# Evolving Collision Avoidance on Autonomous Robots

Lukas König and Hartmut Schmeck

**Abstract** Utilizing the collective behavior of a population of interacting individuals, based on rather simple local algorithms, is a promising approach for achieving complex goals. We use an onboard online evolutionary model, based on finite Moore automata, to develop collective behavior in an artificial swarm of micro-robots. Experiments have been made in simulation to achieve Collision Avoidance. The model is shown to be capable to generate the desired behavior and we present experiments for adjusting the parameters of the evolutionary optimization.

## 1 Introduction

As it has been shown in the past [1, 2], collective behavior can reduce the algorithmic complexity of solving tasks, by exploiting emergent effects in a swarm originating from the interaction between its single individuals. Modern robotics can profit from this concept. However, the emergent collective behavior is hard to predict, and given a task, it is generally not obvious, how programs should be designed to provide, as a collective behavior, the solution of this task. In some cases, one can develop algorithms and strategies manually, but in general, this turns out to be hard even for simple tasks and non-collective applications.

A common strategy for finding potential templates or strategies for the design of a collective behavior is to observe swarms in nature. They show adaptive behavior and, quite often, they are at least close to solving some of their environmental challenges in an optimal way, the most prominent, standard example being the capability of ant colonies to find shortest paths [3]. By a careful analysis of the behavior of natural and artificial swarms, one can hopefully extract appropriate local rules for collective behavior [1]. However, there is no guarantee that the behavior of a natural swarm can be understood sufficiently well to successfully generate the required behavior for an artificial swarm. Also, there are many conceivable tasks for artificial swarms which do not have a related counterpart in nature.

Lukas König
University of Karlsruhe, AIFB, e-mail: koenig@aifb.uni-karlsruhe.de

Hartmut Schmeck
University of Karlsruhe, AIFB, e-mail: schmeck@aifb.uni-karlsruhe.de

One of the suggested approaches to generate local tasks for collections of cooperating agents is to use evolutionary algorithms or genetic programming [4]. In this paper, we use an *onboard online* evolutionary approach to develop collective behavior in a population of robots. *Onboard* means that each robot has an evolutionary program running, which is separated from the other robots and especially is not triggered by some kind of central control with global information. *Online* means that during a run, the robots are supposed not only to achieve the ability to solve a given task, but also to solve the task in the given environment in order to evaluate the feasibility of the current solution. So the idea is to confront collections of robots with a problem, and to let them learn cooperatively through evolution, until an adequate solution is found, and to solve the problem at the same time.

Based on the approach presented in [5] the evolutionary model is built on finite Moore automata and it is defined in a general way to be applicable on different robot platforms. Up to now it has been implemented and tested on the Jasmine IIIp robot platform at the University of Stuttgart and in simulation, where also the Jasmine IIIp robot has been modeled. The Jasmine IIIp series is a swarm of micro-robots sized $26 \times 26 \times 26 mm^3$. It can drive forwards and backwards and turn left and right. Each robot has seven infra-red sensors (two facing to the front, the others being placed in steps of 60 degrees around, each returning values between 0 and 255) to measure distances to obstacles and to communicate with other robots (cf. www.swarmrobot.org).

This paper extends the approach presented in [5] by using extensive simulation experiments to adjust evolutionary parameters and to show that Collision Avoidance can be evolved.

In Sec. 2, we describe the theoretical model and the implementation of the framework for the Jasmine III robot in simulation. In Sec. 3, results of the evolutionary runs are presented. Sec. 4 provides a conclusion and an outlook to future work.

## 2 Model description

The developed behavioral model for robots is based on finite Moore automata defined in a common manner [6]. The output of a state defines an instruction to be executed. The transitions depend on the internal state and on the information provided by the sensors. The automaton is referred to as the *genome* of the robot, while the resulting behavior (i. e., the mapping from a sequence of sensor data to the corresponding sequence of output instructions) is called the phenotype; accordingly, the *genotypic search space* $\Gamma$ is defined to be the space of all Moore automata, while the *phenotypic search space* $\Omega$ is the space of all behaviors. The genome can be modified by mutation and crossover.

Due to space limitations, this section only provides a brief overview of the model and the implementation. For more detailed information see [5].

