International Journal of Intelligent Computing and Cybernetics

DECENTRALIZED EVOLUTION OF ROBOTIC BEHAVIOR USING FINITE STATE MACHINES

LUKAS KÖNIG

Institute AIFB, Karlsruhe Institute of Technology
76128 Karlsruhe, Germany
lukas.koenig@kit.edu
phone: +49 (721) 608-3924
fax: +49 (721) 608-6581

(corresponding author)

SANAZ MOSTAGHIM

Institute AIFB, Karlsruhe Institute of Technology
76128 Karlsruhe, Germany
sanaz.mostaghim@kit.edu
phone: +49 (721) 608-6554
fax: +49 (721) 608-6581

HARTMUT SCHMECK

Institute AIFB, Karlsruhe Institute of Technology
76128 Karlsruhe, Germany
hartmut.schmeck@kit.edu
phone: +49 (721) 608-4242
fax: +49 (721) 608-6581

Received 06 February 2009
Revised 16 August 2009
Accepted 16 August 2009

Abstract

Paper type: Research paper

Purpose: In Evolutionary Robotics (ER), robotic control systems are subject to a developmental process inspired by natural evolution. In this article, a control system representation based on Finite State Machines (FSMs) is utilized to build a decentralized online-evolutionary framework for swarms of mobile robots.

Design/methodology/approach: A new recombination operator for multi-parental generation of offspring is presented and a known mutation operator is extended to harden parts of genotypes involved in good behavior, thus narrowing down the dimensions of the search space. A storage called Memory Genome for archiving the best genomes of every robot introduces a decentralized elitist strategy. These operators are studied in a factorial set of experiments by evolving two different benchmark behaviors such as Collision Avoidance and Gate Passing on a simulated swarm of robots. A comparison with a related approach is provided.

Findings: The framework is capable of robustly evolving the benchmark behaviors. The Memory Genome and the number of parents for reproduction highly influence the quality of the results, the recombination operator leads to an improvement in certain parameter combinations only.

Research limitations/implications: Future studies should focus on further improving mutation and recombination. Generality statements should be made by studying more behaviors and there is a need for experimental studies with real robots.

Practical implications: The design of decentralized ER frameworks is improved.

Originality/value: The framework is robust and has the advantage that the resulting controllers are easier to analyze than in approaches based on Artificial Neural Networks. The findings suggest improvements in the general design of decentralized ER frameworks.

Keywords: Evolutionary Robotics; Swarm Robotics; Online; Decentralized; Onboard; Finite State Machine

1. Introduction

Evolutionary Robotics (ER) is a broad field today, composed of techniques for the development of robotic controllers based on Darwinian evolution, i. e., the principle of a survival of the fittest. Evolution is capable of finding control systems which outperform manually designed solutions in terms of effectiveness in solving the task and simplicity of the controller [Walker *et al*. (2003)]. This has been shown in large swarms of robots, exploiting the emergence of collective behavior [Bonabeau *et al*. (1999)], as well as in single robots, by a previously performed evolution in simulation, or in many other scenarios [Nolfi and Floreano (2001); Floreano *et al*. (2008)].

Most of the learning techniques utilized in ER can be divided into *offline* techniques and *online* techniques. In an offline setting a robotic system (i. e., a swarm or a single robot) learns to solve the desired task in an artificial environment (real or simulated) before performing it in a real environment and actually solving the task. In online settings a robotic system has to learn in an environment in which it simultaneously has to solve the task. This means that during a run, currently evolved robotic behavior is evaluated by observing its performance on the task to solve. In fact, different from offline evolution, one cannot wait for an eventually evolved behavior of sufficiently high quality, but has to employ and evaluate intermediate behaviors *online*. The requirement of learning behaviors online is given when robots have to adapt quickly to a new and possibly unknown situation or when they have to learn how to deal with novel objectives (see examples below).

Another discrimination, which is typical to swarm robotics, is made between *centralized* and *decentralized* (onboard) techniques. Centralized means that there is an observing computer outside of the swarm which can provide the individual robots with global information while decentralized means the absence of such an observing computer. In a decentralized system, every robot has to make all decisions based on local observations, only. Popular examples of centralized evolution in multi-agent systems are experiments with self-assembling *s-bots*, e. g., by Groß and Dorigo [2004]; a popular example of decentralized evolution is *DAEDALUS* (*"Distributed Agent Evolution with Dynamic Adaptation to Local Unexpected Scenarios"*) by Hettiarachchi and Spears [2006] and Hettiarachchi [2007].

While offline and centralized techniques are known to converge quickly to good so-

lutions for several tasks, there is a broad range of applications for which they cannot be used. The framework presented in this article is designed to evolve robotic behavior in an *online* and *decentralized* manner, because there are innumerable important applications in swarm robotics for which these properties are essential. Whenever swarms of robots are supposed to solve tasks in new and (partly) unknown areas which are not accessible to humans or other global observers, a decentralized approach is required. And whenever such an area may change while the robots are inside, which implicitly changes the desired target behavior, an online approach is essential.

Applications range from those which are presently accomplishable, like recovering victims in disaster areas by rescue robots or unmanned discovery of the mars surface by explorer robots, to those which are more futuristic, like injecting a swarm of nano-robots in the human body to heal diseases. Of course, the benchmark behaviors studied in this article are much simpler than these advanced behaviors. However, when further improved, and perhaps in combination with other approaches, we are confident that our framework has the potential of being utilized for such applications.

Controller representation. In the last decade, Artificial Neural Networks (ANNs) have been used frequently as a representation of robotic control systems (inside and outside of ER), since they are easy to implement. Learning operations are well-studied and quite intuitive for ANNs. However, in many cases these benefits come at the cost of resulting controllers which have a complicated structure and are hard to interpret. In the case of ER, this is due to the randomized changes to the genotype introduced by evolutionary operators like mutation or recombination which optimize the phenotypical performance without respecting any structural benefits for the genotype. The consequence is that evolved controllers are hard to analyze from a genotypical point of view (while it may seem plausible by observation that they have learned a certain behavior). This comes as an additional difficulty to the problem observed by Valentin Braitenberg [Braitenberg (1984)] that it is hard to predict robotic behavior from knowledge of the controller only. While this may seem acceptable in some areas, there are many robotic applications for which it is crucial to guarantee that tasks are solved every time with the same precision or in the same manner (e. g., in medical applications where errors can cost lives or on space missions where erroneous behavior can be very expensive). Therefore, we use Finite State Machines (FSMs) for the representation of robotic controllers, introducing a model called Moore Automaton for Robot Behavior (MARB) (based on the preliminary work by König *et al.* [2008a]; [2009]; König and Schmeck [2008b]). The MARB model relies on the rather simple theory of FSMs which is much more comprehensive than, e. g., ANNs that are Turing-complete in general. The approach is justified by an extensive evidence for the effectiveness of evolving FSMs [Fogel, L. J. *et al.* (1966); (1995); Fogel, L. J. (1999); Fogel, D. B. (2006); Spears and Gordon (2003)].

In this article, the structural benefits of the MARB model are exploited by a mutation operator which is designed to *harden* parts of the automaton involved in good behavior. Other parts of the automaton stay loosely connected and can get deleted within few mutations. In this way, the complexity of the automaton gets adapted to the complexity of the

task which is being learned (inspired by an approach by Stanley and Miikkulainen [2004]). Experiments show that this property is achieved in the resulting automata, e. g., for Collision Avoidance which is a simple behavior and can basically be described in a reactive way. Here, in most cases, evolution finds solutions where only two or three states are involved depending on how sophisticated the maneuver is. The hardening works independently of the fitness function which can be designed purely with respect to the task to be learned.

However, the price for analyzability and simplicity of FSMs is their low degree of expressiveness. Yet, we argue that most existing controllers learned with ANNs could as well be represented as FSMs. Another disadvantage is that evolutionary operations on FSMs (i. e., mutation and recombination) are less intuitive, therefore, special attention has to be paid to their design and the genotypic representation.

In a related study, Spears and Gordon [2003] evolve FSMs for the solution of resource protection problems by representing a genotype as a matrix assigning to each state/input pair an action and a next state (this is called a Mealy automaton whereas a Moore automaton assigns actions to states). They observe that evolution is capable of finding the appropriate number of states necessary to learn a behavior of a certain complexity. Furthermore, they find that the recombination operator has a great positive influence on evolution. We can approve that evolution can find the appropriate number of states depending on the complexity of the target behavior, however, in our approach a hardening property is added to the mutation operator to aid this process. In the approach presented here, evolution is also capable of finding the appropriate number of transitions. Using a recombination operator similar to that described by Spears and Gordon we find that it has a rather small effect on evolution which is positive in certain parameter combinations only. However, our approach differs from that of Spears and Gordon by using Moore automata instead of Mealy automata, a different automaton representation (a matrix representation is not feasible, since the input set is huge, consisting of all combinations of possible values of the 7 sensors of a robot, i. e., 256^7 input symbols; see Section 2 for details), a multi-parental recombination operator, and by evolving in a different scenario.

The reality gap. A well-known problem in ER is the transfer of simulated results into real-world applications. It has been shown for many pairs of simulation environments and corresponding real robot hardware that controllers which are evolved in simulation have a very different behavior when transferred onto real robots [Nolfi and Floreano (2001)]. This problem, often referred to as the *reality gap*, arises from several real-world factors which are inherently hard to simulate: unknown changes in the environment, unpredictable locomotive and sensory differences between robots, mechanical and software failures, etc. As there are obvious advantages – in terms of computational and cost efficiency – of utilizing simulation when evolving robot controllers, there have been different attempts to avoid the reality gap. A common method is to use a sophisticated simulation engine with physics computation and artificially added noise to create unpredicted situations. To deal with the high computational complexity of these simulations, different levels of detail can be defined [Jakobi (1997); Ampatzis *et al.* (2005); Ampatzis *et al.* (2006)]. Another way is to combine evolution in simulation with evolu-

tion on real robots [Walker *et al*. (2003)]. Following this approach, it is possible to use the measured differences between simulation and reality as a feedback in an "anticipation" mechanism to adjust the simulation engine [Hartland and Bredeche (2006)]. Another idea of combining simulation and reality is to adjust evolutionary parameters and operators in a quite simple simulation and to actually evolve behaviors onboard of real robots. This avoids the reality gap under the reasonable assumption that the mechanisms of evolution work on real robots basically in the same way as in simulation. This article follows the latter approach. All experiments presented here have been performed in simulation.

