within 30 generations, which take approximately one and a half hours. Other research [4, 5, 10] uses genetic algorithms [7] for almost the same task. A comparison of both approaches show that the evolution-strategy approach is much faster; the genetic algorithm approach takes more than two and a half days. Thus, the obtained speedup is of great practical relevance. For a comparison of evolution strategies and GAs see [1]. Some of the results obtained are further discussed in Section 6, and Section 7 gives

Figure 1: The Khepera robot.

a short conclusion.

2 THE KHEPERA ROBOT

The Khepera robot (cf. Figure 1) is 55 mm in diameter and 32 mm high. The robot is equipped with eight infrared and eight ambient light sensors (cf. Figure 2) as well as two motors, which can be controlled independently. Khepera's sensors and motors are controlled by a Motorola 68331 micro controller. Khepera can operate in two modes. In the first mode, a program is downloaded into the on-board memory, which allows Khepera to operate without any further hardware. In the second mode, Khepera is connected with a workstation via a serial link. In the experiments reported in this paper, the robot was controlled from a workstation and the ambient light sensors were not used. A detailed description of the robot and its electrical parts can be found in [8].

The motors can be controlled independently of each other by sending commands (i.e., function calls) to the robot. Valid speed values are in the range $[-40, 40]$; inside the program, these values are normalized such that they are in the range $[-1, 1]$. To avoid problems caused by the floating-point-to-integer conversion, [0,1]-equal-distributed random numbers are added to the motor speeds at each time step. By setting both motors to the same speed but with different signs, for example, the robot spins on the spot. The robot interprets the speed settings as commands. Internal PID controllers take care of the robot's dynamics. However, rapidly changing motor commands induce additional dynamics, which can cause problems for the fitness evaluation. By means of attached wheel encoders, the robot measures the motor's real speed, which differ from the command setting during situations, in which the robot cannot move. The real speeds can be obtained by sending special commands to the robot.

Khepera is equipped with eight infrared proximity sensors. The sensor readings are of type integer and the values are in the range $[0, 1023]$. Within the program, the sensor readings are normalized such that the values are in the range $[0, 1]$. The sensors give reasonable input values for object distances between 10 mm and 60 mm. A sensor value of 1023 indicates

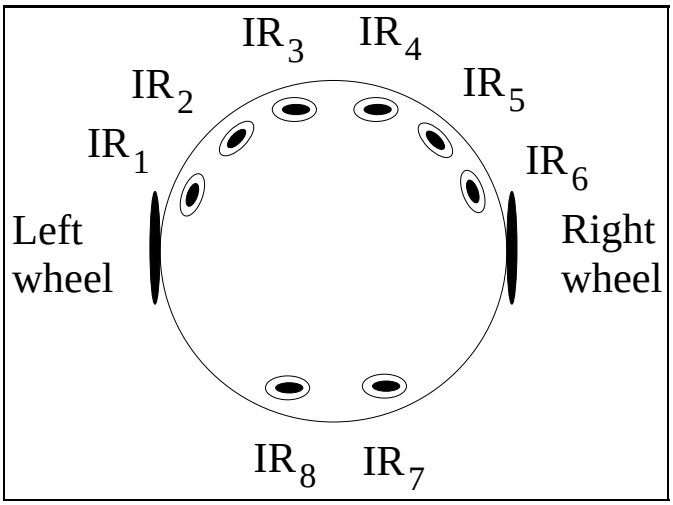

Figure 2: The approximate location of the eight infrared sensors $IR_1 \ldots IR_8$.

that the robot is very close to the object, and a sensor value of 0 indicates that the robot does not receive any reflection of the infrared signal.

A major problem with Khepera is that the sensor readings are very noisy, which causes several problems for the fitness evaluation of a given controller, i.e., the weights of the Braitenberg network. The effect of the noisy sensors to the fitness evaluation can be seen in Figure 3. Figure 3 shows how the fitness contributions f_t are dynamically changing under constant environmental conditions, i.e., constant motor speeds, constant ambient light, and constant position in the arena. Furthermore, the sensor readings are subject to several environmental conditions, such as the material the object is made of, the object's surface,

