Finding Attack Strategies for Predator Swarms Using Genetic Algorithms

Ryan E. Leigh
University of Nevada, Reno
Reno, NV 89557
leigh@cse.unr.edu

Tony Morelli
University of Nevada, Reno
Reno, NV 89557
morelli@cse.unr.edu

Sushil J. Louis
University of Nevada, Reno
Reno, NV 89557
sushil@cse.unr.edu

Monica Nicolescu
University of Nevada, Reno
Reno, NV 89557
monica@cse.unr.edu

Chris Miles
University of Nevada, Reno
Reno, NV 89557
miles@cse.unr.edu

Abstract- Behavior based architectures have many parameters that must be tuned to produce effective and believable agents. We use genetic algorithms to tune simple behavior based controllers for predators and prey. First, the predator tries to maximize area coverage in a large asymmetric arena with a large number of identically tuned peers. Second, the GA tunes the predator against a single prey agent. Then, we tune two predators against a single prey. The prey evolves against a default predator and an evolved predator. The genetic algorithm finds high-performance controller parameters after a short length of time and outpaces the same controllers hand tuned by human programmers after only a small number of evaluations.

1 Introduction

This paper is a preliminary investigation into designing controllers for swarming predators and defending prey for a computer game. Swarms rely on using many easily replaceable and expendable units. For example, by easily replaceable we mean that if one bee is lost defending the hive, there is another to replace it and probably another one soon to be born. To be expendable means that the loss of the unit will not result in critical failure of the swarm. In other words, no task rests entirely on one unit's shoulders nor is there any centralized command. If removing an agent, the swarm should operate just as it did before, possibly reorganizing itself to maintain performance. Again, if a bee dies, even the queen, the swarm can continue to attack.

Prey, on the other hand, have a more difficult task. If the prey cannot fight back, its only tactic is to escape, but with predators capable of attacking from all sides, this tactic may not work. If the prey can fight back, it must ensure that it does not get overwhelmed or cornered and must maneuver to expose its strengths and hide its weaknesses.

Games provide a good application area for studying predator and prey behavior, both for entertainment and for serious simulations like those used in training. A training game needs to provide challenging opponents to be useful. The opponents can be entities in the game world controlled by humans or controlled by game AI. However, there may not be enough humans to control the large numbers of entities that constitute a swarm. Current game AI for controlling many units may be capable of planning an attack en massè, but exhibit limited cohesion or coordinated planning capabilities. We seek a more coordinated control strategy and a game AI appropriate for use in these types of games.

Our goal is to have agents learn how to attack as a swarm or defend against a swarm. However, in order to develop good prey behavior you need to train the prey against a good predator. Similarly in order to develop a good predator, you need to train the predator against a good prey. This cyclical dependence indicates our system's suitability for co-evolution and our long term goal focuses on investigating co-evolutionary approaches to swarming predator-prey systems. We describe in this paper our initial efforts to establish a baseline for our long term goals and investigates the following questions: How should controllers for such agents be designed? How and what should the agents learn? What is a successful attack or defense? Our results show how simple behavioral controllers can be parameterized and optimized via a genetic algorithm to find attack strategies for one or two predators and the defense strategies against the predators. With both predators and prey to evolve, these investigations provide the preliminary steps for using co-evolution.

This paper is organized as follows: the next section provides a brief introduction into behavioral controllers. Section 3 describes the simulation that tests these controllers. Sections 4 and 5 describe our experimental methodology. Section 6 contains the results. Conclusions, observations, and future work are provided in Section 7.

2 Background

2.1 Behavioral Controllers

Behavior-Based Control (BBC) has become one of the most popular approaches for embedded system control in research and in practical applications [2, 6]. Behavior-based systems employ a collection of concurrently executing processes, which take input from sensors or other behaviors, and send commands to actuators. Sensor inputs determine the activation level of a process, for example, whether the process is on or off, and in some systems by how much. These processes, called *behaviors*, represent time-extended actions that aim to achieve or maintain certain goals, and

are the key building blocks for more complex, intelligent behavior.