König *et al*. [2008a] and König and Schmeck [2008b] propose an evolutionary framework which is completely decentralized and, therefore, can be implemented in simulation as well as on real robot platforms. The framework is designed to work onboard of robots (simulated or real), accomplishing an evolutionary algorithm without any central control. In that sense, it is an application of the so-called "embodied evolution" proposed by Watson *et al*. [2002]. Due to the decentralization, the framework scales well to large swarms of robots and can be easily implemented for different simulations and robot platforms. This framework is extended here.

In this article, a new recombination operator *Cross* is presented for the framework which is capable of performing multi-parental offspring generation. Its effects are compared to the use of a trivial reproduction operator which is simply cloning parental genomes, and both are studied in combination with the usage of a newly proposed *Memory Genome*, storing a robot's best controller found so far (a decentralized elitist strategy), and without the Memory Genome. For both scenarios, two objective functions are studied: one for Collision Avoidance and one for Collision Avoidance with an additional task of finding a gate in the middle of the field and passing it as often as possible (this behavior is called *Gate Passing*).

Furthermore, possible implementations of the recombination operator on real robots are discussed, because multi-parental child generation is simple to implement in simulation while in reality, communication and memory constraints have to be considered.

The main contributions of this article to decentralized evolution of robotic behavior are

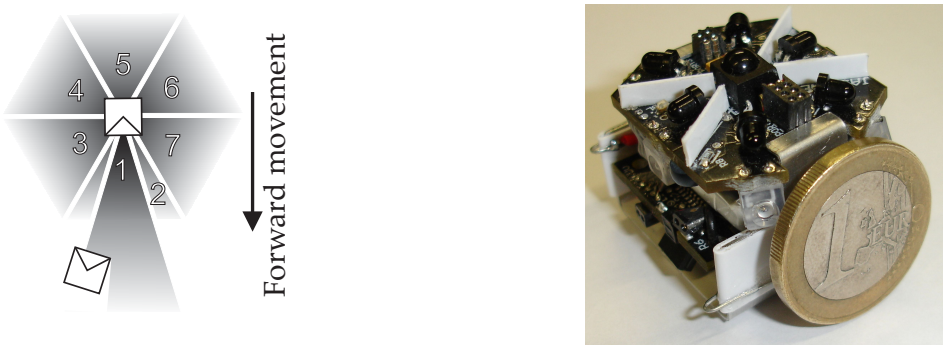
- the property of hardening genotypic structures involved in good behavior through mutation and selection,
- the new recombination operator, and
- a Memory Genome which is stored by every robot to save the best genome found so far, representing a decentralized elitist strategy.

The remainder of this article is structured as follows: In Section 2, the automaton model and the implementation in simulation are described. In Section 3, the evolutionary operators are introduced and differences to earlier implementations are pointed out. Section 4 describes the method of experimentation. Section 5 is dedicated to the results of the evolutionary runs and a discussion of the influence of the evolutionary operators. Section 6 provides a conclusion and an outlook to future work.

2. Automaton Model

This section provides a definition of the evolvable automaton model called *Moore Automaton for Robot Behavior (MARB)*. The states of an automaton define robotic actions to be performed; the next state is chosen by transitions based on conditions over sensors of the robot. Initially, the robot platform is described.

The platform. As the model is defined in a general way, it is applicable to different robotic platforms. It is implemented and tested on the Jasmine IIIp robot which is also simulated. The Jasmine IIIp series is a swarm of micro-robots sized $26 \times 26 \times 26$ mm^3. Every robot can process simple motoric commands like driving forward or backward or turning left or right. Every robot has seven infra-red sensors (as depicted in Figure 1(a)) returning values from 0 to 255 in order to measure distances to obstacles (cf. Figure 1(b) and www.swarmrobot.org). In simulation, the return value of a sensor is calculated by the function $d(x) = \left\lfloor 255 \cdot 51^{(r-(x \cdot a \cdot b))/150} \right\rfloor$, where x is the distance from the middle of the robot to the closest object in the range of the sensor (in millimeters) and r is the distance of the sensor from the center of the robot. The variable a is 1 if the obstacle is a wall, and 2 if it is another robot (simulating different reflecting surfaces); b is 1 for the sensors 2 to 7, and 0.75 for sensor 1; this simulates that one of the sensors facing to the front can sense obstacles at a greater distance by having a more narrow infra-red beam.



(a) (b)

Fig. 1. (a) Placement of infra-red sensors for distance measurement around a (simulated) Jasmine IIIp robot; sensors 2 to 7 are using an infra-red light source with an opening angle of 60 degrees to detect obstacles in every direction of vision. Sensor 1 has an angle of 20 degrees to allow detection of more distant obstacles in the front. (b) Photography of a real Jasmine IIIp robot.

Preliminaries. As mentioned before, the sensors produce byte values to indicate distances. Let $B = \{0, \ldots, 255\}$ and $B_+ = \{1, \ldots, 255\}$ be the sets of all and only the positive byte values, respectively. Let $H = \{h_1, \ldots, h_7\}$ be a set of 7 sensor variables which contain the input from the sensors on the robot, and $v = (v_1, \ldots, v_7) \in V := B^7$ the actual values of the robot's sensors. Here, h_i is associated to the sensor labeled with i in Figure 1(a) and

delivers the value v_i for $1 \leq i \leq 7$ (at a specific time step).

The set of *conditions* C over the sensor variables H is the set defined by:

$$c ::= true \mid false \mid z_1 \triangleleft z_2 \mid (c_1 \circ c_2),$$

where $z_1, z_2 \in B_+ \cup H$,
$\triangleleft \in \{<, >, \leq, \geq, =, \neq, \approx, \not\approx\}$,
$\circ \in \{AND, OR\}$,
c_1, c_2 are conditions (recursively).

The values *true* and $false$ are called atomic constants, $z_1 \triangleleft z_2$ is called an atomic comparison. Therefore, a condition can be an arbitrary combination of atomic comparisons and atomic constants, connected by AND and OR. A function $E\colon C \times V \to \{true, false\}$ evaluates a condition $c \in C$ to *true* or $false$ in the obvious way, depending on the current sensor values $v \in V$. For $a, b \in B$, it is defined that $a \approx b$ if and only if $|a - b| \leq 5$ and $a \not\approx b$ if and only if $\neg a \approx b$.

Example conditions are: *true*; $false$; $h_1 < h_2$; $20 > h_7$; $(h_1 \approx h_2\ OR\ h_2 \not\approx 120)$.

A robot can execute *operations* op which are defined as a pair of a command $cmd \in Cmd$ and a parameter $par \in B_+$:

$$op = (cmd, par) \in Op = Cmd \times B_+,$$

where $Cmd = \{Move, TurnLeft, TurnRight, Stop, Idle\}$. For $X \in B$, the meaning of these operations is:

- $(Move, X)$: drive forward for at most X mm.
- $(TurnLeft, X)$: turn left for at most X degrees.
- $(TurnRight, X)$: turn right for at most X degrees.
- $(Stop, X)$: stop the current action and do nothing (X is ignored).
- $(Idle, X)$: keep performing the current action (X is ignored).

The motion commands $(Move, X)$, $(TurnLeft, X)$ and $(TurnRight, X)$ start an action that is applied repeatedly as long as the specified distance or turning limit X is reached (i. e., multiple simulation cycles can be involved). The robot then moves forward for 4 mm or turns around for 10 degrees each cycle until the maximum of X mm or X degrees is reached. However, each operation except $Idle$ stops the old operation that has been started in earlier cycles.

We assume a function $rand(S)$, which returns a random element out of an arbitrary finite set S, based on a uniform distribution.

Automaton definition. The behavioral model is based on Moore automata (or Moore machines) as defined by Hopcroft and Ullman [1979]. The output $op \in Op$ at a state q of an automaton is interpreted as one of the instructions to be executed by the robot (Op is therefore the output alphabet of the automaton).

The transition function defines transitions between states depending on the evaluation of *conditions*, i. e., every transition, has an associated condition. A transition is performed

if the evaluation $E(c, v)$ of the associated condition c, based on the current sensor values v, results in *true*.

This is somewhat different from the regular definition of Moore automata where a transition is associated to a symbol a from the input alphabet and is performed if the current input symbol is equal to a. From that point of view, the input alphabet can be considered as B^7, i. e., any combination of possible sensor values of the robot is a symbol of the input alphabet. On the other hand, a transition between two states of a *MARB* can be seen as a placeholder for a collection of classical transitions between the two states. Each combination of sensor values which lets a condition evaluate to *true* is covered by this condition. A *MARB* transition, therefore, can represent a fusion of a large set of classical transitions, greatly improving the compactness of the model. For example, two transitions associated to $h_1 < h_2$ and $h_1 \geq h_2$ represent together the set of all 256^7 possible outgoing transitions of a state.

Here, two special cases have to be considered to make the model complete and deterministic. (1) If for a state q none of the outgoing transitions has a condition that evaluates to *true*, the model defines an implicit transition to the initial state; this approach is preferred to the naive idea of simply defining q itself as the next state in such cases, since that could lead to deadlocks. (2) If, on the other hand, more than one condition evaluates to *true*, one of the corresponding transitions has to be chosen. In that case the first transition (in order of insertion during creation of the automaton) which has a satisfied condition is chosen. Figure 2 shows an example MARB with two states and an incomplete definition of transitions. From both states, an implicit transition is inserted by the model for the case that the other transition evaluates to $false$ (dotted transitions).

