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
Jerusalem, Israel
{mberger,jeff}@cs.huji.ac.il

Abstract

This paper explores hybrid agents, which use a combination of a genetic algorithm, a learning algorithm, and a parenting algorithm. We experimentally examined what constitutes the best combination of weights over these three algorithms, as a function of an environment's rate of change.

1. Introduction

Genetic algorithms (GAs) represent every possible solution to a given problem as a sequence of "genes". Over many generations, different sequences evolve towards a near-optimal solution. A disadvantage of GAs is that they are mostly suitable for stationary problems; problems, however, may have a dynamic state, and in such cases, GAs tend to collapse. GAs can be improved by combining them with a learning algorithm [3]. While GAs gather information over many generations, learning gathers information over a single generation, and, unbiased by information from previous generations, is more suitable for dynamic problems. However, learning might miss important information from previous generations, and is thus more wasteful. How would algorithms that are "in-between" fare, when a problem state remains somewhat similar between one generation and the next? Consider problems with the following conditions:

C1 A problem state can only change to an "adjacent" state.

C2 The problem has a very low (positive) rate of change.

We introduce parenting as such an "in-between" algorithm.

2. Environment and Task

A *round* is the basic time unit. The environment consists of a cyclic rectangular grid, $w^{Env} \times h^{Env}$ in size. The grid contains agents and food. In a given round, when food is present in a square, all agents in that square "eat". At the end of a round, an agent may travel to an adjacent square or stay put. A *food patch* is a positive probability function over a group of adjacent squares, which determines the probability that food would appear in a square in a given round. Agents are not aware of food patches. α^{Env} is the probability that in a given round, all food patches move, each in its own random direction. This environment fulfills *C1*. For very low positive values of α^{Env}, it fulfills *C2* as well.

A *generation* is a group of agents that live in the environment simultaneously. In a given generation, all agents start at the same position (determined randomly). An *eating-rate* of an agent is the fraction of rounds in which it eats out of the rounds in which it lives. For a given generation, a *BER (best eating-rate)* is the maximal eating-rate in that generation. In a run of n^{Task}_{GenRun} generations, each consisting of n^{Task}_{Ag} agents that live for n^{Task}_{Rnd} rounds, λ is the mean over the BERs of the last $n^{Task}_{GenTest}$ generations. Our goal is to develop agents that maximize λ.

3. Agent Definitions and Types

A *reward* indicates whether an agent ate (1) or not (0). A *perception* contains an agent's position and reward. An *Action* is one of EAST, SOUTH, WEST, NORTH, HALT. A *memory* is a sequence of the agent's last perception-action pairs. A *MAM* (Memory-Action Mapper) receives a memory as input, and returns an action as output. An *ASF* (Action-Selection Filter) receives several action suggestions as input and returns one of them as output.

The MAM of a genetic agent is its gene sequence. Each gene is composed of a *key* (a possible memory) and a *value* (a possible action). The MAM of a genetic agent is created with the agent, and it is never updated. When the MAM receives a memory as input, it returns an action by returning the value that matches the appropriate key. GAs employ generations of genetic agents, letting each agent run in the environment and then producing a new generation by mating agents from the current generation [1]. When two genetic agents mate, they create two offspring. The MAMs of the offspring are created by passing the gene sequences of the parents through a process of crossovers and mutations.

A learning agent employs the Q-learning algorithm with Boltzmann exploration [4]. In Q-learning, an agent tries to maximize rewards by estimating them in advance. The *Q-value* $Q(s,a)$ is the expected discounted sum of future rewards obtained by taking action a when the memory is s, and following an optimal policy thereafter. When a is selected as the next action, $Q(s,a)$ is updated after receiving the subsequent reward. Boltzmann exploration is responsible for selecting the next action. Every possible action is given a probability of being selected. For any given memory, the probability is initially equal for all actions, but over time, the probability of selecting actions with high Q-values increases. After a while, exploration stops, and the action with the maximal Q-value is always selected thereafter.

A parenting agent selects an action by turning to another parenting agent, its "parent", for advice. The parent, which interacted with the environment in the previous generation, uses its own experience to return an answer. We used Monte Carlo (MC) methods [5] to simulate parenting. MC-methods evaluate and improve memory-action mappings after *episodes* of rounds (i.e., generations). In the evaluation stage, the mapping of each possible memory to each possible action produces an *MC-value*. $MC(s,a)$ is the average of rewards in all rounds following the agent's encounter of memory s and selection of action a as the next action. 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 is mapped to an action by selecting the action with the highest MC-value for that memory. The parenting agent has an ASF, which gives each parent an equal chance for its advice to be taken.

The real agent population consists only of *complex agents*. Each complex agent contains an instance of each simple type (genetic, learning, and parenting), in a subsumption architecture [2]. A complex agent mediates between its inner agents and the environment.

At a generation's end, each complex agent is awarded mating rights according to its eating-rate. Two complex agents mate by mating their respective inner agents and encapsulating the inner offspring in a new complex offspring.

When a complex agent receives a perception, it passes it on to its inner agents, which update their MAMs if needed.

When a complex agent executes an action, it asks its inner agents to suggest it. The complex agent then feeds the suggested actions into its ASF, which selects the suggestion of the genetic agent, the learning agent, or the parenting agent with probabilities P_{Gen}^{Comp}, P_{Lrn}^{Comp}, and P_{Par}^{Comp}, respectively. The complex agent executes the action selected by its ASF and also perceives it in the usual manner.

4. Runs and Results

The constants were: $n_{Ag}^{Task} = 20$, $n_{GenRun}^{Task} = 9500$, $n_{GenTest}^{Task} = 1000$, $n_{Rnd}^{Task} = 30000$, $h^{Env} = 20$, $w^{Env} = 20$. There was one food patch, 5×5 in size. Its center had a value of 0.8. All outer frame squares had a value of 0.2. All other squares had a value of 0.4. The dependent variable was λ. The independent variables were α^{Env} and $\delta \equiv (P_{Gen}^{Comp}, P_{Lrn}^{Comp}, P_{Par}^{Comp})$. Results appear in table 1.

	δ				
α^{Env}	(1,0,0)	(0,1,0)	(0,0,1)	Best δ	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. Result summary of runs

5. Conclusions

An agent's algorithm A is an *action-augmentor* of an agent's algorithm B if: (1) Both algorithms always receive perceptions; (2) B selects the action in most steps; (3) A selects the action in at least 50% of the other steps.

In a stationary environment (*C1* holds, *C2* holds except that the rate of dynamic change is not *positive*), parenting can contribute as an action-augmentor for genetics. In an environment with a low rate of dynamic change (*C1* and *C2* both hold), parenting can contribute as an action-augmentor for learning. In an environment with a high rate of dynamic change (*C2* doesn't hold), parenting loses its effectiveness.

The conclusions above might help determine when parenting would be expected to appear in natural environments.

References

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

[2] R. A. Brooks. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, 2(1):14–23, March 1986.

[3] S. Nolfi and D. Parisi. Learning to adapt to changing environments in evolving neural networks. *Adaptive Behavior*, 5(1):75–98, 1997.

[4] T. W. Sandholm and R. H. Crites. Multiagent reinforcement learning in the iterated prisoner's dilemma. *Biosystems*, 37:147–166, 1996.

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