Math 378 – Final Project: Submarine Warfare
Chad Austin & Taran Shilling
2004-05-06

Overview

Submarine Warfare is an evolutionary system where "submarines" battle on a board. This system is a hybrid of structure and ideas from Sunburn, Symbots and Herbivore. Using some of the coding framework from Herbivore, the submarines are represented by GP-Automata with parse trees for making decisions, along with a string representing weapon allocations.

Unlike Herbivore, submarines are single points in 2D real space, instead of on a grid, and the submarines have an alterable direction. For weapons, they have forward and aft-torpedoes and our special addition: sonic pulse. Weapon effectiveness is a function of both angle and distance to the target. Subs have 8 actions choices: think, fire forward, fire aft, fire pulse, turn left, turn right, speed up, and slow down. Turns are always done in (pi/8) radians increments. Ships will always continue to move at the same speed as the previous turn unless their turn is used to alter their speed. For these experiments, only two submarines will be used in the battles and they will be placed randomly on the board.

Evolution is performed by selecting two pairs of subs in the population, executing combat between each pair, and copying the winners over the losers. The copied ships then have crossover applied to either their weapon allocation strings or the decision trees and single-point mutation applied to the weapon allocation strings or to the GP automata. This is discussed more fully in the Methods section below.

We hope to examine runs of Submarine Warfare and see what types of subs emerge, as well as any potentially interesting evolution paths, by running the Submarine Warfare software several times and plotting the action frequencies and weapons distributions in the resulting populations. We also hope to experience first-hand some of the difficulties and issues in using evolutionary algorithms to solve a complex problem.

There were challenges to overcome in developing this simulation but we were excited that eventually the software produced populations with significant evolutionary improvements. Surprisingly, we managed to obtain good data with several open doors for future work.

Rule Specifics

Submarines consist of a fixed-length weapon string that specifies the number of each type of weapon that this submarine has. 'F' in the string represents a single forward torpedo. 'A' and 'P' represent aft torpedoes and pulses, respectively. For example, FFFAAAPPPA has three forward torpedoes, four aft torpedoes, and three sonic pulses. While in battle, ammunition is limited. If a submarine only has four forward torpedoes and tries to fire them five times, the last fire command is ignored. (Specifically, it becomes a think action, which doesn't really help the submarine at all.)

Battles are performed in real two-space. Each submarine has a position, a direction specified in military degrees (0 degrees = north, 90 degrees = east), a current speed from 1 to 5, a damage rating, and the number of each weapon type available to them. At the beginning of the battle, the two submarines are placed randomly in a circle of some radius centered at the origin. They are given a random direction, a speed of 3, and zero damage.

Right away, the submarines take their initial actions. The battle updates every 0.1 seconds of "game time", and can last up to the battle time limit, usually about 200 seconds. Every battle update, submarines are moved by their propellers in the direction they are headed, and then take actions. Actions are determined by their GP-Automata controllers. GP-Automata, as we're using them, are essentially finite state machines with a parse tree decider that determines the transitions and actions taken. As input to the parse tree, the submarine is given the number of forward torpedoes, the number of aft torpedoes, the number of sonic pulses, and the distance (in integer distance units) and relative angle (in integer degrees) to the other submarine. This enables the decider to make decisions like "If the angle to the other submarine is positive, turn right." or "If the distance is less than 10 units, fire a sonic pulse."

There are three types of weapons: forward torpedoes, aft torpedoes, and sonic pulses. The torpedoes have a long range and act instantly in this implementation. If the enemy submarine is located within a certain angle of the torpedo's direction, it hits and does a fixed amount of damage. For example, if submarine one fires an aft torpedo and the other submarine's location is at 160 degrees (behind and slightly to the right) relative to submarine one, the aft torpedo would hit. We can change the effective angular range of a torpedo, but it is commonly set to plus or minus 22.5 degrees with respect to the submarine's orientation. Pulses are a different type of weapon entirely. They send out a sonic shock wave that damages the other sub, provided it is within a certain range. The amount of damage done depends on the proximity to the source ship. The damage done is specified by the following function: $\dfrac{pulseDamage}{1+(\frac{dist}{5})^2}$ where *pulseDamage* is a constant we can change and *dist* is the distance between the subs. The function above decreases exponentially with distance. To discourage excessive pulsing, we put a distance cap of 20 units on the pulse range.

