In P. Maes, M. Mataric, J.-A. Meyer, J. Pollack, and S.W. Wilson, (Eds.), *From Animals to Animats 4: Proceedings of the Fourth International Conference on Simulation of Adaptive Behavior*, pp. 411-420, Cambridge, MA: The MIT Press/Bradford Books.

Increasing Adaptivity through 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

The automated generation of controllers for real-world autonomous agents by means of evolutionary methods has recently attracted a lot of attention. Most of the pertinent research has employed genetic algorithms or variations thereof. We have applied a different evolutionary method to the generation of a control architecture for Braitenberg vehicles, namely "evolutionary strategies". The application of the ES accelerates the development of such controllers by more than one order of magnitude (a few hours compared to more than two days). This result is very important, since the development process is to be done in real systems. In addition to the dramatic speedup, there is an important theoretical reason for preferring evolutionary strategy over genetic algorithms, namely epistatic interaction.

1 Introduction

Autonomous agents are self-sufficient, embodied systems (robots) [5] that do some useful work. 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; typically, the agent gets some reward after doing useful work. 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 manipulates its environment by actively 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. For an overview of special issues involved in autonomous agent design see, for example, [16, 25].

The control structures of autonomous agents can be developed either by the designer or by applying evolutionary algorithms. Many research projects [6, 8, 9, 14, 18] have used genetic algorithms (GAs) to evolve neural control structures for autonomous agents. As pointed out in Harvey et al. [14], the automated evolution is more appropriate, since it allows the development of complex control structures that exhibit many interactions between sub-systems. In this paper, we too advocate the use of evolutionary algorithms for adaptive systems.

Even though GAs have been successfully applied to rather small tasks, an increasing amount of research gives strong evidence that GAs are very time consuming if the parameters exhibit epistatis. Epistatis describes the interaction of parameters with respect to the fitness of an individual. Results of other research [22, 23] strongly indicate that the independence of parameters is an essential prerequisite of the GA's high global convergence performance; the presence of epistatis drastically slows down convergence. The problem in the context of autonomous agents is that the parameters of control systems for real-world applications are not independent. Consequently, even the evolution of rather simple controllers, such as Braitenberg vehicles, is very time consuming. To relieve this problem, it is suggested [14] to divide the evolution of more complex systems into different stages.

Until now, most research projects have focused on GAs, which are only one option of evolutionary algorithms. Surprisingly little attention has been devoted to the evolution strategy (ES) as defined by Rechenberg [20] and Schwefel [24]. The ES is especially designed for applications that involve real-valued parameters. To this end, Section 2 gives a brief comparison of these two algorithms with respect to autonomous agents. One result of this comparison is that the performance of the ES does not degrade in the presence of epistatis; the ES behaves invariant with respect to epistatis.

The theoretical analysis of Section 2 is supported by a case study on the evolution of Braitenberg vehicles. To this end, Section 3 describes the setup of the case study. The case study consists of the *Khepera*^{TM} robot, which is an example of a small robot that is widely used for research on autonomous agents. Subsection 3.1 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

control system uses the sensor readings and its internal state to determine the agent's next action. Subsection 3.2 describes a simple neural control architecture called a Braitenberg vehicle [4], and Section 3 describes the arena, in which the robots have to operate.

Section 4 demonstrates the benefits of the ES when applied to the development of adaptive control architectures. That section reports some experiments obtained by applying a (3,6)-ES to the evolution of Braitenberg vehicles. A (3,6)-ES evolves reasonable Braitenberg controllers within 30 generations, which takes approximately one and a half hours. Other research [8, 9, 18] uses GAs for almost the same task. However, the GA-based approach requires approximately two and a half days. Thus, the ES speeds up the developmental process by more than one order of magnitude. This speed up is of great practical relevance, since the experiments are to be done in real systems that dynamically interact with the real world. Also, such a time reduction allows for the application to more complex control structures, which are currently intracktable.