BBC's modularity and robust real-time properties make it an effective approach for robot and autonomous agent control. While BBC constitutes an excellent basis for our chosen domain, developing behavior-based systems requires significant effort from the part of the designer. In particular, the choosing parameters intrinsic to the robot's behaviors consumes much time. Programmers need to choose and test numerous parameter values and their combinations in order to achieve a behavior with desired characteristics, such as smooth trajectories, effectiveness, and ability to reach the goal.

In this paper, we equip the robot with a set of behaviors and we propose an approach to learning the behaviors' specific parameters using an evolutionary approach. Genetic algorithms (GA) have been successfully employed for tuning, designing and programming robots. For example, Sims used a variable length tree encoding that built simulated robots from blocks and motor actions to competitively evolve simulated robots from the ground up [9]. Schultz, used GAs to find the rules that decide when to activate a behavior [8]. Gruber used a GA to tune a neural network for a predator in a predator-prey environment [4]. Louis and Li combined memory with the GA to evolve robotic behaviors [5]. In contrast with these previous approaches, our work proposed in this paper examines the preliminary steps needed for co-evolution between predator and prey.

3 Simulation

We have developed a 2D simulation for evaluating the fitness of attack and defense strategies. Our simulation design needed to be simple and fast for quick evaluations of individuals in the GA. We prefer coarse yet quick fitness evaluations to accurate yet lengthy simulations and therefore simulate simple predators and prey. Predators and prey only move with a certain speed and heading. A predator wins if it manages to collide with its prey, otherwise the prey wins if it manages to avoid colliding with the predator and survives until time runs out.

3.1 Environment

Figure 1 shows the environment map is made up of fixed-sized tiles that are either traversable (dark areas) or non-traversable (light areas). Non-traversable tiles allow for fast and easy collision detection and easier finding of surrounding barriers. For this study, the map is 258 tiles by 97 tiles which suits our needs and matched multiples of our monitor's screen resolution.

3.2 Agents

Each predator unit has two sensors and two effectors. The *barrier sensor* returns the distance to, and direction of, the center of all non-traversable tiles within a certain parameterizable range. The *beacon sensor* returns the distance, label, and direction of every other agent's location simulated by

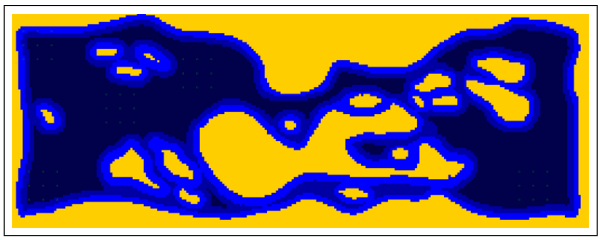

Figure 1: Environment Map

pretending the agent has an omni-directional beacon corresponding to the center of that agent. The two effectors allow the controller to specify a desired direction and desired speed. See Figure 2 for a diagram. The direction and speed are labeled as "desired" because the simulation will slowly change the controller's state to match these parameters rather than instantly set the values. Prey cannot accelerate or turn as fast as the smaller and more nimble predators. Since maneuverability is one of the most important components in any strategy, we model this facet to improve solution quality.