The battle is over when the time limit has been exceeded or one or both of the ships has been destroyed. The winner is determined by the following conditions, in the order listed below:

```
if both ships died at the same time:
    battle is a tie
if one ship is dead:
    the other ship wins
if one ship has a greater hit accuracy than the other:
    it is declared the winner
otherwise:
    the battle is a tie
```

Note that at first we resolved ties due to timeouts by picking the more damaged ship as the loser. However, requiring hit accuracy to resolve ties drove the evolution much faster in the direction we wanted.

Methods

The software used to simulate submarine wars was developed primarily by us, but the integer stack parse tree was used courtesy of Dr. Ashlock. The GP-Automata part of the code was based on Dr. Ashlock's GP-Automata code from the Herbivore project, but designed to more closely fit our purposes.

The main program is responsible for gluing the various components together into several runs of a population evolving up to a fixed number of generations. It also is responsible for reporting statistics used to plot the evolution of our population. It is structured as follows:

```
for each of runCount runs:
    build a randomized population of populationSize size
    for generationCount generations:
        updateGeneration
        reportStatistics

updateGeneration:
    randomly split population into tournaments of size 4
    battle the first and second pairs of submarines, copying the winners over the losers
    perform crossover and mutation on the children
```

Submarine battle follows the game rules specified above. In order to reduce the number of ties, however, submarines are given three tries to win. If neither can win in three tries, both submarines are mutated and neither is declared the winner. This is designed to encourage wins.

Now on to the specifics of breeding:

Since the submarine consists of two components, the weapon string and the behavior automata, submarine mutation flips a coin and picks one and performs mutation on that. String mutation is done with one-point mutation, and automata mutation is done as follows:

10% of the time, the initial state is mutated
10% of the time, the initial action is mutated
20% of the time, a random transition is changed
20% of the time, a random action is changed
10% of the time, a new decider is randomly created and put in a state
10% of the time, two decider trees are crossed-over
10% of the time, two decider trees are switched
10% of the time, one decider is copied over another

Submarine crossover operates in a similar fashion. Half of the time the weapon strings are crossed-over with two-point crossover. The other half of the time the automata states are crossed-over, once again with two-point crossover.

The code is available by reference. Contact aegisk@iastate.edu for it.

Statistics

In order to see what is going on inside of our evolution, our program outputs a comprehensive set of statistics. We keep track of the number of times each action is taken and output their relative frequencies. We output the relative frequencies of weapon allocations as well. We plot average battle time as well as the average minimum, average average, and average maximum battle distances. This gives us an indication as to how close submarines like to get to each other. Finally, we graph hit percentages for each type of weapon. Our primary gauge of population fitness is the average battle time, and the secondary gauge is the summation of the hit percentages.

We ran the software many many times, varying most of our constants, then plotted the various statistics in the resulting populations, and observed several trends.

The software was run using a Mersenne Twister random number generator seeded with 98765.

Results & Discussion

Analysis Techniques for the Graphical Results

Although there is no fitness standard to compare against, the primary determination of fitness was the time to battle completion and the secondary fitness was the summation of the weapons hitting percentages. Therefore, our assessment of a population was primarily based on the time to battle completion. Populations that did not experience a significant decrease in average time to completion consisted of subs with similar capabilities to that of the initial random population. For populations that noticeable evolved, identification of their behavior was based on the action frequencies, weapons allocations, and hit percentages.