**Preliminaries.** We denote a set of byte values and a set of positive byte values as $B = \{0,...,255\}$ and $B_+ = \{1,...,255\}$. The behavior depends on sensor data represented by a set $H$ of $n$ sensor variables $H = \{h_1,...,h_n\}$. The sensor data may originate from real or virtual sensors (the latter being any internal variables of the robot). Each variable $h_i$ can be set to any byte value. The seven main infra-red sensors of the robot are stored as $h_1,...,h_7$, starting with the two sensors facing to the front and then incrementing clockwise. We did not use any other sensors for the experiments.

We assume a set $\mathcal{I} = \{I_1,...,I_m\} \subseteq B_+$ of $m$ instructions, encoded as positive byte values. In general, instructions may be interpreted as arbitrary programs, which are capable to run on a robot; however, up to now only the following simple instructions have been used: (1) Idle (i. e., "keep executing the last instruction"), (2) Stop, (3) Move forward, (4) Turn left, (5) Turn right.

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

**Moore automaton for robot behavior.** A finite Moore automaton for robot behavior (MARB) is a Moore Automaton $A = (Q, \Sigma, \Omega, \delta, \lambda, q_0)$, where:

$Q$ is the set of states ($q_0$ being the initial state).

The input alphabet $\Sigma = B^n$ consists of all possible combinations of sensor values.

The output alphabet $\Omega = \mathcal{I} \times B_+$ consists of the instructions with an additional parameter.

The transition function $\delta$ is defined for any state and each combination of sensor values in $\Sigma$ by specifying for each state $q$ a list $((c_1,q_1),(c_2,q_2),...)$ of conditions and associated following states; it is interpreted like a case-statement, i. e., the first condition evaluating to *true* under the current input determines the next state. The conditions are conjunctive and disjunctive combinations of *false*, *true*, or relational expressions of the type *a rel b*, where $a, b \in H \cup B_+$, $rel \in \{<,>,\leq,\geq,=,\neq,\approx,\not\approx\}$. $\approx$ is true ($\not\approx$ is false) whenever the two operands differ by at most a constant (which is set to 5 in our experiments). If none of the conditions evaluates to true there is a default transition to the initial state (see Fig. 1).

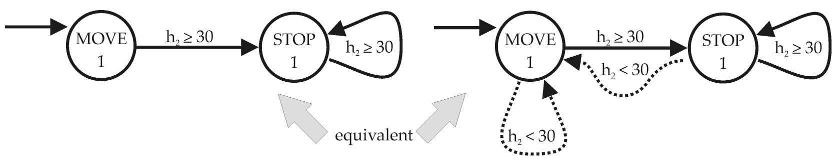

**Fig. 1** Example of implicit transitions to the initial state if no condition is *true*.

The output function $\lambda$ assigns to each state an instruction with a parameter, e. g., (Turn left, 45), which lets the robot turn left by 45 degrees. The parameter can be any positive byte value.

**Mutation.** The mutation operator is defined as a mapping in the genotypic search space: $M : \Gamma \to \Gamma$ (i. e. it maps a MARB $A$ into a MARB $M(A)$). Let $k \in \{0,...,255\}$ be a constant (we set $k = 5$ in all experiments). A mutation randomly selects one of the following atomic transformations:

1. Toggle "inactive" transitions (*syntactic*, i. e., no change of behavior): Remove a random transition, associated with the condition *false* or add a random transition, associated with the condition *false*.
2. Remove an "inactive" state (*syntactic*): Remove a state $q$ without incoming transitions or with all outgoing transitions being associated with the condition *false* and the state being associated with the instruction *IDLE*.
3. Add a new state $q$ (*syntactic*): $q$ has no incoming transitions and no outgoing transitions, random instruction and a random parameter $\leq k$.
4. Change a condition: Let $a, b \in B_+ \cup H$, $c$ a condition. Any part of a condition that matches the following patterns can be mutated (the notation $x \leftrightarrow x'$ means that $x$ may be replaced by $x'$ and vice versa.):

