When to Apply the Fifth Commandment: The Effects of Parenting on Genetic and Learning Agents

Michael Berger Jeffrey S. Rosenschein
School of Engineering and Computer Science
Hebrew University
Hebrew University, School of Engineering and Computer Science,
Givat Ram, Jerusalem 91904 Israel
Telephone: ++972-2-658-5353
Fax: ++972-2-658-5439
{mberger,jeff}@cs.huji.ac.il
(*Received 00 Month 200x; In final form 00 Month 200x*)

This article explores hybrid agents that use a variety of techniques to improve their performance in an environment over time. We considered, specifically, genetic-learning-parenting hybrid agents, which used a combination of a genetic algorithm, a learning algorithm (in our case, reinforcement learning), and a parenting algorithm, to modify their activity. We experimentally examined what constitutes the best combination of weights over these three algorithms, as a function of the environment's rate of change.

For stationary environments, a genetic-parenting combination proved best, with genetics being given the most weight. For environments with low rates of change, genetic-learning-parenting hybrids were best, with learning having the most weight, and parenting having at least as much weight as genetics. For environments with high rates of change, pure learning agents proved best. A pure parenting algorithm operated extremely poorly in all settings.

Keywords:
Parenting; Genetic Algorithms; Learning; Q-learning; Monte Carlo

1 Introduction

1.1 *Genetic and Learning Algorithms*

Among the most prominent types of heuristic algorithms used in dealing with NP-hard problems are genetic algorithms. Genetic algorithms operate by representing every possible solution to a given problem as a series of "genes". Multiple agents, each containing its own series of genes, start working to solve the problem. As in biological evolution, these genes are developed over many generations by a repeated process of genetic crossovers, mutations, evaluations, and "survival of the fittest". Over time, the developed genes contain information that is gathered over these multiple generations, and hopefully, they eventually provide an optimal or near-optimal solution to the problem.

The main disadvantage of genetic algorithms is that they are mainly suitable for dealing with stationary problems. Problems, however, may be of a dynamic nature, i.e., have a hidden parameter, or state, that occasionally changes. In such dynamic situations, when a change occurs in the problem's state, genetic algorithms tend to collapse and start over again, searching for a new optimal solution, often without much success (Cobb and Grefenstette, 1993).

A modification that can help genetic algorithms handle dynamic problems is to combine them with a learning algorithm. Such an approach has been used in developing both agents that handle stationary environments (Hinton and Nowlan, 1996; Nolfi et al., 1994; Parisi and Nolfi, 1996) and agents that handle dynamic environments (Littman, 1996; Littmann and Ackley, 1991; Nolfi and Floreano, 1999; Nolfi and Parisi, 1997; Todd and Miller, 1991).

For more precise interpretation of the term "learning", we need to first examine the term *memory of evolution*. Memory of evolution simply refers to memory that can be directly transferred from an individual in one generation to an individual in the subsequent generation. Note that genetic information is not included in memory of evolution, as the genetic information contains only implicit instructions and not a history of events. Thus, an assumption about the presence or absence of memory of evolution is crucial in the interpretation of the term "learning". If

memory of evolution is assumed to be present, any learning process started at one generation can be continued in the next. Otherwise, any learning process started at one generation must come to an end at that same generation. As the absence of memory of evolution correctly describes natural evolution, and as natural evolution has been the dominant paradigm for artificial evolution, any further interpretation of the term "learning" in this article should assume the absence of memory of evolution (Sebag et al., 1997).

Genetic algorithms and learning algorithms are inherently different, in that genetic algorithms gather information over many generations, while learning algorithms gather information only over a single generation. Herein lies the flexibility of learning—as long as genetic effects are neutralized, learning is unbiased by information gathered in previous generations, and is therefore more suitable for dealing with a dynamic problem, where information from previous generations might not be relevant. On the other hand, a learning process might actually be missing *some* important information from previous generations, causing it to be more wasteful in relearning information, and ultimately less accurate (Mayley, 1996).

1.2 *Parenting Algorithms*

Let us informally define a current decision's level of *substantiation* to be determined by the amount of information gathered as a basis for reaching that decision. Let us also informally define a current decision's level of *relevance* to be determined by the level of similarity between the current environment and the environment at the time the information was gathered. Let us also say that an algorithm's levels of substantiation and relevance are determined by the levels of substantiation and relevance attained by the decisions that are taken when using that algorithm.

In the discussion above of algorithms that deal with dynamic problems, two extremes have been presented: a genetic algorithm, which has a high level of substantiation but a low level of relevance, and a learning algorithm, which has a low level of substantiation but a high level of relevance. One might consider how algorithms that are "in-between" might fare—specifically, for a problem with the following two conditions:

- ***Condition C1:*** The problem state can only change to a state adjacent to it in the state space, with adjacency defined by some metric over that space.
- ***Condition C2:*** The problem has a very low (but positive) rate of dynamic change.

For such problems, it seems that the an in-between algorithm might have a higher level of substantiation than learning, while still preserving an adequate level of relevance, because *C1* and *C2* combined imply that a current problem state is somewhat similar to problem states that were encountered over the last few generations.

Thus, we introduce the notion of parenting. We define a *parenting algorithm* as one in which agents follow instructions from their parents, i.e., agents of the previous generation, and the parents determine their instructions according to what they themselves had learned over the *complete* course of their generation—*a posteriori*. In contrast to learning agents, parenting agents do not execute their own actions according to what they learn for themselves.

A parenting algorithm has higher substantiation than a learning algorithm. Thus, it avoids decisions based on momentary experience, which might be more prone to error—especially in early stages (Johnston, 1996). Conversely, a parenting algorithm has lower substantiation than a genetic algorithm, because its information is gathered over the span of a single generation, and not over many generations. Thus, a parenting algorithm is more prone to error than a genetic algorithm.

As for relevance, in a given generation, a parenting algorithm has lower relevance than a learning algorithm, because its information is gathered in the previous generation and not in the current one. Conversely, a parenting algorithm has higher relevance than a genetic algorithm, because its information is unbiased by whatever happened in generations before the previous one.

A parenting algorithm seems to fit the definition of an "in-between" algorithm. Can it help handle a problem that fulfills conditions *C1* and *C2*? This question constitutes one focus of this article.

1.3 *Overview of the Article*

In Section 2, we describe the overall environment of our experiments, and what constituted our goal for the agents. In Section 3, the hybrid agents themselves are presented, along with their genetic, learning, and parenting aspects. Section 4 describes the various experimental runs that were made, while Section 5 presents the results of those

experiments. Section 6 gives an overview of related work. Our conclusions are discussed in Section 7.

2 Environment and Task

2.1 *Environment*

We define a *round* to be the basic time unit. The environment consists of a rectangular grid, with dimensions $w^{Env} \times h^{Env}$. The grid contains agents and food. In every round, each agent can either "eat" or "starve". This environment is generally similar to that used in Nolfi et al. (1994).

In each round, each square on the grid may contain any number of agents, and it may or may not contain unlimited food—unlimited in the sense that when food is present in a square, all agents in that square are said to have eaten (unlike Nolfi et al. (1994)). The reason for making food unlimited is to avoid dependencies between the actions and performance of different agents, thus avoiding the complexity of social interactions. At the end of a round, an agent may travel to an adjacent square or stay put.

The appearance or non-appearance of food is actually controlled by structures in the environment to which agents are oblivious. These are *food patches*. A food patch is a positive probability function P_{Food} defined over a group of adjacent squares on the grid. If s is a square, then $P_{Food}(s)$ defines the probability that in a given round, food will appear in square s. If several food patches, $P_{Food_1} \ldots P_{Food_k}$, have a non-empty domain intersection, then for each square s in that intersection, $P_{Food}(s)$ is defined as $\max_{i=1}^{k} P_{Food_i}(s)$. The environment contains n_{Patch}^{Env} food patches.

A fundamental attribute of the environment is α^{Env}, which is the probability that in a given round, the food patches move (i.e., their domains move). When this occurs, *all* food patches move one square, each in its own random direction (EAST, SOUTH, WEST, NORTH).

One last important property of the environment is that it is cyclic. This is true both for movements of agents and of food patches. This means that a movement from a square residing on the edge in the direction of the edge results in re-appearance on the other side of the grid. The grid was defined as cyclic so that all squares in it would be homogeneous (thus inner squares, edge squares and corner squares all have the same importance).

2.2 *Mapping the Environment to an Abstract Problem*

If we define an abstract problem as that of finding food in an environment that has a pre-determined set of n food patches, each with a pre-determined shape, then a given position of all food patches actually constitutes one state of that problem. For food patch i, let (x_{Food_i}, y_{Food_i}) be any representational point from within it.

A metric can then be defined on the $2n$-dimensional space of these states by defining any vector v in that space, $v \equiv (x_1, y_1, \ldots, x_n, y_n)$, to denote the positions of the representational points of all food patches in that state, i.e., $\forall i \, (x_{Food_i} = x_i \wedge y_{Food_i} = y_i)$. Thus, the environment presented above in Section 2.1 fulfills condition *C1*. For very low positive values of α^{Env}, it fulfills condition *C2* as well.

2.3 *Task*

The goal in our experiments has been to synthesize an agent that enjoys a maximal *eating-rate*—the fraction of rounds in which it eats, out of the total number of rounds. This task definition requires more detail, in two respects: 1) how to integrate agent generations in the definition, and 2) how to measure the success of agents.

An integration of agent generations in the task definition is required since the development of genetic agents and parenting agents does not occur in a single agent's lifetime, but over many lifetimes. Thus, the concept of *generation* is defined: a generation of agents is a group of agents that start their life in the environment simultaneously, and end their life in the environment simultaneously. To avoid having some agents receive unfair advantages over others, all agents of the same generation start at the same position, a position that is randomly determined for each generation.

It is also important to note that the environment is completely oblivious to the concept of generations. This means that the position of food patches is not reset at the start of a new generation—the environment is "continuous" across generations.

When considering how to measure agent success, such measurement must not consist of only the initial generation; agents that develop over multiple generations must be allowed time to develop. Also, the measurement must include a large enough number of generations so as to increase the statistical validity of the results.

Having presented those aspects above that required more detail, a more accurate task can now be formulated. For a given agent generation, let a *BER (best eating-rate)* be defined as the maximal eating-rate of all agents in that generation (a similar metric for success was used in Sebag et al. (1997)). In a run of n_{GenRun}^{Task} agent generations, each consisting of n_{Ag}^{Task} agents that live for n_{Rnd}^{Task} rounds, let the measure of success λ be defined as the average over the BERs of the last $n_{GenTest}^{Task}$ generations. Our goal is then to develop agents that maximize λ.

3 Agents

The agents used in our experiments are hybrids of a genetically-developing agent, a self-learning agent, and a parenting agent. Before defining these agent types, a few definitions are in order.

3.1 *Definitions*

- ***Perception:*** An element of the set:

$$\{0,\ldots,w^{Env}-1\}\times\{0,\ldots,h^{Env}-1\}\times\{0,1\}$$

  Indicates a position (of the agent) on the grid, and whether the agent found food in that position.
- ***Reward:*** That part of the perception which indicates whether the agent found food (reward is 1) or not (reward is 0).
- ***Action:*** A direction of movement (for the agent to move next). One of EAST, SOUTH, WEST, NORTH, HALT.
- ***Memory:*** A sequence containing the agent's last perception-action pairs. A memory of length m will contain the last perception, preceded by the previous $m-1$ perception-action pairs. Note that in principle, there are $\left(2\cdot h^{Env}\cdot w^{Env}\right)^m\cdot 5^{m-1}$ possible memories of length m. However, since movement in our environment is deterministic, the number of possible memories of length m is $\left(2\cdot h^{Env}\cdot w^{Env}\right)\cdot 5^{m-1}$.
- ***MAM:*** Memory-Action Mapper. Receives a memory as input, and returns an action as output. The MAM actually constitutes an agent's policy.
- ***ASF:*** Action-Selection Filter. Receives several actions as input (i.e., action suggestions), and returns one of them as output.

3.2 *Genetic Agents*

A genetic agent is one that selects its actions according to its predetermined gene sequence, and can mate with other genetic agents to produce new offspring that contain their own gene sequences (Axelrod, 1997).

A genetic agent holds a memory of length m^{Gen}. In addition, the genetic agent employs genes, with each gene composed of a *key* and a *value*. The key is a possible memory, and the value is a possible action.

The MAM of a genetic agent is a gene sequence, containing one gene for each possible memory. The position of a gene in the sequence is defined uniquely by its key (i.e., the gene sequences of all genetic agents in the same environment and with the same memory length contain genes with an identical order of keys).

The MAM of a genetic agent is created when the agent is created, and it is never updated afterwards. When the MAM receives a memory as input, it maps it to the returned action by finding the key that matches the memory, and then returning its accompanying value.

Genetic algorithms employ generations of genetic agents, letting each agent run in the environment and then producing a new generation of agents by mating agents from the current generation. Conventionally, the mating stage consists of two parts: *generational selection* and *offspring creation*. Here, only offspring creation is implemented within the genetic agent. As for generational selection, it is implemented elsewhere, and is described in Section 3.5.1.

Figure 1. Genetic Sequence Example ($m^{Gen} = 1$)

(a)

Key	Value
⋮	⋮
(5,7)	NORTH
(5,8)	NORTH
(5,9)	SOUTH
(6,0)	WEST
(6,1)	HALT
(6,2)	WEST
⋮	⋮

Key	Value
⋮	⋮
(5,7)	EAST
(5,8)	NORTH
(5,9)	WEST
(6,0)	HALT
(6,1)	NORTH
(6,2)	SOUTH
⋮	⋮

(b)

Key	Value
⋮	⋮
(5,7)	NORTH
(5,8)	NORTH
(5,9)	WEST
(6,0)	HALT
(6,1)	NORTH
(6,2)	WEST
⋮	⋮

Key	Value
⋮	⋮
(5,7)	EAST
(5,8)	NORTH
(5,9)	SOUTH
(6,0)	WEST
(6,1)	HALT
(6,2)	SOUTH
⋮	⋮

(c)

Key	Value
⋮	⋮
(5,7)	NORTH
(5,8)	**SOUTH**
(5,9)	WEST
(6,0)	HALT
(6,1)	NORTH
(6,2)	**EAST**
⋮	⋮

Key	Value
⋮	⋮
(5,7)	EAST
(5,8)	NORTH
(5,9)	SOUTH
(6,0)	**HALT**
(6,1)	HALT
(6,2)	SOUTH
⋮	⋮

This figure follows the changes that are undergone by a specific segment of a gene sequence. *(a)* shows the segment in the genes of both parents, prior to mating. *(b)* shows the segments after the crossover stage. A double-line marks the places where a crossover occurred. *(c)* shows the segments after the mutation stage. Text in bold denotes positions where a mutation occurred.

An illustration of the mating process of two genetic agents appears in Figure 1. Whenever two genetic agents mate, they create two offspring, the MAMs of which are created as follows. First, a copy is created for each of the gene sequences of the parents. We will denote the sequence copies as S^1 and S^2. A simultaneous scan is then started over S^1 and S^2, gene by gene. Let g_j^i denote the gene of sequence copy S^i at position j. As previously explained, g_j^1 and g_j^2 have the same key. For any j, when the scan reaches position j, the following events may occur:

- ***Crossover:*** This event occurs with probability P_{Cros}^{Gen}. When it occurs, all genes at positions $k \geq j$ are swapped between S^1 and S^2 (i.e., the complete sub-sequences of genes starting at g_j^1 and g_j^2).
- ***Mutation:*** This is an event that occurs with probability P_{Mut}^{Gen} for each of the sequences. When it occurs for sequence S^i, the value of gene g_j^i changes to a randomly selected value (action).

After the scan is complete, the offspring i receives sequence i as the gene sequence in its MAM.

3.3 *Learning Agents*

A learning agent is an agent that selects its actions according to a learning algorithm. The most appropriate type of learning algorithm for this environment is reinforcement learning (Littman, 1996), in which agents receive after every action a signal that informs them how good their choice of action was (here, the presence or absence of food in their position). The reinforcement learning algorithm that was selected for the learning agent was Q-learning with Boltzmann exploration (Sandholm and Crites, 1996).