Therefore, a Moore Automaton for Robot Behavior is defined as follows (note that there are no final states as a robot should always be capable of reacting in the environment):

Definition 2.1 (Finite Moore Automaton for robot behavior (MARB))
A Finite Moore Automaton for Robot Behavior A is defined as

$$A = (Q, \Sigma, \Omega, \delta, \lambda, q_0).$$

For the set of conditions C, and the set of operations Op, the elements are defined as follows:

- $Q = B_+ \times Op \times (C \times B_+)^*$ *corresponds to the set of states.*

  $$For\ q \in Q\colon\ q = \left(id, op, (c_1, id_1), (c_2, id_2), \ldots, \left(c_{|q|}, id_{|q|}\right)\right),$$

  where
  - *id is the state's unique identifier,*
  - $op = (cmd, par) \in Op$ *is the operation performed (e. g.,* $(MOVE, 3)$*),*
  - *the* (c_i, id_i) *encode the outgoing transitions,* c_i *being the condition,* id_i *the identifier of the following state,*
  - $|q|$ *denotes the number of conditions of state q.*

For any element x of A, the notation x^q denotes that x belongs to state q.

- *The input alphabet* $\Sigma = V = (B_+)^{|H|}$.
- *The output alphabet* $\Omega = Op$.
- *The transition function* $\delta\colon Q \times V \to Q$:

$$\delta(q, v) = \begin{cases} q', & \text{if } \exists\,(c_k, id_k) \text{ in } q\colon id_k = id^{q'} \text{ and } (E[\![c_k, v]\!] = true \text{ and} \\ & \forall\ j \in \{1, \ldots, |q|\} \text{ with } E[\![c_j, v]\!] = true\colon\ k \leq j) \\ q_0 & \text{otherwise} \end{cases}$$

  for $q \in Q$, $v \in V$.
- *The output function* $\lambda\colon Q \to Op\colon \lambda(q) = op^q = (cmd^q, par^q)$, *for $q \in Q$.*
- *The initial state* $q_0 \in Q$.

Transitions of a MARB A are also identified by the set

$$T(A) \in (B_+ \times B_+) \times C$$

where

$((id_1, id_2), c) \in T(A) \Leftrightarrow$ a transition (c, id_2) exists in the state which has the identifier id_1.

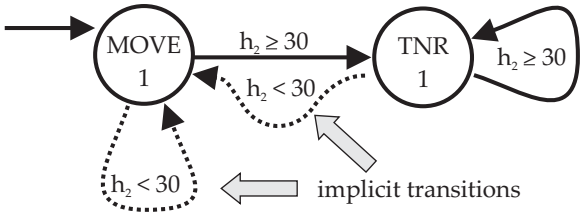


Fig. 2. An example MARB with two states. The dotted transitions are inserted implicitly by the model. The states perform the operations $(Move, 1)$ and $(TurnRight, 1) \in Op$, respectively. The automaton represents a simple collision avoiding behavior moving forward as long as no obstacle is ahead ($h_2 < 30$) and turning right if an obstacle is ahead ($h_2 \geq 30$).

Search spaces. The space of all MARBs is called the *genotypic search space* Γ. The space of all robot behaviors is called the *phenotypic search space* Π. Mutation and recombination are computed on Γ, while fitness evaluation is performed on Π.

3. Evolutionary Operators, Parameters

In this section, the proposed evolutionary operators, namely the mutation and recombination operators, fitness functions and the selection operator, are described. The mutation operator is based on earlier work [König and Schmeck (2008b)], but has a newly presented

quality of hardening parts of the automata which participate in good behavior during the evolutionary process. The recombination operator $Cross$ is presented here for the first time; it is tested against the former method of only cloning one of the parental automata during reproduction (called *trivial reproduction*).

Two fitness functions are studied, one designed for the task of simple *Collision Avoidance*, the other for Collision Avoidance with an additional subtask of passing a gate in the middle of the field (*Gate Passing*). Furthermore, a storing mechanism is presented to store the best genome found so far for each robot in a storage space called *Memory Genome*. It is an elitist strategy for decentralized evolution which has been applied in our earlier work [König *et al*. (2009)], but is described and analyzed here extensively for the first time.

Fitness function. A robot's behavior depends on the environment, and as in most cases in ER, there is no direct mapping from the genotype to a fitness value. Instead, a mapping has to be employed at the phenotypic level (robot's behavior) by some kind of fitness prediction. This has been studied in evolutionary computation, e. g., by Ong *et al*. [2003], Jin *et al*. [2001] and Jin [2005]. Due to the decentralized approach presented here, the fitness cannot be calculated by some global mechanism, but has to be determined by the sensor values and the inner state of each single robot. Also, the behavior has to be observed over time before an adequate fitness value can be calculated. This inherently leads to a delayed fitness, rating not the current behavior, but the one performed in the recent past.

In this work, a robot's fitness is calculated as the weighted sum of many *fitness snapshots* where a snapshot is an integer calculated from the sensor values received by the robot at a certain moment. Snapshots are taken at a constant time interval for every robot separately. In principle, every change of the genome (i. e., mutation or recombination) leads to a temporarily wrong fitness value, since the old value is not valid anymore, and the new behavior has not been observed, yet. However, it is not likely that a single mutation or recombination operation causes a drastic change of the behavior (by definition of these operators, see below). Therefore, it is reasonable not to reset the fitness value after each of these operations, since a recalculation from scratch is time-consuming. Instead, the fitness value is only adapted to the changed automaton which is expected to be similar. For this to work, the snapshots of old behaviors should lose their influence (*evaporate*) over time (since otherwise, e. g., a formerly bad automaton which changed to a good one would need a lot of time to compensate for the bad behavior in the past). This loss of influence is achieved by an exponential decrease of the weights for the old snapshots over time making them less influential to the total fitness sum.

The fitness calculation is divided into two parts which are constantly repeated during the runs:

(1) *Fitness snapshot:* A rating $snap_X(t)$ of the current situation, perceived through the sensors and the inner state at time step t, is added to the fitness f of the robot: $f \coloneqq f + snap_X(t)$ $(X \in \{CollAvoid, GatePass\}, t \in \mathbb{N})$. After a while, as more and more snapshots are summed up, the expected error caused by the preconditions of a robot is reduced and f is expected to adequately reflect the behavior.
(2) *Fitness evaporation:* The robot's fitness f is divided by a constant E: $f \coloneqq f \,/\, E$. The

evaporation accomplishes the exponential decrease in snapshot weight over time.

Both parts of the fitness calculation are performed separately, the fitness snapshot every $t_{snap} = 50$ simulation cycles, the evaporation every $t_{evap} = 300$ simulation cycles. The evaporation constant is set to $E = 2$.

More formally, the fitness f_R of a robot R at a time step $t \in \mathbb{N}$ is:

$$f_R = \sum_{i=0}^{t} s_X(i) \cdot c_i$$

where

$$s_X(i) = \begin{cases} snap_X(i), & \text{if } i \bmod t_{snap} = 0 \\ 0 & \text{otherwise} \end{cases}$$

and

$$c_i = E^{-\lfloor t/t_{evap} \rfloor + \lfloor i/t_{evap} \rfloor}$$

Note that this idea of fitness calculation can be compared to that of *eligibility traces* which are one basic mechanism of reinforcement learning [Sutton and Barto (1998)]. Based on the assumption that an automaton A loses credit for the current behavior when it is repeatedly replaced by mutated versions $M(A), M(M(A)), \ldots$ of itself, the snapshots calculated during the execution of the original automaton A should lose their influence on the current fitness value.

For the fitness snapshot of Collision Avoidance, two properties are considered: the robot must not stand still and it must not collide. The snapshot for Collision Avoidance is calculated by Algorithm 3.1.

Algorithm 3.1: Computation of fitness snapshot $snap_{CollAvoid}$ for Collision Avoidance.

---

input : Current operation $op \in Op$ of a robot R at a time step t; number of collisions $|Coll|$ of R since last snapshot before t.

output: Fitness snapshot for R at time step t.

int $snap \coloneqq 0$;
if $op = (Move, X), X \in B$ **then**
  $snap \coloneqq snap + 1$;
end
$snap \coloneqq snap - 3 \cdot |Coll|$;
return $snap$;

The snapshot for Gate Passing has the additional property that a reward is given when the robot passes the gate. It is calculated by Algorithm 3.2 which is based on Algorithm 3.1.

Algorithm 3.2: Computation of fitness snapshot $snap_{GatePass}$ for Gate Passing.

input : Current operation $op \in Op$ of a robot R at a time step t; number of collisions $|Coll|$ of R since last snapshot before t; Boolean value *Gate* indicating if the gate was passed since the last snapshot before t.
output: Fitness snapshot for R at time step t.

int $snap := snap_{CollAvoid}(op, |Coll|)$;

if *Gate* **then**
  $snap := snap + 10$;
end

return $snap$;

Mutation – general idea. The proposed mutation operator is intended to harden parts of the automaton which are highly involved in the behavior. Through this mechanism, an adequate search space structure for the evolving task is supposed to be found during the evolution.

Using MARBs as genotypic representation leads to a "flexible" search space. This means that the search space is infinite, but it is finite if the topology of the automata is fixed and the condition size is limited. Therefore, inserting a state or a transition into a MARB or extending a condition (by *AND*, *OR*) can be seen as a *complexification* of the search space (or as adding a dimension to it), while removing states or transitions or reducing a condition can be seen as a *simplification* (or as removing a dimension from the search space). As described by Stanley and Miikkulainen [2004], with a flexible search space there is no need to define the search space structure before the runs. Instead, there is a possibility to include the search for an adequate search space complexity for a desired behavior in the evolutionary process. To achieve this, the mutation operator in this article is designed to have the following two properties:

(1) Both complexification and simplification are allowed to occur during a run, keeping the search space flexible.
(2) By complexification, those parts of the topology are *hardened* which are expected to have high influence on the behavior.

The first property is similar to the approach by Stanley and Miikkulainen, which, however, allows one of the two to occur in a single run only.