One important feature of GAs is their ability to use genomes with variable length. Section 5 briefly shows how this important feature can be incorporated in the ES. Finally, Section 6 concludes with a summary of the work presented in this paper.

2 Evolutionary Algorithms

Evolutionary algorithms can be seen as a framework that includes genetic algorithms, evolutionary strategies, evolutionary programming, and genetic programming. All these algorithms are heuristic population-based search procedures based on natural selection and population genetics. Typically, such population-based search procedures generate offspring in each generation. Then a fitness value (defined by a fitness function) is assigned to each offspring. Depending on the fitness, each population member survives with a certain probability.

All evolutionary algorithms implement different strategies, and therefore, each algorithm is best suited for different application domains. To this end, the following subsections give a brief description of GAs and ES. A good comparison of evolutionary algorithms can be found in [2]. Section 2.3 compares both schemes in more detail with respect to autonomous agents, adaptivity, and performance issues under epistatis. Furthermore, both algorithms are well grounded by an extensive theory, which can be found in, for example, [3, 12, 17, 24].

2.1 Genetic Algorithm

The GA technique is based on Holland [13] and a good introduction to GAs can be found in [12]. There does not exist *one* GA, but rather a variety of variants, each covering different applications and aspects. According to [24], the canonical GA can be described as follows:

- Step 0: Initialization of the population's individuals and evaluation of the individuals' fitness
- Step 1: Selection of the parents according to a preselected selection scheme (e.g., roulette wheel, linear ranking, truncation selection)
- Step 2: Recombination of selected parents by exchanging parts of their genes
- Step 3: Mutation of some genes by a prespecified probability
- Step 4: Go to Step 1

Before using a GA in a particular application domain, the designer is concerned with the coding of the parameters, the implementation of the mutation and recombination/crossover operators, and the choice of the selection scheme. Traditionally, a GA treats every parameter as a bit string. Depending on the required precision, a coding scheme can encode each parameter by a certain number of bits and an attached mapping function. The mapping function maps the bit representation to real-valued parameters. Another coding scheme is to directly use the bit representation used by the programming language. In such a case, no mapping function is needed. Other algorithms like the BGA [17] treat each parameter as a real-valued data type and implement mutation by adding or subtracting small random numbers.

In each generation, a GA typically draws pairs of parents and applies mutation and recombination with probabilities p_m and p_r respectively. After generating a specified number of offspring, the GA evaluates each offspring and selects the best population members as parents for the next generation. Typical selection schemes are roulette wheel selection, linear ranking, or truncation selection. Very often, GAs use an elitist selection scheme, which preserves the best individual in order to maintain gained success. The influence of different selection schemes on the resulting convergence can be found in, for example, [26].

When using GAs, it is very important to find appropriate parameter settings. Normally, the mutation probability p_m is set to small values $p_m \approx 1/n$ [2, 7, 12, 17, 19], where n denotes the number of parameters, e.g., the number of weights in a neural network controller. Generally, GAs prefer rather high recombination rates [2, 12, 17]. If using one-point or two-point crossover, the recombination probability p_r is set to values between 0.5 and 0.9, and if using uniform recombination, the probability is set to $p_r = 0.5$.

This GA framework has been successfully applied in various domains, such as function optimization, VLSI design, neural network learning, and autonomous agent development. Applications of GAs in the field of autonomous agents can be found, for example, in [6, 8, 9,

14, 18].

2.2 Evolution Strategy

The ES has been introduced in [20] (see, also, [24]). The ES is similar to GAs. A $(\mu \dagger \lambda)$-ES maintains a population of μ parents and generates λ offspring in each generation. In a (μ, λ)-ES, the μ parents are selected from best λ offspring, whereas in a $(\mu + \lambda)$-ES, the μ parents are selected from the union of parents and offspring. For further details see, for example, [2]. Currently [2], the (μ, λ)-ES is recommended, especially in noisy environments.