The learning agent holds a memory of length m^{Lrn}. When the agent's MAM receives memory as input, it maps it to the returned action according to the Q-learning with Boltzmann exploration algorithm.

In the general Q-learning algorithm, an agent tries to maximize rewards that it receives from the environment by improving its action selection. Q-learning's strategy for attaining this goal is to evaluate actions according to the expected discounted sum of future rewards that these actions are likely to yield under the most optimistic assumptions that can be derived from present knowledge.

Let r_j denote the reward given at learning step j. Then at learning step n, the discounted sum of future rewards is defined as $\sum_{i=0}^{\infty} \gamma^i r_{n+i}$, where $0 \leq \gamma < 1$ is the discount factor. γ is a pre-determined parameter, which determines the importance of the reward at the current learning-step, compared to the importance of all future rewards (a value of 0 means that the agent only values its current reward; values approaching 1 mean that the agent attributes more and more importance to future rewards).

The Q-learning algorithm works by estimating the expected discounted sum of future rewards for all possible actions, in light of the current memory. Let s denote a memory, and let a denote an action. Then the value $Q(s, a)$ (termed a *Q-value*) is defined as the expected discounted sum of future rewards obtained by taking action a when the memory contains s, and following an optimal policy thereafter (this is the "most-optimistic assumption" mentioned earlier). In our setting, all Q-values are initialized to 0.5, to prevent bias in the initial selection of actions.

Q-learning uses an additional pre-determined parameter, which is termed the "learning rate". This parameter can receive values between 0 and 1. The learning rate determines how quickly Q-learning converges to final memory-action evaluations—the higher the learning rate, the quicker the convergence.

Formally, the Q-learning algorithm is defined as follows:

(i) Initialize Q-values to arbitrary values.
(ii) From the current memory s, select an action a. This will cause receipt of an immediate payoff r, and arrival at a next memory s'.
(iii) Update $Q(s, a)$ based on this experience as follows:

$$\Delta Q(s,a) = \alpha^{Lrn} \left[r + \gamma^{Lrn} \max_b Q(s', b) - Q(s,a) \right]$$

where α^{Lrn} is the learning rate and γ^{Lrn} is the discount factor.
(iv) Go to 2.

An illustration of the Q-table use and update appears in Figure 2.

For a current memory s, what action should be selected next? One trivial answer is to select the action with the highest Q-value for s, thus "exploiting" the Q-value. However, according to Sandholm and Crites (1996) (and, in another context, Carmel and Markovitch (1999)), the algorithm operates better if an element of exploration is added to action-selection, thus improving the comparison that a Q-learning agent makes among different actions.

In Sandholm and Crites (1996), a Boltzmann exploration method was selected. According to this method, every possible action a_i has the following probability to be selected:

$$p(a_i) = \frac{e^{\frac{Q(s,a_i)}{t}}}{\sum_a e^{\frac{Q(s,a)}{t}}}$$

where t is a computational parameter that controls the amount of exploration, and in learning step n, $t = f^{Lrn}_{Temp}(n)$. t is an annealing temperature that decreases over time, thus increasing exploitation and decreasing exploration (the action with the highest Q-value receives probabilities that approach 1). Also, a freezing temperature t^{Lrn}_{Freeze} has

Figure 2. Q-Table Example ($m^{Lrn} = 1$)

(a)

Key	NORTH	EAST	SOUTH	WEST	HALT
⋮	⋮	⋮	⋮	⋮	⋮
(4,7)	0.10	**0.20**	0.15	0.13	0.08
⋮	⋮	⋮	⋮	⋮	⋮
(5,7)	0.12	0.33	0.79	0.12	0.30
⋮	⋮	⋮	⋮	⋮	⋮

(b)

Key	NORTH	EAST	SOUTH	WEST	HALT
⋮	⋮	⋮	⋮	⋮	⋮
(4,7)	0.10	**0.60**	0.15	0.13	0.08
⋮	⋮	⋮	⋮	⋮	⋮
(5,7)	0.12	0.33	0.79	0.12	0.30
⋮	⋮	⋮	⋮	⋮	⋮

This figure shows an example of the use and update of a Q-table. Assume that $\alpha^{Lrn} = 0.3$, $\gamma^{Lrn} = 0.8$, and the agent is in position (4,7). *(a)* shows the agent's Q-table before selecting the the next action. *(b)* shows the Q-table after the agent selected action EAST and moved to (5,7) (which would occur for example if there is no exploration, since EAST is the action with the highest Q-value for (4,7)). Assume that the agent's reward for this action was 1. Then the table cell in bold gets updated as follows: $\Delta Q = 0.3 \cdot [1 + 0.8 \cdot 0.79 - 0.2] \approx 0.4$

been defined, so that when $t < t^{Lrn}_{Freeze}$, exploration (which is better for early steps) ceases completely, and full exploitation kicks in.

3.4 *Parenting Agents*

A parenting agent is an agent that differentiates between *learning* experiences from the environment, and *performing actions* based on that experience. In fact, when a parenting agent decides on an action to perform, its experience has absolutely no weight in that decision. Instead, the agent turns to another parenting agent, its "parent", and seeks advice about what action to perform. The parent, in turn, uses its own experience to return an answer. Note that the parent is an agent that interacted with the environment in the previous generation, but no longer does so.

A group of methods that seems appropriate for such parenting agents is Monte Carlo methods, or MC-methods for short (Sutton and Barto, 1998). MC-methods are used for evaluating and improving policies, i.e., memory-action mappings. In contrast to Q-learning, with MC-methods the evaluation and improvement are performed after complete *episodes* of rounds, and not after every single round. In the current context, an episode is simply a generation. This was the reason for preferring MC-methods over Q-learning for the parenting algorithm—MC-methods are more suitable for creating policies that take into account several dynamic states of the environment in the course of a generation, thus avoiding convergence of policy according to a single state at the end of that generation. This allows a bird's-eye view of what were *on average* the "correct" actions during the previous generation, lessening the chance for errors, which is a characteristic that we would like the parenting algorithm to have.

In the evaluation stage, the mapping of each possible memory to each possible action is evaluated. The evaluation produces a value, termed an *MC-value*. Let s be a memory and a an action. The MC-value of (s, a) is the average of rewards in all rounds following the agent's encounter of memory s and selection of a as the next action. In our setting, all MC-values are initialized to 0.5, to prevent bias in the initial relative evaluation of actions.

The MC-value definition above leaves one open question: What happens if the agent encountered memory s and responded with action a more than once? MC-methods typically use one of two approaches:

(i) Set the MC-value to the average of rewards received in all rounds following the agent's first encounter of (s, a). This is termed a *first-visit* MC-method.

(ii) For every encounter of the agent with (s, a), calculate the average of rewards received in all rounds following that encounter. Then, set the MC-value to the average of these averages. This is termed an *every-visit* MC-method.

The advantage of a first-visit MC-method is that the computation of MC-values is based on the largest possible sample of data. However, we chose to use an every-visit MC-method, because in our case, the environment may be dynamic; specifically, the environment may change significantly between the parent's first encounter with (s, a) and the end of the parent's generation. An illustration of the evaluation stage for parenting agents appears in Figure 3.

Figure 3. Parenting Agent's Evaluation Example ($m^{Par} = 1$)

(4,7)	(4,8)	(4,7)	(4,6)	(4,7)	(5,7)	(4,7)	(3,7)	(4,7)	(4,7)	(4,8)
(S,1)	(N,0)	(N,0)	(S,0)	(E,1)	(W,1)	(W,0)	(E,1)	(H,0)	(S,1)	

(4,7)	(3,7)	(4,7)	(4,8)	(4,7)	(4,6)	(4,7)	(5,7)	(4,7)	(4,7)	(3,7)
(W,0)	(E,0)	(S,1)	(N,0)	(N,1)	(S,0)	(E,0)	(W,1)	(H,1)	(W,1)	

(a)

Key	NORTH	EAST	SOUTH	WEST	HALT
⋮	⋮	⋮	⋮	⋮	⋮
(4,7)	0.5	0.667	**0.75**	0.5	0.5
⋮	⋮	⋮	⋮	⋮	⋮

Key	NORTH	EAST	SOUTH	WEST	HALT
⋮	⋮	⋮	⋮	⋮	⋮
(4,7)	0.667	0.75	0.625	**0.75**	1
⋮	⋮	⋮	⋮	⋮	⋮

(b)

This figure shows an example of the use of an MC-table by parents. *(a)* shows the history of two parents. The top row is the parent's position, and the bottom row is the resultant action and reward received from that action. *(b)* shows the calculated MC-tables as a result of each parent's history (respectively). Cells in bold denote memories that occurred more than once (and hence their MC-values were calculated according to the every-visit principle). If these parents had a common offspring that is located at (4,7), the first parent would suggest it move SOUTH, while the second parent would suggest it HALT.

In the improvement stage, the parenting agent stores in its MAM a mapping of memories to actions, to be used only by the agent's offspring. Any possible memory s is mapped to an action a^* by simply selecting $a^* = \arg\max_a MC(s, a)$. Since this mapping is computed from scratch, without taking into account the mapping of the parenting agent's parent, this method is termed an *off-line* MC-method.

In addition to the MAM, the parenting agent holds a memory of length m^{Par}. Also, the parenting agent has an ASF component.

So far, the functions of parents and offspring have been discussed, without mentioning the number of parents that each offspring has. In this experiment, we used one of nature's models—every offspring has two parents. If they both give the offspring advice, which advice does it take? In principle, this is controlled by the ASF component. In our experiments, each of a given offspring's parents has an equal chance for its advice to be accepted by the offspring.

One important point is the exploration issue. As noted in Sutton and Barto (1998), MC-methods usually require exploration for them to be effective. Otherwise, deterministic policies lead to the exploration of only a single action for every memory. However, the intention here was purposely to develop a "pure" parenting function, where the parent always gives advice that it deems the best—i.e., the most exploitive one. In contrast to learning, here exploration is not essential, because a parenting agent does not start its life with zero-experience—it has its parents'

experience.

That said, there is still some degree of non-determinism that slips in through the back door, in three spots:

(i) A parenting agent's ASF does not always select the same parent.
(ii) A given memory could have more than one action with a maximal MC-value (in which case, an action is chosen for advice by selecting randomly from among these actions).
(iii) A complex agent might not always use the parenting agent to execute an action (see Section 3.5).

In concluding this section, it is important to re-iterate the most important difference between learning agents and parenting agents: Learning agents select actions on-the-fly, according to their own experience. Parenting agents select actions according to their parents' experience, with complete information about the eventual rewards that their parents received after selecting their actions.

3.5 *Complex Agents and Processes*

As mentioned above, the agents that are situated in our environment are actually hybrids of genetic, learning, and parenting agents. The agent types mentioned in Sections 3.2, 3.3 and 3.4 are not situated directly in the environment. Rather, the agent population is made up of agents that are termed *complex agents*. Each complex agent contains within it an instance of each of the mentioned types (genetic, learning, and parenting), in a subsumption architecture (Brooks, 1986). As such, it can be said that each complex agent contains three *inner agents*—one genetic agent, one learning agent, and one parenting agent. The complex agent mediates between the genetic, learning, and parenting agents, on the one hand, and the environment, on the other. The complete processes of mating, receiving a perception, and executing an action are detailed in the following subsections.

3.5.1 ***Mating.*** At the end of a generation's run, the performance of each complex agent is evaluated by its eating-rate. The evaluations' average m and standard deviation d are calculated, and the agents are classified in strata of width d around m. Each agent is then awarded mating rights, determined by its stratum. Mating rights specify the number of matings to which the agent is entitled. This is the generational selection part of the genetic algorithm, which was mentioned in Section 3.2.

Specifically, mating rights are awarded as follows (Axelrod, 1997): first, all agents that have an eating-rate of at least $m + d$ receive two mating rights. Afterwards, the strata are passed in descending order, and as long as the sum of awarded mating rights is smaller than n_{Ag}^{Task}, all agents in the next lower stratum receive one mating right. Finally, mating pairs are determined randomly, while assuring that no agent participates in more matings than entitled to, and that the total number of offspring is equal to n_{Ag}^{Task}. Note that this method preserves, to a high degree, the following principles:

- ***Survival of the fittest.*** Agents that exhibit good performance (compared to others) receive a mating right, and those with exceptional performance receive an extra mating right.
- ***Genetic variance.*** It is not the case that only those agents with exceptional performance receive mating rights. Thus, we avoid putting all of our eggs into one basket.

The mating process of two individual agents is depicted in Figure 4. Two complex agents mate and create two offspring by mating their respective inner genetic agents, as well as their respective inner parenting agents, and then encapsulating the inner offspring in new complex agent offspring (in the figure, only one of the new offspring is shown, for the sake of clarity). There are three points that should be noted:

(i) The inner genetic offspring receives genetic information from its parents during mating, while the inner parenting offspring does not.
(ii) Let function $g(c)$ mark the inner genetic agent of complex agent c. Let function $p(c)$ mark the inner parenting agent of complex agent c. Let functions $P_1(a)$ and $P_2(a)$ mark the first and second parent of any agent a. Let function $C(a)$ mark the complex agent containing inner agent a. Then for every complex agent c:

$$\left(P_1\left(p\left(c\right)\right), P_2\left(p\left(c\right)\right)\right) = \left(p\left(C\left(P_1\left(g\left(c\right)\right)\right)\right), p\left(C\left(P_2\left(g\left(c\right)\right)\right)\right)\right)$$

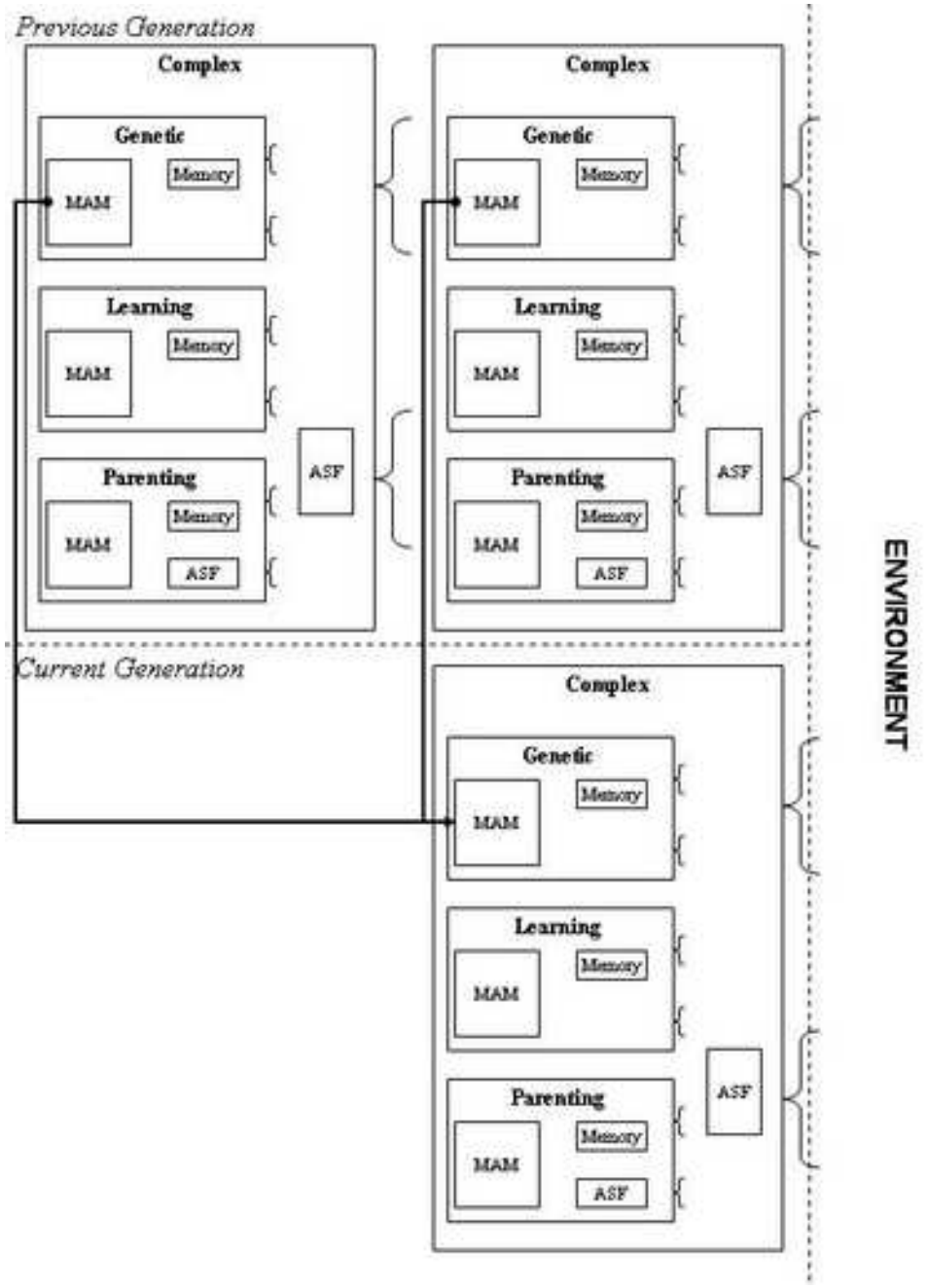


Figure 4. Mating Flow

In other words, in any complex offspring, the parents of its inner parenting agents are those associated with the parents of the inner genetic agents—i.e., the inner parenting agents take advice from their real parents, and not from foster parents.