The second property is new to the approach proposed. Here, hardening means lowering the probability of getting removed by mutation for certain parts of an automaton. On the level of states, the states with many incoming transitions are expected to be most involved in the behavior, so it should be unlikely to remove them through mutation. For transitions,

the complexity of the associated condition and its proximity to *true* can be used as an indicator for the transition's impact on the behavior.

During mutation, the hardening is only a random process, however, it gets directed by selection. Since selection is based on behaviors, hardened parts are expected to influence selection more than other parts. Over time, this leads to a hardening of those parts of automata which are most involved in a *good* behavior. For the other parts, mutation is only a random change and they are not expected to get hardened meaning they are loosely connected and can get removed within a few mutations.

Therefore, the complexity of the genome (i. e., the automaton) is expected to adapt to the complexity of the behavior which has to be learned. This process is implicitly split in two parts, where first the states, then the transitions are hardened.

Another advantage of using this idea for mutation is that most of the mutations defined below (namely those of the kinds (1) and (2) and some of (3)) naturally affect only the structure of the automaton, but have no impact on the behavior. This leads to large neutral plateaus which have been shown to affect evolution positively [Kimura (1985)]. These mutations are called *syntactic* while those which (potentially) affect the behavior are called *semantic*.

Definition of the mutation operator. The mutation operator M is defined as a mapping in the genotypic search space, depending on the state ξ of a random number generator: $M^{\xi} : \Gamma \to \Gamma$.

M consists of atomic mutations $M_1, \ldots, M_{11}$ which, by the above reasoning, can be divided into three general kinds:

(1) insert or remove states;
(2) insert or remove transitions;
(3) change labels (a: operations on state level, b: conditions on transition level).

The atomic mutations are listed here:

(1) States can always be inserted, but removed only if they do not have impact on the behavior:

M_1 (syntactic): Insert a state without incoming or outgoing transitions, with random operation and a random parameter.
M_2 (syntactic): Remove a random state with no incoming transitions (except the initial state which can only be deleted if it is the only state in the automaton).
M_3 (syntactic): Remove a random state associated with an $Idle$-operation and all outgoing transitions being associated with $false$.

(2) Transitions can be inserted with the associated condition $false$ and removed if they are associated to $false$ (no impact on behavior):

M_4 (syntactic): Insert a transition associated with $false$ between two arbitrary states (this hardens the state to which the transition points).
M_5 (syntactic): Remove a random transition associated with $false$.

(3a) The state labels are mutated by randomly choosing a state and changing its parameter. In addition to it, if the parameter would fall below zero by the change (which would be inconsistent with the MARB definition), the operation of the state gets changed (the idea is that, e. g., for small X, Y, $(Move, X)$ is similar to $(TurnLeft, Y)$). This can still have a drastic behavioral impact, therefore this mutation is used with a low probability (see Table 1):

M_6 (semantic): For a random state's output $op = (cmd, par) \in Op$, change this state's output to $(cmd', |par + c| + 1)$, where

$$c = rand(\{-5, \ldots, 5\}),\ cmd' = \begin{cases} cmd, & \text{if } par + c > 1 \\ rand(Cmd) & \text{otherwise} \end{cases}.$$

(3b) Transition labels are mutated by selecting a random transition and either performing a simplification or complexification on its condition with no impact on the behavior (M_7, M_8) or by changing a random atomic part of the condition with slight impact on the behavior (M_9), hardening the transition by complexifying and by moving towards *true*. Additionally, single values and sensor variables in a condition can be changed (M_{10}, M_{11}); this, however, can have a drastic impact on the behavior and is used with a low probability:

M_7 (syntactic): For $c \in C$ change:
(c *AND true*) to c,
(c *AND* $false$) to $false$,
(c *OR true*) to *true*, or
(c *OR* $false$) to c.
M_8 (syntactic): For $c \in C$ change:
c to (c *AND true*), or
c to (c *OR* $false$).
M_9 (semantic): An atomic part of a condition can be moved in small steps closer to *true* or $false$. "$P \leftrightarrow Q$" means that P can get changed to Q and vice versa. When mutating *true* and $false$ into atomic comparisons, a, b are chosen randomly. Let $a, b \in B_+ \cup H$ where $H \backslash \{a, b\} \neq H$:

$$false \leftrightarrow a = b \leftrightarrow a \approx b \leftrightarrow \begin{matrix} a \leq b \leftrightarrow a < b \\ a \geq b \leftrightarrow a > b \end{matrix} \leftrightarrow a \not\approx b \leftrightarrow a \neq b \leftrightarrow true$$

M_{10} (semantic): Change a number $i \in B$ within a condition to $i + rand(\{-5, \ldots, 5\})$.
M_{11} (semantic): Change a sensor variable $h \in H$ within a condition to $rand(H)$.

One single application of the mutation operator M is defined to choose one of the atomic mutations randomly, based on the distribution shown in Table 1. These probabilities have been determined by extensive studies and by reasonable assumptions like performing mutations more often if the expected change of behavior is small. If the chosen atomic mutation

cannot be performed (e. g., because no $false$ transition exists to be deleted in the case of M_5), the mutation has no effect for that application.

Table 1. Probability distribution for $M_1, \ldots, M_{11}$.

	M_1	M_2	M_3	M_4	M_5	M_6	M_7	M_8	M_9	M_{10}	M_{11}
Prob.	$\frac{4}{49}$	$\frac{3}{49}$	$\frac{3}{49}$	$\frac{7}{49}$	$\frac{7}{49}$	$\frac{1}{49}$	$\frac{4}{49}$	$\frac{6}{49}$	$\frac{10}{49}$	$\frac{3}{49}$	$\frac{1}{49}$

This mutation operator is complete in the sense that for each two MARBs $A, A' \in \Gamma$, there exists an $n \in \mathbb{N}$ and states of the random number generator $\xi_1, \ldots, \xi_n$ with $M^{\xi_1}(\ldots(M^{\xi_n}(A))\ldots) = A'$. This is apparent by the following proof sketch:

The empty automaton can be generated from every automaton A by

(1) reducing all conditions to $false$ using M_7 and M_9;
(2) deleting all transitions using M_5;
(3) deleting all states (which now cannot have any incoming transitions) using M_2.

From the achieved empty automaton every topology can be derived by using M_1 and M_4. At this point, all conditions are $false$. Using M_7, ..., M_9, every condition can be produced from $false$ by complexification. As all state labels already are arbitrary, this approach can prove that every automaton A' can be derived from every automaton A by a repeated application of M, i. e., M is complete.

Recombination and selection. Due to the onboard approach, selection cannot be defined as a population-based operator. In the approach proposed by König *et al.* [2008a], two robots mate when they come spatially close to each other. Offspring is produced by cloning the parent with the best fitness. Since reproduction in this approach occurs unpredictably, it is difficult to control the reproduction rate and the selection pressure.

In simulation, it is possible to control the reproduction rate and still use an approach similar to that of König *et al.* For this, a clock triggered by the simulation environment is used to synchronize reproduction. Therefore, the robots no longer reproduce when they meet each other, but all robots reproduce simultaneously according to this global clock; every robot mates with the one (or more) robot(s) it is spatially closest to. Although this is opposed to the decentralized paradigm, it can be used to get better insights into the effects of reproduction rate and selection pressure on evolution. These insights can then be used in a purely decentralized approach.

Definition of the recombination operator. The recombination operator $Cross$ is a mapping of a constant number p of parental genotypes $G_1, \ldots, G_p$ with corresponding fitness values $f_1, \ldots, f_p$ to p offspring genotypes: $Cross : (\Gamma \times \mathbb{Z})^p \to \Gamma^p$.

Given $(G_1, f_1), \ldots, (G_p, f_p) \in (\Gamma \times \mathbb{Z})^p$, one single offspring genotype is produced as follows:

(1) Choose the topology for the offspring fitness proportionally (based on $f_1, \ldots, f_p$) from one of the parental topologies.
(2) For all states and transitions which are present in all of the parents, choose the according label of the state or transition fitness proportionally from one of the parents and insert it into the offspring.

The exact algorithm for the production of an offspring automaton from p parental automata is shown in Algorithm 3.3. The p offspring automata are then produced by repeating the algorithm p times.

Algorithm 3.3: Computation of one child by the recombination operator $Cross$.

---

input : $\left((G_1, f_1), \ldots, (G_p, f_p)\right) \in (\Gamma \times \mathbb{Z})^p$.
output: $G \in \Gamma$.

Select a MARB $G' \in \{G_1, ..., G_p\}$ by fitness proportional distribution*;
Let $G := G'$;

for *all states q which exist in G* **do**
  if *a state q' exists in all MARBs in $\{G_1, ..., G_p\}$ with $id^q = id^{q'}$* **then**
    Select a MARB $G' \in \{G_1, ..., G_p\}$ by fitness proportional distribution*;
    Let q' be the (only) state in G' with $id^q = id^{q'}$;
    Set $Op^q := Op^{q'}$;
  end
end

for *all transitions $t = ((id^{q_1}, id^{q_2}), c) \in T(G)$* **do**
  if *a transition $t' = \left(\left(id^{q'_1}, id^{q'_2}\right), .\right)$ exists in all MARBs in $\{G_1, ..., G_p\}$ with $id^{q_1} = id^{q'_1}$ and $id^{q_2} = id^{q'_2}$* **then**
    Select a MARB $G' \in \{G_1, ..., G_p\}$ by fitness proportional distribution*;
    Let $t' = \left(\left(id^{q'_1}, id^{q'_2}\right), c'\right)$ be the (only) transition in G' with $id^{q_1} = id^{q'_1}$ and $id^{q_2} = id^{q'_2}$;
    Set $c := c'$;
  end
end

return G;

* The probability to be selected is fitness proportional for positive fitness values only; negative values are treated as 0. If all values are 0, the distribution is defined to be uniform.

The recombination operator $Cross$ does not affect the topologies of the parental automata, this is being done by mutation only. Instead, recombination on the one hand distributes the topologies in a new way according to the parental fitnesses. On the other hand,

Table 2. The eight main sets of experiments; for every set, the number of parents for reproduction was varied from 1 to 10.