In contrast to GAs, the ES encodes each parameter as floating-point numbers, and it applies mutation to *all* parameters simultaneously, i.e., $p_m = 1$. Mutations are typically implemented by adding $(0, \sigma)$-normal distributed random numbers. The key concept of the ES is that it, in its simplest form, maintains one global step size σ for each individual. This step size is self-adapted by the following mechanism: Each offspring inherits its step size from its parent(s). This step size is modified by log-normal random numbers prior to mutation. By this means, the step size is self-adapted to nearly optimal values, since, in a statistical sense, those offspring survive that have the best adapted step size. For further details of different step size schemes see [2, 24]. In addition, the ES can use the same recombination schemes as GAs [2].

The self-adaptation mechanism of the step size σ has a great advantage. It allows the ES to self-adapt to different fitness landscapes. Therefore, besides the population size, the ES does not have any parameters that have to be tuned by the designer. Since the ES directly encodes each parameters as floating-point numbers, the ES is better suited for problems, such as neural networks, that are specified by a set of parameters.

Evolutionary programming (EP) has been introduced by Fogel [10]. EP is another evolutionary algorithm and very similar to ES. Recent applications to general function optimization can be found in [11]. Since both schemes are very similar, it can be expected that EP yields very similar results when applied to the evolution of Braitenberg vehicles.

2.3 Genetic Algorithm vs. Evolution Strategy

Even though the differences between GAs and ES seem rather small, they significantly influence the performance of both algorithms and, consequently, they aim at different problem domains. The remainder of this subsection discusses some important aspects.

The main difference between both algorithms is that GAs apply mutations to only a few parameters per offspring, whereas the ES applies mutation to *all* parameters, i.e., $p_m = 1$. Furthermore, the ES encodes each parameter as floating-point number, whereas GAs have, in the general case, to worry about the coding scheme. Traditionally, GAs encode each parameters as bitstrings. However many applications that involve real-valued parameters [17, 18] directly use floating-point numbers. In addition, the ES features a self-adaptation mechanism, which self-adapts the step size by itself so that no parameter settings have to be done by the user.

From the coding mechanisms it should become clear that ES is rather suited for real-valued parameters, whereas GAs with traditional bit coding schemes are preferred for combinatorical tasks like the traveling-salesman problem. Furthermore, the ES increases adaptation, since the *algorithm is inherently adaptive*.

Many applications [1, 2, 7, 17] report high performance when applied to various optimization task, especially the optimization of multimodal function that contain millions of local optima but only one global optimum. The high performance that was achieved in these applications suggest that GAs easily escape from local optima and that they have very good global convergence. However, recent results [22] show that the performance of GAs drastically degrades if a rotation is applied to the coordinate system. Furthermore, theoretical analysis [23] shows that the computational complexity of GAs can increas up to $O(n^n)$, when applied to multimodal functions with n parameters that depend on each other. Even when applied to simple unimodal functions, epistatis drastically slows down the convergence of GAs [22]. Moreover, that analysis suggests that the independence of the parameters is an essential prerequisite for high performance of GAs.

A rotation of the coordinate system does not change the fitness function (fitness landscape), but it induces *epistatis*. Epistatis describes the interaction of different parameters with respect to the fitness function. In other words, if epistatis between parameters is present, all parameters involved have to be adapted simultaneously in order to achieve any improvement of the fitness function; adapting only one parameter leads to a worse fitness. Thus, applying mutation to only one parameter is not sufficient in such situations.

In a particular application, one important question is, whether the parameters are independent or if they depend on each other, i.e., whether epistatis is present. When looking at the control structures of autonomous agents, the controller's parameters are not independent in the general case. To this end, Section 4 investigates the evolution of Braitenberg controllers. It turns out that the weights of the Braitenberg network are not independent. The results show that even a simple (3,6)-ES is more that one order of magnitude faster than GA-based approaches.