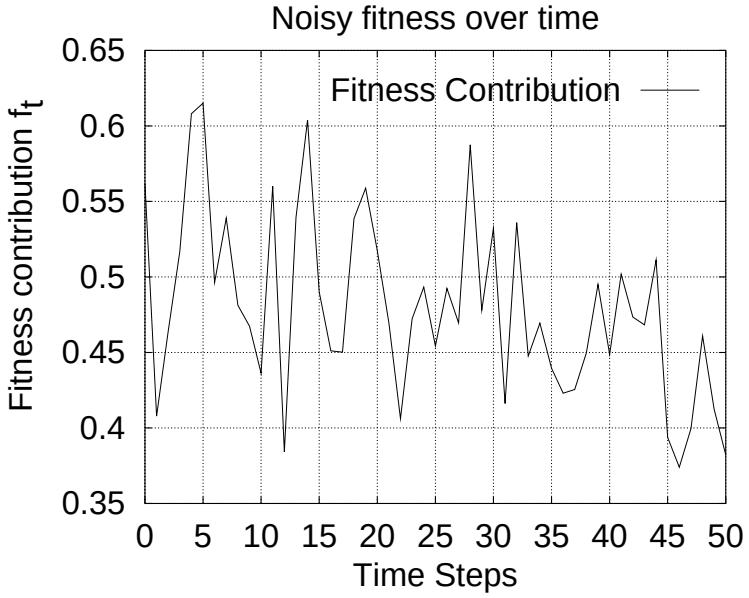

Figure 3: The noisy sensor readings cause a dynamical change of the individual fitness contributions f_t even under constant environmental conditions.

and the ambient light. During a long series of experiments, the environment cannot be kept constant. In summary, the fitness evaluation is extremely noisy.

3 BRAITENBERG VEHICLES

As outlined in the introduction, an autonomous agent cannot sit somewhere while doing nothing, since it would consume energy and would eventually die. Rather, the agent has to *operate* in its environment. Thus, moving around while avoiding obstacles is a key issue in autonomous agent research. Braitenberg [2] has proposed a simple architectures for such tasks. Figure 4 shows a control architecture inspired by a Braitenberg type-3c vehicle. The main idea is that a sensor with a high proximity activation accelerates the motor on the sensor's side whereas this sensor slows down the motor on the opposite side. By this principle, the presence of an obstacle leads to different motor speeds, which causes the robot a turn. Depending on the activation of all proximity sensors, the robot either turns, spins on the spot, or even backs up.

Braitenberg type-3c architectures are simple and straight forward, but finding appropriate weights is anything but easy. To this end, the control architecture is typically implemented as a neural network. The activation of the left motor M^l is calculated by the following formula

