Will Rice
wdr0016@auburn.edu
COMP 5660 Fall 2020 Assignment 2c

Methodology

In this Competitive Coevolution Search, there were two separate GP controllers evolved. The first was a controller for Pac-Man, and the second was a controller shared by the three ghosts. The two controllers were evolved in separate populations that were in competition with each other.

The Pac-Man controller was evolved using five terminal inputs: G, P, W, F, and #.#. Here, G represented the distance to the nearest ghost, P represented the distance to the nearest pill, W represented the number of surrounding walls, F represented the distance to the nearest fruit, and #.# represented a static random number between -10 and 10. A custom caveat for F was that if there was no fruit on the board, then a configurable number was put in its place instead. All distances were measured using Manhattan Distance.

The Ghost controller was evolved using three terminal inputs: M, G, and #.#. Here, M represented the distance to Pac-Man, G represented the distance to the nearest Ghost excluding itself, and #.# again represented a static random number between -10 and 10.

Both controllers had access to the following five operations: add, subtract, multiply, divide, and random. One custom note is that if there was an attempted division by zero, then the program would instead multiply the numerator by a large number, specifically 9e10. This was chosen because it would imitate nearing positive infinity or negative infinity, depending on the sign of the numerator.

Both controllers were represented in the Python programming language as a TreeNode class with variables: left, right, and val. These variables are used in the typical fashion of Trees in computer science. During mutation and recombination however, the controllers switch from using the TreeNode class to manipulating a string version of the TreeNode. This was done mostly out of trouble with copying Python classes and passing Python classes by reference. These are issues that likely could have been resolved given more time and pressure to do so, but the string manipulation performs the part well without a noticeable performance hit.

Both controllers approach recombination in the __add__ methods of their respective classes. The recombination itself, after going through the switch from TreeNode to string and back, simply takes a random subtree from one parent and replaces it with a random subtree with the second parent. The second parent does the same. This results in two offspring children to be added to the new population.

Mutation for both classes is done by choosing a random node in the Tree and replacing it with a random choice of the same type as the original node. For example, a multiplication node could be changed to a subtraction node. This creates one new child to be added to the new population.

Both controllers were created in their own custom classes. When the time came for them to compete, it was done in a simple while loop. While there was at least one controller in either population that had not yet competed, there was a game instance where a Pac-Man controller faced off with a Ghost controller. The program chose a controller that had not competed if there was one available, and if there were none, then it chose a random member of the population and took the average of all of the scores given to it.

Each controller also used the same idea when it came to parsimony. If the tree that had been evolved was over a configurable number of nodes, then the score was demerited by the number of nodes over the threshold times the configurable parsimony for each class. To imitate a traditional parsimony where every tree is punished by some amount, a threshold of one could be put in place. This would punish each controller by a value proportional to the number of nodes it contained.

Experimental Setup

In the informal experiments run, it was clear that a change in lambda had an impact on the outcome of Pac-Man's final local best fitness. This is the variable that was chosen to be formally experimented on. Lambda for both Pac-Man and the Ghosts was tried at 100, 250, and 475. These numbers were chosen because they offered large enough changes in the informal experiments to possibly provide interesting results. They were all run with mu of 100 to give a reasonable size of the beginning population.

They were also run with the following evolutionary parameters: Parent Selection of Over-Selection, Survival Selection of Truncation, max depth of three, parsimony of five. These parameters were chosen because they provided reasonable outputs in all of the informal experiments. They could each be tuned in their own way, but they were kept the same in the experiments below in order to capture the effect of changing lambdas only.

The rest of the parameters are as follows: Over-Selection Percentage of 0.75, node threshold of forty, and fruit constant of one hundred. These again were chosen because of their steady performance in the informal experiments run, and kept the same to monitor only the lambdas.

The configuration files can be seen in the configs subdirectory as trial1.json, trial2.json, and trial3.json. They contain lambdas of 100, 250, and 475 respectively. Solution, world, and log files can be found in similarly named fashion in each of their subdirectories. Results of the experiments run can be found below.