	Memory Genome	Fitness function	Recombination	Parents
1	no	Collision Avoidance	trivial reproduction	1-10
2	yes	Collision Avoidance	trivial reproduction	1-10
3	no	Gate Passing	trivial reproduction	1-10
4	yes	Gate Passing	trivial reproduction	1-10
5	no	Collision Avoidance	new recombination $Cross$	1-10
6	yes	Collision Avoidance	new recombination $Cross$	1-10
7	no	Gate Passing	new recombination $Cross$	1-10
8	yes	Gate Passing	new recombination $Cross$	1-10

the labels are combined from the parental automata producing intermediate genotypes.

Memory Genome. Each robot has a *Memory Genome* which is a storage for the automaton with the highest fitness obtained so far. The Memory Genome's function is to keep the good genotypes over the course of evolution. In a constant interval, the current genotype is replaced by the stored genotype if the current fitness is lower than the old one, and only if the current fitness is negative or zero. The reason for the latter is that it could be shown that non-positive fitness is a good indicator for trivial behavior while positive fitness is a good indicator that the behavior is more complex and adapted to the desired task (cf. Section 4). Therefore, by reactivating the stored genome only in the case of non-positive fitness, the deletion of good new automata is avoided.

4. Method of Experimentation

Experiments have been performed to study the influence of

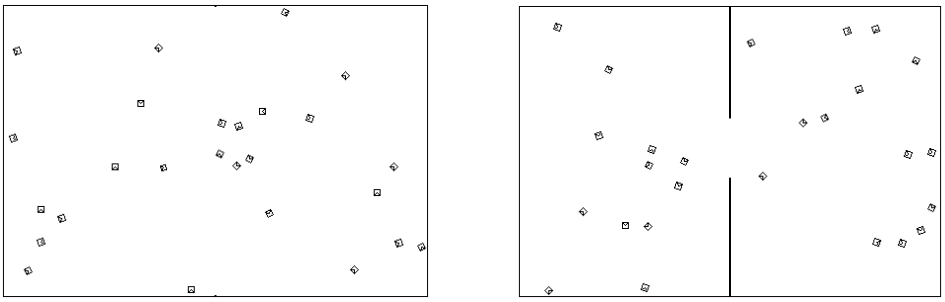
- the new recombination operator $Cross$ against a trivial reproduction mechanism based on cloning parental automata, cf. [König and Schmeck (2008b)],
- the number of parents for reproduction (varied between 1 and 10), and
- the Memory Genome against the classical approach without Memory Genome

on the quality of the resulting behaviors using the two different fitness functions for Collision Avoidance and Gate Passing, respectively. Every possible combination has been tested to provide evidence of potential dependencies. This leads to the eight main sets of experiments which are shown in Table 2.

For each of these sets, the number of parents for reproduction varied from 1 to 10, therefore 80 different parameter combinations have been tested overall. Every combination has been simulated for $80,000$ cycles, and every simulation has been repeated 26 times with different random seeds to gain statistical significance. A total of $8 \cdot 10 \cdot 26 = 2080$ simulations have been run.

The field. All experiments are performed on a rectangular field. For the Collision Avoidance task the field is designed to be empty (Figure 3(a)), where for the Gate Passing task a

wall with a gate in the middle is placed in the field (Figure 3(b)). No additional objects are placed in the field which means that walls and robots are the only further obstacles. The field is sized 1440 mm × 980 mm, the gate has an opening of 190 mm.



(a) (b)

Fig. 3. Experimental field with 26 robots for Collision Avoidance (a) and for Gate Passing (b).

General settings. 26 robots were placed randomly (positions and angles) on the field. Their initial genome was set to be empty (i. e., an automaton without any states or transitions). The experiments were run for 80, 000 simulation cycles; this complies with a real-world driving distance of about 320 m for a robot which is driving only straight forward or a real-world time of about 15 min. Mutation and recombination were performed every 100 and 200 cycles, respectively. For reproduction, the number of parents ranged in $p \in \{1, ..., 10\}$ for each of the eight experimental sets. The runs with $p = 1$ served as a control group only, as they did not provide any selective reproduction. The only way to direct evolution towards better fitness areas in these runs was to mutate and reuse the Memory Genome. The fitness snapshot was calculated and added every $t_{snap} = 50$ cycles and the fitness evaporation was performed every $t_{evap} = 300$ cycles dividing the fitness by $E = 2$. The interval for the Memory Genome was 1000 cycles for the runs it is used in.

Definition of *success*. In this article, robotic behavior is said to be *successful (according to a fitness function)* if it eventually leads to a positive (non-zero) fitness value when being executed in an environment. As argued before [König and Schmeck (2008b)], for the described fitness functions, a negative or zero-fitness (after a proper time of execution) implies a non-adapted behavior with a high probability, while a positive fitness implies with a high probability an adapted behavior. The reason for this lies in the way fitness is calculated, namely in the repeated rewards and punishments by adding the fitness snapshot. A bad behavior is expected to get many negative snapshots during a run while a good one is expected to get many positive snapshots; this leads over time to a discrimination at the zero-fitness level. Structural analysis of evolved automata with negative fitness at the end of the runs showed that in nearly all cases, they did not have both a reachable *Move*-state and a reachable *Turn*-state. This means that they were not capable of performing any non-

trivial behavior. On the other hand, about 90 % of the robots with positive fitness had both state types reachable. (A state s of automaton A is said to be *reachable* if there exists a path in A along the edges from the initial state q_0 to s; here, satisfiability of the conditions is not considered.)

Therefore, an evolved robot is called *successful* if it has a positive fitness in the final population of a run. An evolutionary run (experiment) is called *successful* if at least one successful robot exists in the final population.

Negative fitness. As argued above, negative fitness correlates with trivial behavior. Therefore, it does not provide much information by the amount of negativity; rather it distorts results if it is taken into consideration. For this reason, in the following evaluations negative fitness is always treated as zero.

Comparison to previous results. As described by König and Schmeck [2008b], studies were performed before in a similar scenario with the following differences: reproduction was based on simple cloning without recombination; only two parents at a time were reproduced; mutation was slightly altered; there was no Memory Genome; only Collision Avoidance was used as fitness function; interval parameters like the fitness snapshot interval t_{snap}, the evaporation interval t_{evap}, mutation and recombination intervals, etc. differed slightly (this could not be avoided as the intervals were defined in real-world time in the previous experiments; here, simulation cycles were used to define intervals which allows for a more objective comparison between simulations on different computers).

The runs with two parents for reproduction, no Memory Genome, and no recombination operator, in the scenario where Collision Avoidance was evolved, are supposed to reflect these previous experiments (cf. Section 5). As the results show, despite the aforementioned slight differences in the experimental setup, the outcomes are quite similar and comparisons between the old and the new results seem to be valid.

5. Experimental results

This section describes the influences of the Memory Genome, the new recombination operator and the number of parents for reproduction on the quality of the resulting behaviors at the evolution of Collision Avoidance and Gate Passing, respectively. Also, it provides a general analysis of the resulting automata.

Using the new operators, all four sets of experiments greatly outperformed previous results [König and Schmeck (2008b)] where 2.6% of the robots evolved a successful behavior, and 11.0% of the runs were successful. The runs lasted for about $2,000,000$ simulation cycles compared to $80,000$ cycles here (a reduction by a factor of 25). For the eight main sets of experiments conducted here, the following percentages of successful robots and successful runs have been achieved:

(1) Successful robots: $74,0\%$, successful runs: $93,1\%$.
(2) Successful robots: $87,6\%$, successful runs: $99,5\%$.
(3) Successful robots: $73,6\%$, successful runs: $87,2\%$.

(4) Successful robots: 83, 0%, successful runs: 99, 5%.
(5) Successful robots: 75.3%, successful runs: 92.7%.
(6) Successful robots: 88.7%, successful runs: 99.6%.
(7) Successful robots: 65.0%, successful runs: 83.3%.
(8) Successful robots: 84.0%, successful runs: 100.0%.

For these numbers, the experiments with only one parent for reproduction have been left aside. The reasons for the general improvement seem to be mainly

- the increased number of parents for reproduction (as two parents in the previous experiments seem to gain too little selection pressure), and
- the Memory Genome which prohibits the loss of already found good behavior.

The recombination operator seems to have different influences when used with or without the Memory Genome. Below is a detailed analysis of these factors.

5.1. *Influence of the Memory Genome*

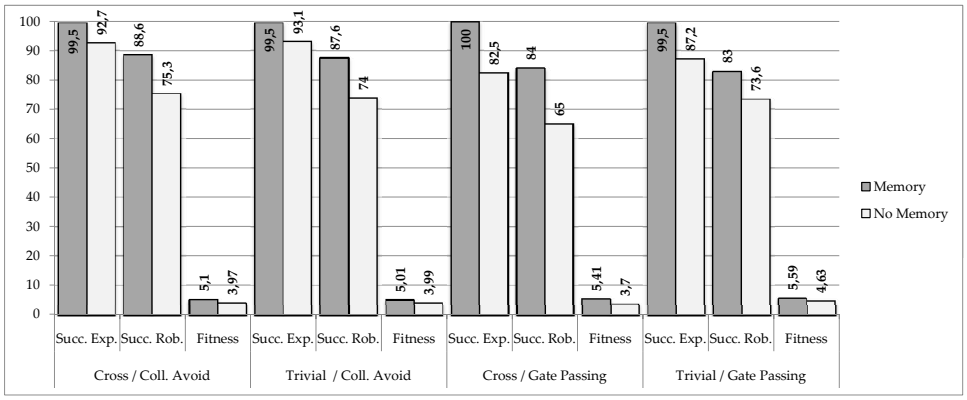


Fig. 4. Average results in terms of percentage of successful experiments, percentage of successful robots, and mean fitness in the end of the eight main sets of experiments. Runs with Memory Genome (dark) and runs without Memory Genome (light) are plotted next to each other.