(iii) The inner learning agents of the complex offspring have absolutely no association with the inner learning agents of the complex parents.

3.5.2 ***Receiving a Perception.*** The process of receiving a perception is depicted in Figure 5. When the complex agent receives a perception, it passes it on to the inner genetic, learning, and parenting agents. The genetic agent updates its memory accordingly. The learning agent updates its memory and its MAM. The parenting agent updates its memory and its MAM (this is equivalent to updating the MAM only at the generation's end, because the MAM is not used during the current generation).

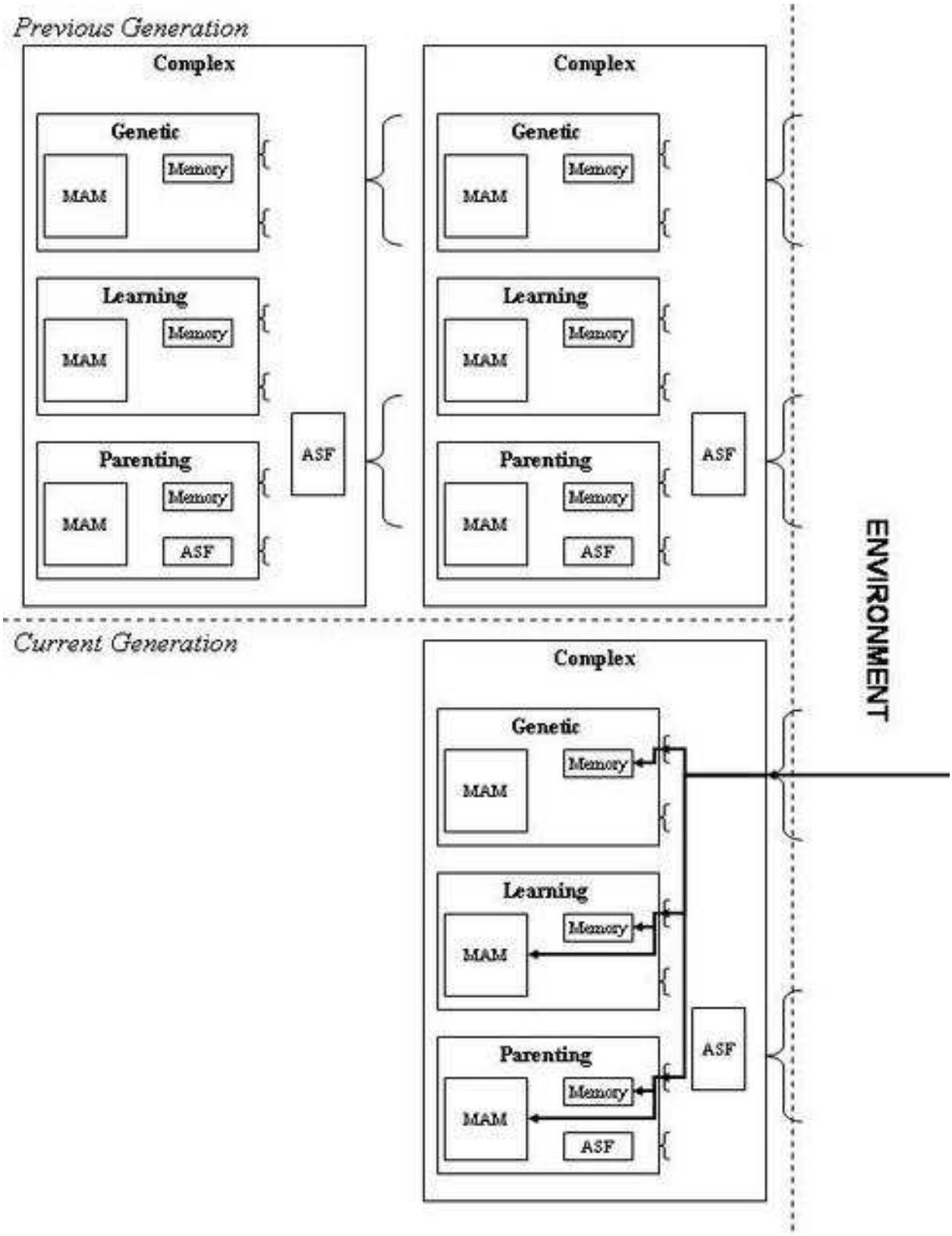


Figure 5. Perception Flow

Note that the parents from the previous generation are completely unaware of the perception—they do not learn any new information.

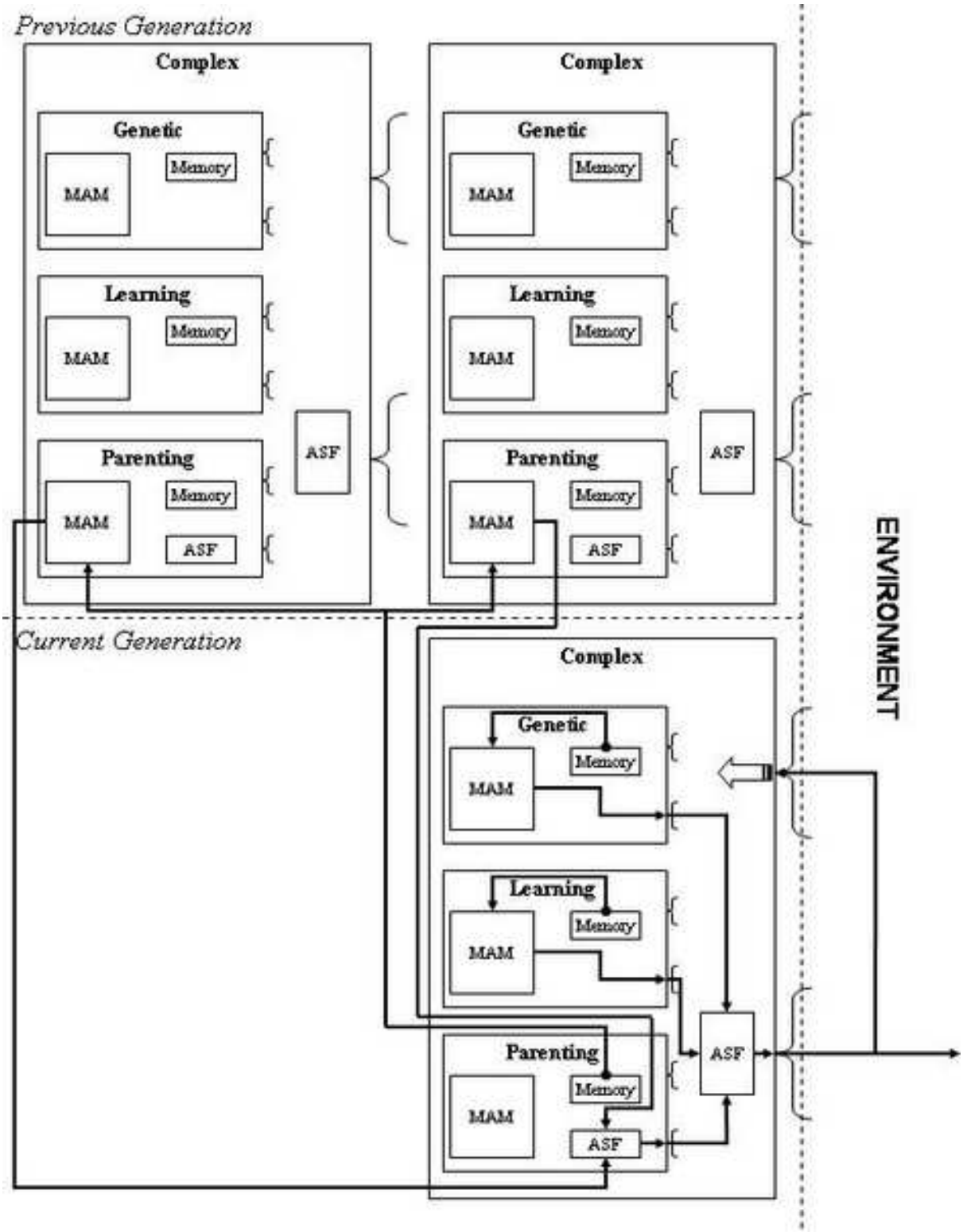


Figure 6. Action Flow

3.5.3 ***Executing an Action.*** The flow of the action execution process is depicted in Figure 6. When the complex agent is about to execute an action, it asks its inner agents to suggest it.

The genetic agent feeds its memory to its MAM, and suggests the returned action as the action to execute.

The learning agent feeds its memory to its MAM, and suggests the returned action as the action to execute.

The parenting agent feeds its memory as input to each of its parents' MAMs. The returned actions are fed through the parenting agent's ASF, and the parenting agent suggests the action selected by its ASF as the action to execute.

The actions suggested by the inner agents are fed to the complex agent's ASF, which selects the genetic agent's suggestion with probability P_{Gen}^{Comp}, the learning agent's suggestion with probability P_{Lrn}^{Comp}, and the parenting agent's suggestion with probability P_{Par}^{Comp}. The complex agent executes the action selected by its ASF, and also perceives it by causing the inner agents to perceive it in the usual manner (see above, Section 3.5.2).

Note that the flow of perceptions and actions in the complex agent is a general one, and it could be used to combine any number of given algorithms (as long as each algorithm has an inner agent that executes it). Any algorithms used will update their own internal structures correctly, but they are not guaranteed to have their own suggested action executed in every round. The effect of this fact on different algorithms may vary.

4 Runs

Sections 2 and 3 specified all the different parameters that determine the behavior of the agent and its environment. This section specifies which parameters were set to constant values, which were treated as independent variables, which constituted the dependent variables, and finally summarizes the number of runs performed.

4.1 *Constants*

4.1.1 *Task.*

- $n_{Ag}^{Task} = 20$.
  This value has been selected because it was used by the genetic algorithm implementation in Axelrod (1997), and seems to provide a sufficient genetic base.
- $n_{GenRun}^{Task} = 9500$.
  This value has been determined experimentally to allow for enough generations for a genetic algorithm to be effective when $\alpha^{Env} = 0$, and to also allow for a sufficient number of generations for testing.
- $n_{GenTest}^{Task} = 1000$.
  This value has been determined experimentally to smooth out the measured results and to prevent flukes from having a noticeable effect.
- $n_{Rnd}^{Task} = 30000$.
  This value has been determined experimentally to allow for enough rounds for a Q-learning algorithm to be effective.

4.1.2 *Environment.*

- $h^{Env} = 20$.
  $w^{Env} = 20$.
  These values have been selected to allow for a large enough grid so that search for food patches would not be trivial, and to allow for a large enough set of possible perceptions.
- $n_{Patch}^{Env} = 1$.
  This value has been selected to allow an easier analysis of the results.
  The single food patch is defined in relation to a central point $c_{Patch} = (x_{c_{Patch}}, y_{c_{Patch}})$. For a point (x, y), let $d_{(x,y)}^{c}$ be defined as :
  $\max\left(\min\left(|x - x_{c_{Patch}}|, \left|x - x_{c_{Patch}} - w^{Env}\right|\right), \min\left(|y - y_{c_{Patch}}|, \left|y - y_{c_{Patch}} - h^{Env}\right|\right)\right)$.
  $d_{(x,y)}^{c}$ denotes the larger between the horizontal distance between c_{Patch} and (x, y) and the vertical distance

between c_{Patch} and (x, y), taking into account that the environment is cyclic. Then, the single food patch function is defined as follows:

$$P^{Env}_{Food_1}(x, y) = 0.2 \cdot \left\lfloor 2^{2-d^c_{(x,y)}} \right\rfloor$$

In effect, this describes a 5x5 food patch. Its peak is at the center with a value of 0.8. All squares in the inner frame have a value of 0.4. All squares in the outer frame have a value of 0.2.

4.1.3 *Genetic agents.*

- $m^{Gen} = 1$.
  This value has been selected experimentally, because higher values increase the number of possible memories, and hence the number of genes, causing the genetic algorithm to have a much slower convergence rate when $\alpha^{Env} = 0$.
- $P^{Gen}_{Cros} = 0.02$.
  $P^{Gen}_{Mut} = 0.005$.
  These values were selected in the general vicinity of values from Axelrod (1997) and Cobb and Grefenstette (1993), and they have been fine-tuned experimentally.

4.1.4 *Learning agents.*

- $m^{Lrn} = 1$.
  This value has been selected experimentally, because higher values increase the number of possible memories, causing the Q-learning algorithm to "spread out" too much by updating each Q-value less often.
- $\alpha^{Lrn} = 0.2$.
  $\gamma^{Lrn} = 0.95$.
  $f^{Lrn}_{Temp}(n) = 5 \cdot 0.999^n$.
  These values have been selected because they were used by the Q-learning algorithm implementation in Sandholm and Crites (1996).
- $t^{Lrn}_{Freeze} = 0.2$.
  This value has been selected in light of the selected values for n^{Task}_{Rnd} and f^{Lrn}_{Temp}, so that exploration would stop after a small but significant fraction of rounds (3910 rounds, or approximately the first 13% of rounds in every generation).

4.1.5 *Parenting agents.*

- $m^{Par} = 1$.
  This value has been selected experimentally, because higher values increase the number of possible memories, causing the Monte Carlo algorithm to "spread out" too much by updating each MC-value less often.

4.2 *Independent Variables*

The parameters that constituted independent variables were as follows: α^{Env}, P^{Comp}_{Gen}, P^{Comp}_{Lrn}, P^{Comp}_{Par}. For shorter notation, (G, L, P) will sometimes be used instead of $(P^{Comp}_{Gen}, P^{Comp}_{Lrn}, P^{Comp}_{Par})$; the two are synonymous.

4.3 *Dependent Variables*

The only dependent variable is λ.

4.4 *Summary of Runs*

For a given α^{Env}, all values of (G, L, P) that fulfill the following conditions were tested:

(i) $P_{Gen}^{Comp} \geq 0$, $P_{Lrn}^{Comp} \geq 0$, $P_{Par}^{Comp} \geq 0$.
(ii) $P_{Gen}^{Comp} + P_{Lrn}^{Comp} + P_{Par}^{Comp} = 1$.
(iii) $P_{Lrn}^{Comp} = 0.1 \cdot n$,
$P_{Gen}^{Comp} = 0.1 \cdot k \cdot \left(1 - P_{Lrn}^{Comp}\right); \quad n, k \in N.$

Under the above conditions, there are 11 different values that P_{Lrn}^{Comp} can receive (i.e., $0, 0.1, \ldots, 1$). For each such value, there are 11 values that P_{Gen}^{Comp} can receive, except for $P_{Lrn}^{Comp} = 1$, where P_{Gen}^{Comp} can receive only one value (0). Therefore, for a given α^{Env}, there are 111 values of (G, L, P) that fulfill the above conditions.