Results

Below are the statistical analysis results of the experiments as described above. Each analysis compares the final local best Pac-Man fitness in each population of each run. First is comparing lambda=100 against lambda=250.

lambda=100	lambda=250
118	128
114	153
117	152
107	113
174	136
180	145
157	135
179	129
161	137
112	163
135	131
145	197
159	176
150	102
99	138
172	126
131	184
168	127
128	122
121	146
144	121
132	191
85	174
161	155
119	149
155	136
119	118
119	127
171	117
95	129

Standard Deviations

26.94654137	23.69395383

F-Test Two-Sample for Variances

	lambda=100	lambda=250
Mean	137.5666667	141.9
Variance	726.116092	561.4034483
Observations	30	30
df	29	29
F	1.293394428	
P(F<=f) one-tail	0.246406748	
F Critical one-tail	1.860811434	

mean(Variable 1) < mean(Variable 2)
F < F Critical
Assume Unequal Variances

T Test: Two-Sample Assuming Unequal Variances

	lambda=100	lambda=250
Mean	137.5666667	141.9
Variance	726.116092	561.4034483
Observations	30	30
Hypothesized Mean Difference	0	
df	57	
t Stat	-0.661463387	
P(T<=t) one-tail	0.255490159	
t Critical one-tail	1.67202883	
P(T<=t) two-tail	0.510980318	
t Critical two-tail	2.002465403	

|t Stat| < |t Critical Two - Tail|
Thus lambda of 250 is not statistically better than lambda of 100,
Nor is lambda of 100 statistically better than lambda of 250.

These results show that, although the mean is higher, lambda=250 is not statistically better than lambda=100. These findings are interesting considering that the lambda is fairly larger, but not directly better because of it. Next the lambda=100 is compared to lambda=475.

lambda=100	lambda=475
118	169
114	123
117	168
107	157
174	155
180	171
157	184
179	122
161	151
112	156
135	167
145	127
159	179
150	200
99	182
172	136
131	138
168	170
128	159
121	181
144	154
132	153
85	148
161	128
119	124
155	168
119	135
119	131
171	157
95	192

Standard Deviations

26.94654137	21.6891572

F-Test Two-Sample for Variances

	lambda=100	lambda=475
Mean	137.5666667	156.1666667
Variance	726.116092	470.4195402
Observations	30	30
df	29	29
F	1.543550023	
P(F<=f) one-tail	0.12418512	
F Critical one-tail	1.860811434	

mean(Variable 1) < mean(Variable 2)
F < F Critical
Assume Unequal Variances

T Test: Two-Sample Assuming Unequal Variances

	lambda=100	lambda=475
Mean	137.5666667	156.1666667
Variance	726.116092	470.4195402
Observations	30	30
Hypothesized Mean Difference	0	
df	55	
t Stat	-2.945172614	
P(T<=t) one-tail	0.002361694	
t Critical one-tail	1.673033907	
P(T<=t) two-tail	0.004723388	
t Critical two-tail	2.004044732	

|t Stat| > |t Critical Two - Tail|
Thus lambda=475 is statistically better than lambda=100.

These results show that lambda=475 actually is statistically better than lambda=100. This suggests that increasing the lambda does have a positive effect on the local best fitness at the end of each run. Next is the comparison between lambda=250 and lambda=475.

lambda=250	lambda=475
128	169
153	123
152	168
113	157
136	155
145	171
135	184
129	122
137	151
163	156
131	167
197	127
176	179
102	200
138	182
126	136
184	138
127	170
122	159
146	181
121	154
191	153
174	148
155	128
149	124
136	168
118	135
127	131
117	157
129	192