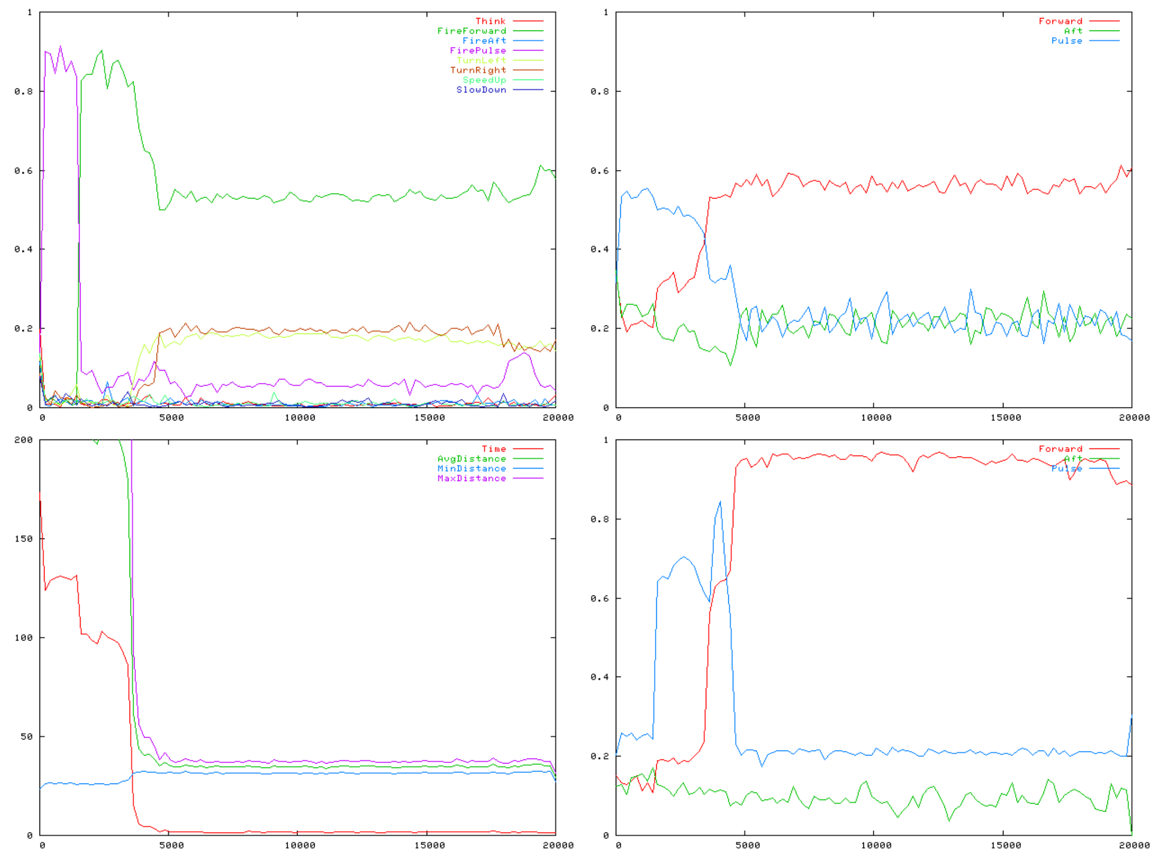

Figure 1: Base parameters, 20,000 generations, run 2

As seen in the population portrayed in Figure 1, the time to battle completion (red line in lower left graph) drops as the hit accuracy goes up (lower right graph). Although it may be difficult to exactly pinpoint the cause of increased hit accuracy, the decreased time to battle completion changes with changes in the usage of pulses. The pulse actions increase dramatically (purple line in the upper left graph) causing a temporarily increased fitness for the first fraction of the run. However, the pulses are only hitting the enemy ship a low percentage of the time (blue line in lower right graph). When the action frequency of using pulses decreased, the hit percentage of pulses increased, implying better weapons logic for when to use the pulse. Instead of firing pulses, the frequency of firing forward torpedoes substantially increased (green line in upper left graph) resulting in another decreased step in the time to solution. Then at around the 4000th generation, the frequency of forward torpedoes decreased and was replaced by increased turning frequencies. This change resulted in the final significant drop in time to battle completion and the increased forward torpedo hit accuracy. Shortly after the 5000th generation, the population mostly stabilizes and based on the action frequency, weapons allocation, and hit percentages for forward-torpedoes, it's clear that this population became specialized in forward torpedoes. Based on this pseudo-stable behavior, we're calling these subs "Forward-Shooters."

In Figure 2, again we observe some of the trends described for Figure 1 as well as the evolution of an "Aft-Shooter." Notice again that early increases in pulses are beneficial but the population can only reach full stability when specialization occurs in torpedoes and the turning logical becomes functional.

