Final Project Report: Predator-Prey Evolution

Caitlin Donahue, Julia Kroll, and Valerie Lambert

June 8, 2015

1 Project Goal

Our goal was to create a simulation of wolves and sheep, where wolves try to eat sheep and sheep try to eat food while staying away from the wolves. Our ultimate goal was to coevolve the wolves and sheep, but our primary goal was to create wolves who evolve as they try to catch sheep that are hard-coded to evade the wolves.

We designed a map for the agents (wolves and sheep) to occupy, organized in a grid layout where each square of the grid can contain one agent. Squares contain different amounts of food (0 through 3 units, randomly selected). Agents are able to move one square at a time, in the directions forward, backward, left, and right. The grid has defined rectangular dimensions, but it does not have walls, so if an agent is on an edge and moves farther in the direction of the edge, it loops around to the other side of the map. Therefore, it is impossible for an agent to be trapped against a wall or in a corner.

One agent moves at a time, with the wolves moving first and the sheep moving afterward. Any agent who tries to move onto a square already occupied by another agent stays in its original square. A sheep who moves onto a square eats one unit of food if that square contains food. A wolf catches a sheep by moving onto a square that a sheep occupies. The wolf then eats the sheep and the sheep is removed from the map. With this setup, we hoped to implement an evolutionary algorithm that would allow the wolves to evolve to eat as many sheep as possible in a given number of moves.

2 Our Evolutionary Algorithm

2.1 Representation

To solve our problem we decided to use a GP with distinct subpopulations of wolves. Each subpopulation contained 250 individuals, and each individual participated in several trials, each trial with a random member of each other subpopulation to determine their fitnesses. We set up our GP such that terminals are an integer mod 4, and the output of the tree is one action. A value of 0 corresponds to the 'move forwards' action, a value of 1 corresponds to 'move right', a value of 2 corresponds to 'move backward,' and a value of 3 corresponds to 'move left.'

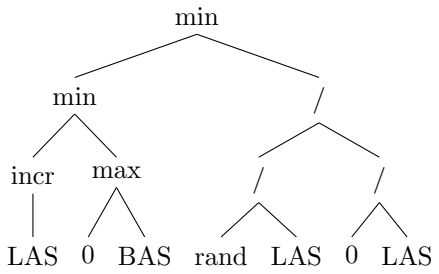
We defined nine different terminals that our GP uses. Four terminals are the constant values 0 through 3, another terminal outputs a random value from 0 to 3, and the other four terminals are sensors that output a value based on whether a corresponding square contains another sheep or wolf. For example the LeftAgentSensor outputs 0 if the square to the left of the wolf contains another wolf, 1 if the square to the left of the wolf contains a sheep, and 2 if the square to the left of the wolf does not contain an agent. There are three other sensors that correspond to the squares in front of, behind, and to the right of the wolf (ForwardAgentSensor, BackwardAgentSensor, and RightAgentSensor). We also tried implementing sensors that output the food amount of a certain adjacent square. However, including these terminals actually made it more difficult to evolve effective wolves because we never implemented sheep that changed their behavior based on trying to get food.

For our GP's functions, we defined the basic operators: addition, subtraction, multiplication, and division. To handle divide-by-zero errors, in the division function we defined anything divided by zero to be 3. We also implemented max, min, increment, decrement, and if/then/else functions, as well as a

function that evaluated one of its two children at random. Each function outputs an integer mod 4.

To determine the fitnesses of a set of individuals during a certain run, we ran a trial of our problem with one individual from each subpopulation, evaluating the trees of each individual to determine their moves. At the end of the trial, we counted up how many sheep the wolves ate collectively and divided that number by the number of wolves on the board. Intuitively, this is acting as if each wolf is sharing each sheep it catches with each other wolf equally.