It could be argued that an acceleration of the development of controllers by one order of magnitude is not that important. However, such an improvement is of

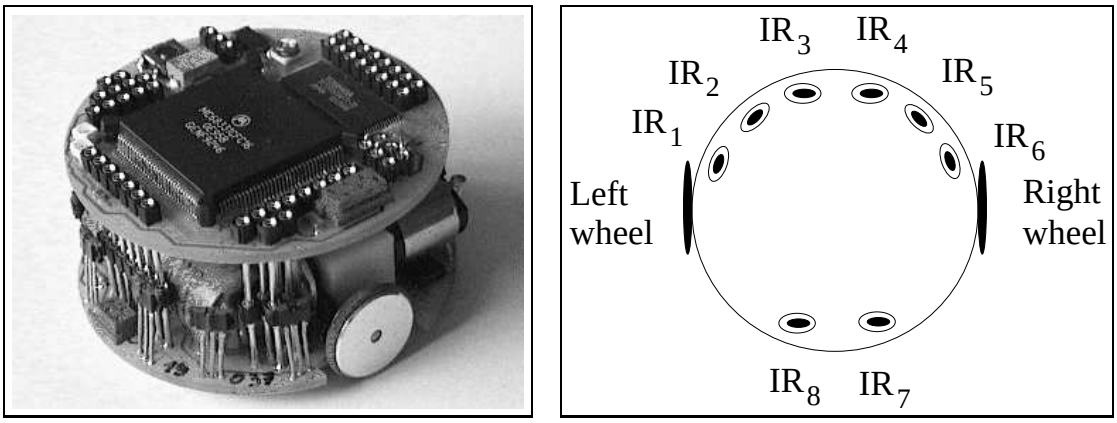

Figure 1: The Khepera robot. The right part shows the approximate location of the infrared sensors $IR_1 \ldots IR_8$.

great practical interest. First of all, control architectures that allow the agent to adapt to the environment have to be developed in real hardware. Thus, it is important whether the developmental process consumes only a few hours or if it take several days. Second, a moderate convergence speed becomes more important as the complexity of the control architectures increases. From a theoretical point of view, any randomized algorithm eventually finds a solution with a probability strictly greater than zero. However, from a practical point of view, time matters. Thus, good convergence speed allows for the development of more complex control architectures, which, in turn, allows the agent to better adapt to its environment and the tasks, which are to be done by the agent. In summary, the results presented in Section 4 strongly suggest that the research community should devote more attention to the evolution strategy or evolutionary programming.

3 The Experimental Setup

This section describes the experimental setup for the evolution and optimization of Braitenberg vehicles. First, Subsection 3.1 describes the Khepera robot, which is widely-used in research on autonomous agents. Then, a Braitenberg architecture is described in Subsection 3.2, and, finally, Subsection 3.3 describes the arena, in which the robot has to operate.

3.1 The Khepera Robot

The Khepera robot (cf. Fig. 1) is 55 mm in diameter and 32 mm high. The robot is equipped with eight infrared and eight ambient light sensors 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 onboard 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 [15].

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 in situations, where 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 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 Fig. 2. Fig-

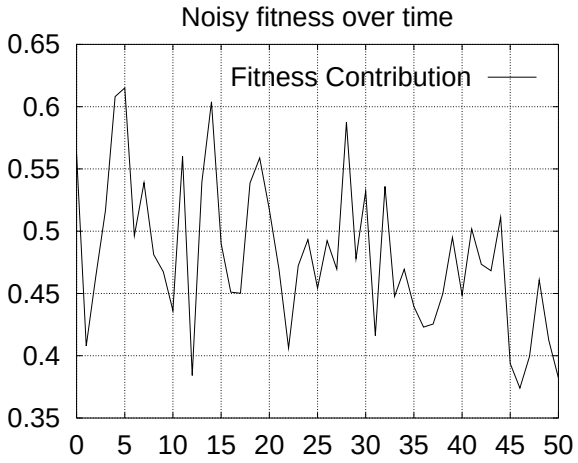

Figure 2: The noisy sensor readings cause a dynamical change of the individual fitness contributions f_t even under constant environmental conditions.

ure 2 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, and the ambient light. During a long series of experiments, the environment cannot be kept constant. In summary, the fitness evaluation is extremely noisy.