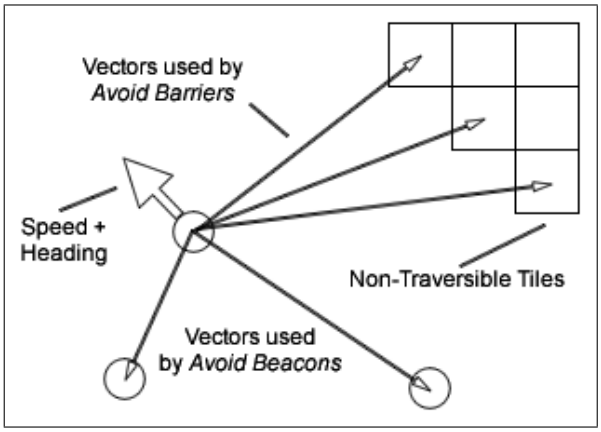

Figure 2: Diagram of Sensors and Effectors. Circles represent agents

Four behaviors control each predator: *wander*, *track target*, *avoid barriers*, and *avoid beacons*. *Wander* and *track target* specify desired agent direction while *avoid barriers* and *avoid beacons* specify desired agent speed. *Wander* chooses a random new desired heading after a random time interval. *Track target* moves the controller towards a target, in this case, the prey. All entities in the simulation have a unique label that also identifies them as predator or prey. *Wander* provides a way to get around obstacles and explore the environment. The controller will not start using *track target* until the target is within a certain range, otherwise an over zealous attack approach could have the predator getting stuck on a barrier. The GA finds the value of this range parameter. *Avoid Barriers* and *avoid beacons* will slow the agent down if it gets too close to land or another entity respectively, otherwise, they will allow the controller to go full speed. The GA also finds how close an obstacle needs to be and how much to slow down. While attacking a target, *avoid beacons* suppresses itself if the obstacle that would

slow the agent down is, in fact, the target the predator is attacking.

Each behavior outputs a value corresponding to the magnitude and direction of an effector's action. For example, speed values between 0 and 1 determine how fast the predator moves. Thus each behavior outputs a vector for each effector and a weighted vector sum of these behavior-output-vectors determines the agent's observed action. Behavior priorities determine the weights in the vector sum. For example, the closer one agent gets to another agent, the greater the increase in the weight for *avoid beacons* resulting in a greater push away from the other agent [1, 7].

The prey unit in this experiment has two goals. First, follow the non-traversable tiles. Second, follow boundaries without running into any non-traversable tile or any other entities in the world. If the prey detects an attack, it can suspend its boundary following task and evade the predator, however at any point avoiding non-traversable tiles takes a higher priority. Once a prey determines it is under attack, it will do whatever is possible to avoid contact with the predator outside of running into non-traversalbe tiles. Two ways determine if a prey is under attack. First, if any agent comes within a certain distance of the prey, the prey assumes it is under attack. Second, the prey will attempt to predict where all other entities are going and if they are on a collision course, the prey will attempt to change its path in such a way that the projected collision will never happen. Once the prey notices it is no longer under attack, it will go back to following boundaries.

3.3 Simulation Process

One evaluation consists of a number simulation trials over a length of time with discrete time steps. The length of time and number of time steps are given in milliseconds. A 5 minute simulation at $50ms$ length time steps is equivalent to $6,000$ steps. Each trial restarts all the entities in the environment at one of three new locations and starts the simulation over again. Thus, the GA will not tune the controller to a certain starting location.

All agents move according to their set speed, multiplied by the length of the time step. Agents have access to all the information within the simulation and there is no noise. We are working on games and therefore these perfect sensors do not provide an issue.

4 Methodology

Our GA evaluates population members in parallel across multiple machines. The ability to run multiple simulations drastically speeds up the run time of the GA. We use a master/slave model with one server and multiple clients or nodes as shown in Figure 3. The server sends an individual to be evaluated to each of the clients. Each client then starts up the simulation and passes the individual to the controllers that are being evolved. The simulation begins after the agents are properly initialized. At the simulation end, the controller measures its fitness and hands it to the simulation. The simulation combines these fitnesses to describe that individual's final fitness. The simulation sends that fitness back to the GA and the simulation receives a new individual. The process continues until all individuals have been evaluated.

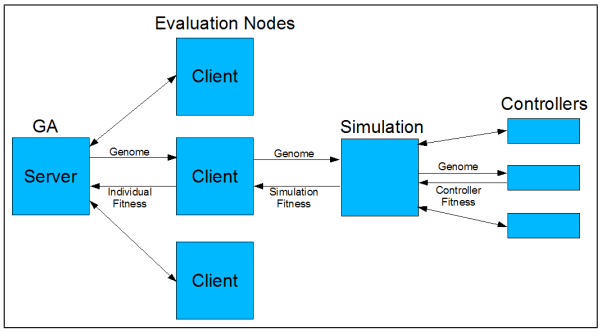

Figure 3: GA Implementation

4.1 Parameters

We ran the GA for the predator experiments over 20 generations with a population of 50. The probability of crossover is 0.7 with a 0.01 probability that a bit will be flipped. The GA used fitness proportional selection to choose parents. Offspring doubled the size of the population and we kept the best n, where n is the population size, for further processing [3]. The prey experiments differed by using a mutation rate of 0.1 and only had a population of 20.