   a. (*semantic*, i. e., potential change of behavior)
   $$false \leftrightarrow a = b \leftrightarrow a \approx b \leftrightarrow \begin{matrix} a \leq b \leftrightarrow a < b \\ a \geq b \leftrightarrow a > b \end{matrix} \leftrightarrow a \not\approx b \leftrightarrow a \neq b \leftrightarrow true$$
   b. One of the following (*syntactic*):
   $$\begin{array}{rcccl} (c\ AND\ true) & \leftrightarrow & c & \leftrightarrow & (true\ AND\ c) \\ (c\ OR\ true) & \rightarrow & true & \leftarrow & (true\ OR\ c) \\ (c\ AND\ false) & \rightarrow & false & \leftarrow & (false\ AND\ c) \\ (c\ OR\ false) & \leftrightarrow & c & \leftrightarrow & (false\ OR\ c) \end{array}$$
   c. Change a number $i$ within a condition (*semantic*): Let $i' = i + rand(\{-k,...,k\})$, Replace $i$ as follows:
   $$i \rightarrow \begin{cases} i', & \text{if } 1 \leq i' \leq 255 \\ 1, & \text{if } i' < 1 \\ 255, & \text{if } i' > 255 \end{cases} .$$
   d. Change a sensor variable $h$ within a condition (*semantic*): Replace $h$ with $rand(H)$.

5. Change a state $q$ (*semantic*): Let $(I, P)$ be the output of $q$.
   Replace $(I, P)$ with $(J, |P + c| + 1)$, where
   $$c = rand(\{-k,...,k\}),\ J = \begin{cases} I, & \text{if } P + c > 1 \\ rand(I) & \text{otherwise} \end{cases} .$$

Mutation was performed once within a time interval $S$. This interval was studied in the experiments (see Sec. 3).

Obviously, an appropriate sequence of mutations can transform any MARB $A$ into any other MARB $A'$ by changing its topology, conditions, and the output function (i. e. the mutation operator is complete). In order to make the mutations "smooth" in the sense that a single mutation causes only a "small" change in the phenotypic search space, the focus is on keeping the semantic mutations (i. e., mutations that potentially change the behavior) few and small.

**Reproduction / Selection.** Since in the onboard concept there is no central unit with information about the whole population, it is not possible to implement a global selection operator. Instead, similar to the diffusion model of evolutionary algorithms (cf. [7]) we use local selection: A robot produces a child genome together with its closest neighbor. As in [5], a very simplified recombination operator is used which assigns to both robots the parental genome having the better fitness. Note, however, that this could easily be replaced with a more standard stochastic crossover operator combining parts of both parental genomes into a child genome. Future work will feature such a crossover operator.

On real robots, reproduction is performed each time two robots meet, i. e., come closer to each other than some threshold. In simulation, we implemented a similar solution which, however, is easier to analyze: Using a constant time interval $T$, each robot reproduces with the robot which, after $T$ time units, is the spatially closest to itself. However, this does not mean that all robots recombine simultaneously; for each robot a separate timer is running which, due to possibly delayed requests, drifts apart during the experiment. The parameter $T$ was studied in the experiments (see Sec. 3).

**Fitness function.** As the fitness of a MARB has to be evaluated locally, it has to be based on the observed sequence of sensor data which is influenced by the generated behavior of the robot. Therefore, every $U$ time units, the fitness value is updated by a "fitness snapshot" (see below).

Since mutations modify the behavior, the fitness value has to be adjusted. This is done by using "evaporation", i. e., every $V$ time units, the fitness value of the robot is divided by 2. $U$ and $V$ are parameters that are studied in Sec. 3. Furthermore, undesirable events (like collisions) should modify the fitness appropriately.

For *Collision Avoidance*, we used a fitness measure, which states that moving around is good, but being near an obstacle is bad; colliding with an obstacle is even worse. The fitness assignment is shown in Alg. 1. It holds that *NOT_MOVING* is 0, if the robot's current instruction is "Move", 1 otherwise; *OBST_NEAR* is 1, if a close obstacle is sensed (i. e., $\exists\, h \in H : \; h > 100$), 0 otherwise. Initially, the fitness value of every robot is set to 0.

```
if *U expired* then  // Add snapshot to fitness.
  fitness += (1 - NOT_MOVING - OBST_NEAR);
end if
if *Collision* then fitness -= 3; end if
if *V expired* then fitness /= 2; end if
```

**Alg. 1**: Fitness assignment and update.

## 3 Experiments

When doing evolutionary computation, it is usually required to adjust a set of parameters, before good results can be achieved. We made experiments to adjust the four parameters mutation interval $S$, reproduction interval $T$, fitness snapshot interval $U$ and fitness evaporation interval $V$. However, there are more parameters than these four, which have to be studied in future experiments (e. g., number of robots, size of field, and more complex parameters like the mutation and crossover operators).