$$M^l = \sum_{i=1}^{8} IP_i w_i^l + w_0^l \qquad , \tag{1}$$

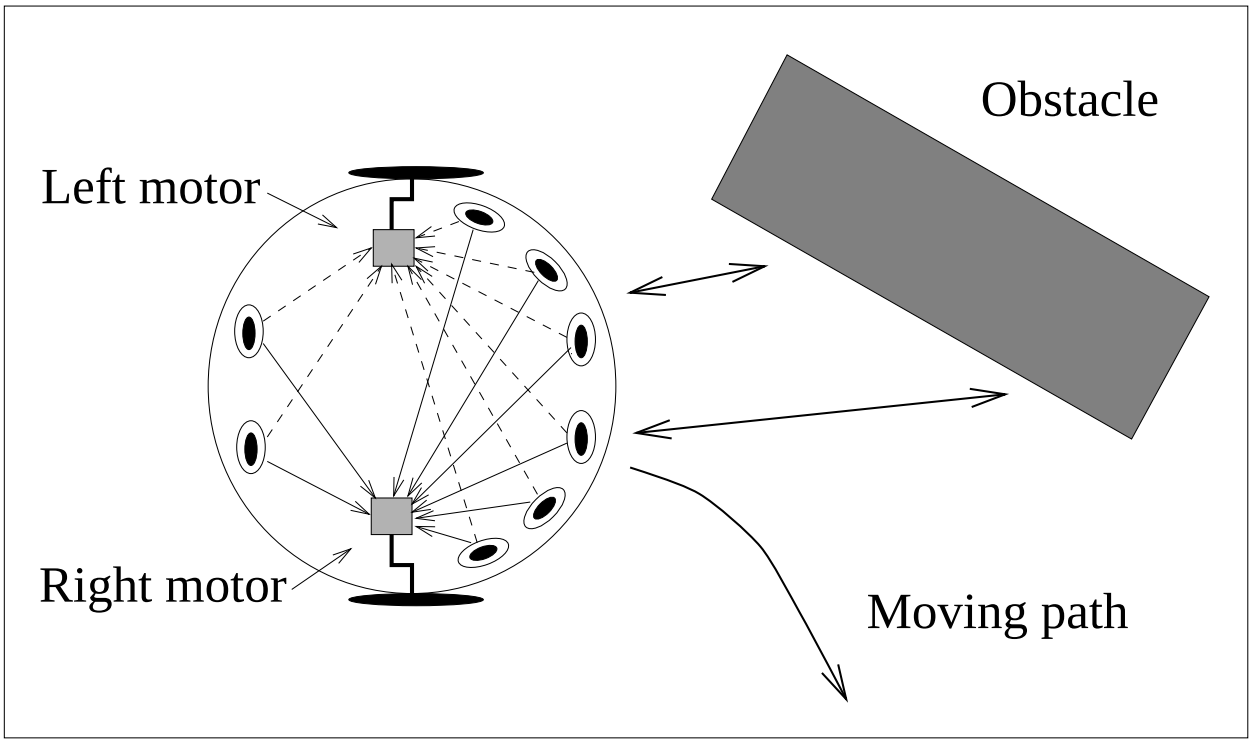

Figure 4: A control architecture inspired by a Braitenberg type-3C vehicle. The sensory information controls the motors via inhibitory and excitatory connections.

where IP_i denotes the activation of the proximity sensor i and w_i^l denotes the weight that connects proximity sensor IP_i with the left motor. The weight w_0^l represents the idle activation of the left motor. This "bias" weight is responsible for the robot's forward movement in the absence of any obstacle. The calculation of the right motor's activation is similarly given by

$$M^r = \sum_{i=1}^{8} IP_i w_i^r + w_0^r \qquad . \tag{2}$$

The main problem is to determine the weights w_i^l, w_i^r, w_0^l, and w_0^r such that the robot is moving around while it avoids obstacles. Mostly, this is done in a trial and error process. Standard neural network training procedures cannot be used, since it is not reasonable to determine a set of training patterns prior to training. Alternatively, evolutionary algorithms are an adequate candidate for this optimization problem.

4 THE EXPERIMENTAL SETUP

The experimental setup has been chosen as close as possible to setups proposed in other research [4, 5, 10]. Figure 5 shows the arena in which the robot has to move. The arena is of size 60x45 cm and the walls are made from wood with a height of 3 cm. The width of the corridors is chosen such that always at least one proximity sensor has a less-than-maximum value. In all experiments, the robot was started in the indicated position. Although averaging