To initialize our population, 50% of trees are created using ECJ's FULL algorithm, and 50% of trees are created using ECJ's GROW algorithm. The GROW algorithm can generate a random tree of up to depth 9, while the FULL algorithm generates random trees of depth 2 through 9. An example of one initialized individual is below. For legibility we have abbreviated backward-agent-sensor as BAS and left-agent-sensor as LAS, as well as / for division.



2.2 Parameters

Our GP uses tournament selection with tournament size 10.

Crossover is performed by selecting a random node in each tree, and swapping the subtrees of those two nodes. The pipeline defined in ECJ attempts to do this once, and if children with illegal depths are produced, no crossover occurs.

Mutation is performed by selecting a random node in the tree, and then creating a random subtree to replace that node. The pipeline defined in ECJ tries to produce a tree of legal depth once, otherwise no mutation occurs.

We defined our trees to have a total max depth of 17 when mutating and crossing over. When selecting nodes for crossover and mutation, terminals were selected with 10% chance and non-terminals selected with 90% chance.

We decided to have three subpopulations to represent three wolves, with each subpopulation having a population size of 250.

In addition to the pre-existing parameters, we created new parameters for agent intelligence. For each species, we programmed multiple levels of intelligence that enable individuals to gather different amounts of information about the squares around them. The wolves have two levels of intelligence. On level 0 they move randomly, and on level 1 they evolve and move based on a GP. The sheep have five levels of intelligence. On level 0 they move randomly. On level 1 they sense whether each of the four squares adjacent to them contains a wolf. On level 2 they have all of the senses of level 1, plus they also can sense diagonals. On level 3 they retain everything from level 2, plus they can sense an extra square in each of the four cardinal directions (so two squares front/back/left/right, and one square diagonally). On level 4, the sheep evolve using GPs.

For our problem-specific parameters, we decided to use a map size of 10 by 10, containing the 3 wolves and 30 sheep. Each trial consisted of 40 turns (one turn representing one action from each wolf and each sheep).

2.3 Reasons for Parameters

The above parameters (excluding agent intelligence) were already set by ECJ, and they seemed to be working well, so overall we did not see a reason to modify them. We did experiment with various maximum and minimum tree depth settings. However, while we looked at the trees and saw that they were larger or smaller corresponding to our modified settings, the wolves' performance was the same no matter the tree depth. We concluded that changing the depth produced trees that had more or less structural complexity but produced basically the same logic. As a result, we decided to use the original depth settings.

2.4 Best Individuals

We ran our algorithm with the stated parameters and sheep intelligence level 2 for 1000 generations. The trees of the best individuals from each subpopulation are shown in the attached file best_individual_trees.pdf. For our experiments, we used a maximum tree depth of 19. However, the resulting trees are too complex to display legibly, so the trees in the attached file were generated using a maximum depth of 10.

3 Hypothesis and Results

We hypothesized that as sheep intelligence increased, cooperation among wolves would also increase. When the sheep move randomly, the wolves do not need to cooperate at all, because the sheep do not sense wolves. Therefore, a lone wolf can easily catch an oblivious sheep. However, by sheep intelligence level 1, a sheep can move away from a wolf who moves next to it, and by level 3, a sheep can avoid moving next to a wolf. As a result, when the sheep have higher intelligence, a wolf can rarely catch a sheep without the help of another wolf. (The one exception is when two or more sheep are next to each other and a farther sheep blocks a closer sheep from escaping.) In order for wolves to eat sheep and thus gain fitness, we predicted that they would cooperate to trap sheep, who can escape one wolf but less easily two or more wolves.

After running our algorithm with random wolves and evolving wolves for each sheep intelligence level, we have found that at every level of sheep intelligence, evolving wolves perform significantly better than random wolves.

For sheep intelligence level 0, where the sheep were moving randomly, random wolves gained an average best fitness of 8.587 in the last 100 generations, while evolving wolves earned an average best fitness of 10.000 (eating all of the sheep). This means that with our problem parameters, and all agents acting randomly, we can expect that most sheep are going to be eaten, but evolving wolves were able to reliably eat all of the sheep.