Collision Avoidance as target behavior has been used, because it is a simple and analyzable behavior. However, as it was not a priori clear if this is even evolvable with a model based on finite Moore automata and how it would work, we present also two of the resulting automata.

Since there was a large number of evolved robots (in total 21 060 robots in 810 simulations), it was not possible to look at all results in detail. Instead, we checked only those automata, which finally achieved a positive fitness value. To justify this, we separately tested automata with fitness $> 0$ (A) and those with fitness $\leq 0$ (B), whether they had a reachable "Move"-state (since otherwise, they could not move and, therefore, especially did not perform Collision Avoidance). It turned out that 91% of the automata in (A) had a reachable "Move"-state, but only 28% of those in (B) did. Also, the observation of random samples convinced us that zero is a reasonable fitness threshold to distinguish between "good" and "bad" Collision Avoidance behavior.

**Adjusting the parameters.** As listed in Tab. 1, $3^4 = 81$ different parameter combinations have been used, setting each of the four parameters to three constant values. Each of these experiments was repeated 10 times. The setting was a rectangular field, $60 \times 80cm$, where 26 robots were placed randomly (based on uniform distribution in position and angle). Their initial automaton was completely empty, i. e., had no states. Simulation was performed for about 2000000 simulation steps (a robot would drive about 8 $km$ in one experiment if only repeating the Move-instruction).

Fig. 2 shows the number of "successful" experiments, i. e., experiments in which there existed robots with a positive fitness in the end, distributed on the 81 different parameter combinations. On average, there were about

**Table 1** Parameter values used in the experiments.

| | Mutation Int. *S* | Reproduction Int. *T* | Snapshot Int. *U* | Evaporation Int. *V* |
|---|---|---|---|---|
| 1. | 5000 ms | 1000 ms | 250 ms | 10000 ms |
| 2. | 10000 ms | 2000 ms | 500 ms | 20000 ms |
| 3. | 20000 ms | 10000 ms | 1000 ms | 30000 ms |

7 robots with a positive fitness in the final populations of successful experiments (see Sec. 4 for a discussion on selective pressure).

As Tab. 2 shows, some of the results indicate a tendency, in which direction parameters should be shifted.

**Table 2** Distribution of successful experiments for each parameter separately.

| | Mutation Int. *S* | Reproduction Int. *T* | Snapshot Int. *U* | Evaporation Int. *V* |
|---|---|---|---|---|
| 1. | 5000 ms: 18% | 1000 ms: 37% | 250 ms: 37% | 10000 ms: 31% |
| 2. | 10000 ms: 29% | 2000 ms: 29% | 500 ms: 36% | 20000 ms: 30% |
| 3. | 20000 ms: 53% | 10000 ms: 34% | 1000 ms: 27% | 30000 ms: 38% |