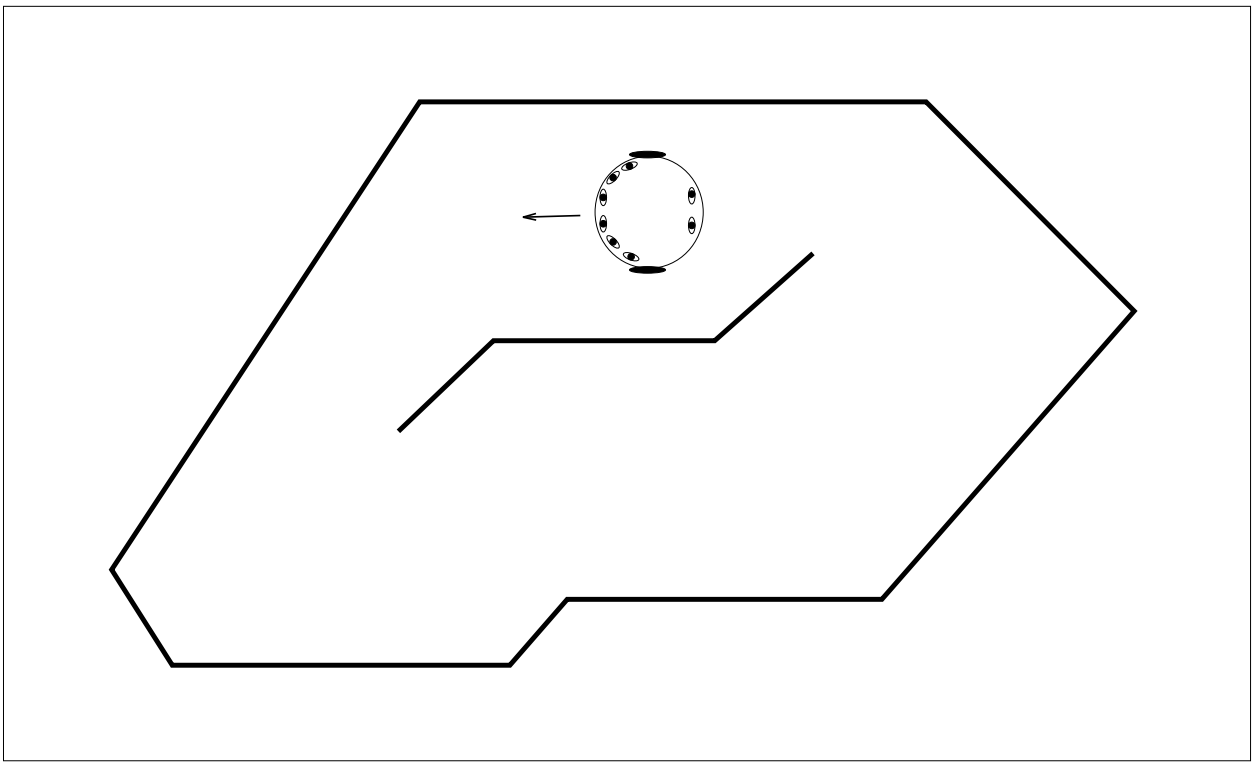

Figure 5: The arena of approximate size 60x45 cm. The indicated position (facing left) is the starting point for fitness evaluation.

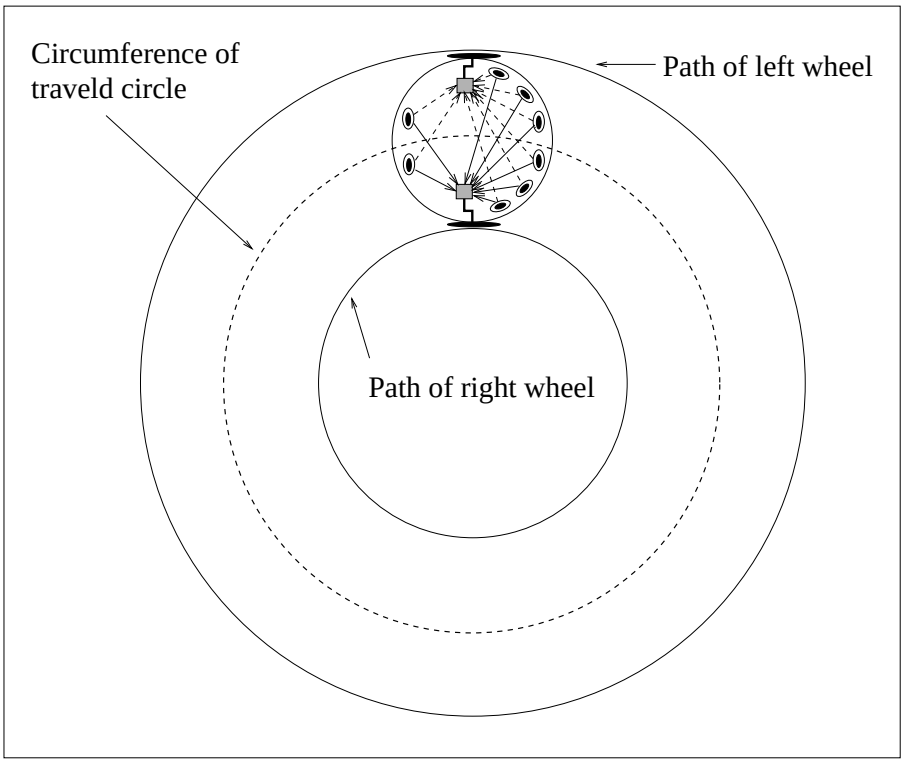

Figure 6: With different motor speeds V_l and V_r, the robot travels on a circle.

over several different starting positions would be very useful, but fitness evaluation already takes so much time that averaging over different starting positions would be infeasible.

As already discussed, the robot in such an arena has to move forward quickly while it has to avoid obstacles. In order to evolve good Braitenberg vehicles, the fitness function has to incorporate the motor speeds and the distance to obstacles. However, using speed and distance only is not sufficient. In such a case, a robot that is spinning on the spot with a high speed far away from any obstacle would have a high fitness. But such a robot would not do anything useful (e.g., collecting trash or cleaning a floor). Therefore, the fitness function is to be enhanced by a third term that favors straight movements by penalizing turns (see also [4, 5, 10]).

The fitness is measured as follows. At a particular time step t, both motor speeds V_l and V_r as well as all eight proximity sensors IR_i are measured. Then, the speed of the robot's center $V_t = (V_l + V_r)/2$, the penalty term $\Delta v_t = |V_l - V_r|$, and the sensor with the highest activation $\hat{IP} = \max_i IP_i$ is calculated. The fitness contribution f_t for step t is then

$$f_t = V_t(1 - \sqrt{\Delta v_t})(1 - \hat{IP_t}) \quad . \tag{3}$$

Finally, the total fitness is the sum over $t_{\max}$ (e.g., 240) time steps

$$F = \sum_{t=1}^{t_{\max}} f_t = \sum_{t=1}^{t_{\max}} V_t(1 - \sqrt{\Delta v_t})(1 - \hat{IP_t}) \quad . \tag{4}$$