Standard Deviations

23.69395383	21.6891572

F-Test Two-Sample for Variances

	lambda=250	lambda=475
Mean	141.9	156.1666667
Variance	561.4034483	470.4195402
Observations	30	30
df	29	29
F	1.193410138	
P(F<=f) one-tail	0.318570952	
F Critical one-tail	1.860811434	

mean(Variable 1) < mean(Variable 2)
F < F Critical
Assume Unequal Variances

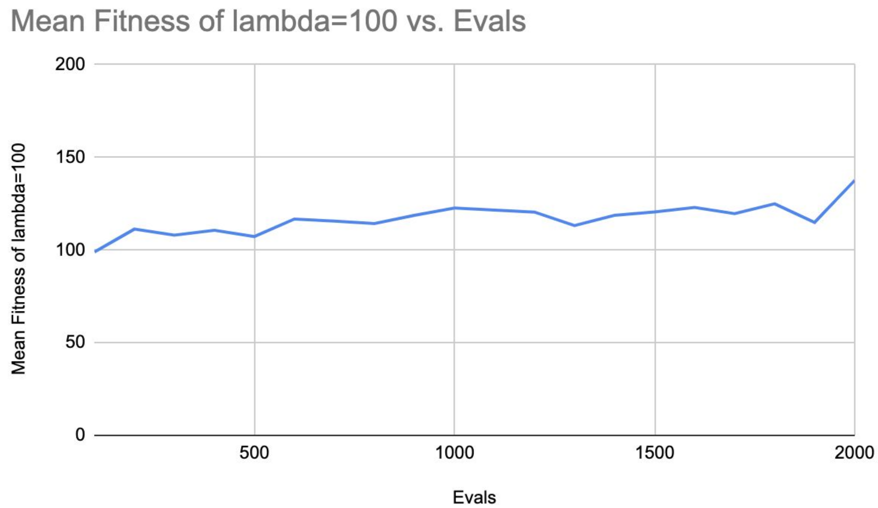
T Test: Two-Sample Assuming Unequal Variances

	lambda=250	lambda=475
Mean	141.9	156.1666667
Variance	561.4034483	470.4195402
Observations	30	30
Hypothesized Mean Difference	0	
df	57	
t Stat	-2.432655114	
P(T<=t) one-tail	0.009075155	
t Critical one-tail	1.67202883	
P(T<=t) two-tail	0.01815031	
t Critical two-tail	2.002465403	

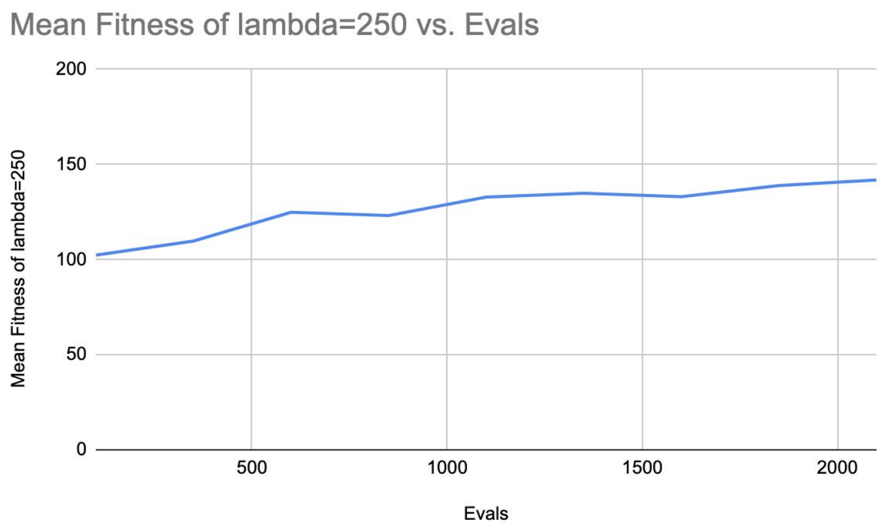
|t Stat| > |t Critical Two - Tail|
Thus lambda=475 is statistically better than lambda=250.

These results show that lambda=475 is also statistically better than lambda=250. This could be predicted because lambda=475 beat lambda=100 which is statistically similar to lambda=250. Still, the experiment proves it to be true.