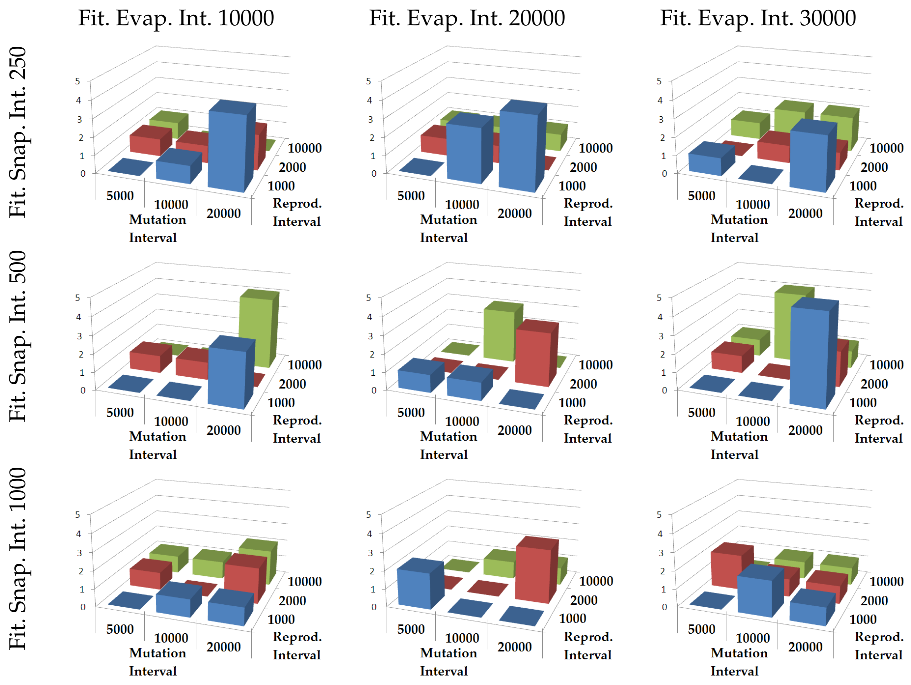

**Fig. 2** Distribution of successful experiments (i. e., number of experiments out of 10 repetitions, where at least one robot had a positive fitness – for the 81 parameter combinations).

The number of successful experiments increases with a larger *mutation interval S*. In particular, a mutation interval of 5000 *ms* almost never yielded successful experiments (18 %). So, looking at all parameter combinations, it seems to be reasonable to increase the mutation interval even more.

For the *reproduction interval T*, no clear statements can be derived. Apparently, the values between 1000 *ms* and 10000 *ms* have to be studied more precisely.

For the *fitness snapshot interval U*, it seems as if the value should be decreased. However, the differences are smaller than at parameter *S*.

For the *fitness evaporation interval V*, higher values seem to be better.

However, due to dependencies between the parameters, it may not be possible to find the perfect parameter combination by only optimizing each parameter separately. Rather than doing that, we will continue to perform experiments with a large set of parameter combinations to learn more about these dependencies. We will use these results, however, to draw conclusions about the directions, in which the search should be extended.

**The evolution of Collision Avoidance.** In our experiments, 545 robots (2.6%) had a positive final fitness. Some of these achieved the expected Collision Avoidance behavior, i. e., moving (arbitrarily) until an obstacle appears, then turning until the way is clear, then moving again; some did some other forms like driving in circles or ellipses. However, it is often hard to characterize a behavior accurately, since it could depend on circumstances, which are hard to understand; e. g., there were robots which avoided obstacles only when a specific constellation of sensor values from the sensors in the back was received. Therefore, we characterize a robot's behavior only by its fitness value; in future the fitness function should be refined to avoid unwanted behaviors.

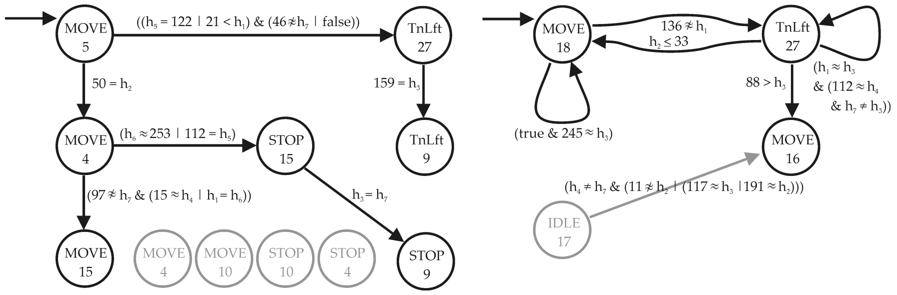

**Fig. 3** Two examples of evolved automata – performing the expected Collision Avoidance behavior (left, fitness: 730); driving a circle (right, fitness: 132); unreachable states set gray.

Fig. 3 shows two automata that evolved during the experiments. The left one lets the robot essentially drive straight forward, except when sensing an obstacle: then it turns left. The right one is driving a circle – without sensing obstacles at all. However, due to the implementation of crash simulation, it turns out that a circle-driving robot finds, after a few initial collisions, enough space to drive with nearly no colliding and, therefore, is successful in the sense of the fitness function.

## 4 Discussion

**Conclusion.** The results presented in this paper show that the onboard approach of online evolution of robot behavior based on a Moore Automaton control works in principle. A set of four parameters has been selected quite arbitrarily and set to some almost arbitrary values to study their influence on the quality of the resulting behavior with respect to the fitness function. Even within this random selection of the parameter search space, a parameter combination has been found, which in 5 out of 10 samples yielded successful experiments (and others did so in 3 or 4 out of 10 trials). It can be expected that even better parameters can be found by extending the search in the directions indicated by the results of the experiments and by modifying other parameters, which have not been considered in this paper.