The fitness function (4) is adopted from other research and could have been defined differently. One possible alternative might incorporate the circumference of the traveled circle as depicted in Figure 6. The circumference is already indirectly included by the penalty term $\sqrt{\Delta v_t}$. When traveling on a straight line, the penalty term vanishes, since both speeds V_l and V_r are equal, and when traveling on a small circle, the penalty term obtains positive values, which depend on the circle's radius. Also, the fitness function can be composed as the sum of the three terms as opposed to the product specified in (3). It can be expected that such a definition would yield similar results. The main reason for using (3) is that it is the original design proposed in earlier work, and a comparison with other research would be problematic if a different fitness function is used. Therefore, (3) is used throughout this paper.

Khepera's on-board controller allows for sending all sensor values every each 100 milliseconds (ms). That means, the robot moves for 100 ms with constant motor speeds. Then, the controller receives the new values and calculates new motor speeds for the next time step. Meanwhile, the individual fitness component f_t is calculated and added to the total fitness.

5 EXPERIMENTS

This section reports some typical results when evolving Braitenberg vehicle by means of a $(3, 6)$-ES with self-adaptation of the step size [14]. Some of the results are discussed in Section 6. A $(3, 6)$-ES generates 6 new offspring per generation and selects the 3 fittest individuals as parents for the next generation. A $(3, 6)$-selection scheme implies that the strategy does not use any elitist selection scheme. A non-elitist selection scheme was chosen, since the fitness evaluation is extremely noise (cf. Figure 3), from which spurious candidates may emerge. In the experiments reported in this paper, the evolution strategy generates two offspring without crossover, two with uniform recombination, and two with intermediate recombination, and the initial standard deviation was set to 0.5. A further source of additional noise is that also the individual's starting conditions vary. Over a long time, the environmental conditions cannot be hold constant; this phenomenon is intrinsic to real-world applications. For each series of experiments, six to eight runs were performed and all presented figures are typical with respect to these runs.

In the evolution of Braitenberg vehicles, a main problem is to find a first controller that exhibits an even tiny reasonable behavior. Initializing all weights at random leads to an agent that immediately crashes into the wall, where it gets stuck. To help guide the evolution process, all weights were initialized with very small negative random weights w_i^l and w_i^r (i.e., $[-0.5, 0]$) and the weights w_0^l and w_0^r were set to very small positive values (i.e., $[0, 0.1]$). Furthermore, the first series of experiments exploit the morphology of the robots, i.e., the left and right part of the controllers are constrained to be equal $w_0^l = w_0^r$ and $w_i^l = w_{(14-i) \bmod 8+1}^r$. This constraint leads to a reduced search space of nine parameters. In all experiments, fitness evaluation was done over 240 time steps. Since one time step requires 100 ms, the evaluation of each controller takes about 24 seconds.

A typical run of the evolution of a *constrained* Braitenberg controller can be seen in Figure 7. Figure 7 shows the fitness of the population's fittest agent and the average of the whole population. In the first few generations, most vehicles are sitting at the initial position or