For sheep intelligence level 1, with sheep moving away from a wolf next to it, random wolves had an average best fitness of 7.683 in the last 100 generations, while evolving wolves had an average best fitness of 9.623. With slightly smarter sheep, random did not do as well, because it guaranteed that if a wolf moved next to a sheep (and that sheep had room to move away), the sheep would try to run away. This didn't, however, prevent a sheep from moving next to a wolf. Thus, wolves could rely somewhat on chance for a sheep to randomly move next to them, but through cooperation they would be able to catch more sheep.

For sheep intelligence level 2, sheep were able to avoid stepping next to a wolf if the wolf was in a square diagonal to them. Random wolves had an average best fitness of 6.228 in the last generations, while evolving wolves had an average best fitness of 8.960. Now a wolf would only be able to catch a sheep by itself if the sheep was two squares away (not diagonals) stepping on a square next to them. Thus both groups of wolves catch fewer sheep than before, but evolving wolves are still able to catch most of the sheep.

For the final sheep intelligence level 3, where sheep avoid stepping next to a wolf, random wolves had an average best fitness of 4.924, while evolving wolves had an average best fitness of 8.063. Here, it is obvious to see that random wolves are only benefiting from sheep colliding into each other, and only

through random, unintentional cooperation can a lone wolf catch a sheep. Random wolves only caught about half of the sheep, but evolving wolves were able to learn to catch the majority, usually leaving 6 or so sheep alive at the end, out of 30 total. We believe that it is not a stretch to say that this good performance is due in part to cooperation between wolves.

These results support our hypothesis that wolves will develop cooperation to catch more sheep. The random wolves provide a baseline for the high sheep intelligence levels that tells us how often random wolves cooperate by chance, and how often sheep are caught because they are trapped by other sheep. Comparing our evolved wolves to the random wolves, then, provides evidence that the wolves were evolving to cooperate.

4 Thoughts on the Project

Overall we achieved what we originally hoped to accomplish. We have a working model of wolves and sheep moving and reacting to one another, and the evolved individuals are able to eat more food and consequently gain a higher fitness than individuals who move randomly. We would have liked to spend more time exploring coevolution with both species evolving, but we think we chose a practical route by focusing on having the wolves evolve against hard-coded sheep, in order to make a polished finished product before progressing to two-species coevolution.

Our greatest challenge was choosing GP software and then using the software to implement our own GP. The software that we chose, ECJ, is extremely complex and contains a large variety of features that were irrelevant to us. It was a challenge to sift through the many components of ECJ and identify which few features would enable us to create our evolving wolves and sheep. Then once we discovered the parts of ECJ that we wanted to use, we had to figure out how to put together the pieces and implement/extend the appropriate classes. The only solution to this problem was studying the docs, manual, and provided examples, and putting time into getting our code running. As we programmed a GP of our own and made implementation and design choices along the way, we gained more knowledge of ECJ and we became familiar with the details of GP nodes, structure, and evaluation.