3.2 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 [4] has proposed a simple architectures for such tasks. Figure 3 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 to 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. The control architecture is typically implemented as a neural network. The activation of the left motor M^l is calculated by the following formula

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

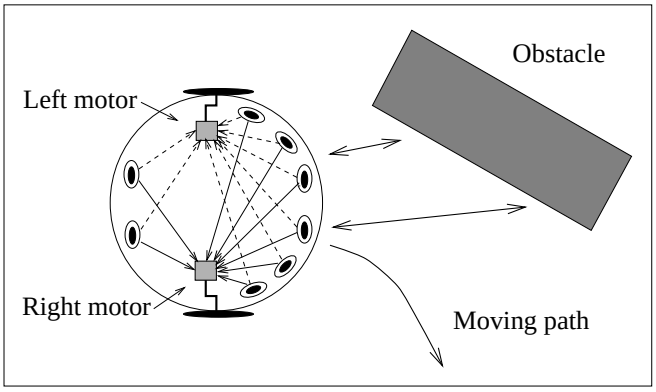

Figure 3: A control architecture inspired by a Braitenberg type-3C vehicle. The sensory information controls the motors via inhibitory and excitatory connections.

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 motion in the absence of any obstacle. The calculation of the right motor's activation is similarly given by

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

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. Thus, the evolution strategy is an ideal candidate for this optimization problem.

3.3 The Experimental Setup

The experimental setup has been chosen to be as close as possible to setups proposed in other research [8, 9, 18]. Figure 4 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.

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. Therefore, the fitness function

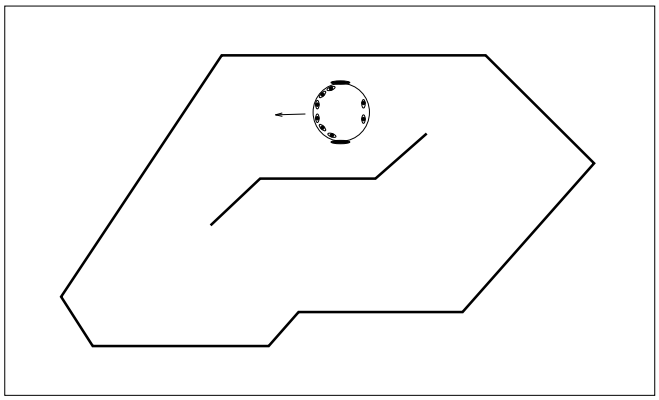

Figure 4: The arena of approximate size 60x45 cm. The indicated position (facing left) is the starting point for fitness evaluation.

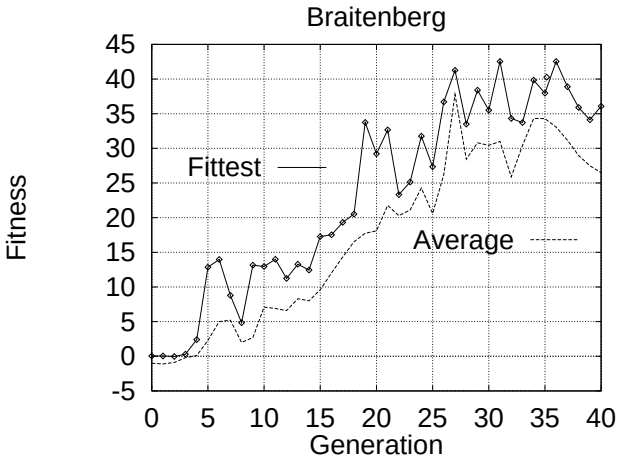

Figure 5: A typical run of the evolution of a *constrained* Braitenberg vehicle. The upper graph represents the best individual whereas the lower graph prepresents the average of the whole population.

is to be enhanced by a third term that favors straight movements by penalizing turns (see also [8, 9, 18]).