There were 7 different values of α^{Env} that were tested: $0, 10^{-6}, 10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}$. Since each combination of α^{Env} and (G, L, P) was tested (in one run), this amounted to a total of 777 runs.

5 Results

This section details the performance of agents by analyzing the value of λ and the eating-rates in general.

5.1 *Scale of λ*

First, it is worth noting the scale by which a value of λ should be evaluated. The food patch had a peak of 0.8. Thus, an agent that can direct itself to that peak would be expected to have an eating-rate of about 0.8. Therefore, an agent could be considered "perfect" if it has an eating-rate of 0.8. Even though a higher value is possible, because the food patch is a probabilistic function, for the purpose of defining a scale this fact is negligible.

5.2 *Main Results*

Table 1. Result Summary of Runs

	(G, L, P)				
α^{Env}	(1,0,0)	(0,1,0)	(0,0,1)	Best (G, L, P)	Best λ
0	0.2425	0.7233	0.0746	$(0.7, 0, 0.3)$	0.7988
10^{-6}	0.2139	0.7188	0.0730	$(0.2, 0.6, 0.2)$	0.7528
10^{-5}	0.1806	0.6912	0.0670	$(0, 0.9, 0.1)$	0.7011
10^{-4}	0.1550	0.5454	0.0520	$(0.03, 0.9, 0.07)$	0.6021
10^{-3}	0.1462	0.3252	0.0291	$(0.02, 0.8, 0.18)$	0.3647
10^{-2}	0.0805	0.1834	0.0167	$(0, 1, 0)$	0.1834
10^{-1}	0.0366	0.0698	0.0177	$(0, 1, 0)$	0.0698

Table 1 summarizes the main results—the λ-value of the pure agent types and the best performer for each value of α^{Env}. Two points are derived from this table:

- Not surprisingly, λ becomes degraded as α^{Env} rises.
- The performance of agents can be classified according to three ranges of α^{Env}: 0, $(0, K]$ and $[K, 10^{-1}]$, where $10^{-3} \leq K \leq 10^{-2}$.

5.3 *Pure Parenting ($P_{Par}^{Comp} = 1$)*

Pure parenting is ineffective. This is evident from Table 1, and though it is true for all tested values of α^{Env}, it is most striking in Figure 8 (parenting probability is implicit in the figure since $P_{Gen}^{Comp} + P_{Lrn}^{Comp} + P_{Par}^{Comp} = 1$).

A probable explanation for this degraded performance is the decision not to include exploration in the parenting agent's algorithm. Indeed, a closer look at the eating-rates of pure parenting agents lends support to this explanation. The eating-rates go through a two-generation cycle. As an example, for $\alpha^{Env} = 0$, the eating-rates of all agents in one generation usually have values between 0.005 and 0.025, while in the next generation almost all eating-rates are 0 (nicknamed a "zero-generation"). This cycle repeats. The zero-generations occasionally have up to four eating-rates higher than 0.05, and there are rare zero-generations with one or two medium or high eating-rates. This behavior is demonstrated for a sample range of generations in Figure 7.

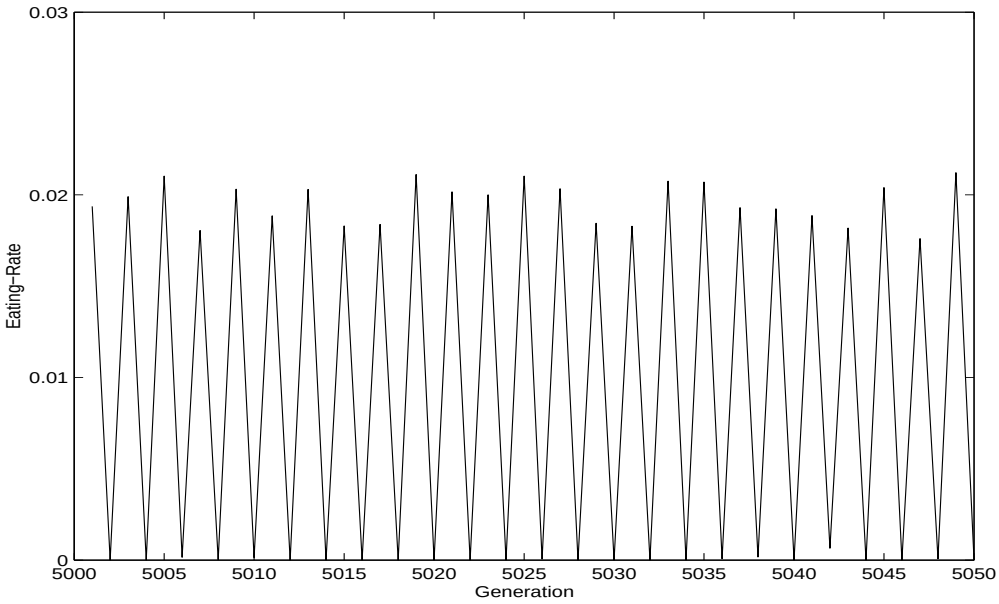


Figure 7. 75^{th} Percentile of Eating-Rates for Pure Parenting Agents ($\alpha^{Env} = 0$)

Lack of exploration could be the culprit. What seems to be happening is that in the non-zero-generation, agents start a search for food, which is beneficial to a certain extent. However, the search is not good enough for the MC-method to consider it a success (since the initial MC-values are 0.5). Therefore, the offspring in the next generation, which is a zero-generation, receive advice from the parents not to go where the parents went, i.e., abandon the parents' search. The obedient offspring do just that, and search in other directions, which are usually barren. Now, the offspring in the next non-zero-generation will receive truly valuable advice, not to go in the barren directions. However, for every memory, the non-zero-generation agents still face three or four other directions in which they can go, so they will be able to start a beneficial search, but only to a certain extent. Hence, the cycle.

5.4 *Runs with* $\alpha^{Env} = 0$

The results for the runs with $\alpha^{Env} = 0$ are summarized in Table 2, and they are represented visually in Figure 8. The points that can be derived from these results are as follows:

- The highest λ-value, 0.7988, is achieved for $(G, L, P) = (0.7, 0, 0.3)$. Taking into account the scale that was mentioned earlier (0.8 being achieved by a perfect agent), this means that the vast majority of generations that eventually evolved for this (G, L, P) have each contained a near-perfect agent—over a span of 1000 generations!
- Surprisingly, pure genetic agents perform poorly, while the hybrids of genetic agents with learning and parenting agents achieve a very high λ. Still, in view of what is known of genetic algorithms, and since the environment is stationary, the pure genetic agent is expected to have eventually achieved a near-perfect λ had it been given enough generations to develop. Indeed, Hinton and Nowlan (1996) demonstrated that "learning" organisms can evolve much faster than their non-"learning" equivalents. And according to Littman (1996), evolution and "learning" complement each other. On the one hand, "learning" may assist evolution by making near-perfect individuals receive high fitness scores, since such individuals adapt, via "learning", to exhibit optimal behavior. On the other hand, evolution accelerates "learning" by making individuals nearly perfect to begin with. This interaction

Table 2. λ for $\alpha^{Env} = 0$

$\frac{P_{Gen}^{Comp}}{1-P_{Lrn}^{Comp}}$	P_{Lrn}^{Comp} 0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
0	0.0746	0.1074	0.2241	0.4866	0.7736	0.7715	0.7626	0.7538	0.7448	0.7373	
0.1	0.0988	0.1032	0.2470	0.7093	0.7792	0.7717	0.7627	0.7531	0.7445	0.7375	
0.2	0.1567	0.1515	0.4812	0.7816	0.7805	0.7746	0.7659	0.7547	0.7451	0.7368	
0.3	0.2629	0.3432	0.7380	0.7877	0.7821	0.7773	0.7706	0.7574	0.7472	0.7369	
0.4	0.3552	0.6807	0.7834	0.7890	0.7858	0.7841	0.7776	0.7636	0.7495	0.7373	
0.5	0.6457	0.7523	0.7851	0.7903	0.7871	0.7846	0.7797	0.7719	0.7560	0.7378	0.7233
0.6	0.7542	0.7770	0.7870	0.7896	0.7887	0.7868	0.7830	0.7763	0.7613	0.7362	
0.7	0.7988	0.7824	0.7865	0.7893	0.7893	0.7883	0.7842	0.7785	0.7634	0.7363	
0.8	0.7982	0.7849	0.7880	0.7900	0.7898	0.7881	0.7856	0.7811	0.7721	0.7360	
0.9	0.7953	0.7867	0.7894	0.7906	0.7906	0.7868	0.7838	0.7820	0.7745	0.7345	
1	0.2425	0.7948	0.7915	0.7869	0.7796	0.7736	0.7668	0.7569	0.7447	0.7368	

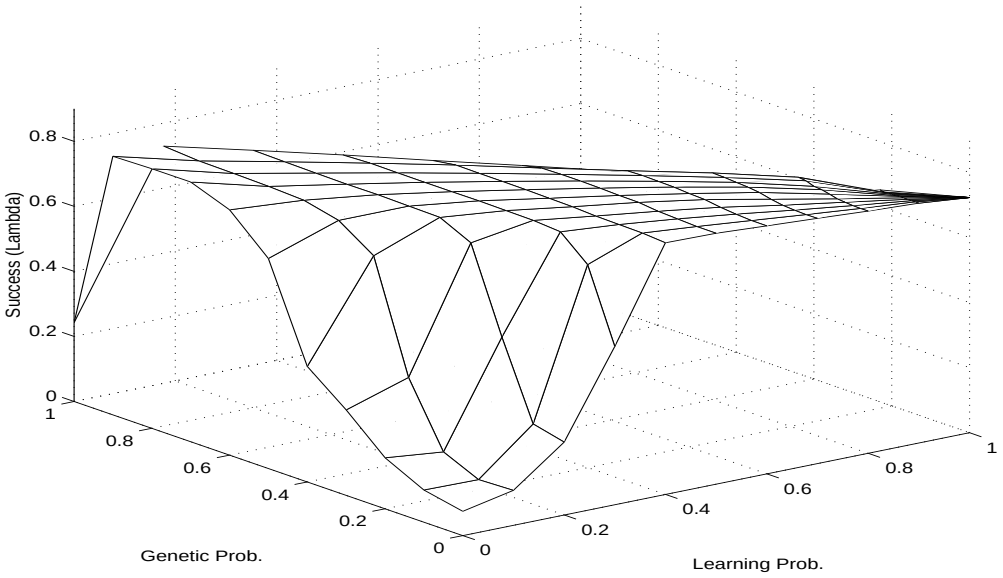


Figure 8. λ for $\alpha^{Env} = 0$

between evolution and "learning" was first applied to natural evolution, and is termed the *Baldwin Effect* (Baldwin, 1896). In our experiments, "learning" is manifested both by learning agents and by parenting agents. Note that the acceleration of "learning" by evolution explains why parenting has such a positive contribution to the genetic-parenting hybrid, compared to the poor performance of pure parenting agents.

- Generally, for a given P_{Par}^{Comp}, λ rises as P_{Lrn}^{Comp} decreases.
- When P_{Par}^{Comp} reaches values as high as 0.6–0.7, λ drops sharply from values above 0.7 to values in the range 0.35–0.5 and lower. So here, not only is pure parenting catastrophic, but also a high enough degree of parenting causes a severe drop in performance.

5.5 *Runs with a Low Positive α^{Env}*

The results for the runs with $\alpha^{Env} \in \{10^{-6}, 10^{-5}, 10^{-4}, 10^{-3}\}$ can be classified together. The results for these values of α^{Env} are summarized in Tables 3, 4, 5, and 6, and they are represented visually in Figures 9, 10, 11, and 12 (respectively). The points that can be derived from the results for these α^{Env}-values are as follows:

- As expected, pure genetic agents perform much worse than pure learning agents.
- Pure learning agents are not the best performers. They are outperformed by hybrids where the following conditions hold:
  (i) $P_{Lrn}^{Comp} > P_{Gen}^{Comp} + P_{Par}^{Comp}$

Table 3. λ for $\alpha^{Env} = 10^{-6}$

$\frac{P_{Gen}^{Comp}}{1-P_{Lrn}^{Comp}}$ \ P_{Lrn}^{Comp}	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
0	0.0730	0.1102	0.2341	0.4547	0.7439	0.7516	0.7514	0.7432	0.7387	0.7337	
0.1	0.0906	0.1070	0.2459	0.4162	0.7248	0.7366	0.7439	0.7354	0.7263	0.7272	
0.2	0.1428	0.1408	0.2803	0.6201	0.7111	0.7346	0.7338	0.7287	0.7348	0.7216	
0.3	0.1886	0.2125	0.4540	0.6256	0.7388	0.7350	0.7230	0.7389	0.7259	0.7204	
0.4	0.2905	0.4339	0.5361	0.6640	0.7255	0.7525	0.7311	0.7318	0.7323	0.7270	
0.5	0.3516	0.4756	0.5898	0.6723	0.7374	0.7255	0.7528	0.7313	0.7246	0.7125	0.7188
0.6	0.4049	0.5101	0.6547	0.7029	0.7248	0.7418	0.7286	0.7382	0.7156	0.7153	
0.7	0.5549	0.6005	0.6620	0.7096	0.7320	0.7085	0.7470	0.7096	0.7217	0.7216	
0.8	0.5663	0.5891	0.6327	0.6869	0.7226	0.7386	0.7156	0.7342	0.7135	0.7097	
0.9	0.5181	0.6060	0.7244	0.7031	0.7196	0.7166	0.7330	0.7281	0.7362	0.7192	
1	0.2139	0.6298	0.6073	0.7278	0.7137	0.7369	0.7160	0.7135	0.6980	0.7190	

Table 4. λ for $\alpha^{Env} = 10^{-5}$

$\frac{P_{Gen}^{Comp}}{1-P_{Lrn}^{Comp}}$ \ P_{Lrn}^{Comp}	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
0	0.0670	0.1163	0.2338	0.3910	0.5970	0.6702	0.6824	0.6943	0.6987	0.7011	
0.1	0.0789	0.1125	0.2396	0.4014	0.5765	0.6567	0.6745	0.6764	0.6976	0.6981	
0.2	0.0822	0.1279	0.2530	0.4071	0.5635	0.6399	0.6607	0.6811	0.6897	0.6953	
0.3	0.1308	0.1558	0.2904	0.4141	0.6032	0.6336	0.6535	0.6772	0.6836	0.6918	
0.4	0.1571	0.2168	0.3288	0.4423	0.5670	0.6367	0.6478	0.6664	0.6745	0.6889	
0.5	0.1234	0.2433	0.3395	0.4785	0.5778	0.6300	0.6660	0.6555	0.6740	0.6878	0.6912
0.6	0.1919	0.3134	0.4130	0.5540	0.5990	0.6364	0.6517	0.6643	0.6712	0.6863	
0.7	0.2504	0.3374	0.4127	0.5286	0.5589	0.6260	0.6416	0.6484	0.6681	0.6818	
0.8	0.2288	0.3973	0.4783	0.5716	0.5937	0.6032	0.6593	0.6482	0.6723	0.6802	
0.9	0.3085	0.4670	0.4585	0.5399	0.5583	0.5849	0.6281	0.6325	0.6532	0.6643	
1	0.1806	0.5329	0.5166	0.5296	0.5039	0.5880	0.5654	0.6060	0.6279	0.6537	

Table 5. λ for $\alpha^{Env} = 10^{-4}$

$\frac{P_{Gen}^{Comp}}{1-P_{Lrn}^{Comp}}$ \ P_{Lrn}^{Comp}	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
0	0.0520	0.1233	0.2317	0.3236	0.4163	0.4816	0.5304	0.5641	0.5823	0.5987	
0.1	0.0445	0.1157	0.2272	0.3360	0.4182	0.4903	0.5321	0.5644	0.5897	0.6019	
0.2	0.0462	0.1204	0.2282	0.3332	0.4166	0.4856	0.5334	0.5626	0.5818	0.5994	
0.3	0.0533	0.1239	0.2325	0.3277	0.4116	0.4805	0.5274	0.5640	0.5850	0.6021	
0.4	0.0570	0.1353	0.2321	0.3330	0.4028	0.4777	0.5209	0.5546	0.5816	0.5989	
0.5	0.0765	0.1461	0.2411	0.3430	0.4127	0.4772	0.5216	0.5519	0.5803	0.5909	0.5452
0.6	0.0685	0.1678	0.2530	0.3371	0.4020	0.4683	0.5167	0.5447	0.5683	0.5912	
0.7	0.0893	0.1700	0.2907	0.3477	0.4136	0.4670	0.5058	0.5359	0.5616	0.5820	
0.8	0.0703	0.2008	0.2824	0.3486	0.4121	0.4584	0.4925	0.5326	0.5582	0.5705	
0.9	0.0890	0.2411	0.3127	0.3729	0.4124	0.4564	0.4673	0.5038	0.5415	0.5591	
1	0.1550	0.4180	0.4163	0.4269	0.4266	0.4369	0.4482	0.4712	0.5023	0.5431	

(ii) $P_{Par}^{Comp} \geq P_{Gen}^{Comp}$

In other words, the best performers are those hybrids where learning predominates (but is not absolute), and parenting has at least as much weight as genetics. For $\alpha^{Env} \in \{10^{-5}, 10^{-4}, 10^{-3}\}$, the inequality of the second condition is a strong one—the best performer contains a mix where parenting has more than twice the weight of genetics. Presumably, for $\alpha^{Env} = 10^{-6}$ environment changes occur at such a low rate that genetics can handle them well (one change every $33\frac{1}{3}$ generations on average).