4.1.1 Encoding

Table 1 shows the encoding of the predator genome and consists of 118 bits used for tuning. The last three parameters, *moveToWeight*, *moveToC1*, and *moveToC2*, all tune an inverted sigmoid function. This function's output is used as a weight for *track target*. If the target is far away, *track traget* has less weight and the predator is less inclined to move towards the prey. The sigmoid function was chosen to allow for a smooth transition from wandering to tracking. For the prey, we used 51 bits as shown in Table 2. The distances use *TooClose* as a baseline value, and the next farthest is greater than the previous by a certain offset. For example, if *TooClose* was 20 yards and *Close* had a value of 10, then an entity would be considered close at 30 yards. The same applies to the speed values.

Parameter	Bits	Description
Far	8	Threat is Far
Near	8	Threat is Near
Close	8	Threat is Close
TooClose	8	Threat is Too Close
turningRate	4	Rate to turn the prey
visionRange	3	Finds if something near prey
normalSpeed	7	Prey travels at normal speed
fastSpeed	7	Prey travels at fast speed

Table 2: GA Parameter Encodings for Prey

Parameter	Range	Bits	Description
turnRate	0-0.128	7	Rate agent turns when stuck for *avoid barrier* and *avoid beacons*
minSpeedTillTurn	0-0.128	7	Speed at which agent gets stuck for *avoid barrier* and *avoid beacons*
changeHeadingTime	0-10240	10	Random interval for new direction in *wander*
maxBarrierSpeed	0-0.512	9	Max speed that *avoid barrier* will allow
maxBarrierDistance	0-256	8	Distance that *avoid barrier* looks at
stopBarrierDistance	0-256	8	Distance that the agent stops for barriers in *avoid barrier*
BarrierFOV	0-128	7	Arc in front of agent that *avoid barrier* looks at
maxBeaconSpeed	0-0.512	9	Max speed that *avoid beacons* will allow
maxBeaconDistance	0-256	8	Distance that *avoid beacons* looks at
stopBeaconDistance	0-256	8	Distance the agent stops at for beacons in *avoid beacons*
beaconFOV	0-128	7	Arc in front of agent that *avoid beacons* looks at
moveToWeight	0-102.4	10	Effects distance at which predator starts tracking prey in *track traget*
moveToC1	0-102.4	10	Effects distance at which predator starts tracking prey in *track traget*
moveToC2	0-102.4	10	Effects distance at which predator starts tracking prey in *track traget*

Table 1: GA Parameter Encodings for Predator

5 Experiments

In this study, we performed four experiments to test various aspects of the controllers as well as the performance. The four experiments can be broken into two categories: wandering and attack. For every experiment, we hand-tuned a controller to find what was the best overall behavior. We then run the simulation and find the hand-tuned controller's fitness in each of the four experiments.

5.1 Wandering

Wandering tests to see how much area the controllers could cover with no target on the map and the area covered determined fitness. We overlaid a grid on the environment map and added a point for every new grid cell covered by the agent and removed one hundredth of a point for every time cycle the agent remained in that cell..

With this scoring system in mind, we tried two grid sizes. The fine grid has cells sized the same as the tiles (Figure 4). This makes it easier for controllers to discover many new cells while receiving a smaller penalty. We hypothesized that this would cause robots to not explore as much as there is a wealth of new tiles in only a small range.

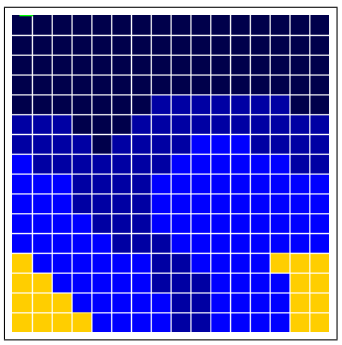

Figure 4: Fine Grid

To counteract this local exploration, a coarser grid was designed that that would force the controllers to strike out in new directions (Figure 5). Staying local would incur larger penalties to fitness to the agent.

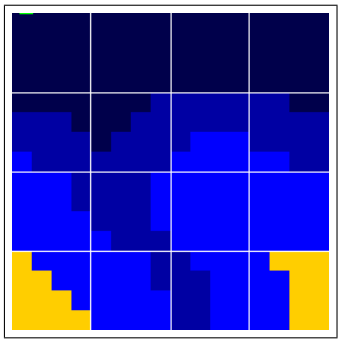

Figure 5: Coarse Grid

We ran each simulation with 27 agents placed around the map, all sharing the same genome, for 10 minutes at $50ms$ time steps. An agent crashing into either a barrier or another agent would receive a fitness of zero.