The Memory Genome seems to be largely responsible for the aforementioned general improvement. Figure 4 shows the average percentage of successful experiments, of successful robots, and the mean population fitness in the last generation for the eight main sets of experiments (note that the fitness value is expressed absolutely and not in percent; it is still depicted in the same chart for compactness). The runs with Memory Genome are plotted next to those without Memory Genome while the other parameter combinations are grouped along the X-axis. In every parameter combination, and for all three indicators, the runs with Memory Genome (dark bars) outperform the runs without Memory Genome

(light bars). The plots in Figures 8, 9, 10 and 11 (see below) show that this outperformance is present for all different numbers of parents for reproduction (except for one).

When studying the fitness during the runs, the reasons for this expected success of the Memory Genome could be confirmed. In the previous experiments, populations tended to lose already found good behaviors by mutation. Therefore, studying only the last generations did not take into account that good behaviors might have existed earlier in the runs. Also, these good behaviors might have disappeared before evolution had a chance to improve them, so they were not able to reach their full potential.

In the current experiments with Memory Genome, the fitness is prevented from falling back into negative areas permanently, once a good, robust behavior has been achieved. Figure 5 shows the average population fitness during an example run with 6 parents where no Memory Genome is used. At about 50, 000 simulation cycles, a high fitness is achieved, however, when approaching 60, 000 cycles, the fitness decreases drastically indicating that a formerly good behavior has been lost. Later a fairly high fitness is reached again, but it oscillates a lot and does not stabilize. In contrast, Figure 6 shows the average fitness during a run with the same parameters except that the Memory Genome is used. A quite high fitness is achieved at about 35, 000 cycles and it is conserved at a stable level until the end of the run. We interpret this as an effect of the Memory Genome.

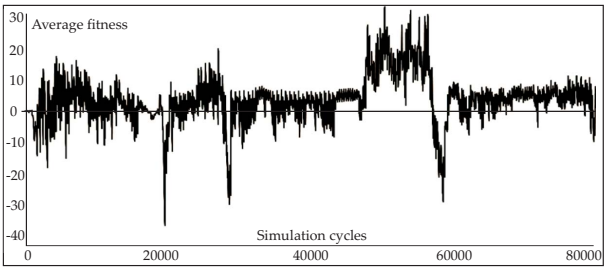


Fig. 5. Average fitness in a population during a run with 6 reproduction parents *without* Memory Genome evolving Gate Passing.

As it is essential in an online-evolutionary approach that not only a good behavior is resulting in the end of a run, but that it can be trusted in the stability of intermediate behaviors [König *et al.* (2008a)], we suggest to always use the Memory Genome or a similar mechanism in online-evolutionary approaches.

5.2. *Influence of the recombination operator*

Figure 7 (as Figure 4) shows the average percentage of successful experiments, of successful robots, and the mean population fitness in the last generation for the eight main sets of experiments. The chart depicts the same data as Figure 4, except that the bars are ordered differently to highlight the comparison between runs with trivial reproduction and runs with the recombination operator $Cross$.

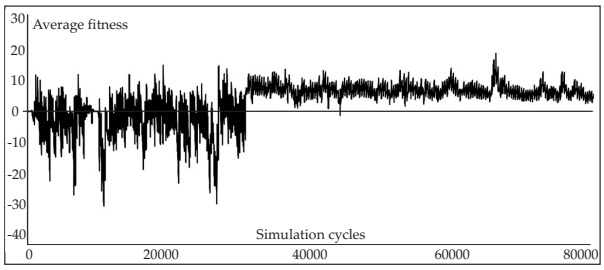


Fig. 6. Average fitness in a population during a run with 6 reproduction parents *with* Memory Genome evolving Gate Passing.

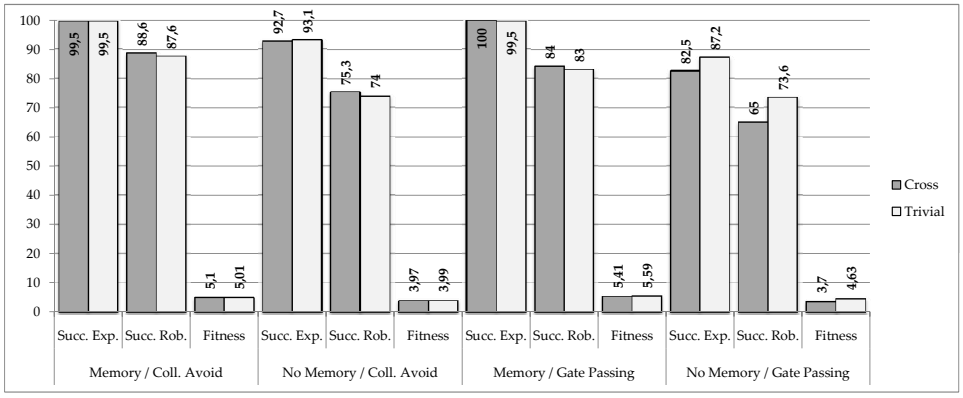


Fig. 7. Average results in terms of percentage of successful experiments, percentage of successful robots, and mean fitness in the end of the eight main sets of experiments. Runs with new recombination (dark) and runs with trivial reproduction (light) are plotted next to each other.

The chart shows that for runs *without Memory Genome* the trivial reproduction outperforms the recombination operator *Cross* (quite clearly for the right group evolving Gate Passing, and rather slightly for the second group from the left evolving Collision Avoidance).

For both groups *with Memory Genome* (first and third group from the left), however, the recombination operator *Cross* slightly outperforms the trivial reproduction.

We interpret this results as follows: as the recombination operator *Cross* introduces additional diversity in the population, the runs without Memory Genome are less capable of keeping good behavior and the probability increases that it gets lost. In the runs with Memory Genome, however, the operator is capable of improving the results as intended, since already found good behavior is preserved and the additional diversity has a positive effect here. As the usage of the Memory Genome seems reasonable in most cases, the recombination operator *Cross* can be seen as a general improvement. However, as the enhancement is rather small, the operator should be further studied and eventually replaced

by a better one.

5.3. *Influence of the number of parents for reproduction*

Figures 8 and 9 show the average fitness of robots and the average number of successful robots in the final populations for runs with trivial reproduction (i. e., main experimental sets 1-4); different plot types denote the parameter usage (Memory Genome vs. no Memory Genome and Collision Avoidance vs. Gate Passing); the X-axis plots the number of parents for reproduction. Figures 10 and 11 show the same data using the recombination operator *Cross* instead of trivial reproduction (i. e., main experimental sets 5-8).

As expected, the runs with only one parent were not able to achieve successful behavior in neither case. Not even the runs using the Memory Genome reached a fitness significantly above zero despite the possibility to improve behavior by a simple random search using mutation only and storing the best genome found by pure chance. However, since negative fitness values are treated as zero, the values for one parent are still slightly positive in all charts.

The performance in runs with two parents was disproportionately better with the Memory Genome than without the Memory Genome (compared to runs with three and more parents). This indicates that the Memory Genome is able to compensate for the lack of selection pressure (as observed by König and Schmeck [2008b]) when having only two parents for reproduction. However, independently of the Memory Genome, both fitness and number of successful robots increase with the number of parents for reproduction until an optimum is reached. The optimal number of parents seems to be between four and seven.

Over all numbers of parents, the runs with Memory Genome perform best which is not unexpected as it is a known result that elitist strategies can have positive effects if evolution is not capable of keeping good solutions in the population.

5.4. *Analysis of evolved automata*

Among the non-trivial (successful) runs, the evolved behaviors can roughly be divided into three groups: (1) Collision Avoidance, (2) “Altruistic” Gate Passing with Collision Avoidance, and (3) “Egoistic” Gate Passing. In each of the groups, behaviors of different complexity and robustness were evolved. The assignment of evolved behaviors to the groups has been performed manually by observation and for a minor portion of the 2080 runs only. Therefore, no exact count of behaviors per group can be given. However, in runs evolving Collision Avoidance only behaviors from the first group occurred while in runs evolving Gate Passing there were behaviors from all three groups.

Group 1: Collision Avoidance. The behaviors from this group use obstacle avoiding strategies which range from very simple to highly adapted. Some simple automata are performing a circle-driving behavior without obstacle detection. This can lead to a situation where nearly no collisions occur if the entire population is doing the same, however, the achieved fitness is rather low. Some automata, on the other hand, are more sophisticated by not only

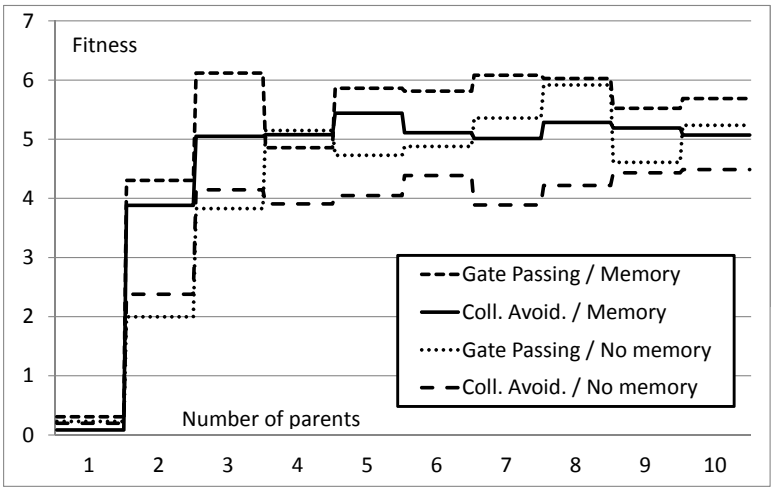


Fig. 8. Average fitness of robots in last populations of the runs with trivial reproduction. The number of parents for reproduction is depicted on the X-axis. The Y-axis shows the fitness.