The result for $\alpha^{Env} = 10^{-6}$ might actually imply that the shift from the best performer type for $\alpha^{Env} = 0$ to the best performer type for low positive α^{Env}-values is a gradual one, rather than an abrupt one. For $\alpha^{Env} = 0$, the best performer has $P_{Lrn}^{Comp} = 0$ and $P_{Par}^{Comp} < P_{Gen}^{Comp}$. For $\alpha^{Env} \in \{10^{-5}, 10^{-4}, 10^{-3}\}$, all best performers have $P_{Lrn}^{Comp} \geq 0.8$ and $P_{Par}^{Comp} > 2 \cdot P_{Gen}^{Comp}$. Therefore, the best performer for $\alpha^{Env} = 10^{-6}$, which has $P_{Lrn}^{Comp} = 0.6$ and $P_{Par}^{Comp} = P_{Gen}^{Comp}$, can be considered "in-between" these two states.

Table 6. λ for $\alpha^{Env} = 10^{-3}$

$\frac{P_{Gen}^{Comp}}{1-P_{Lrn}^{Comp}}$	P_{Lrn}^{Comp} 0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
0	0.0291	0.0930	0.1565	0.2200	0.2623	0.2959	0.3246	0.3456	0.3580	0.3505	
0.1	0.0254	0.0919	0.1623	0.2225	0.2702	0.3102	0.3350	0.3519	0.3647	0.3566	
0.2	0.0261	0.0925	0.1615	0.2231	0.2730	0.3090	0.3408	0.3571	0.3601	0.3530	
0.3	0.0292	0.0911	0.1607	0.2209	0.2714	0.3037	0.3354	0.3580	0.3641	0.3561	
0.4	0.0314	0.0947	0.1635	0.2222	0.2702	0.3039	0.3329	0.3556	0.3634	0.3560	
0.5	0.0343	0.0974	0.1624	0.2191	0.2634	0.3008	0.3343	0.3480	0.3574	0.3522	0.3252
0.6	0.0382	0.1002	0.1605	0.2144	0.2648	0.2957	0.3272	0.3519	0.3524	0.3492	
0.7	0.0402	0.1018	0.1623	0.2161	0.2579	0.2935	0.3207	0.3365	0.3530	0.3465	
0.8	0.0430	0.1143	0.1737	0.2160	0.2589	0.2861	0.3157	0.3270	0.3431	0.3395	
0.9	0.0461	0.1324	0.1862	0.2228	0.2573	0.2836	0.3025	0.3222	0.3265	0.3336	
1	0.1462	0.2943	0.3000	0.3072	0.3028	0.3068	0.3110	0.3222	0.3219	0.3303	

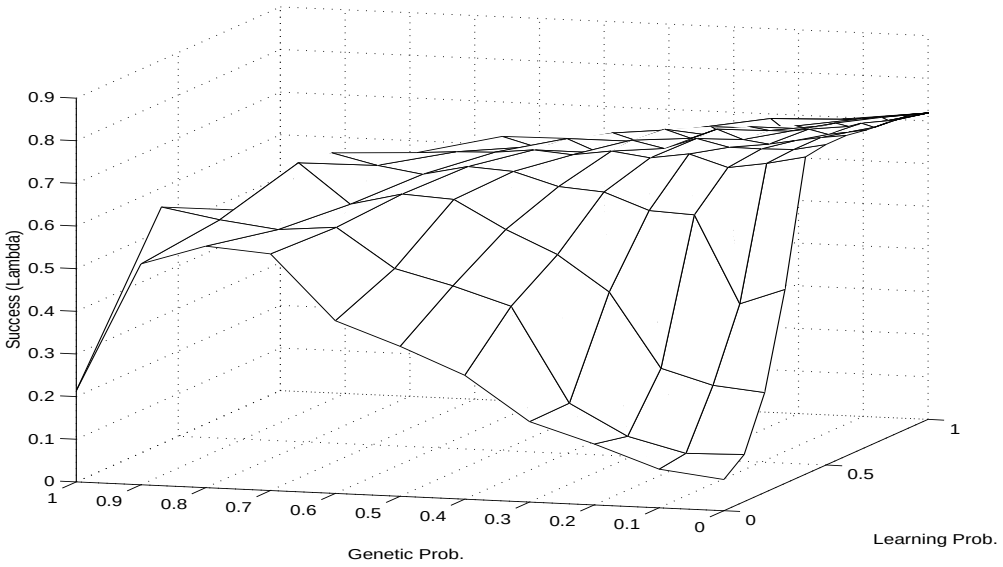


Figure 9. λ for $\alpha^{Env} = 10^{-6}$

Unfortunately, the possibility of this gradual trend for $\alpha^{Env} < 10^{-6}$ cannot be further explored experimentally with the given parameter values. The reason is that the random variable that denotes the number of food patch movements throughout the last $n_{GenTest}^{Env}$ generations is a binomial one. For high enough α^{Env}-values, it is approximately a normal distribution, but already for $\alpha^{Env} = 10^{-5}$ and more so for $\alpha^{Env} = 10^{-6}$, the distribution is not so close to normal anymore (this can be seen visually by comparing the jerkiness of the graphs in Figures 9 and 10 to the relative smoothness of all other graphs).

- The best performers are not the only ones that are better than pure learners. Best performers are surrounded by an (experimentally) continuous environment[1] of performers, all of which perform better than pure learning agents. This is important, because it indicates that it is not a matter of chance or a fluke that the best performers in our experiments performed better than pure learning agents. Table 7 summarizes these continuous environments.
- The best performers of α^{Env}-values 10^{-3} and 10^{-4} perform significantly better than the pure learning agent (12.1% and 10.4% higher, respectively). For α^{Env}-values 10^{-5} and 10^{-6}, the difference is less significant (1.4% and 4.7% higher, respectively).

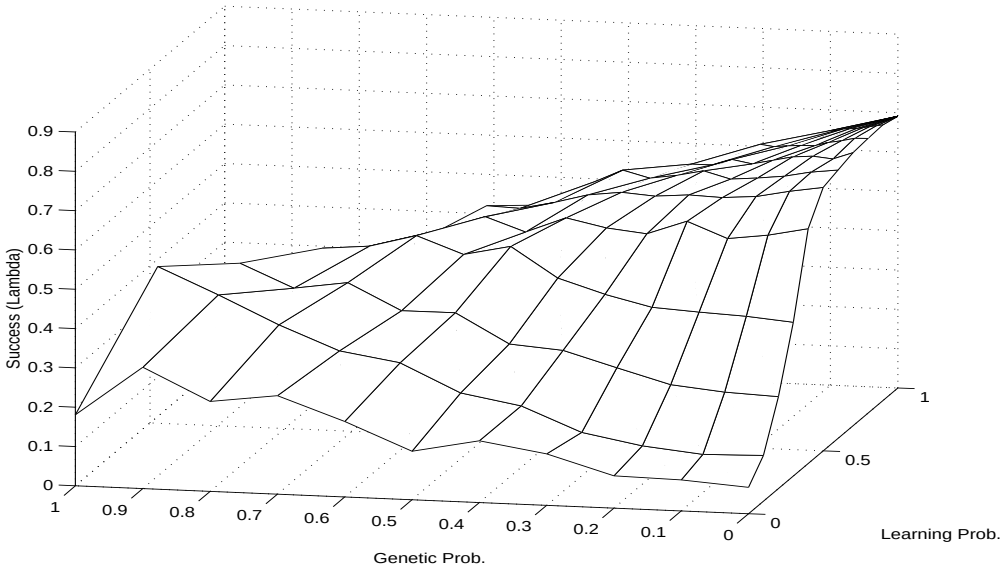


Figure 10. λ for $\alpha^{Env} = 10^{-5}$

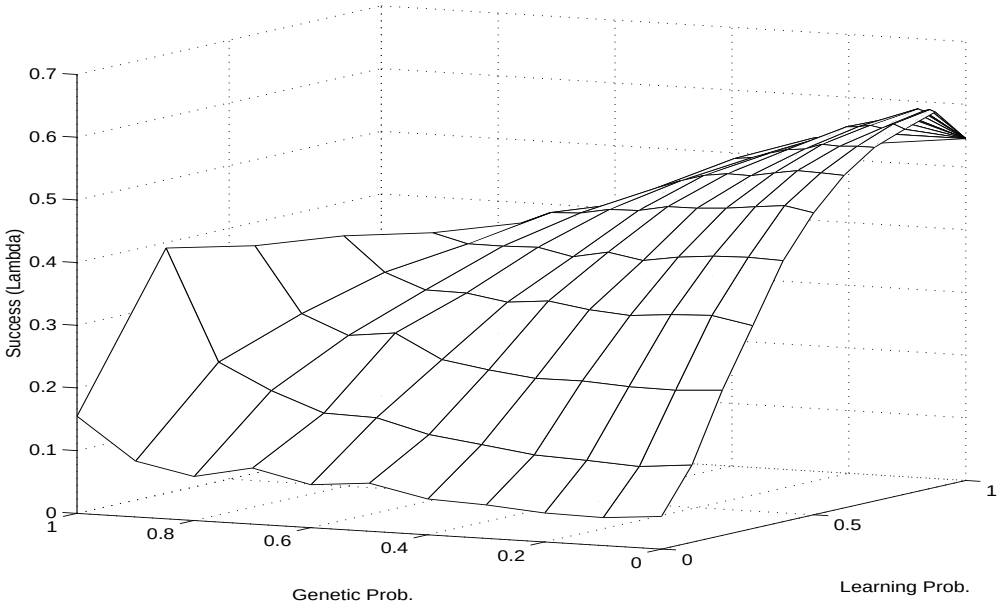


Figure 11. λ for $\alpha^{Env} = 10^{-4}$

Table 7. Environments of Performers that are Better than Pure Learning Agents

α^{Env}	Best Performer $(P_{Lrn}^{Comp}, \frac{P_{Par}^{Comp}}{1-P_{Lrn}^{Comp}})$	P_{Lrn}^{Comp} in Environment	$\frac{P_{Par}^{Comp}}{1-P_{Lrn}^{Comp}}$ in Environment
10^{-6}	(0.6,0.5)	0.5–0.7 0.4–0.7	0–0.6 0.3–0.6
10^{-5}	(0.9,1)	0.8–0.9	0.9–1
10^{-4}	(0.9,0.7)	0.8–0.9 0.7–0.9	0.2–1 0.5–1
10^{-3}	(0.8,0.9)	0.7–0.9	0.2–1

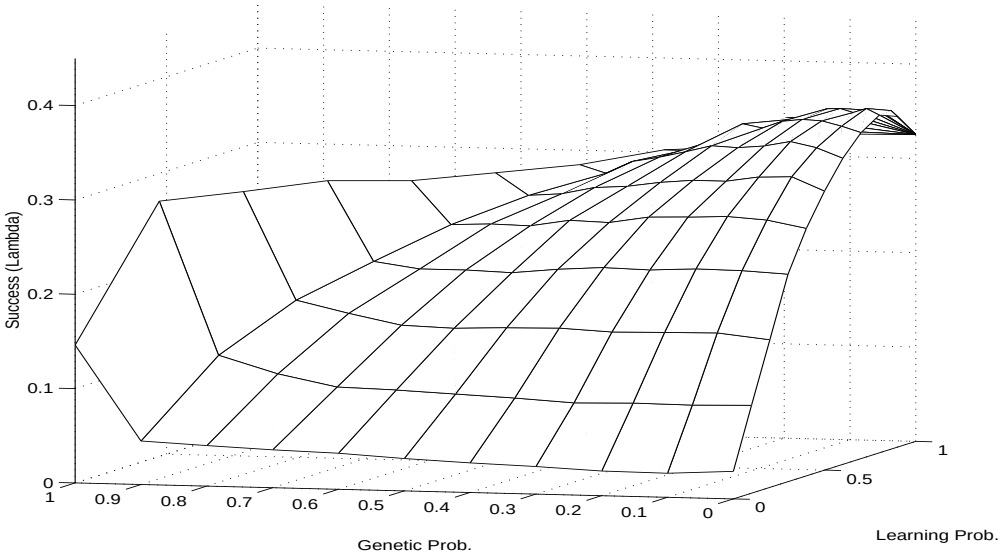


Figure 12. λ for $\alpha^{Env} = 10^{-3}$

Table 8. λ for $\alpha^{Env} = 10^{-2}$

$\frac{P_{Gen}^{Comp}}{1-P_{Lrn}^{Comp}}$ \ P_{Lrn}^{Comp}	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
0	0.0167	0.0427	0.0600	0.0787	0.0887	0.1017	0.1087	0.1162	0.1208	0.1247	
0.1	0.0186	0.0430	0.0624	0.0785	0.0934	0.1006	0.1144	0.1174	0.1221	0.1289	
0.2	0.0195	0.0432	0.0618	0.0789	0.0932	0.1054	0.1142	0.1182	0.1223	0.1260	
0.3	0.0211	0.0440	0.0632	0.0796	0.0956	0.1060	0.1146	0.1167	0.1237	0.1261	
0.4	0.0230	0.0447	0.0640	0.0793	0.0933	0.1037	0.1119	0.1201	0.1225	0.1258	
0.5	0.0256	0.0453	0.0652	0.0812	0.0948	0.1047	0.1128	0.1182	0.1222	0.1282	0.1834
0.6	0.0273	0.0448	0.0636	0.0811	0.0946	0.1049	0.1131	0.1189	0.1238	0.1276	
0.7	0.0278	0.0465	0.0647	0.0793	0.0919	0.1030	0.1132	0.1181	0.1216	0.1253	
0.8	0.0293	0.0492	0.0671	0.0812	0.0942	0.1059	0.1115	0.1181	0.1252	0.1318	
0.9	0.0323	0.0573	0.0744	0.0865	0.0963	0.1070	0.1122	0.1202	0.1267	0.1318	
1	0.0805	0.1319	0.1513	0.1576	0.1599	0.1607	0.1555	0.1523	0.1484	0.1437	

Table 9. λ for $\alpha^{Env} = 10^{-1}$

$\frac{P_{Gen}^{Comp}}{1-P_{Lrn}^{Comp}}$ \ P_{Lrn}^{Comp}	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
0	0.0177	0.0253	0.0287	0.0320	0.0351	0.0367	0.0382	0.0401	0.0423	0.0461	
0.1	0.0190	0.0256	0.0292	0.0325	0.0351	0.0374	0.0396	0.0405	0.0425	0.0462	
0.2	0.0221	0.0258	0.0290	0.0321	0.0350	0.0379	0.0395	0.0413	0.0427	0.0457	
0.3	0.0227	0.0259	0.0294	0.0321	0.0356	0.0378	0.0402	0.0412	0.0431	0.0459	
0.4	0.0233	0.0262	0.0297	0.0324	0.0356	0.0376	0.0399	0.0411	0.0430	0.0461	
0.5	0.0232	0.0260	0.0298	0.0329	0.0357	0.0386	0.0401	0.0415	0.0434	0.0464	0.0698
0.6	0.0231	0.0265	0.0297	0.0331	0.0362	0.0382	0.0399	0.0415	0.0438	0.0463	
0.7	0.0226	0.0267	0.0304	0.0332	0.0364	0.0387	0.0410	0.0422	0.0439	0.0466	
0.8	0.0303	0.0283	0.0313	0.0349	0.0371	0.0392	0.0412	0.0425	0.0446	0.0477	
0.9	0.0317	0.0299	0.0341	0.0374	0.0388	0.0415	0.0430	0.0441	0.0458	0.0485	
1	0.0366	0.0450	0.0541	0.0551	0.0556	0.0560	0.0542	0.0526	0.0514	0.0521	

5.6 *Runs with a Higher Positive α^{Env}*

The results for the runs with $\alpha^{Env} \in \{10^{-2}, 10^{-1}\}$ can be classified together. The results for these α^{Env}-values are summarized in Tables 8 and 9, and they are represented visually in Figures 13 and 14 (respectively). The points that

[1]The term "environment" in this bullet refers to a mathematical environment, not the environment in which the agents operate.