5.2 Single predator attack strategy

A lone predator's accomplishments provide a baseline for evaluating multiple-predator attack strategies. One predator cycles between three starting locations on one side of the map and the prey starts at one location on the other side of the map. One evaluation consists of six simulation trials. An empirically chosen constant minus the distance between the predator and prey equals the fitness for a single trial as given below.

$$\text{fitness} = C - \text{Distance(predator, prey)} \qquad (1)$$

A collision between the predator and the prey receives a maximum score. If the predator dies before it reaches the prey, it receives zero points. If the prey crashes before the predator could arrive, the predator receives zero points also. The predator does not get rewarded for mistakes on the prey's behalf. The downside lies in that the predator receives no reward if the predator causes the prey to crash.

5.3 Dual predator attack strategy

The setup for dual predators follows the single predator strategy except that the each predator gets a different starting point at each trial. Thus, if during the first trial starting points 1 and 2 were used, in the next points 2 and 3 would be used. The highest fitness of the two predators determines the overall fitness for that trial. Only one predator has to succeed for the whole team to succeed.

5.4 Prey defense strategies

We designed the prey experiments follows: First, the evaluator finds the baseline fitness by running the hand-tuned predator against the hand-tuned prey. Next, we evolved the prey against the hand-tuned predator and also evolved the predator against the hand-tuned prey. We used the evolved prey and the evolved predator to evaluate against each other. Finally, the GA evolved a new prey against the evolved predator to find the final fitness results.

6 Results

6.1 Wandering

The results of fine grid wandering and coarse grid wander are given in Figures 6 and 7 respectively. Both experiments were done over five runs each with a different random number generator seed. Table 3 shows the averages of maximum fitnesses for the Hand-Tuned controller and GA-Tuned controller as well as percent improvement.

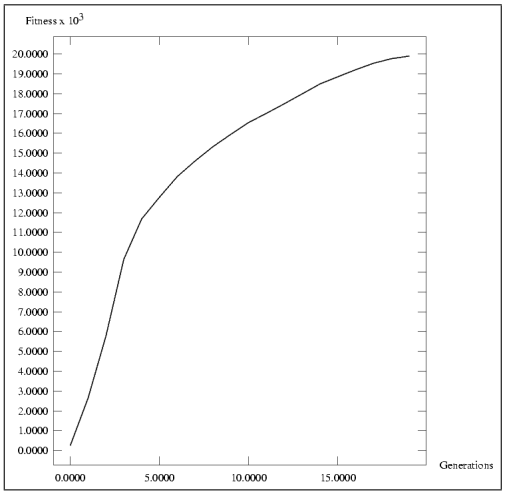

Figure 6: Fine Grid Fitness

We find it interesting that the fine grid wandering produced better results although not enough runs were performed to find these results statistically significant. We cannot say what is optimal performance with either grid. Large areas of the map are inaccessible due to barriers and an agent cannot cover a new tile every time tick because they cannot travel that fast. Despite this, the evolved controller fitness increased around 400 percent, a vast improvement.

The nature of the each differently evolved controllers from each seed varied greatly. In either fine grid or coarse

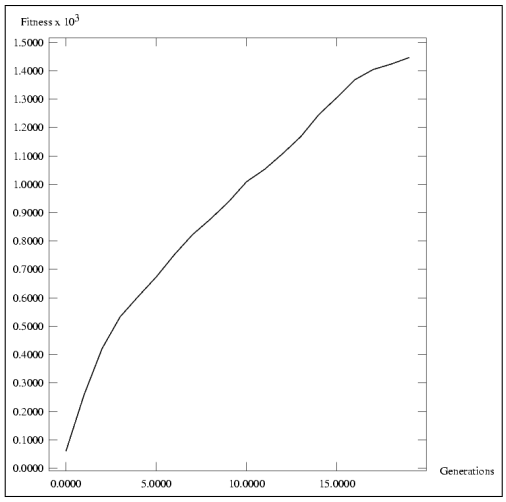

Figure 7: Coarse Grid Fitness

grid wandering, some controllers would dart for long periods of time, some would loop in circles and slowly work across the map, and others would fall somewhere in between. No one style correlated to any particular ranking in the set of trials. Choosing the best could just be picking the most aesthetically pleasing.