However, the experiments also uncovered some problems, which have to be considered in future work. The process of reproduction, which is also the mechanism of selection, is based on the assumption that the robots are moving around at least now and then. If too many are standing idle, the rule of producing offspring always with the closest neighbor determines an incestuous exchange of genomes, always with the same partner. Since we argued that most of the robots had a negative fitness in the end and that of their automata only about 28 % even had a reachable Move-state, we can expect that this assumption was not fulfilled sufficiently in many populations. Since a similar problem arose in experiments with real robots, the movement of robots during evolution has to be studied more carefully and probably a new mechanism of reproduction has to be developed. Maybe using a flexile mutation interval (e. g., mutating robots with lower fitness more often), rather than a constant one as we did so far, could also help avoiding this. A second problem was the fitness function, which induced unexpected solutions like driving circles without sensing obstacles. Though these solutions were successful in the sense of the fitness function, it was not the intended behavior. Obviously, the design of a fitness function for a target behavior has to be studied in correlation with the expected and the actual outcome.

Another issue is the measure of diversity (as defined in [8]) in the population during the experiments, which could give a more detailed insight into the evolutionary process. Besides inherent problems of the diversity measure, in this case, it is not even clear, how to measure the difference between two single individuals. As proposed in [9], this difference could be measured on the phenotypic level through the reaction of individuals to stimuli (i. e., some kind of fitness function, again). However, the fitness function is delayed and dependent on the environment, so a more precise measure on the genotypic level should be found. For two automata $A$ and $A'$ such a measure could, e. g., be based on the number of input sequences (i. e., sequences of sensor combinations), where the corresponding output sequences produced by $A$ and $A'$ differ.

So far, we considered fitness values and genomes at the end of experiments, only. However, fitness, genome, and movement of each robot are recorded during the experiments. A careful analysis of this data should allow to gain more information about selective pressure, the benefits of a flexible mutation interval, the problem with incestuous exchange of genomes and it could help to get a better idea of an appropriate evolving time.

**Outlook to future work.** More experiments with a larger variety of parameter combinations have to be performed and the statistical significance of the results has to be checked. Especially studying the mutation interval seems to be promising for achieving better results. A recombination operator which generates offspring as a mixture of the genomes of both parents has already been developed, but still has to be implemented and tested. Also, we are planning to make experiments with a flexible mutation interval. A measure for diversity on the genotypic level is being developed. The development of the population during evolutionary runs is going to be studied in detail.

## References

1. Bonabeau, E.; Theraulaz, G.; Dorigo, M.: Swarm Intelligence. From Natural to Artificial Systems. Santa Fe Institute Studies in the Sciences of Complexity, Oxford University Press, 1999
2. Kube, C. R.; Zhang, H.: Collective Robotics: From Social Insects to Robots. Adaptive Behavior, 2, 2, 189-218, 1993
3. Goss, S.; Aron, S.; Deneubourg, J. L.; Pasteels, J. M.: Self-organized shortcuts in the argentine ant. Naturwissenschaften, 76, 579-581, 1989
4. Branke, J.; Schmeck, H.: Evolutionary design of emergent behavior. In ”Organic Computing”, Series: Understanding Complex Systems. Wrtz, Rolf P. (Ed.), Springer Verlag, 123-140, 2008
5. König, L.; Jebens, K.; Kernbach, S.; Levi P.: Stability of on-line and on-board evolving of adaptive collective behavior. Euros, Prague (accepted), 2008
6. Hopcroft, J. E.; Ullman, J. D.: Introduction to Automata Theory, Languages, and Computation. Addison-Wesley, 1979
7. Schmeck, H.; Branke, J.; Kohlmorgen, U.: Parallel implementations of evolutionary algorithms. In: Solutions to Parallel and Distributed Computing Problems, Zomaya, A.; Ercal, F.; Olariu, S. (Ed.). John Wiley, New York, Wiley Series on Parallel an Distributed Computing, 47-66, 2001
8. Nehring, K.; Puppe, C.: A Theory of Diversity, *Econometrica*, 70, 1155-1198, 2002
9. Curran, D.; O’Riordan, C.: Increasing population diversity through cultural learning, *Adaptive Behavior*, 14, 4, 2006