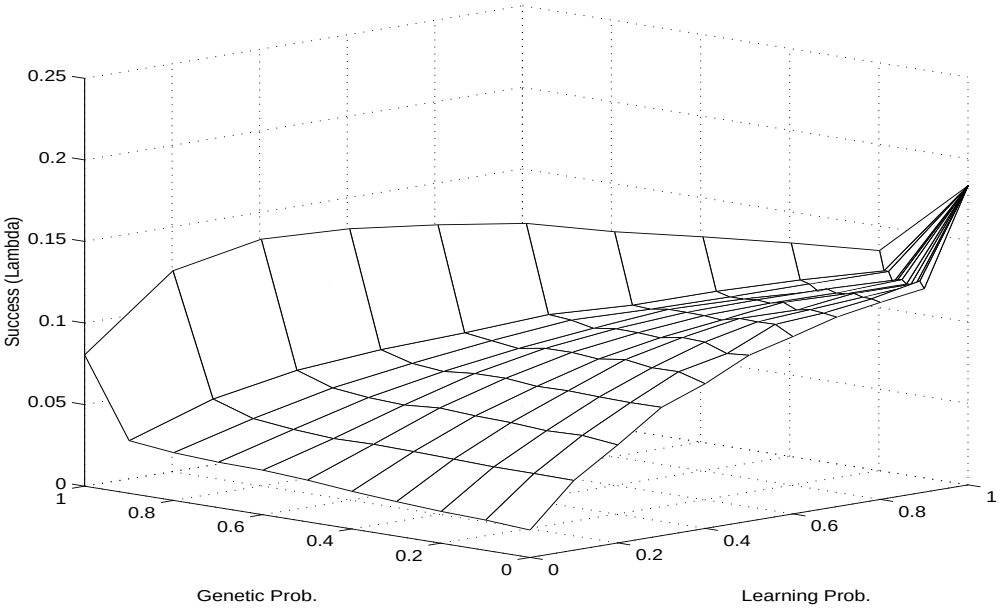


Figure 13. λ for $\alpha^{Env} = 10^{-2}$

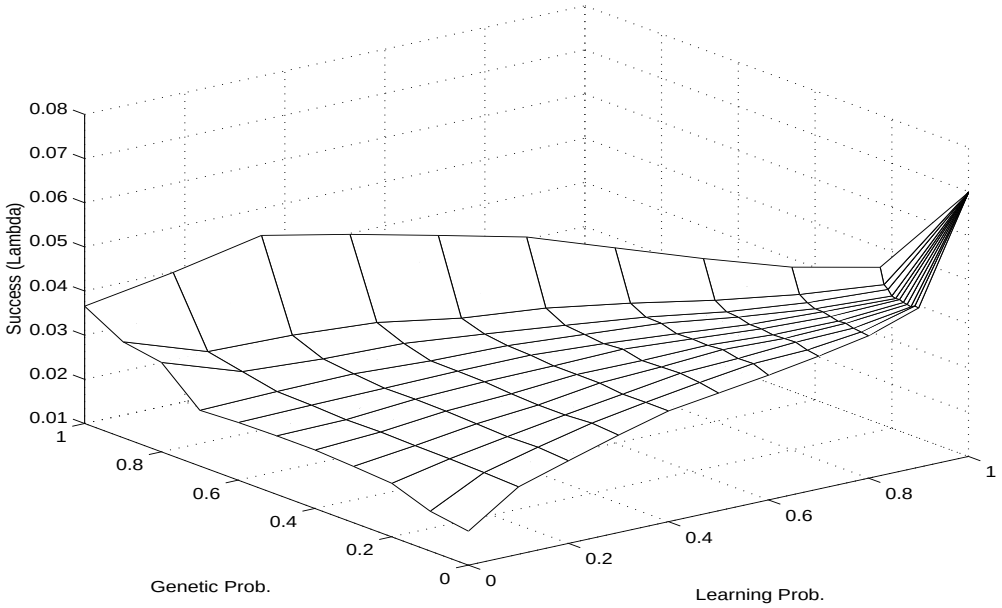


Figure 14. λ for $\alpha^{Env} = 10^{-1}$

can be derived from the results for these values of α^{Env} are as follows:

- As expected, pure genetic agents perform much worse than pure learning agents.
- Pure learning agents are the best performers.
- For any given P_{Lrn}^{Comp}, a hybrid with $P_{Par}^{Comp} = 0$ performs better than any other hybrid with the same P_{Lrn}^{Comp}. In other words, even very small amounts of parenting hinder the performance of agents.

5.7 *A Qualitative Angle*

In Sections 5.4, 5.5, and 5.6, the discussion centered around λ, which is an aggregate measure of BERs (in the last 1000 generations). However, it is also worth exploring the qualitative behavior of BERs that lies behind λ. This section describes the different behaviors of the BER graphs for different types of best performers, and examines the effects of adding a parenting element to a pure learning agent.

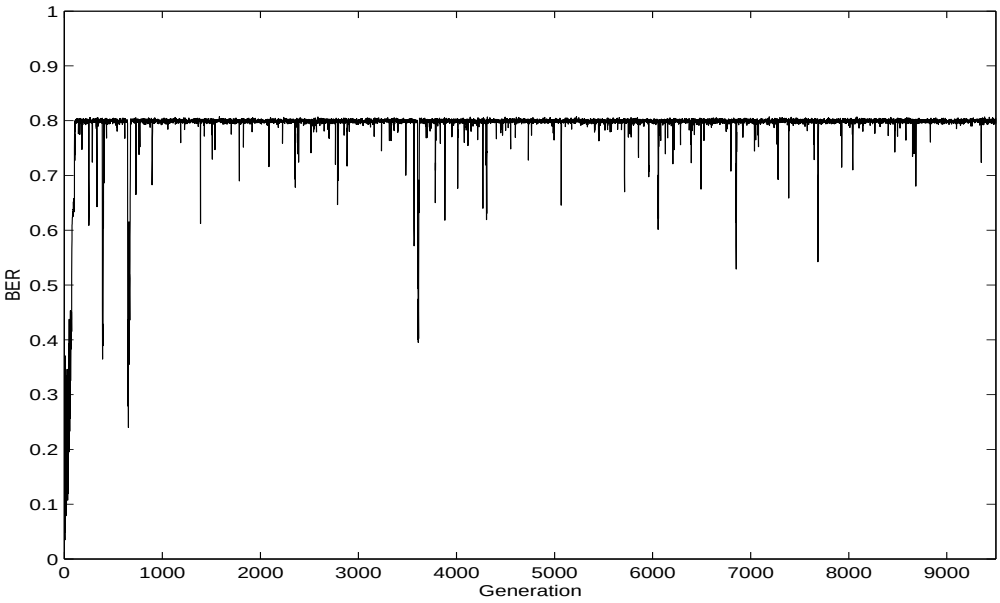


Figure 15. BER of Best Performer for $\alpha^{Env} = 0$

5.7.1 ***BERs of Best Performers.*** For $\alpha^{Env} = 0$, the BER graph for the best performer is shown in Figure 15. This genetic-parenting hybrid evolves over a short period of about 100 generations, and then reaches a very high performance with occasional drops. These drops in performance become rarer as generations pass, and the main, crisp graph line indicates a pretty high consistency in performance. For the last 1000 generations, the measurement period of λ, $\sigma^2 = 4.72 \cdot 10^{-5}$.

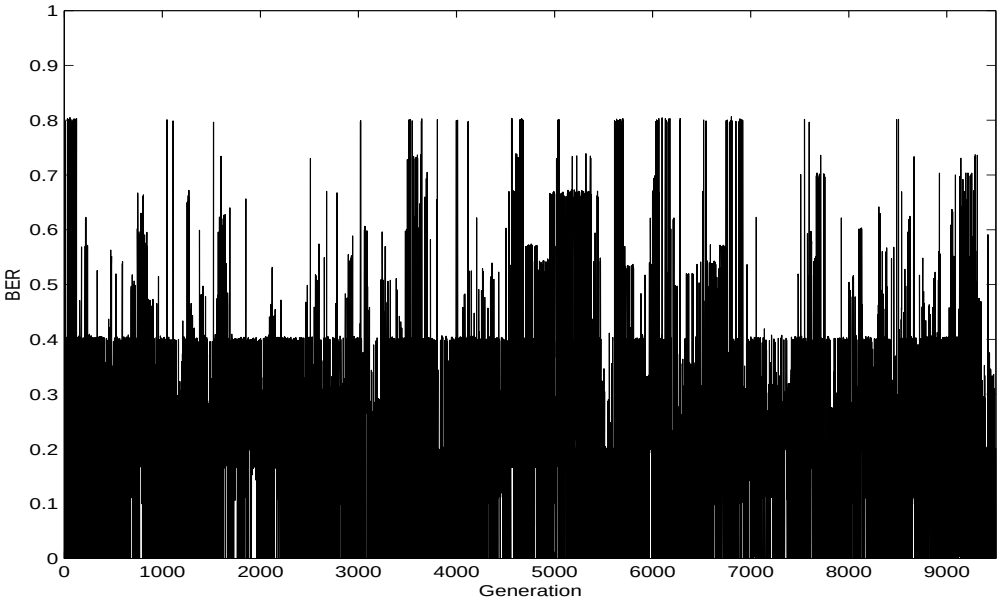


Figure 16. BER of Pure Genetic Agent for $\alpha^{Env} = 0$

When comparing the BER graph of the best performer with the BER graph of a pure genetic agent (Figure 16), it is evident that parenting carries out two roles. First, it manages to nudge genetics so that it converges quickly in the correct direction. Second, it provides a safety net in later generations, in the sense that even when performance falls, it does not fall too deep (with extremely high probability). These roles have already been mentioned in Section 5.4.

As it turns out, learning provides the same roles as parenting (see Figure 17), which means that learning, instead of parenting, can be added to genetics. While this does slightly hurt the average of BERs in later generations ($\lambda = 0.7948$ instead of 0.7988), it greatly improves the minimum of BERs in these generations (with extremely

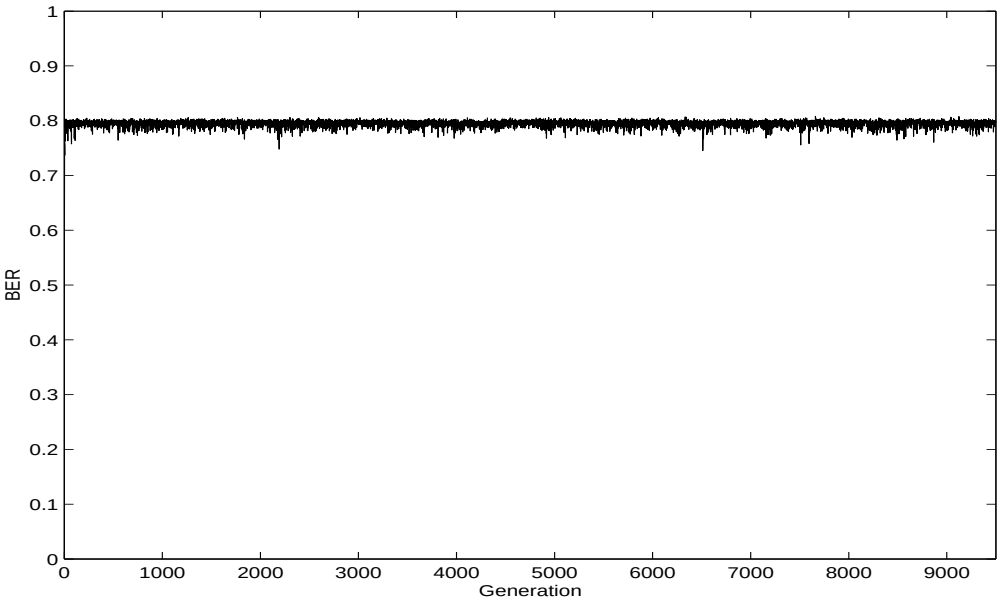


Figure 17. BER of $(G, L, P) = (0.9, 0.1, 0)$ for $\alpha^{Env} = 0$

high probability). The loss in λ is caused by the fact that the main band of the graph in Figure 17 is wider than in Figure 15. To translate the meaning of a "main band" into numerical variance, we considered the graph in Figure 17 to be made up of a single main band with no results outside that band. The minimum of this graph in the measurement period was 0.7606. Thus we compared the variance of all results larger than or equal to 0.7606 in both graphs. This meant that in Figure 15 there were 6 rouge results (out of 1000), that were not included in the definition of the "main band" (this is visually apparent from the figure). For Figure 17, $\sigma^2 = 3.07 \cdot 10^{-5}$. While smaller than the original $\sigma^2 = 4.72 \cdot 10^{-5}$ measured for Figure 15, it is larger than the variance of the main band in that figure— $\sigma^2 = 1.54 \cdot 10^{-5}$. This is caused by the inevitable need of the learning agent to relearn information each new generation and to perform exploration, while a parenting agent performs no exploration at all. As seen here, lack of exploration in parenting also has a disadvantage in that there is a small chance that it would lead agents in a wrong direction, as manifested by the rogue results. But ultimately, in this scenario, the benefits of parenting's lack of exploration still outweigh those of learning in the long run.

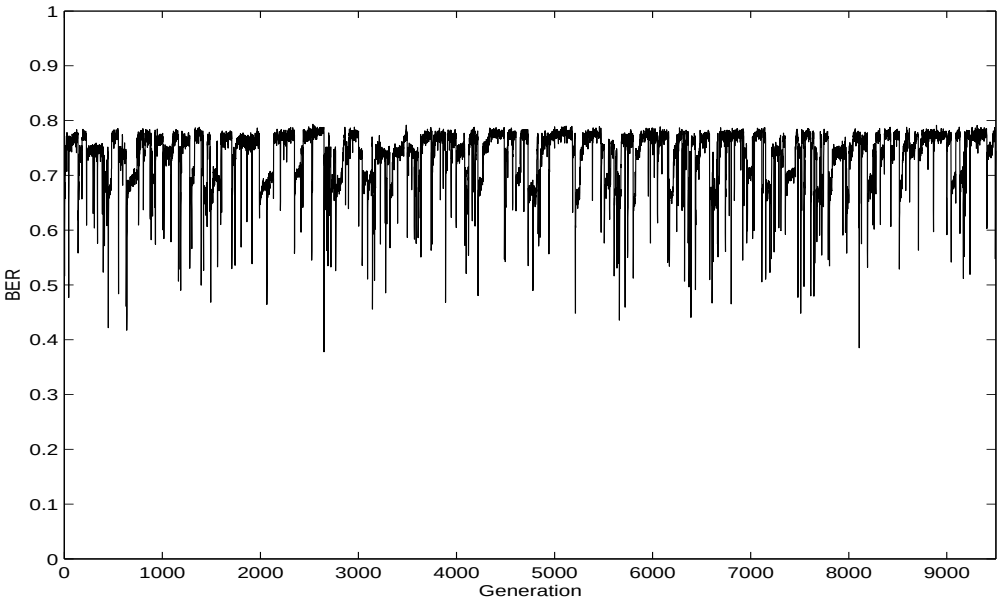


Figure 18. BER of Best Performer for $\alpha^{Env} = 10^{-6}$

For $\alpha^{Env} = 10^{-6}$, the BER graph for the best performer is shown in Figure 18. This genetics-learning-parenting hybrid that has a fair amount of genetics has the advantage of a relatively high starting-point, since the genetics are boosted by learning and parenting (see Section 5.4). This performer manages to evolve from its high starting-point to a very high position (between 0.7 and 0.8), and hold its ground there for a short while. Performance drops occur at a higher rate than they do for the BER performance of $\alpha^{Env} = 0$, and sometimes the drops bring down the performance for longer than an instant (presumably, when the food patch moves). However, learning and parenting act like a safety net, ensuring that the performance drops are not too steep, and genetics always fight (successfully) to bring the performance back up to very high levels. $\sigma^2 = 0.0015$.