6.2 Attack

The hand-coded controller never completed an attack in the simulation. The controller worked fine in simpler scenarios, but could not deal with more complicated routes to the prey. This controller uses a more passive, lay-in-wait type of strategy. In the future, the prey could take a more aggressive stance in defense by destroying predators that are attacking, thus the hand-coded controller tries to look as innocent as possible, waiting until the last second to strike. The GA quickly exploits the fact the prey cannot strike back and the predator moves as quickly as possible towards the prey. The maximum possible fitness for a run is 30,000: 5,000 for a successful attack over six trials (Figure 8). This weakness lets the average fitness gets quite close to the optimal fitness and the best individual achieves this fitness.

Contrary to our expectations, the single predator's fitness overtakes the dual predator fitness in the last few generations. Our prior considerations held that multiple lines of attack would increase the chances of success. What we can observe tells us that as the GA makes the controllers more and more aggressive, both predators reach the prey more often. Because both are heading towards the same target, they often get in each others path and slow down to avoid hitting each other. This courtesy among predators means the prey can keep away from the predators thus lowering their total fitness. Therefore, almost the entire time the two predators do better than one, but get too competitive as they get better.

6.3 Defense

Table 4 shows the results of various prey versus different predator configurations by giving the average fitnesses of four runs through the simulation each with a different seed.

Average Fitness	Hand Tuned	GA	Improvement
Fine Grid	4679.24	19894.47	425.16
Coarse Grid	431.05	1725.03	400.19

Table 3: Average Fitness for Fine and Coarse Grid

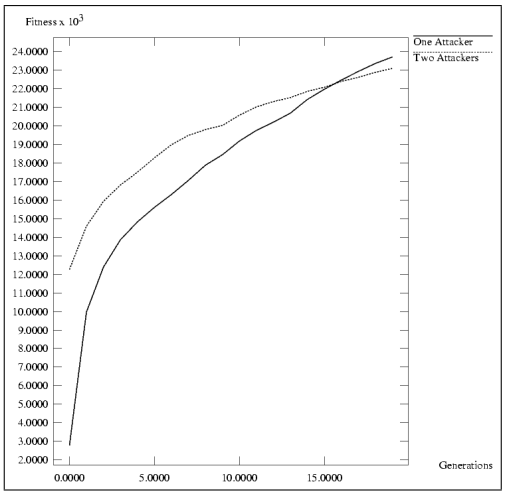

Figure 8: Comparative Attack Fitness

The evolved prey and predators use the maximum fitness individuals from a run through the GA.

Experiment	Fitness	Improvement
Def. Prey vs. Def. Predator	90310	N/A
Ev. Prey vs. Def. Predator	199819	+221%
Def. Prey vs. Ev. Predator	43476	-48%
Ev. Prey vs. Ev. Predator	27671	-70%
ReEv. Prey vs. Ev. Predator	193972	+214%

Table 4: Average Fitness for Prey against Predators
Avg: Average, Ev: Evolved, ReEv: Re-Evolved

As expected, the evolved prey did very well against the hand-tuned predator and the evolved predator did very well against the hand-tuned prey. Surprisingly, the evolved predator had better success against the evolved prey. We can explain this by the fact that the prey learns a specific evasive tactic to avoid a specific attacker. The similar results between evolving the prey against the evolved predator and and evolving the prey against the hand-tuned predator shows this fact.

The GA successively created an evasive tactic for a prey against a predator. However, the poor performance against a different opponent than the one it evolved against illustrates that the GA over specializes. This result can be expected since there was no history kept of the past strategies used against different opponents.

7 Discussion

Our current work has shown many abilities of genetic algorithms in tuning agent controllers. First, effective performance can be found even with simple behaviors. The resulting performance exceeds by far those controllers designed by extensive hand-tuning.

Second, the GA produces these results much faster. A day of meticulous nudging and tweaking takes only a few minutes for the GA to start exceeding the hand-tuned controller and an hour to finish with a five times gain in performance.