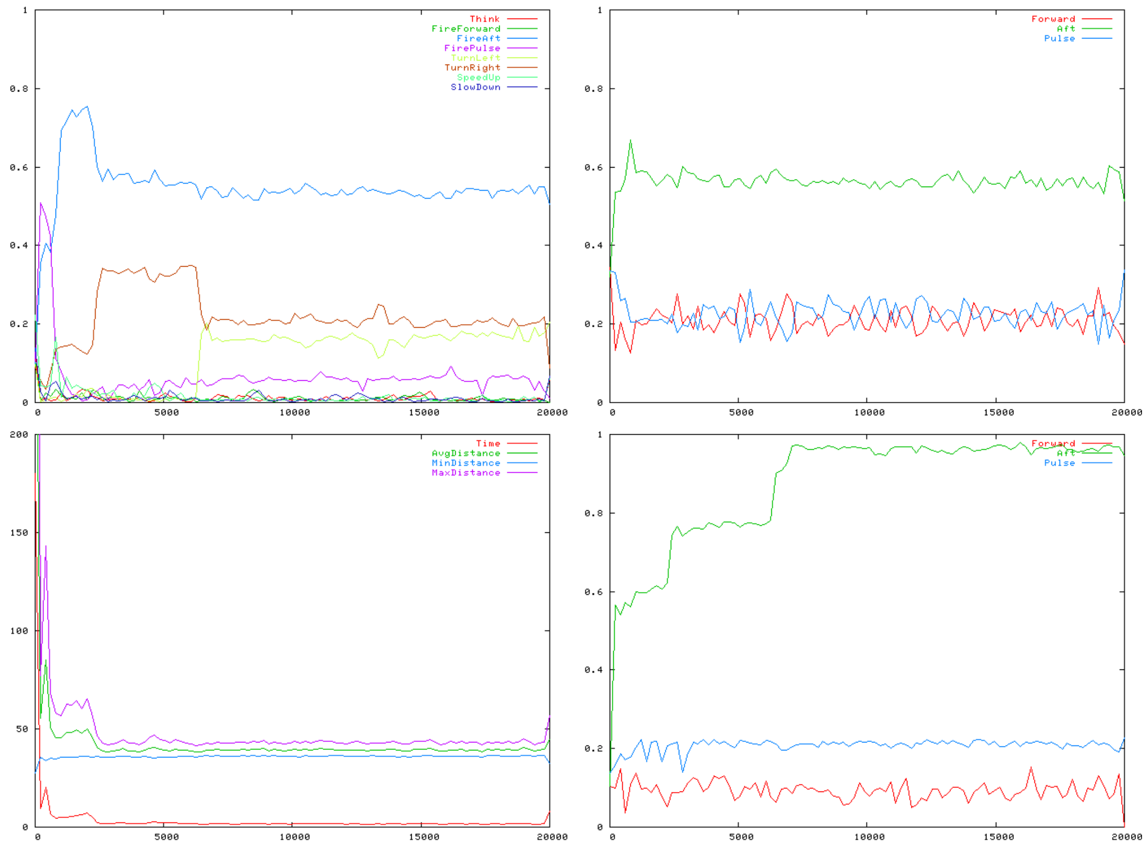

Figure 2: Base parameters, 20,000 generations, run 24

Varying Number of States in GP-Automata

In the base parameter case, we have 8 states in our GP-Automata. When lowering the number of states to 6, we see a drop in performance [average time to battle completion] from about 20% not evolving significantly in 1000 generations to about 50% not evolving significantly. When we lower it even further to 2 states, about 60% of the runs do not show a population that evolves significantly.

Varying Decider Tree Sizes

We also tried varying the initial and maximum sizes of the integer parse trees used as deciders in the GP-Automata. In the base case, the integer parse trees start with a size of 8 and a maximum size of 12. We lowered the initial decider size to 4 and the maximum size to 8. This increased our "failure rate" of populations evolving to be effective from 20% to 60%. Increasing the initial decider size to 12 and the maximum decider size to 20 reduced the failure rate to about 0%. We did not see any populations in our 10 runs fail to evolve.

Varying Mutation Rate

We doubled the mutation rate from the base case and observed no significant changes. However, in this run, we did observe a population that evolved high hit percentages for both pulses and forward torpedoes as shown in Figure 3. By further increasing the mutation rate to four times the base case, we again observed no significant changes. As expected, the action frequency showed more volatility for higher mutation rates, but the populations behaviors were unchanged.