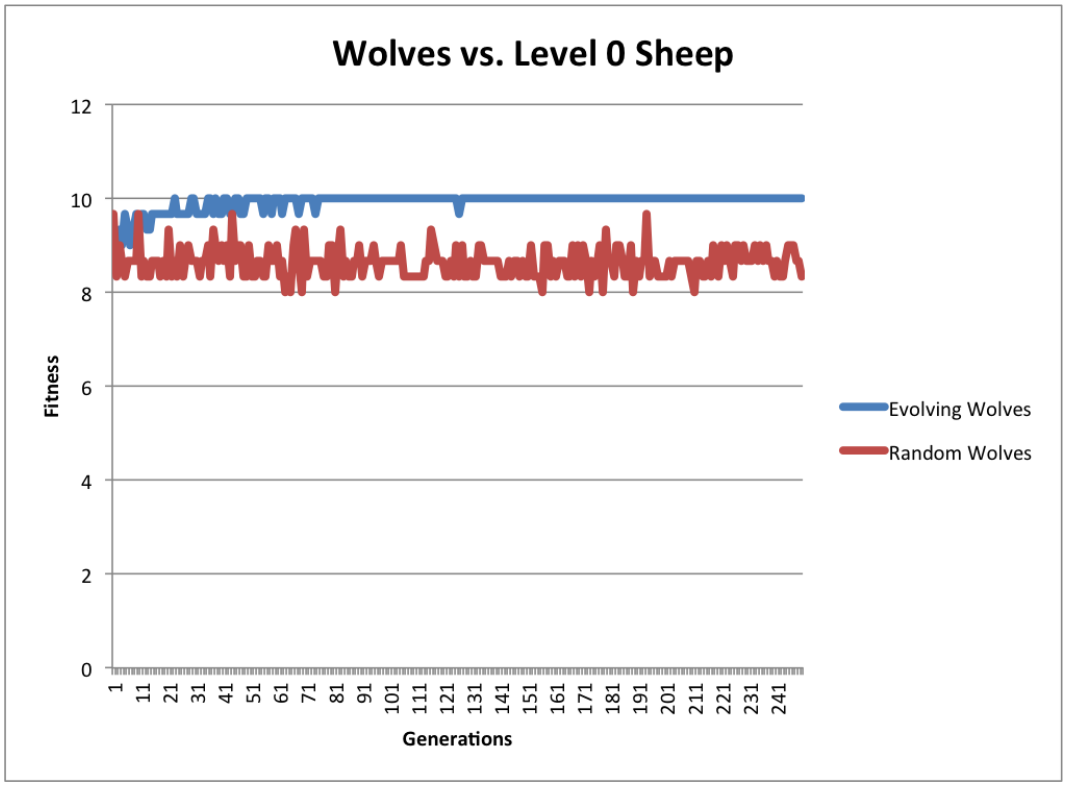
The most exciting aspect of the project was once we had evolution working and we could finally run trials with our own choice of parameters. This was the point when our previous efforts became meaningful and we could examine how the wolves were evolving, looking at the trees and stepping through individual moves to study the often surprising strategies that the wolves evolved to cooperate and capture sheep.