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 $\hat{IP} = \max_i IP_i$ are 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

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

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.

4 Experiments

This section reports some typical results when evolving Braitenberg vehicles by means of a $(3, 6)$-ES with self-adaptation of the step size [24]. Some of the results are discussed in Subsection 4.3. 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 noisy (cf. Fig. 2). In the experiments reported in this paper, the ES 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 held constant; this phenomenon is intrinsic to real-world applications.

For each experiment, at least four runs have been performed. Figures 5 and 6 present typical runs that resemble average performance. Each figure shows the fitness of the best individual as well as the average fitness of the entire population. No average of several runs is shown, since averaging eliminates interesting details. The other runs are within a 20 percent interval of the presented 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 $w_i^l = w^r_{(14-i) \bmod 8+1}$. 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.

4.1 Constrained Controllers

A typical run of the evolution of a *constrained* Braitenberg controller can be seen in Fig. 5. Figure 5 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 crashing backwards into the wall, resulting in a negative fitness. After a few more generations, some agents are moving forward, but after some steps, got stuck at the wall; 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. But sometimes, they are hitting the walls again. After hitting a wall, good controllers set the motor speeds to negative values, which causes 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 asymmetric 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. But a Braitenberg vehicle adapts its behavior to both the environment and the unspecified or even changing characteristics of its electrical components.

4.2 Unconstrained Controllers

Figure 6 shows a typical run of the evolution of an *unconstrained* Braitenberg controller. Such a controller has 18 parameters and the ES 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. 5 and 6, the evolution of an unconstrained Braitenberg controller requires more time, and also, the final performance is lower than the 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 cause 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.

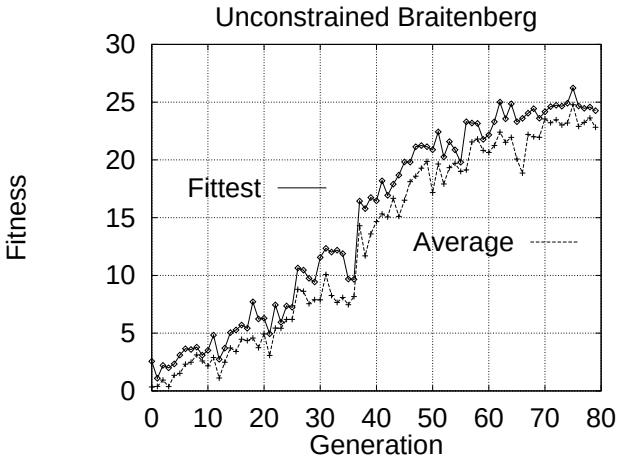

Figure 6: A typical run of the evolution of a *unconstrained* Braitenberg vehicle. The upper graph represents the best individual whereas the lower graph prepresents the average of the whole population.

4.3 Discussion

The previous two subsections have investigated the development of two different controller types, 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 relationships between both controller types were further investigated in a series of control 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 losing their behavior and the fitness dropped to values around 10. Afterwards, these unconstrained controllers started to develop the wall-following behavior as already discussed. Thus, the developed controller is adapted not only to the environment but also to the control architecture itself.