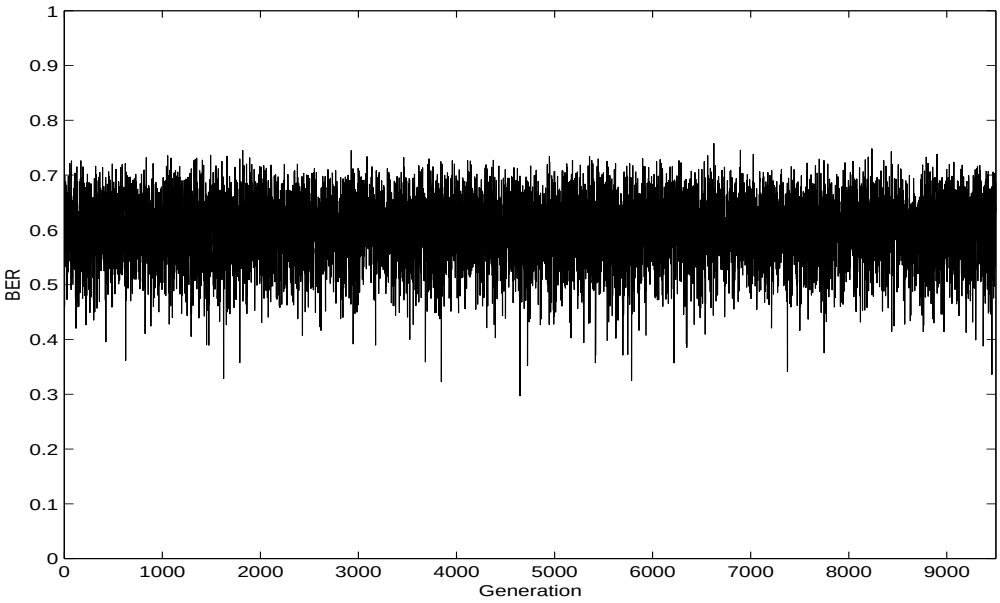


Figure 19. BER of Best Performer for $\alpha^{Env} = 10^{-4}$

For $\alpha^{Env} \in \{10^{-5}, 10^{-4}, 10^{-3}\}$, a representative BER graph for the best performer is shown in Figure 19 (for $\alpha^{Env} = 10^{-4}$). This genetics-learning-parenting hybrid has minute (or no) amounts of genetics. Therefore, the BER graph does not show an element of evolution—the end of the graph is not much higher than its start. On the other hand, parenting and learning provide a safety net that is manifested by the fact that all BER values are higher than 0.3, and a vast majority of them are also higher than 0.4. The graph is very shaky, and the width of its main band is about 0.2. $\sigma^2 = 0.0038$.

For $\alpha^{Env} \in \{10^{-2}, 10^{-1}\}$, a representative BER graph for the best performer is shown in Figure 20 (for $\alpha^{Env} = 10^{-2}$). The best performer is a pure learning agent—and thus, the BER graph does not show any element of evolution. Here too, learning provides a safety net, which is manifested by the fact that all BER values are higher than 0.1. This graph is not as shaky as was the case for $\alpha^{Env} \in \{10^{-5}, 10^{-4}, 10^{-3}\}$—the width of its main band is about 0.1. $\sigma^2 = 4.88 \cdot 10^{-4}$.

5.7.2 ***The Effects of Parenting on Learning.*** Section 5.5 has shown that the best performer for low α^{Env}-values is one where learning predominates but is not absolute, and parenting predominates the rest. Section 5.6 has shown that for higher α^{Env}-values, parenting hinders performance, and pure learning is best.

The effect of parenting on learning can be isolated when comparing the performance of $(G, L, P) = (0, 1, 0)$ with that of $(G, L, P) = (0, 0.9, 0.1)$. This is presented in Figures 21, 22, 23, 24, 25, and 26. Note that for $\alpha^{Env} \in \{10^{-6}, 10^{-5}, 10^{-4}, 10^{-3}\}$ performance is improved by adding parenting, while for $\alpha^{Env} \in \{10^{-2}, 10^{-1}\}$ performance is worsened.

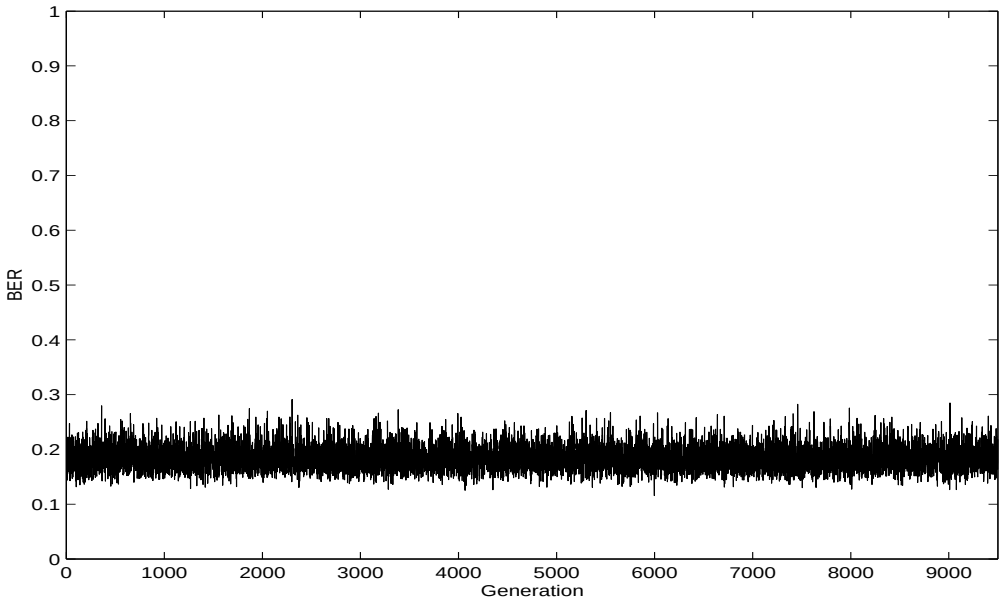


Figure 20. BER of Best Performer for $\alpha^{Env} = 10^{-2}$

Figure 21. BER of $(G, L, P) = (0, 1, 0)$ (left) and $(G, L, P) = (0, 0.9, 0.1)$ (right) for $\alpha^{Env} = 10^{-6}$

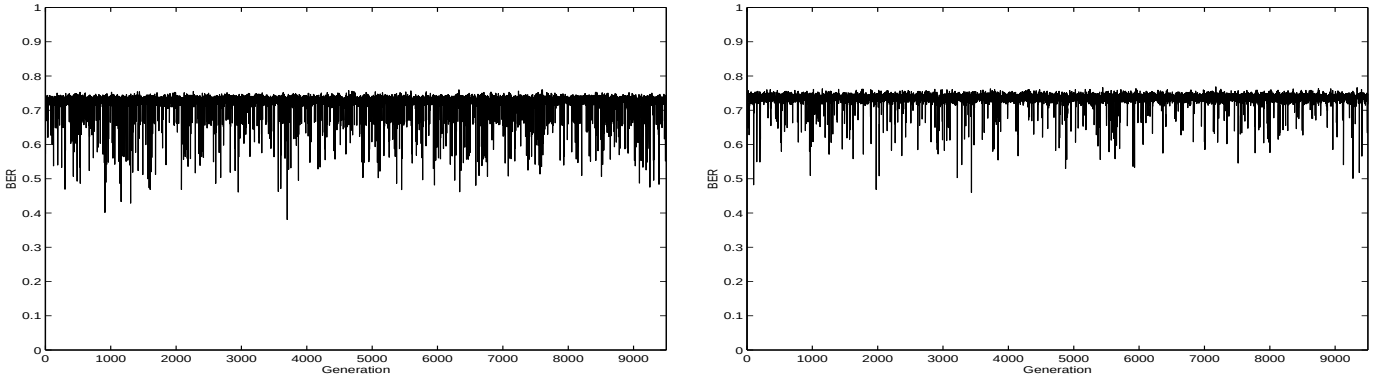
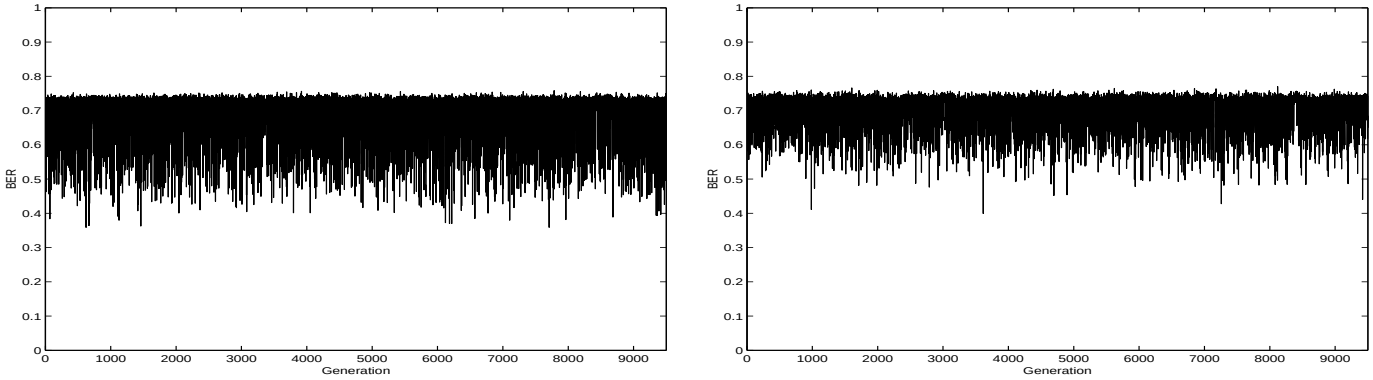


Figure 22. BER of $(G, L, P) = (0, 1, 0)$ (left) and $(G, L, P) = (0, 0.9, 0.1)$ (right) for $\alpha^{Env} = 10^{-5}$



6 Related Work

Related work can be divided into two types. The first type is research that relates to methods that have certain aspects similar to parenting. The second type is research that simulates combinations of genetic algorithms and learning algorithms only.

Figure 23. BER of $(G, L, P) = (0, 1, 0)$ (left) and $(G, L, P) = (0, 0.9, 0.1)$ (right) for $\alpha^{Env} = 10^{-4}$

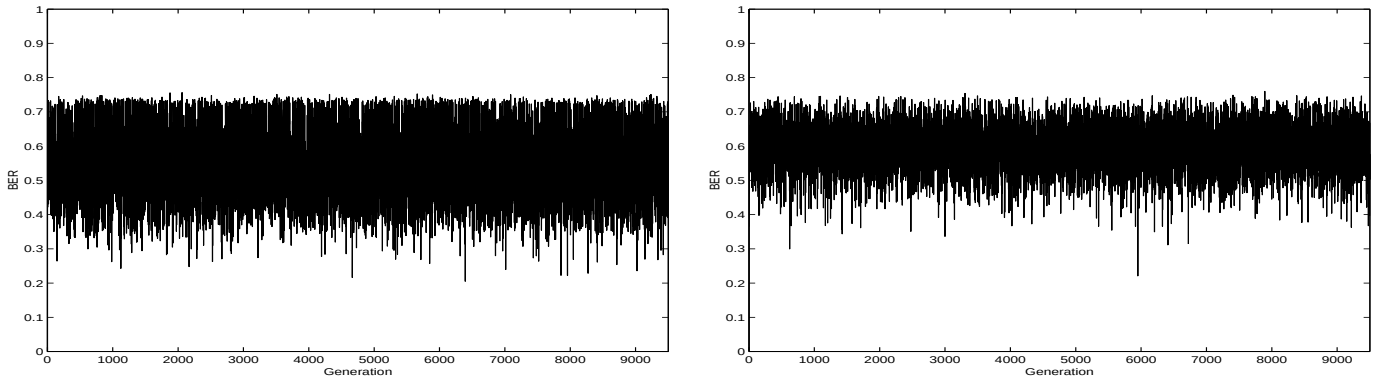


Figure 24. BER of $(G, L, P) = (0, 1, 0)$ (left) and $(G, L, P) = (0, 0.9, 0.1)$ (right) for $\alpha^{Env} = 10^{-3}$

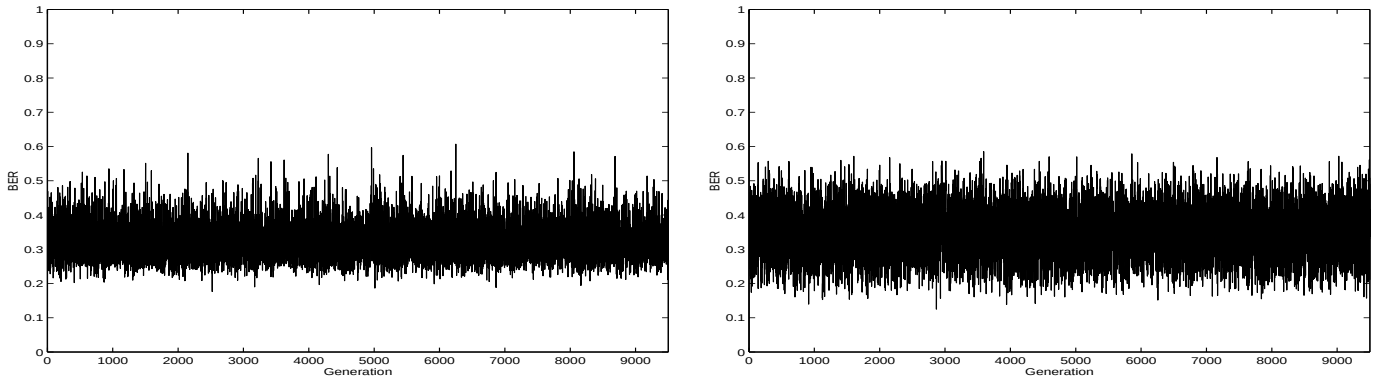


Figure 25. BER of $(G, L, P) = (0, 1, 0)$ (left) and $(G, L, P) = (0, 0.9, 0.1)$ (right) for $\alpha^{Env} = 10^{-2}$