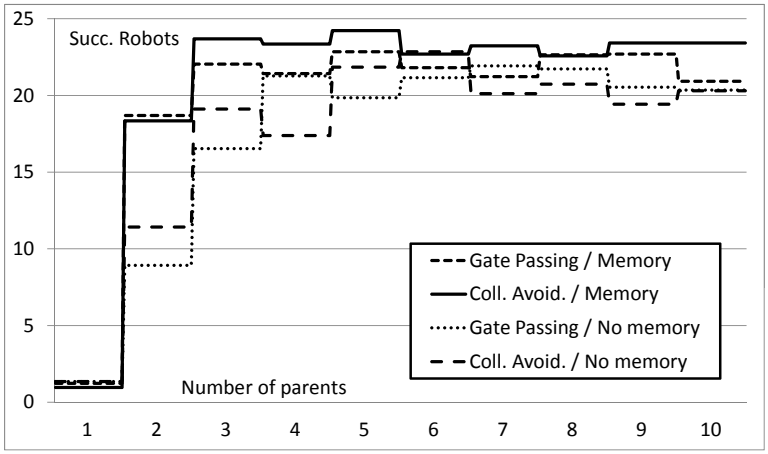


Fig. 9. Average number of successful robots in last populations of the runs with trivial reproduction. The number of parents for reproduction is depicted on the X-axis. The Y-axis shows the number of successful robots.

avoiding collisions, but, e. g., deciding on which side of an obstacle to drive past to minimize the need for turning, thus maximizing the number of *Move*-operations.

It is notable that many behaviors which were evolved for Gate Passing still were assigned to this group. The reason seems to be that Collision Avoidance is a subtask of Gate Passing and forms a local optimum in the fitness landscape where evolution tends to get stuck in. To check this conjecture, a more objective and automated method for categorizing behaviors is needed.

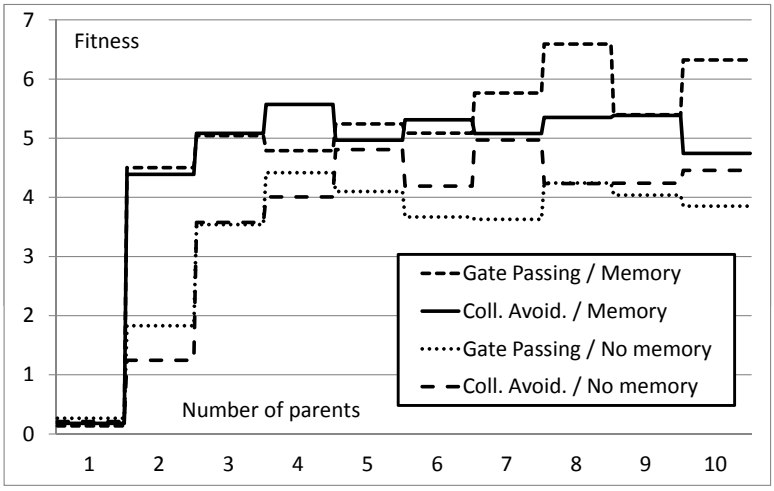


Fig. 10. Average fitness of robots in last populations of the runs with recombination operator *Cross*. The number of parents for reproduction is depicted on the X-axis. The Y-axis shows the fitness.

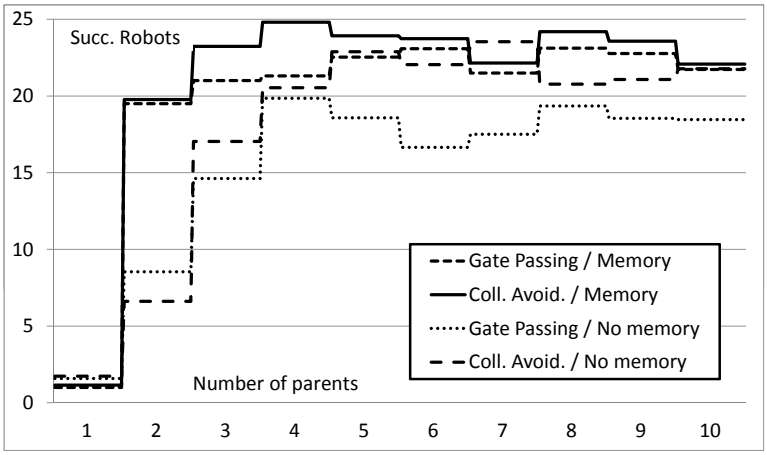


Fig. 11. Average number of successful robots in last populations of the runs with recombination operator *Cross*. The number of parents for reproduction is depicted on the X-axis. The Y-axis shows the number of successful robots.

Figure 12 shows a trajectory of a robot from Group 1 in an empty field without other robots. The X marks the starting position. Figure 13 shows a trajectory of the same robot in a more complex environment. It collides one time at the beginning with the small obstacle in the middle of the field, but afterwards it is capable of driving around without further collisions.

Group 2: "Altruistic" Gate Passing with Collision Avoidance. Behaviors in this group

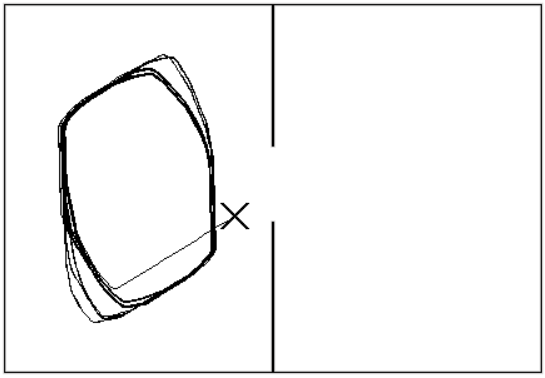


Fig. 12. Trajectory of an evolved robot doing Collision Avoidance without passing the gate.

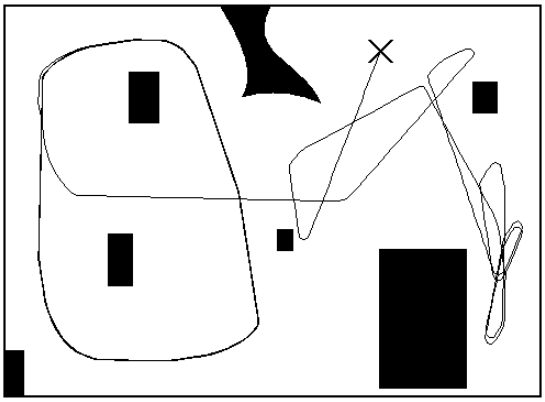


Fig. 13. Trajectory of the same robot as in Figure 12 in a more complex environment.

are the most effective ones in gaining fitness for the entire population. They have in common that the behavior includes an altruistic component which leads to a suboptimal individual fitness for the collective good. Figure 14 shows the trajectory of a robot from this group. The robot is driving close to the wall avoiding collisions until it recognizes the gate and passes it (this does *not* happen every time it drives past, but sufficiently rarely that the gate is never crowded). Then it performs a small loop on the other side and drives back through the gate. Since the robot does not pass the gate every time, there is enough space for the whole population to profit from the behavior. Figure 16 shows the corresponding automaton which is analyzed below.

Group 3: "Egoistic" Gate Passing. This group consists of robots which found a fitness niche exploitable for at most one or two robots at a time, gaining, however, great fitness for them. They developed a mechanism to recognize the gate and then drive constantly through it, back and forth. In some populations, the gate passing robots interchanged, in others, one

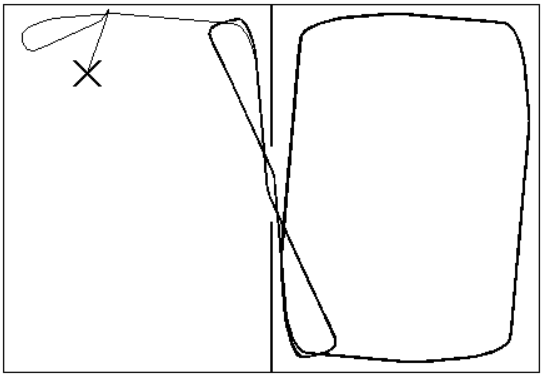


Fig. 14. Trajectory from group 2 produced by the automaton in Figure 16.

single robot performed the gate passing while other robots could not recognize the blocked gate. In many of these populations, Collision Avoidance was not evolved since the constant Gate Passing gained a high fitness. However, population fitness was lower than in group 2. Figure 15 shows a trajectory of a robot performing this behavior.

Automata from groups 2 and 3 are typically hardly transferrable into new environments as they implicitly need the gate to perform the evolved behavior accurately.

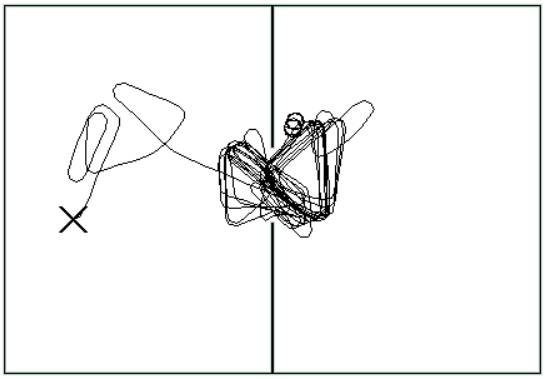


Fig. 15. Trajectory of an evolved robot from group 3 driving constantly through the gate.

Analysis of the structure of an example automaton. Figure 16 shows an automaton from group 2, evolved to do Gate Passing as depicted in Figure 14. The automaton consists of six states, however, the gray painted states are not reachable. Also, the gray transitions are inactive. By the design of the mutation operator, the gray parts can be deleted within a small number of mutations, as they are not hardened. If further evolved, these parts are expected to either be involved in the behavior or deleted. Therefore, the automaton essentially consists of 3 states and 5 transitions. The transitions have quite complex conditions which is

plausible, since the robot is capable of recognizing the gate which is non-trivial. However, there are also neutral elements in the conditions, e. g., all constant parts (*true*, $false$); they could be removed leading to equivalent simpler conditions. During evolution, the inactive elements serve as neutral plateaus which eventually get deleted by mutation if they are not needed. In contrast, the hardened parts of the automaton are unlikely to get removed.