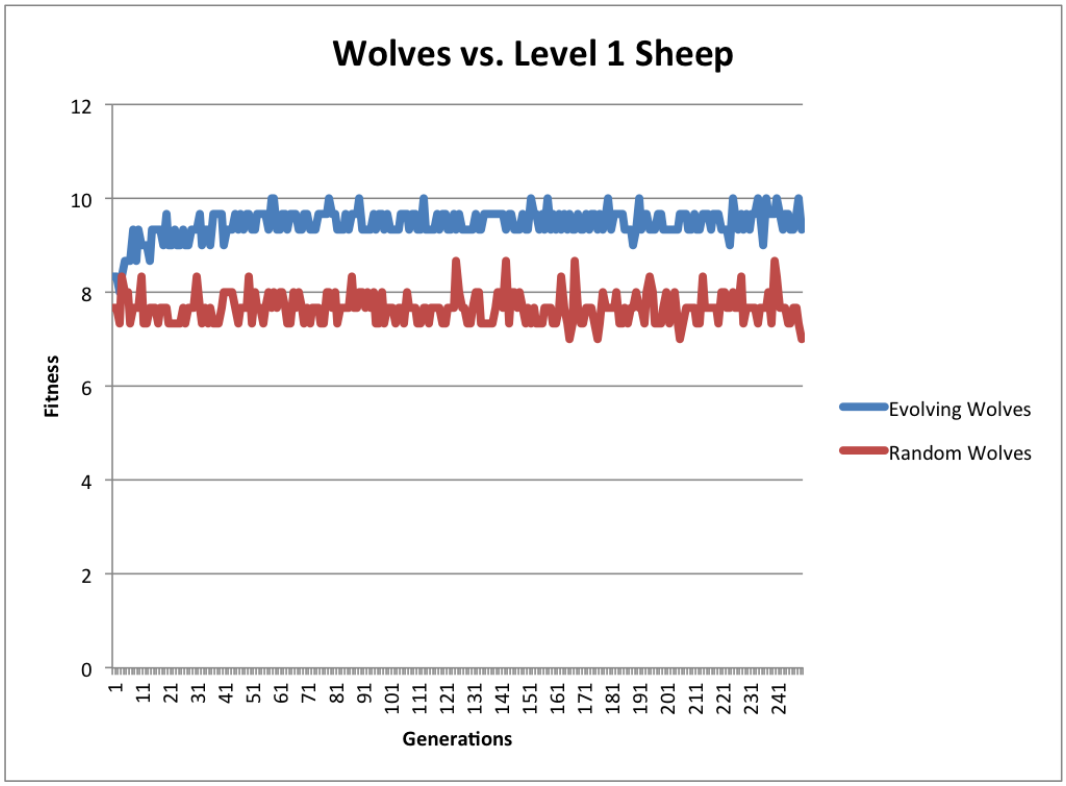
5 Data/Graphs

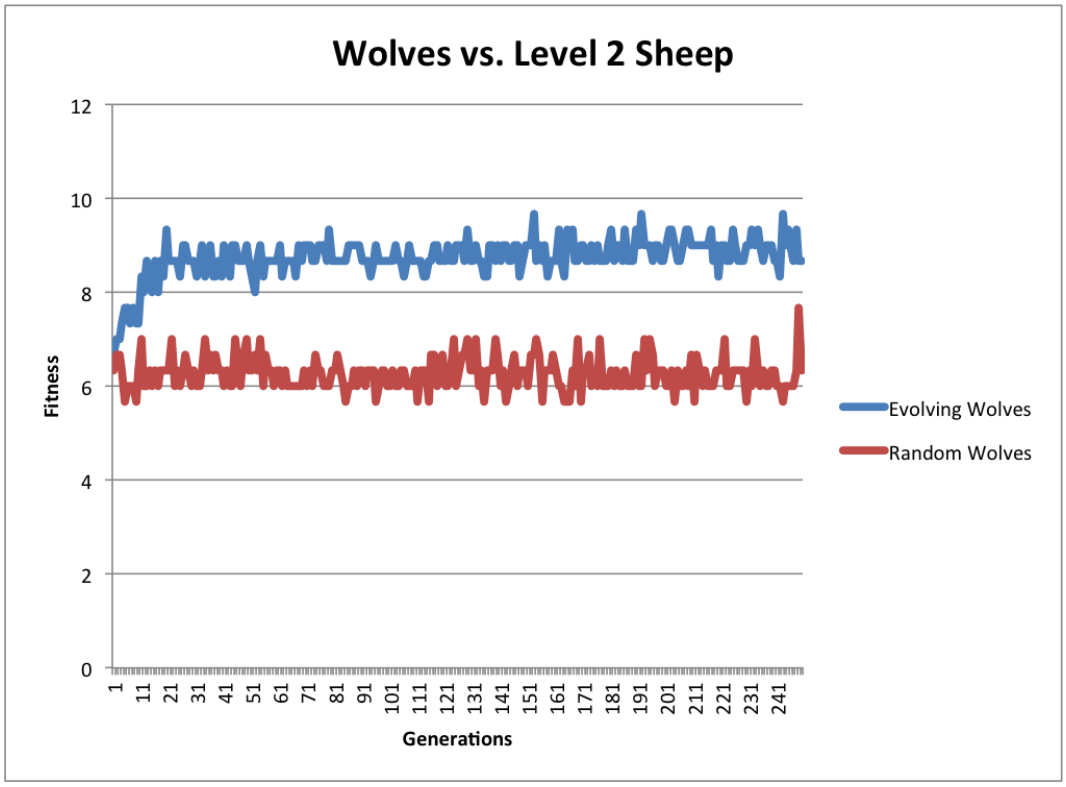
Table 1 contains the best fitnesses found over an entire run with varying sheep intelligence levels for both random and evolving wolves. This data, as well as the data presented in the subsequent graphs, clearly shows that by using our GP we can evolve wolves that eat sheep better than random.