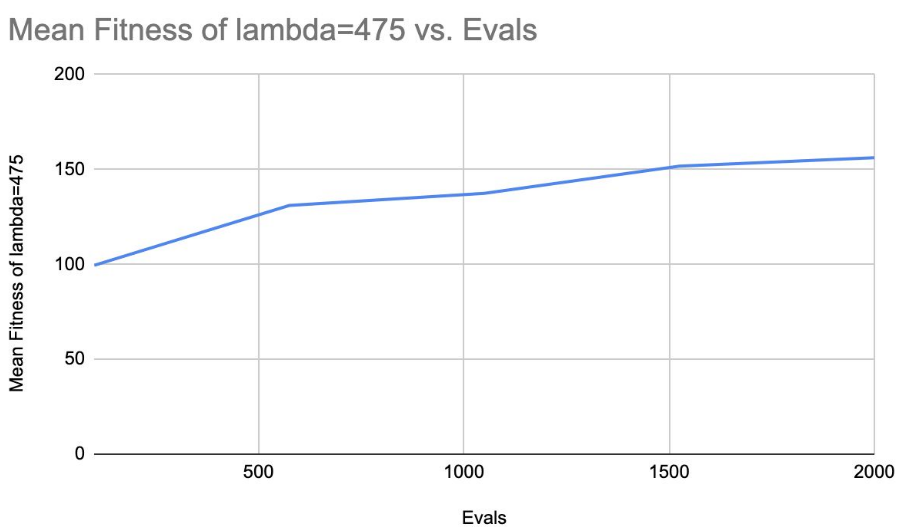
The next section of the results show the plots of evaluations versus population mean fitness averaged over the 30 runs. The plots are below and titled accordingly.



Mean Fitness of lambda=100 vs. Evals
200
150
100
50
0
500
1000
1500
2000
Mean Fitness of lambda=100
Evals



Mean Fitness of lambda=250 vs. Evals
200
150
100
50
0
500
1000
1500
2000
Mean Fitness of lambda=250
Evals



Mean Fitness of lambda=475 vs. Evals
200
150
100
50
0
500
1000
1500
2000
Mean Fitness of lambda=475
Evals

These results are quite interesting as they show that a larger lambda results in a more steady growth in mean fitness per generation. This will be discussed further in the Discussion section below.

Discussion

The findings for the experiments testing the different values of lambda show that a lambda of 475 is statistically better than both lambda of 250 and lambda of 100. Interestingly though, a lambda of 250 is not statistically better than a lambda of 100. This suggests that a lambda has to be significantly bigger to increase performance when it comes to the Pac-Man final local best fitness. There is likely a situation where the lambda could be too large though, but more formal experiments would have to be run to find such a threshold.

These experiments were run with only 2000 fitness evaluations, but the plots of average fitness per evaluation show that increasing the number of fitness evaluations could yield interesting results. This for a few reasons, first being that lambda of 100 does not show a constant shift upwards. There are enough peaks and valleys to suggest that increasing the number of evaluations per run might not result in a continually increasing average fitness. Secondly there appears to be a more steady increase in average fitness per evaluation in both lambda of 250 and lambda of 475. This could suggest that increasing the number of fitness evaluations could result in even more increases in average fitness. Increasing fitness evaluations might even be enough to make lambda of 250 statistically better than lambda of 100, but this still would have to be formally tested.

Conclusion

Overall, the results show that lambda of 475 is the best of the three. More experiments with an increased number of fitness evaluations would be a logical next set of experiments based on the results of the experiments done in this report. Another set of experiments could be gauging how changing only the lambda of one population affects the final outcome, as the experiments in this report kept both lambdas the same. Lessons learned include that lots could be done to tweak and optimize this EA. With each configurable parameter added there is more that could be done to optimize, but each formal experiment run takes lots of time and computational power.

Bibliography

N/A

Appendices

N/A