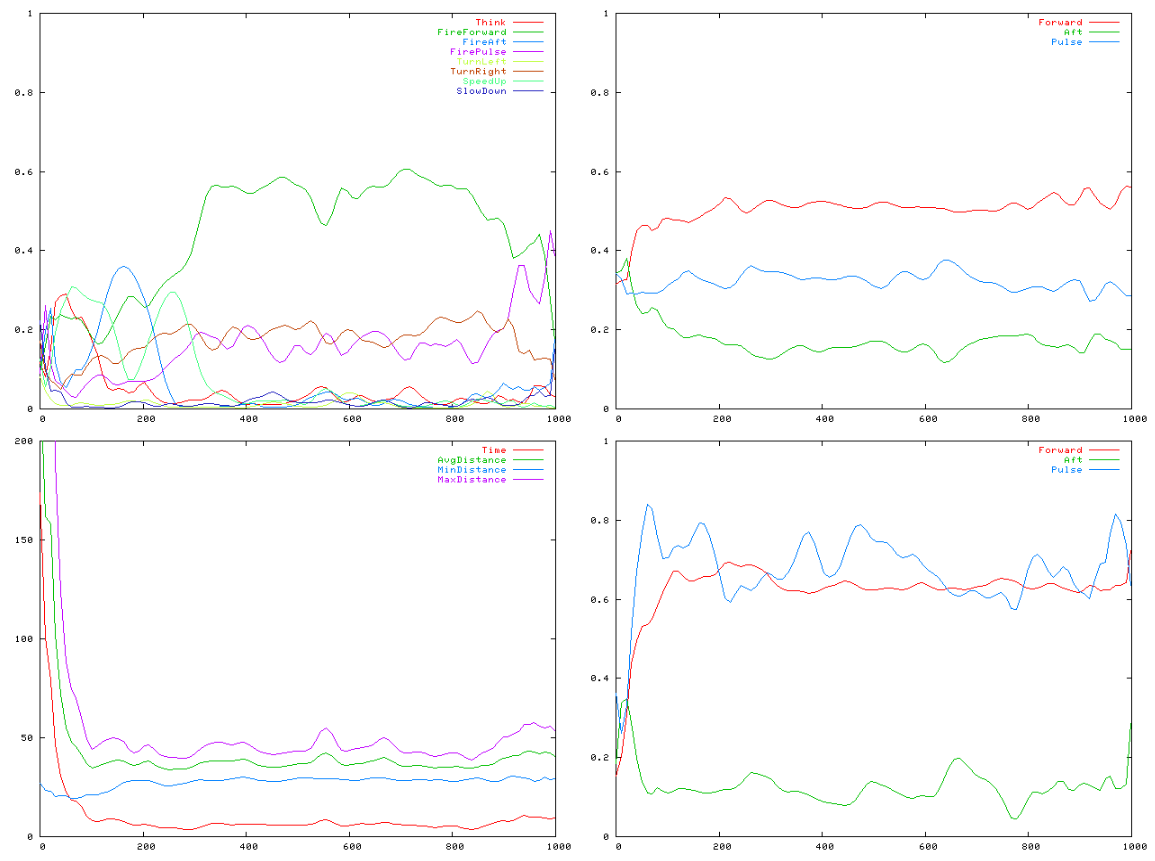

Figure 3: Base parameters, double mutation rate, 1000 generations

Varying Pulse Strength

It's pretty clear that populations can learn to fire forward torpedoes or aft torpedoes, so we wanted to see what effect varying the sonic pulse strength would have. We lowered the pulse damage from 5 to 1 and noticed that the initial burst of pulses in the population became much less important. The population instead learned torpedoes earlier without even bothering with pulses. The pulse hit rates stayed roughly the same, and allocations did as well, at least when the torpedoes were learned. Interestingly, reducing the pulse damage resulted in none of the 10 runs failing to evolve. When we increased pulse damage to 10, one population of the ten failed to evolve. Since we only have 10 runs, these aren't statistically accurate numbers, but there does not seem to be much of a difference between pulse damages of 5 and 10. Increasing the pulse damage to 40 and the pulse range to 40, however, showed big changes. Six of the ten populations focused entirely on pulses, as expected. It's difficult to say if these populations failed to show any improvement on the same criteria we used before, because using pulses takes longer to complete a battle than using torpedoes, because the ships have to be able to move close enough to do effective damage. Two of the ten populations did learn aft torpedoes, but still allocated a relatively high number of pulses.