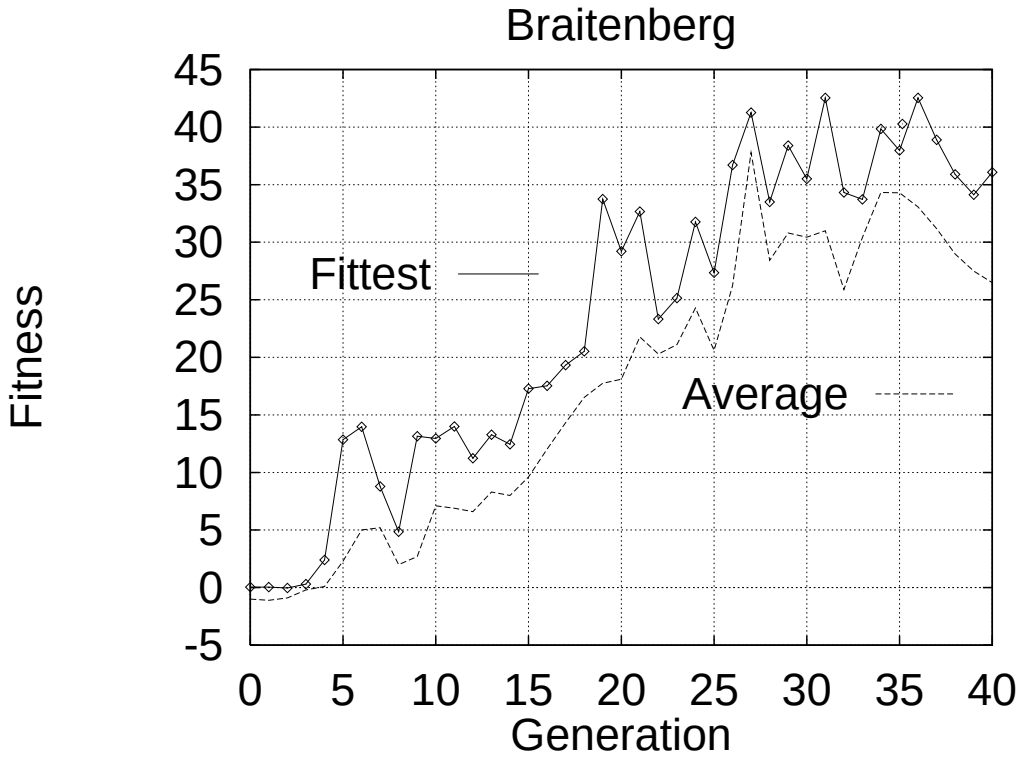

Figure 7: A typical run of the evolution of a *constrained* Braitenberg vehicle. The upper graph represents the best individual whereas the lower graph prepresents the average of the whole population.

crashing backwards into the wall resulting in a negative fitness. After a few more generations, some agents are moving forward, but got stuck at the wall after some additional steps; at that time, the avoidance behavior was not sufficient. However, such a short forward movement results in a small positive fitness, which is a first step towards a useful agent. After about eight to ten generations, the fittest agents are moving slowly inside the corridor, although at time they continued hitting the walls. After hitting a wall, good controllers set the motor speeds to negative values, which cause the robots to back up from the wall, and after turning, they continue moving inside the corridor. After 30 generations, the fittest agents perform up to three complete laps in the arena. Even though both controller sides are constrained to be equal, identical weights do not ensure total symmetric behavior. Tolerances in the electrical characteristics of the motors and sensors impose an unsymmetric behavior. Consequently, such controllers have to find a good compromise. As a result, Braitenberg vehicles with constrained controllers move forward in an almost straight line, and turn if they approach an obstacle. Overall, a Braitenberg vehicle with a constrained controller moves with a high speed but with a rather rough trajectory.

Figure 8 shows a typical run of the evolution of an *unconstrained* Braitenberg controller. Such a controller has 18 parameters and the evolution strategy has to evolve both sides of the controller, i.e., the connections for the left motor activation M^l as well as the right motor activation M^r. Thus, as can be seen in Figs. 7 and 8, the evolution of an unconstrained Braitenberg controller requires more time, and also, the final performance is lower than the