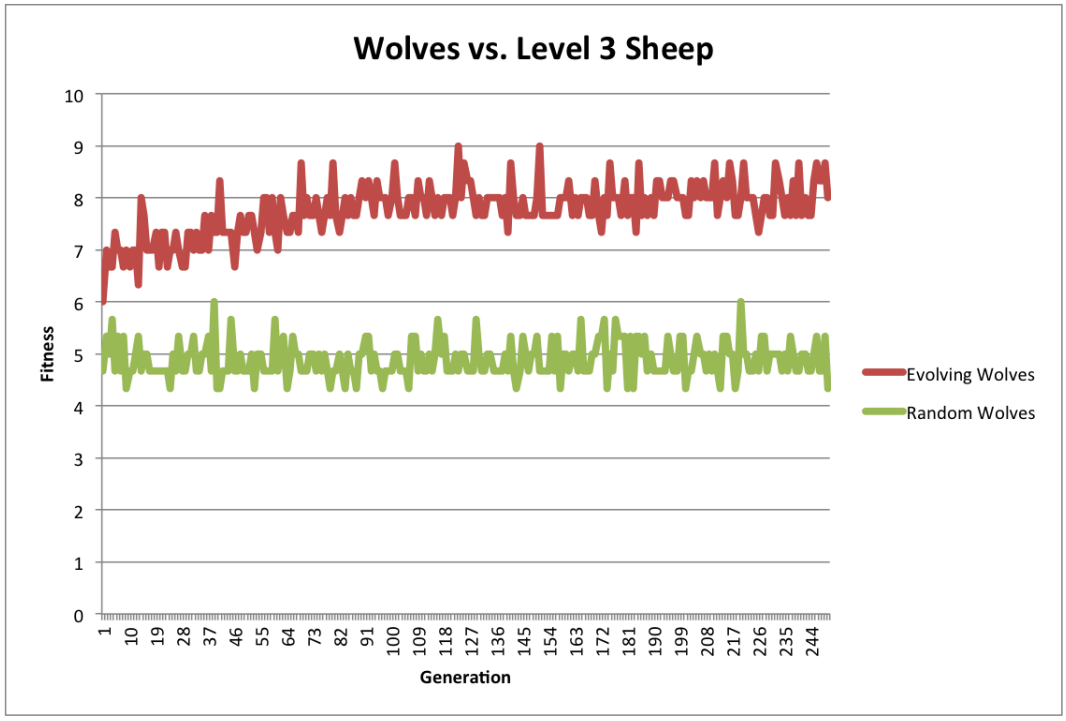
Table 1: Average best wolf fitness over the last 100 generations

	Sheep level 0	Sheep level 1	Sheep level 2	Sheep level 3
Random Wolves	8.587	7.683	6.228	4.924
Evolving Wolves	10.000	9.623	8.960	8.063
Percent Increase Random → Evolving	16%	25%	44%	64%


Wolves vs. Level 0 Sheep
Fitness
Generations
Evolving Wolves
Random Wolves


Wolves vs. Level 1 Sheep
Fitness
Generations
Evolving Wolves
Random Wolves


Wolves vs. Level 2 Sheep
Fitness
Generations
Evolving Wolves
Random Wolves


Wolves vs. Level 3 Sheep
Fitness
Generation
Evolving Wolves
Random Wolves