Further investigations have shown that a controller reacts very sensitively to parameter changes; the controller's parameters exhibit high epistatis. This epistatis results from the sensor's non-linearities and the 100 ms time interval between two subsequent sensor readings. It is not reasonable to change, for example, only the bias weight w_0^l. This would result in a too high speed and

the robot would crash into the next wall. Conversely, changing only one "avoidance" weight w_i^l would degrade the fitness, since the robot would tend to wriggle. Consequently, all parameters have to be adapted simultaneously. The ES obeys this requirement by using a mutation probability $p_m = 1.0$, i.e., the ES applies mutation to all parameters at the same time. It is suspected that the observable epistatis is the main reason for the inefficiency of GA-based approaches [8, 9, 18]. 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 ES approach. This coincides with the research discussed in [22], which investigates the performance of GAs when applied to artificial fitness functions. The main results presented in [22] 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 epistatis.

Even though this paper points to other GA-based research, the main focus of this paper is the application of the ES 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 addition, we did a series of control experiments, in which we used a GA instead of the ES. The control experiments yield roughly the same performance as reported in [8, 9, 18]. The GA needs approximately ten times more fitness evaluations than the ES. The GA suffers from the high epistatis and the need of a rather large population of about 80 individuals. In each generation, the GA generates ten times more offspring than the ES. Thus, in this application, the ES converges in a time period, in which the GA performs only five to ten generations.

5 Enhancements

For the research on GAs, it is very important to consider genomes with varying length. Several applications, e.g., [14], explicitly use this feature for the development of more complex control structures. The main underlying idea is that first, the GA develops a small solution with the most important properties. In the ongoing evolution process, the genome is allowed to grow in size, which enables the system to add more beneficial features.

It is often argued that this dynamical growth of the genome is proprietary for GAs and not reasonable for the ES. However, as outlined above, both types of algorithms are very similar. Both apply mutation as well as recombination. For a mutation operator, the length of the genome does not matter. If applying recombination/crossover, these operators have to ensure the correctness of a new genome anyway.

In [21] a hybrid method has been proposed that allows for the development of neural networks with minimal topology. Essentially, this hybrid method works as follows: Each offspring randomly adds and removes neurons as well as connections from the network that is has inherited from its parents. Similar to the self-adaptation of the step size, this hybrid method self-adapts the probabilities for adding and removing of connections and units. By these means, the network can dynamically grow and shrink depending on the actual environment.

6 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 ES-based approach is much faster than the GA-based approach. It is suspected that the high epistatis between the controller's parameters drastically slows down the GA-based approach. The ES 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 systems. The ES converges in a few hours compared to 66 hours required for the GA-based approach.

This paper has also argued that the evolution strategy is more adaptive, since it self-adapts parameters, such as the step size or mutation probability. This allows to better embed the whole approach in more complex tasks. It could be argued that the small mutation rate as is usually used in GAs is biologically more plausible than a mutation rate $p_m = 1$ as is used in the ES. However, we argue that it is the other way around. If one looks at living things, all individuals differ in almost all attributes. For example, if comparing two arbitrary selected humans, these two humans will differ in all perspectives. It seems that nature applies mutation to only a few genes. However, most genomes do not encode all parameters of the resulting individual. Rather, the genome encodes developmental processes. Thus, modifying one gene results in different developmental programs, and consequently, the resulting individual differs in (almost) all perspectives. Since most evolutionary approaches do not involve real developmental processes, such as growing, we argue that the ES better reflect nature's principles.

Furthermore, the experiments also indicate that the evolution process can highly benefit from an exploitation of physical matters.

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. Optimal Mutation Rates in Genetic Search. In S. Forrest (ed.) *Proceedings of the Fifth International Conference on Genetic Algorithms*, pp. 2-8. Morgan Kaufmann, San Mateo, CA, 1993.

[2] Bäck, T. and Schwefel, H.-P. An Overview of Evolutionary Algorithms for Parameter Optimization, *Evolutionary Computation* 1(1), pp. 1-23. MIT Press, Cambridge, MA, 1993.

[3] Beyer, H.-G. Toward a Theory of Evolution Strategies: On the Benefits of Sex – the $(\mu/\mu, \lambda)$ Theory. *Evolutionary Computation* 3(1), pp. 81-111. The MIT Press, Cambridge, MA, 1995.