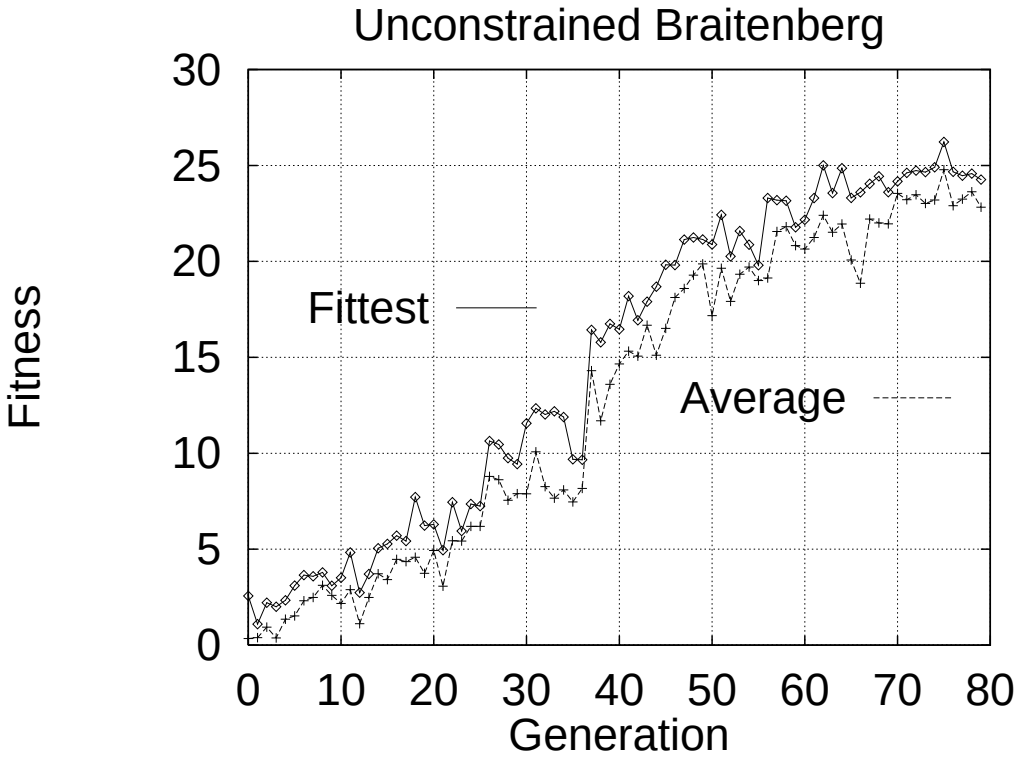

Figure 8: A typical run of the evolution of a *unconstrained* Braitenberg vehicle. The upper graph represents the best individual whereas the lower graph prepresents the average of the whole population.

performance of the constrained controller. The unconstrained controller develops a different survival strategy. From the very beginning, the first controllers have different bias weights w_0^l and w_0^r, which causes the robot to turn in small circles. Circling around results in a small positive fitness. During the next generations, this circling process is preserved, but the radius becomes larger and larger. After approximately 30 generations, one controller side improves its object-avoidance behavior so that it prevents the robot from crashing into the wall; the resulting behavior can be interpreted as wall following. This wall-following behavior is not encoded in the fitness function. Rather, it is emerged from the evolution process. In the ongoing evolution process, this wall-following behavior is further improved, and after about 80 generations, most robots perform two complete laps in the arena.

6 DISCUSSION

In Section 5, two different controller types were investigated, namely constrained and unconstrained controllers. The constrained control architecture is inspired by the biological observation that most animals are of symmetric shape. A constrained controller has less parameters, which accelerates the evolution process, but, at the same time, limits the number of potential solutions. As a consequence, both controllers develop different survival strategies.

The relations between both controller types were further investigated in a series of con-

trol experiments, in which the weights of unconstrained controllers were initialized with weights taken from an evolved constrained controller. In the first five to ten generations, these controllers were loosing their behavior and the fitness dropped to values around 10. Afterwards, these unconstrained controllers start developing the wall-following behavior as already discussed.

Further investigations have shown that a controller reacts very sensitively to parameter changes; the controller's parameters exhibit high epistasis, which describes a non-linear interaction of the parameters with respect to the fitness function. This epistasis results from the sensor's non-linearities and the 100 ms time interval between two subsequent sensor readings. Consequently, all parameters have to be adapted simultaneously. The evolution strategy obeys this requirement by using a mutation probability $p_m = 1.0$, i.e., the evolution strategy applies mutation to all parameters at the same time. It is suspected that the observable epistasis is the main reason for the inefficiency of GA-based approaches [4, 5, 10]. That research reports that the GA needs about 50 to 100 generations with a population of 80 individuals. Such an optimization process takes approximately 66 hours, which is approximately 40 times longer than that of the evolution-strategy approach.

The performance difference observed in these experiments coincides with the research discussed in [12], which investigates the performance of GAs when applied to artificial fitness functions. The main results presented in [12] indicate that the independence of the parameters to be optimized is an essential prerequisite for GAs and that the GA's performance significantly degrades under epistasis. Both experiments, the optimization of Braitenberg controllers and the optimization of artificial fitness functions under a coordinate rotation, suggest that evolution strategies might be better suited for continuous parameter optimization which incorporate epistatic interaction between the parameters, whereas GAs seem to perform with higher efficiency when applied to combinatorical problems, such as the the traveling salesman problem or the evolution of neural network topologies. In the future, more research is to be done that provides more insight in this phenomenon. When applied to continuous parameter optimization, the dynamic self-adaptation of the step size is a further advantage of evolution strategies. This dynamic adaptation supports the designer, because the designer is not forced to find a good value for that important parameter. In other words, such an adaptation mechanism allows the designer to concentrate more on the engineering aspect, rather than tackling parameter settings that belong to the optimization procedure itself.

Even though this paper points to other GA-based research, the main focus of this paper is the application of evolution strategies to the evolution and optimization of Braitenberg vehicles. A comparison across remote research would be very problematic, since not all parameters of such a real-world application can be replicated. In additional series of control experiments, a GA was used instead of an evolution strategy. The control experiments yield roughly the same performance as reported in [4, 5, 10]. The GA needs approximately ten times more fitness evaluations than the (3,6)-ES. The GA suffers from the high epistasis and the need of a rather large population of about 80 individuals. In each generation, the GA generates ten times more offspring than the evolution strategy. Thus, in this application, the evolution strategy converges in a time period in which the GA performs only five to ten generations.