Table 1: Summary of Submarine Types for Different Base Case Alterations

	Forward-Shooter	Aft-Shooter	Pulsers
Base Case	2	6	0
2 GP-Automata States	1	3	0
6 GP-Automata States	2	3	0
Decider Tree (Initial 4, Max 8)	2	1.5	0
Decider Tree (Initial 12, Max 20)	2	8	0
Mutation Rate x2	2	7	0
Mutation Rate x4	3	7	0
Pulse Damage of 1, Range 20	3	7	0
Pulse Damage of 10, Range 20	1	8	0
Pulse Damage of 40, Range 40	0	2	7

As shown in Table 1, Aft-Shooters are the dominant specialization which emerged in the different modifications to the base case run in this experiment. Note that specializing in pulses, referred to as "Pulsers", only emerged when the pulse damage and range was increased substantially. We suspect that this experiment is biased towards Aft-Shooters. This may be because the subs are always moving forwards, so once they pass each other, forward torpedoes don't work until the sub has turned around again. Also, if a Forward-Shooter gets really close to another sub, it could be damaged more by pulses. Finally, torpedoes have a very long range, so it's okay to turn away from the other submarine for a long time. One thing we noticed when comparing the ratios of Forward-Shooters to Aft-Shooters was that Forward-Shooters tend to have a slightly lower average distance in battle, which is expected.

Conclusions and Recommendations

- Initially, when logic structures were underdeveloped, pulses were much more effective.
- In the long run, when logic could support it, torpedoes were the best way to win battles. The key development here is being able to rotate the sub to aim torpedoes.
- Increasing the number of states might enable faster and better evolution.
- Increasing the decision tree size enables more effective submarines as well.
- Increasing the mutation rate did not have a noticeable effect on the battle times or resulting populations, but slightly increased volatility.
- As expected, pulse became the dominant weapon as we increased its damage and range. At lower pulse damage and range values, the population was encouraged to use torpedoes.

- We saw improvement even when some of the submarine inputs were screwed up, but not nearly as much as when we fixed them.
- Working with an evolutionary system with much complexity and many variables is challenging to debug and analyze.

Problems Collecting Data

We had several problems related to the geometric representation of the world. Initially, the geometry calculations were off, which caused the submarines to receive nonsensical angle data. This means they couldn't learn to fire torpedoes effectively, so we suspected that for a while they were just spinning and shooting. We did see about 30% improvement, however, and in some cases they did learn to use pulses. Before we gave them correct data, many of the battles would result in ties. In order to help them evolve, we added code that would mutate both submarines if they tied three times in a row. Once we addressed all of the bugs in the geometric calculations, we quickly saw improvement in the populations.

Another problem in developing this application is that it is really complex and has a highly iterative nature. This makes it especially difficult to debug because you cannot see without the aid of graphs and statistics how the program is operating.

There are so many variables that it's difficult to do a comprehensive study of the effects of the different parameters and what types of populations will evolve.

Recommendations

In the future, we would like to more exhaustively tune the variables and see their effects. We would also like to vary the effective angular range of torpedoes as well as perhaps differentiate between forward and aft torpedoes. We also want to change the number of weapons, sub hit points, speed, turn increments, firing frequencies, and number of subs.

In the initial implementation of this system, the torpedoes could not control their direction directly: they had to do so through a rudder. Like the speed of the sub, the rudder position remained the same from the previous turn so the subs could go into turns without wasting additional actions. However, we decided that they would not be smart enough to learn how to effectively control this, so we changed turns to be direct heading changes. We would like to investigate the behavior with the original rudder system.

Work that could add a lot to this system would be a visualization system for battles. Right now, it's difficult to see exactly what's going on when two submarines are engaged in battle. We have overall statistics, but we probably lose some interesting data in the process of averaging it all.

Just like the Herbivore Project, it would be interesting to break down a GP-Automata and its associated parse trees and examine its logic structure. This would help us understand how these submarines are actually behaving.

A potentially powerful addition would be to modify the evolutionary model to evolve a population, then evolve other populations against that one, and repeat. This might help us reach our goal of developing submarines that could accurately and efficiently use all of their weapons.