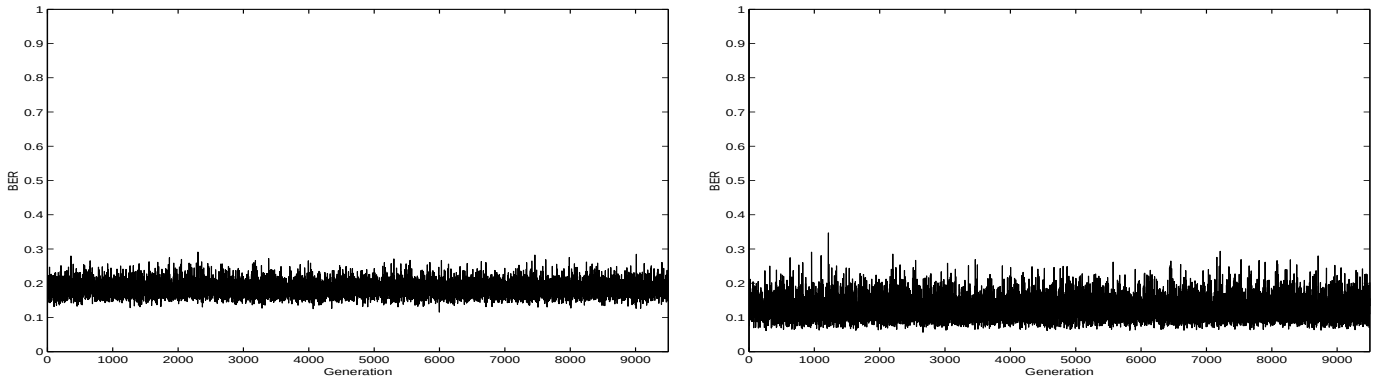
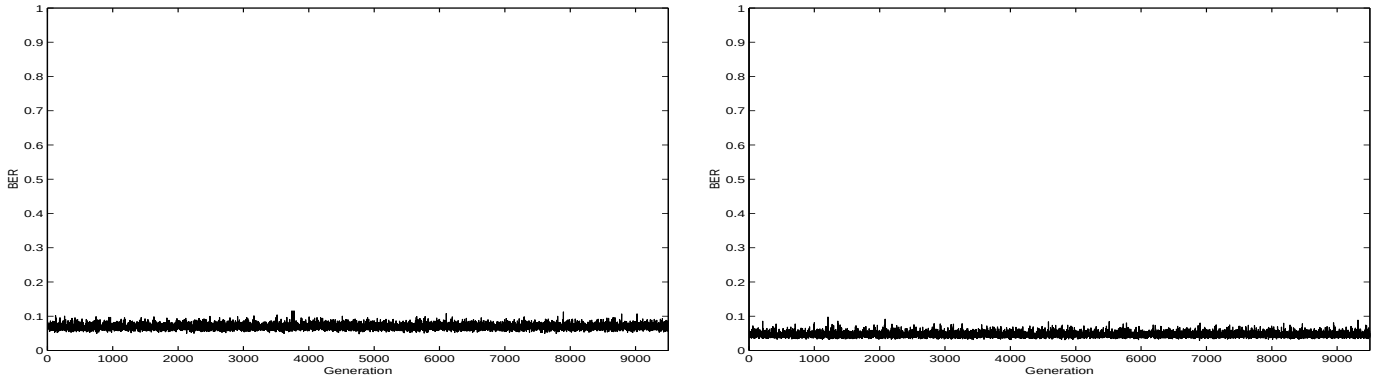


Figure 26. BER of $(G, L, P) = (0, 1, 0)$ (left) and $(G, L, P) = (0, 0.9, 0.1)$ (right) for $\alpha^{Env} = 10^{-1}$



6.1 *Notions Similar to Parenting*

A notion that is somewhat related to parenting is taken from the realm of teamwork—training a new worker on a team to fulfill the job of an older worker that left the team (except here the whole team gets replaced at once). This scenario is dealt with in Denzinger and Ennis (2002), where the notions of genetics, parenting, and learning are replaced by notions of abilities, seed strategy, and online learning, respectively. The abilities determine what a worker can perform. The seed strategy is a strategy that the worker starts with, but it does not cover all situations—it is only a guideline. Online learning fills in the blanks. When a new worker is brought in to replace an old worker, the new worker receives the seed strategy of the old worker, which is analogous to parenting. Also, an interesting parallel between our research and Denzinger and Ennis (2002) is that in the latter, a new worker can replace an old worker only if it has similar abilities; In our research, as mentioned in Section 3.5.1, the parenting in complex agents is performed by real parents, i.e., parents that share genetic material similar to that of the offspring.

Parenting is also somewhat reminiscent of the idea of layered learning Whiteson and Stone (2003). In layered learning, an agent is composed of layers, that are sorted according to their "proximity" to the environment. The layer that is closest to the environment learns first. Afterwards, it trains the next level, which now becomes the learning layer. This process is repeated, with every learning layer turning into a training layer for the subsequent layer. One of the training methods is pruning of the output set—which is similar to what occurs in parenting (each parent provides only one allowed action). However, there are several important differences between parenting and layered learning:

- In parenting, all "layers" (agents) are applied to the same task. In layered learning, layers are applied to different tasks. In this sense, parenting can be said to represent a specific case of layered learning.
- Parenting always deals with no more than two "layers" (which are different agents), while layered learning holds on to previous layers. As a consequence, parenting is extendible to any number of generations, in contrast to layered learning (because of memory limitations).
- In pure parenting, the child "layer" (agent) does not learn at all, while in layered learning each layer learns. However, this difference does not exist in non-pure parenting where some learning is performed, in which case the child "layer" (agent) learns as well.

In parenting, all "layers" (agents) are of identical proximity to the environment.

6.2 *Other Combinations of Genetic Algorithms with Learning*

All research mentioned in this subsection explored learning. However, it appears that what was referred to in this article as parenting might also be branded as a type of "learning" in the work mentioned in this subsection, in that parenting is also a type of local search for a solution that operates side by side with global search, which is performed by a genetic algorithm.

Hinton and Nowlan (1996) presents an extreme example of a stationary environment. In this environment, organisms have 20 switches that they have to either connect or disconnect. Organisms receive additional mating awards only if they set the switches in a single (fixed) pattern, out of the 2^{20} possible patterns. During their lifetime, the organisms have 1000 attempts to find the correct setting. The organisms carry a gene for each switch, with three possible values: 1 (connected), 0 (disconnected) and "?"(to be learned). The organisms "learn" by selecting random values for the switches that correspond to genes marked with "?". It was found that when the organisms were allowed to learn, the evolved populations contained organisms that reached the correct solution much faster than in the case where organisms were not allowed to learn.

Nolfi et al. (1994) presents a more "natural" example of a stationary environment, which is similar to the one we used. The environment is a two-dimensional grid, in which agents search for pieces of food that are located in an environment. The input given to an agent is the position of the food element nearest to it. In contrast to our model, when food is eaten by an agent, it disappears. This is a disadvantage from an analysis standpoint because it influences the performance of agents; thus, we could have an agent that performs well, but that did not find food just because others beat him to it. Agents receive mating rights according to the amount of food that they find during their lifetime. Here too, a combination of a genetic algorithm with learning evolved faster than a pure genetic algorithm. Also, it was shown that the genetic algorithm improves learning by evolving individuals that have a predisposition for learning to behave efficiently.

Nolfi and Parisi (1997) presents a different "natural" environment, which is dynamic. In this environment there is a grid with a single target area, which contains food. The grid is surrounded by walls, and if the agent hits a wall, it gets stuck in it. This fact complicates analysis of performance, and we avoided this in our environment by eliminating walls altogether and making the environment cyclic. In Nolfi and Parisi (1997), there are actually two types of walls, which activate the proximity sensors of agents from different distances. The environment in every even generation is surrounded by the first type of walls, while the environment in every odd generation is surrounded by the second type. This is what makes the environment dynamic. However, there are two important differences from our environment. First, this environment changes only between generations, and not within them as in our environment. Second, the change in this environment is deterministic, unlike in ours. Again, the results were similar—learning speeded up evolution, while the genetic algorithm improved learning.

7 Conclusions

Returning to the original abstract question raised in Section 1.2, let us define an agent's algorithm A to be an *action-augmentor* of an agent's algorithm B if the following conditions hold: (1) both algorithms are always used for receiving perceptions; (2) B is applied for executing an action in most steps; (3) A is applied for executing an action in at least 50% of the other steps.

The main results of our experiments can be rephrased as follows:

- When the environment is stationary, parenting can provide a positive contribution when used as an action-augmentor for genetics. Note that in such an environment, conditions *C1* and *C2* (from Section 1.2) basically hold, although without the condition of *C2* that the rate of dynamic change be *positive* (but zero is indeed very low).
- When the environment has a very low rate of dynamic change, parenting can provide a positive contribution when used as an action-augmentor for learning. Note that in such an environment, conditions *C1* and *C2* both hold.
- When the environment has a higher rate of dynamic change, parenting loses its effectiveness, and pure learning is the best performer. Note that in such an environment, condition *C2* does not hold.

In all cases, pure parenting performs extremely poorly.

When the environment is stationary, learning too can provide a similar positive contribution when used as an action-augmentor for genetics. Thus, the contribution of parenting is unique only to environments with a low positive rate of dynamic change—i.e., where conditions *C1* and *C2* both hold (fully).

While this article only presents a single testbed, the conclusions presented above may very well be applicable to many settings. Continuing with the abstract solution presented above, when a task is given in an agent-environment setting and the agent has a large enough number of rounds to solve the problem, one suggested method for developing suitable agents would be to reduce the task to the setting discussed in this article, as follows:

(i) Divide the sequence of rounds into generations. Each generation should be sufficiently long, and there should be a sufficient number of generations.
(ii) Define a metric over the state of the environment.
(iii) Calculate the expectancy of the "rate of change" in the environment's state, with respect to the metric.
(iv) Categorize the expectancy of the "rate of change". If it is 0, use a genetic-learning combination (as suggested here and in Hinton and Nowlan (1996); Nolfi et al. (1994); Nolfi and Parisi (1997)), or use a genetic-parenting combination (as suggested here). If the expectancy is lower than an arbitrary limit, use some combination of parenting with learning, perhaps with a very small amount of genetics. If the expectancy is higher than that limit, use pure learning.

Finally, we offer a thought which is beyond the strictly computational. In nature, parenting has two functions: providing for the young, and educating them. However, there are settings where the young have access to sufficient resources, eliminating the need to provide for them. An interesting point for biologists to explore might be whether for animals in such settings, a parenting role evolves, as a function of the environment's rate of change. According to this article's results, it would be expected to evolve if and only if the environment does not change too quickly.

Appendix A: Further Research

The model presented in this article contains many parameters and details, which can be modified in search of a better way to synthesize agents that cope with static, slowly changing, and quickly changing environments. Further research can be done on the following questions:

- Would performance improve by tweaking parameter values?
- Parenting in our model was performed with two parents. Is there an optimal number of parents? Is it one, two, or more?
- Parenting in our model was performed with real parents, as mentioned in Section 3.5.1. Is there any importance for the parents to be real parents? Is there some hidden effect when the offspring receives advice from parents with which it shares its genetic material? This question especially concerns cases where best performers had positive values of P_{Gen}^{Comp} and P_{Par}^{Comp}.
- A parenting agent in our model selected between its parents' advice with equal probability. Is there a different method that would improve on this? For example, should the probability that the parenting agent selects a parent's advice be influenced by how well that parent performed in the previous generation? Should it be influenced by how well that parent is sure of its advice (e.g., the actual MC-values, the number of times that the parent encountered the given memory, how late during its life-time the parent encountered the given memory, etc.)?
- Should the parenting model be changed so that parents track the performance of their offspring and update their MC-tables accordingly? This question is analogous to the idea presented in Whiteson and Stone (2003)—would layered learning operate better concurrently (in which case a training layer still learns), or should there only be a single learning layer at any given time (which is termed "traditional" layered learning)?
- What difference would it make to add exploration to the parenting algorithm? Would this significantly improve or worsen the effects of parenting shown in this article?
- What happens when α^{Env} is positive but approaches zero? Would learning still predominate for best performers, or would best performers shift more and more weight toward genetics, up to a point where genetics predominate?
- Would there be any significant difference when memory lengths are larger than 1?
- Would there be any significant difference with more than one food patch, or with a food patch that has multiple local minima or maxima?
- Should the model be changed so that (G, L, P) does not remain constant throughout the lifetime of agents? To take human settings as an example, as people grow older, they tend to listen less to their parents and rely more on their own experiences. Should P_{Par}^{Comp} decrease and P_{Lrn}^{Comp} increase during an agent's lifetime?
- Is it possible for agent generations to evolve a suitable (G, L, P), instead of having it predetermined?
- The metric used in defining agent success relied on the best eating rate (BER) measured per generation. Would the conclusions regarding the appropriate mix of genetics, learning, and parenting change significantly if an average eating rate was measured instead?

This list is certainly not exhaustive, and many other aspects can be further explored.

References

R. Axelrod: 1997, *The Complexity of Cooperation: Agent-Based Models of Competition and Collaboration*. Princeton University Press.

J. Baldwin: 1896, 'A New Factor in Evolution'. *American Naturalist* **30**, 441–457, 536–554.

R. A. Brooks: 1986, 'A Robust Layered Control System for a Mobile Robot'. *IEEE Journal of Robotics and Automation* **2**(1), 14–23.

D. Carmel and S. Markovitch: 1999, 'Exploration Strategies for Model-based Learning in Multiagent Systems'. *Autonomous Agents and Multi-agent Systems* **2**(2), 141–172.

H. G. Cobb and J. J. Grefenstette: 1993, 'Genetic Algorithms for Tracking Changing Environments'. In: *Proceedings of the Fifth International Conference on Genetic Algorithms*. San Mateo, pp. 523–530.

J. Denzinger and S. Ennis: 2002, 'Being the New Guy in an Experienced Team - Enhancing Training on the Job'. In: *Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems*.

Bologna, Italy, pp. 1246–1253.

G. E. Hinton and S. J. Nowlan: 1996, 'How Learning can Guide Evolution'. In: *Adaptive Individuals in Evolving Populations: Models and Algorithms*. Addison-Wesley, pp. 447–454.

T. D. Johnston: 1996, 'Selective Costs and Benefits in the Evolution of Learning'. In: *Adaptive Individuals in Evolving Populations: Models and Algorithms*. Addison-Wesley, pp. 315–358.

M. Littman: 1996, 'Simulations Combining Evolution and Learning'. In: *Adaptive Individuals in Evolving Populations: Models and Algorithms*. Addison-Wesley, pp. 465–477.

M. Littmann and D. Ackley: 1991, 'Adaptation in Constant Utility Non-Stationary Environments'. In: *Proceedings of the Fourth International Conference on Genetic Algorithms*. San Mateo, pp. 136–142.

G. Mayley: 1996, 'Landscapes, Learning Costs, and Genetic Assimilation'. *Evolutionary Computation* **4**(3), 213–234.

S. Nolfi, J. L. Elman, and D. Parisi: 1994, 'Learning and Evolution in Neural Networks'. *Adaptive Behavior* **3**(1), 5–28.

S. Nolfi and D. Floreano: 1999, 'Learning and Evolution'. *Autonomous Robots* **7**(1), 89–113.

S. Nolfi and D. Parisi: 1997, 'Learning to Adapt to Changing Environments in Evolving Neural Networks'. *Adaptive Behavior* **5**(1), 75–98.

D. Parisi and S. Nolfi: 1996, 'The Influence of Learning on Evolution'. In: *Adaptive Individuals in Evolving Populations: Models and Algorithms*. Addison-Wesley, pp. 419–428.

T. W. Sandholm and R. H. Crites: 1996, 'Multiagent Reinforcement Learning in the Iterated Prisoner's Dilemma'. *Biosystems* **37**, 147–166.

M. Sebag, M. Schoenauer, and C. Ravise: 1997, 'Revisiting the Memory of Evolution'. In: T. Bäck (ed.): *Proceedings of the Seventh International Conference on Genetic Algorithms (ICGA97)*. San Francisco, CA.

R. S. Sutton and A. G. Barto: 1998, *Reinforcement Learning: An Introduction*. The MIT Press.

P. M. Todd and G. F. Miller: 1991, 'Exploring Adaptive Agency II: Simulating the Evolution of Associative Learning'. In: *From Animals to Animats: Proceedings of the First International Conference on Simulation of Adaptive Behavior*. pp. 306–315.

S. Whiteson and P. Stone: 2003, 'Concurrent Layered Learning'. In: *AAMAS 03 : Proceedings of the Second International Joint Conference on Autonomous Agents and Multiagent Systems*. Melbourne, Australia, pp. 193–200.