7 CONCLUSIONS

This paper has discussed the practical application of the evolution strategy to the evolution and optimization of Braitenberg vehicles. Braitenberg vehicles are autonomous agents with a simple control architecture, which is typically implemented as a neural network. Autonomous agents are very important tools in New Artificial Intelligence, since they study intelligence as the result of a system environment interaction, rather than understanding intelligence on a computational level.

In the practical experiments, constrained as well as unconstrained Braitenberg controllers have been investigated. These two controllers typically develop different survival strategies, which have also been discussed. A comparison with other research that apply genetic algorithms to very similar tasks shows that the evolution-strategy-based approach is much faster than the GA-based approach. It is suspected that the high epistasis between the controller's parameters drastically slows down the GA-based approach. The evolution strategy speeds up the development of Braitenberg controllers by more than one order of magnitude, which is very important, since experimentation is to be done with real system. The evolution strategy converges in one and a half hours compared to 66 hours required for the GA-based approach.

The experiments also indicate that the evolution process can benefit from an exploitation of physical matters, such as the physical symmetry of the robots body.

Further research will be devoted to more complex control architectures for object avoidance, navigation, and manipulation as well as other neural controllers, which are designated to enhance the robots capabilities/competences

Acknowledgements

This work was supported in part by a Human Capital and Mobility fellowship of the European Union, grant number ERBCHBICT941266. Thanks to Rolf Pfeifer and Peter Eggenberger for helpful discussion.

References

[1] Bäck, T. and Schwefel, H.-P. (1993) An Overview of Evolutionary Algorithms for Parameter Optimization, *Evolutionary Computation* **1**, No. 1, pp. 1-23.

[2] Braitenberg, V. (1984) *VEHICLES, Experiments in synthetic psychology* (MIT-Press, Cambridge, Massachusetts).

[3] Brooks, R. (1986) A robust layered control system for a mobile robot, *IEEE Journal of Robotics and Automation*, **2**, pp. 14-23.

[4] Floreano, D. and Mondada, F. (1994) 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.W. Wilson (eds.), (MIT Press-Bradford Books, Cambridge, Massachusetts) pp. 421-430.

[5] Floreano, D. and Mondada, F. (1996) Evolution of Homing Navigation in a Real Mobile Robot, to appear in: *IEEE Transactions on Systems, Man and Cybernetics*.

[6] Fogel, D.B. (1995) *Evolutionary Computation: Toward a New Philosophy of Machine Learning Intelligence* (IEEE Press, NJ).

[7] Goldberg, D.E. (1989) *Genetic Algorithms in Search, Optimization and Machine Learning* (Addison-Wesley Publishing Company).

[8] *Khepera Users Manual*, Laboratoire de microinformatique, Swiss Federal Institute of Technology (EPFL), 1015 Lausanne, Switzerland.

[9] Maes, P. (ed.) (1991) Designing Autonomous Agents: Theory and Practice from Biology to Engineering and Back (The MIT Press, Cambridge, Massachusetts).

[10] Nolfi, S. and Parisi, D. (1995) Learning to Adapt to Changing Environments in Evolving Neural Networks, Technical Report 95-15, Institute of Psychology, National Research Council, Rome, Italy, <www http://kant.irmkant.rm.cnr.it/public.html>.

[11] Rechenberg, I. (1973) *Evolutionsstrategie* (Frommann-Holzboog, Stuttgart).

[12] Salomon, R. (1996) Reevaluating Genetic Algorithm Performance under Coordinate Rotation of Benchmark Functions; A survey of some theoretical and practical aspects of genetic algorithms. *BioSystems* **39**, No. 3, Elsevier Science, pp. 263-278.

[13] Salomon, R. (1996) Neural Networks in the Context of Autonomous Agents: Important Concepts Revisited, *Proceedings of the Artificial Neural Networks in Engineering Conference (ANNIE'96)*, C.H. Dagli, M. Akay, C.L.P. Chen, B.R. Fernández, and J. Ghosh (Eds.), (ASME Press, NY, NY) pp. 109-116.

[14] Schwefel, H.P. (1995) *Evolution and Optimum Seeking* (John Wiley and Sons, Inc, New York, Chicester, Brisbane, Toronto, Singapore).

[15] Simon, H.A. (1976) *The sciences of the artificial* (2nd edition) (The MIT Press, Cambridge, Massachusetts).

[16] Steels, L. (ed.) (1995) The Biology and Technology of Intelligent Autonomous Agents. Special issue of *Robotics and Autonomous Systems* **15** (Elsevier).