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

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

[6] Cliff, D., Husbands, P., Harvey, I. Evolving Visually Guided Robots. In J.-A. Meyer, H.L. Roitblat, and S.W. Wilson (eds.), *From animals to animats 2*, pp. 374-383. MIT Press, Bradford Books, Cambridge, MA, 1992.

[7] De Jong, K.A. *An Analysis of the Behavior of a Class of Genetic Adaptive Systems*. Ph.D. Thesis, University of Michigan, 1975.

[8] Floreano, D. and Mondada, F. Automatic Creation of an Autonomous Agent: Genetic Evolution of a Neural-Network Driven Robot. In D. Cliff, P. Husbands, J. Meyer, and S.W. Wilson (eds.), *From Animals to Animats III: Proceedings of the Third International Conference on Simulation of Adaptive Behavior*, pp. 421–430. MIT Press, Bradford Books, Cambridge, MA, 1994.

[9] Floreano, D. and Mondada, F. Evolution of Homing Navigation in a Real Mobile Robot, to appear in: *IEEE Transactions on Systems, Man, and Cybernetics-Part B*, Vol.26, No.3, June 1996.

[10] Fogel, L.J. "Autonomous Automata", *Industrial Research*, vol. 4, pp. 14-19, 1962.

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

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

[13] Holland, J.H. *Adaptation in Natural and Artificial Systems* Ann Arbor, Michigan, University of Michigan Press, 1975.

[14] Harvey, I., Husbands, P., Cliff, D., Thompson, A., and Jakobi, N. Evolutionary Robotics: the Sussex Approach. In R. Pfeifer and R. Brooks (eds.), *Robotics and Autonomous Systems, Special Issues on "Practice and Future of Autonomous Agents"*, 1996.

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

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

[17] Mühlenbein, H. and Schlierkamp-Voosen, D. The Science of Breeding and its Application to the Breeder Genetic Algorithm. *Evolutionary Computation* 1(4), pp. 335-360. The MIT Press, Cambridge, MA, 1993.

[18] Nolfi, S. and Parisi, D. 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. 1995.

[19] Potter, M.A. and De Jong, K.A. A Cooperative Coevolutionary Approach to Function Optimization. In: Y. Davidor, H.P. Schwefel, and R. Männer (eds.), *Proceedings of Parallel Problem Solving from Nature 3*, pp. 249-257. Springer-Verlag, 1994.

[20] Rechenberg, I. *Evolutionsstrategie*. Frommann-Holzboog, Stuttgart, 1973.

[21] Salomon, R. A Hybrid Method for Evolving Neural Network Topologies. In C.H. Dagli, B.R. Fernández, J. Ghosh, and R.T.S. Kumara (eds.), *Proceedings of*

the Artificial Neural Networks in Engineering (ANNIE'94), pp. 147-152. New York: ASME Press, 1994.

[22] Salomon, R. Performance Degradation of Genetic Algorithms under Coordinate Rotation. To appear in *Fifth Annual Conference on Evolutionary Programming EP'96*, to be held at San Diego, USA, February 29 - March 3, 1996.

[23] Salomon, R. Implicit Independence Assumptions; a Notorious Problem for Genetic Algorithms. To appear in *Proceedings of the International Symposia on Soft Computing and Intelligent Industrial Automation SOCO'96*, to be held in Reading, UK, March 26 - March 28, 1996.

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

[25] L. Steels (ed.), The Biology and Technology of Intelligent Autonomous Agents. Special issue of *Robotics and Autonomous Systems* **15** Elsevier, Amsterdam, Lausanne, New York, Oxford, 1995.

[26] Thierens, D. and Goldberg, D. Convergence Models of Genetic Algorithm Selection Schemes. In Y. Davidor, H.P. Schwefel, and R. Männer (eds.), *Proceedings of Parallel Problem Solving from Nature 3*, pp. 119-129. Springer-Verlag, 1994.