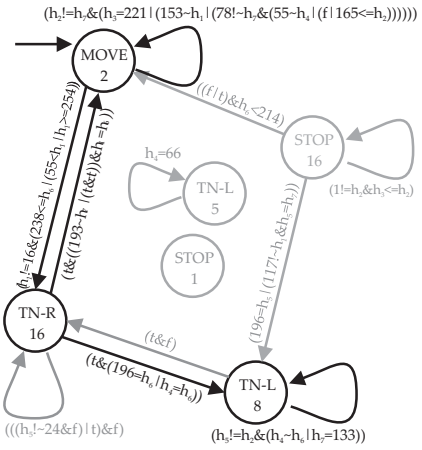


Fig. 16. Automaton avoiding collisions and driving through the gate; cf. trajectory in Figure 14. Unreachable states and unsatisfiable conditions are painted gray; they are not hardened and therefore can get deleted in a small number of mutations. *true* is denoted by 't', $false$ by 'f'.

5.5. *Summary and overall results*

The results show that a major improvement can be achieved by increasing the number of parents at reproduction from two to about four to seven which raises selection pressure, and by using the Memory Genome for preservation of good behavior. The recombination operator *Cross* has been shown to improve performance when used together with the Memory Genome and to decrease it otherwise. This can be explained by the additional diversity which is introduced by using a recombination operator.

Analyzing the resulting behaviors, roughly three groups can be identified: (1) Collision Avoidance, (2) "Altruistic" Gate Passing with Collision Avoidance, and (3) "Egoistic" Gate Passing.

Analyzing the structure of the resulting automata indicates that the hardening of good parts of genomes leads to an adaptation of genome complexity to target behavior complexity.

6. Conclusion and Outlook

It has been shown that a decentralized approach for online evolution based on the MARB model is capable of robustly evolving Collision Avoidance and Gate Passing in a swarm of simulated Jasmine IIIp robots. Overall, 78.3 % of the robots achieved positive fitness (called successful robots) and in 93.9 % of the runs there were robots with positive fitness in the final population (called successful runs). The runs lasted for 80, 000 simulation cycles which can be compared to 15 minutes in an experiment with real Jasmine IIIp robots.

Due to the special design of the mutation operator, the complexity of the generated automata is adapted to the problem complexity. This does not necessarily mean that the automata have few states or transitions or very simple conditions. But those states or transitions which are not involved in the behavior can be deleted anytime within few mutations and in the same way parts of conditions which do not affect the behavior can be simplified. Therefore, evolution has the possibility of exploring large neutral plateaus which are only loosely connected to the automaton meaning that these parts can be deleted if they are found to be useless.

The Memory Genome has been shown to greatly improve the quality of the evolved behaviors while the new recombination operator $Cross$ seems to work only when used together with the Memory Genome. The best number of parents for reproduction seems to be between four and seven.

As the framework is working fine with simple behaviors, an important goal for the future is the evolution of more sophisticated behaviors. Wall Following is planned to be evolved as well as the recognition of shapes and collaborative behaviors like moving heavy objects. Also, dynamic environments will be studied. As the recombination operator seems to affect evolution positively, similar operators will also be tested and the improvement will be further optimized. Another objective is to improve the analysis on the genotypic level to allow for more objective categorizations of evolved behaviors. Such an analysis has been performed to confirm that positive fitness is a good indicator for evolutionary success (cf. Section 4). However, a more advanced analysis on the genotypic level is planned for future work to exploit the structural simplicity of the MARB model.

References

Ampatzis, C., Tuci, E., Trianni, V. and Dorigo, M. (2005). Evolving communicating agents that integrate information over time: a real robot experiment. In *CD–ROM Proceedings of the Seventh International Conference on Artificial Evolution (EA 2005)*. Springer Verlag.

Ampatzis, C., Tuci, E., Trianni, V. and Dorigo. M. (2006). Evolution of signalling in a group of robots controlled by dynamic neural networks. In E. S. et al., editor, *Proceedings of the Second Workshop on Swarm Robotics*. Springer Verlag.

Bonabeau, E., Theraulaz, G. and Dorigo, M. (1999). *Swarm Intelligence: From Natural to Artificial Systems (Santa Fe Institute Studies in the Sciences of Complexity)*. Oxford University Press.

Braitenberg, V. (1984). *Vehicles: Experiments in Synthetic Psychology*. Cambridge, MA: MIT Press.

Floreano, D., Husbands, P. and Nolfi, S. (2008). *Evolutionary Robotics in Springer Handbook of Robotics*. Springer Verlag.

Fogel, L. J., Owens, A. J. and Walsh, M. J. (1966). *Artificial Intelligence through Simulated Evolution*. John Wiley.

Fogel, L. J., Angeline, P. J. and Fogel, D. B. (1995). An evolutionary programming approach to self-adaptation on finite state machines. In *Proceedings of the Fourth Annual Conference on Evolutionary Programming*, 355–365. MIT Press.

Fogel, L. J. (1999). *Intelligence Through Simulated Evolution: Forty Years of Evolutionary Programming.* Wiley Series on Intelligent Systems.

Fogel, D. B. (2006). *Evolutionary Computation: Toward a New Philosophy of Machine Intelligence.* Wiley-IEEE Press.

Groß, R. and Dorigo, M. (2004). Cooperative transport of objects of different shapes and sizes. *Ant Colony Optimization and Swarm Intelligence, 4th International Workshop*, 106–117. Springer Verlag

Hartland, C. and Bredeche, N. (2006). Evolutionary robotics, anticipation and the reality gap. *3rd IEEE International Conference on Robotics and Biomimetics*, 1640–1645.

Hettiarachchi, S. D. (2007). *Distributed Evolution for Swarm Robotics (PhD thesis).* University of Wyoming.

Hettiarachchi, S. D., Spears, W. M. (2006). DAEDALUS for agents with obstructed perception. *2006 IEEE Mountain Workshop on Adaptive and Learning Systems*, 195–200.

Hopcroft, E. J. and Ullman, J. D. (1979). *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley.

Jakobi, N. (1997). Evolutionary robotics and the radical envelope-of-noise hypothesis. *Adapt. Behav.*, 6(2):325–368.

Jin, Y. (2005). A comprehensive survey of fitness approximation in evolutionary computation. *Soft Computing - A Fusion of Foundations, Methodologies and Applications*, 9(1):3–12.

Jin, Y., Olhofer, M. and Sendhoff, B. (2001). Managing approximate models in evolutionary aerodynamic design optimization. In *In Proceedings of IEEE Congress on Evolutionary Computation*, 592–599. IEEE Press.

Kimura, M. (1985). *The Neutral Theory of Molecular Evolution*. Cambridge University Press.

König, L., Jebens, K., Kernbach, S. and Levi, P. (2008a). Stability of on-line and on-board evolving of adaptive collective behavior. In *Springer Tracts in Advanced Robotics*, 293–302.

König, L. and Schmeck, H. (2008b). Evolving collision avoidance on autonomous robots. In M. Hinchey, A. Pagnoni, F. Rammig, and H. Schmeck, editors, *Biologically Inspired Collaborative Computing*, 85–94.

König, L., Mostaghim, S., and Schmeck, H. (2009). Online and onboard evolution of robotic behaivior using finite state machines. In *8th International Conference on Autonomous Agents and Multiagent Systems*, 1325–1326.

Nolfi, S. and Floreano, D. (2001). *Evolutionary Robotics. The Biology, Intelligence, and Technology of Self-Organizing Machines.* The MIT Press, Cambridge, Massachusetts.

Ong, Y., Nair, P., and Keane, A. (2003). Evolutionary optimization of computationally expensive problems via surrogate modeling. *AIAA Journal*, 41(4):687–696.

Spears, W. M., Gordon, D. F. (2003). Evolution of strategies for resource protection problems. *Advances in evolutionary computing: theory and applications*, 367–392.

Stanley, K. O. and Miikkulainen, R. (2004). Competitive coevolution through evolutionary complexification. *Journal of Artificial Intelligence Research*, 21:63–100.

Sutton, R. S. and Barto, A. G. (1998). *Reinforcement Learning – An Introduction.* MIT Press.

Walker, J., Garrett, S. and Wilson, M. (2003). Evolving controllers for real robots – a survey of the literature. *Adaptive Behavior*, 11:179–203.

Watson, R., Ficici, S. and Pollack, J. (2002). Embodied evolution: Distributing an evolutionary algorithm in a population of robots. In *Robotics and Autonomous Systems*, pages 1–18.

List of Figures

List of Tables

Biography

Lukas König is a PhD Student and a Research Associate at Karlsruhe Institute of Technology (KIT), Germany. He graduated from University of Stuttgart, Germany, with a degree in Computer Science in 2007. He started research in Evolutionary Swarm Robotics in 2006 which is still his main research topic. Besides doing experimental studies with real and simulated robots, he is interested in obtaining a theoretical understanding of the mechanisms influencing evolution.

Lukas König,
lukas.koenig@kit.edu

Sanaz Mostaghim is currently working as a Lecturer and Research Assistant at Karlsruhe Institute of Technology (KIT), Germany. She received her PhD degree in Electrical Engineering from the University of Paderborn in Germany in 2004. After her PhD, she worked as a Post Doctoral Fellow at the Swiss Federal Institute of Technology (ETH) in Zurich, Switzerland. She has worked on multi-objective optimization algorithms using evolutionary algorithms and particle swarm optimization and applied them to several different applications from geology and computational chemistry. Parallel optimization, multi-objective optimization, Particle Swarm Optimization, Grid Computing, and Organic Computing are her research topics.

Sanaz Mostaghim,
sanaz.mostaghim@kit.edu

Hartmut Schmeck,
hartmut.schmeck@kit.edu

Hartmut Schmeck is a Full Professor of Applied Informatics at the Karlsruhe Institute of Technology (KIT), Germany. His current major research interest is on self-organization and adaptivity in complex technical systems. He is a Key Member of the Organic Computing Initiative and Coordinator of the German Priority Programme on Organic Computing. At the KIT, he is the Scientific Spokesperson of the newly formed KIT-Focus Area COMMputation addressing the inherent combination of communication and computation which is a characteristic feature of smart application systems.