Third, the GA serves to highlight the difficulty in evaluating robotic performance. There exists a cyclic dependency between predator and prey. To program a good predator and evolve it, there has to be a challenging prey and to program that prey, a good predator is needed. Without either, the GA quickly exploits weaknesses of either the predator or the prey. We believe that co-evolution can solve this problem.

Fourth, the GA will exploit any flaw in the design of the scenario, the prey, or the predator. The GA does not strive to make an intelligent or believable predator, it can only maximize the fitness, Evaluators cannot simply judge intelligence and believability. As such, additional methods need to be employed to nudge the agent in the right directions and a more versatile controller needs to be designed.

Last, there are many directions in which the prey can become better. Greater predator detection efficiency would allow the prey more time to plan, thus spending less time trying to lose the predator. With co-evolution, we should be able to develop an evasive tactic that works for an entire generation of attackers.

7.1 Conclusions and Future Work

We have demonstrated an evolutionary approach for tuning Behavior-Based Controllers that provide preliminary steps for using co-evolution. We experimented in a predator-prey scenario in which a predator is evolved against a hand-tuned prey and prey that evolves against a hand-tuned predator and an evolved predator. We find that the GA-tuned controller outperforms the hand-tuned controller by 400 percent. The GA over specializes both the predator and the prey to their opponent and we expect using co-evolution would train the predators and prey against a variety of opponents, thus allowing for more general knowledge. We believe evolving efficient predators and prey is a natural candidate for co-evolution. The first run of this experiment only had predators evolving against a static prey and vice versa. We could provide a variety of interesting avenues to explore using co-evolution of predators and prey.

As one of our reviewers pointed out, we may be able to increase the performance of two predators if we add a communications channel to facilitate the emergence of cooperative behavior. In this particular example, if one predator gets too close to the other predator, a message can cause the

other predator to slow down thus allowing an unimpeded attack by the first predator. We believe that allowing communication will allow more sophisticated plans through synchronization and teamwork.

Additional forms of learning are also needed. A GA can only optimize based on an evaluator function, but due to the multi-modal nature of the search space, many different personalities can arise as was seen in the wandering behavior. Biasing though case-injection or parameterizing the behavior based on control from a human player could push the GA in the right direction. Alternatively, an existing controller can be nudged into a more human-like behavior by recording human-play.

8 Acknowledgments

This material is based upon work supported by the Office of Naval Research under contract number N00014-03-1-0104.

Bibliography

[1] Ronald Arkin. Motor schema based navigation for a mobile robot: An approach to programming by behavior. In *Proceedings of the IEEE Conference on Robotics and Automation*, pages 264–271, 1987.

[2] Ronald C. Arkin. *Behavior-Based Robotics*. MIT Press, CA, 1998.

[3] Larry J. Eshelmann. *The CHC Adaptive Search Algorithm: How to Have Safe Search When Engaging in Nontraditional Genetic Recombination*. Morgan Kauffman, 1990.

[4] Kerry Gruber, Jason Baurick, and Sushil J. Louis. Evolution of complex behavior controllers using genetic algorithms. In *Proceedings of the International Conference on Artifcial Intelligence*, Las Vegas, Nevada, 2002.

[5] Sushil J. Louis and Gan Li. Combining robot control strategies using genetic algorithms with memory. In *Lecture Notes in Computer Science, Evolutionary Programming VI*, pages 431–442.

[6] Maja J. Matarić. Behavior-based control: Examples from navigaton, learning, and group behavior. *Journal of Experimental and Theoretical Artificial Intelligence*, 9(2–3):323–336, 1997.

[7] David Payton. Internalized plans: A representation for action resources. In *Robotics and Autonomous Systems*, volume 6, pages 88–103. 1990.

[8] A. Schultz, J.J. Gregenstette, and W. Adams. Roboshepherd: Learning a complex behavior. In *RoboLearn '96, The Robots and Learning Workshop a FLAIRS*, May 1996.

[9] Karl Sims. Evolving 3d morphology and behavior by competition. In P. Maes R. Brooks, editor, *Artifical Life IV*, pages 29–39. MIT Press/Bradford Books, 1994.