COMP SCI 5401 FS2018 Assignment 2c

William Lorey
wwlytc@mst.edu

Contents

Methodology

For assignment 2c, an Evolutionary Algorithm (EA) employing Genetic Programming (GP) alongside a Coevolutionary Search was used to coevolve controllers to play pacman. Controllers for both pacman and the ghosts were evolved in tandem. In addition to the implementation of standard EA and GP paradigms as outlined as part of the requirements for this assignment, the custom parts of this EA include a controller tree mutation strategy, the function and terminal sets, and mechanics of pairing controllers with other controllers and also with pacman game worlds.

Mutation was probabilistically performed (with a user-configurable probability) on each member of both controller populations (both for pacman and for the ghosts) following the completion of recombination to obtain the child population members. The mutation itself was a sub-tree mutation. If mutation was determined to act on a population member, it functioned by first selecting a random node from the controller tree. Then the sub-tree of the chosen node was 'nullified'. As the tree was stored as an array maintaining the heap property (each child node can be found from the parent index in a predictable way), this functionality was required to set the value of each node in the chosen sub-tree to a null value so rouge nodes would not impact the tree's functioning after mutation. A 'grow' method was then applied to the chosen node, randomly adding `n` subsequent levels to the sub-tree, where the inclusive range of `n` was user-configurable. In this implementation, a full tree was grown from the chosen node, ensuring that leaf nodes were part of the terminal set and that interior nodes were part of the function set. This form of mutation was chosen due to its simplistic nature

and effectiveness in slightly modifying controllers with (typically) minimal destruction of the original controller tree.

In order for a GP tree to produce reasonable, predictable output, node selections from two sets of nodes were made: the function set of nodes and the terminal set of nodes. As taken from standard GP and backed up by logical reasoning, the GP tree in this assignment was constructed with function nodes always making up the interior nodes of the controller trees, and terminal nodes always making up the leaf nodes of the controller trees. The compatibility of nodes was also important. For ease of constructing and evaluating trees, all function nodes took two inputs of floating point numbers, and all terminal nodes evaluated to floating point numbers. This ensured that when trees were interpreted to produce a crisp numerical output, they could be compiled correctly without errors.

The function nodes used for both the pacman and ghost controller trees were standard and provided as part of the assignment: addition, subtraction, multiplication, division, and random (choose a random floating point number within a given range). The implementation of these operations were straightforward except for division and random. For division, division by zero was protected for. If a number was to be divided by zero, the result would become zero. For the random operation, as implemented in Python, a random selection made using a call to the `random` library required the lower bound to be smaller than the upper bound. This was solved by always supplying the minimum floating point value as the lower bound and the maximum floating point value as the upper bound.

Terminal nodes differed for the pacman and ghost controllers. The pacman controllers used the following terminal nodes: Manhattan distance between pacman and the nearest ghost, Manhattan distance between pacman and the nearest pill, Manhattan distance between pacman and the nearest fruit, number of walls adjacent to pacman, a floating point constant, and Manhattan distance to the nearest other pacman (for the BONUS2 assignments). These nodes were implemented in accordance with the main assignment deliverables. One main comment about implementation particulars for the pacman nodes would be that if a fruit was not placed in the world, the distance to nearest fruit node would evaluate to a user-configurable arbitrary large number. Similarly for the ghost terminal nodes, which include the Manhattan distance to the nearest pacman, Manhattan distance to the nearest ghost, and a floating point constant, the majority of nodes were implemented as part of the main assignment deliverables. The floating point constant was not required for the ghost controller terminal nodes, but it seemed useful to include it, as it may provide useful information to the controller while not increasing the search space drastically.

To complete runs of the experiment, pairings of pacman controllers to ghost controllers had to be performed. To facilitate this process, a data structure was created which included the unit controller class and the fitness of that controller. Once both populations of controller-fitness objects were generated (either by the initial population generation or through recombination and mutation), the controllers were randomly paired by taking, for each index in both unordered populations, one controller from each population. To evaluate the controllers, each pairing was matched with a pacman world with its own generated wall and pill placements. The controllers were applied to this world by running a game to determine the controller fitnesses. This approach allowed for sufficient abstraction to keep the process easy to understand and implement while providing stochastic pairing of controllers and pacman worlds.

Experimental Setup

All experiments for this assignment were run using custom configuration files, with each file testing the algorithm against a slightly different set of parameters. Each configuration that was employed is included as part of the git repository in the `config` directory. The experiments themselves were each run on one of the S&T campus linux machines, with each employed machine running one EA on a time-shared basis with other users of the machines using them for their own purposes.

When employing a custom configuration file in experimentation, the following command (for example) was used:

```
./run.sh config/config_file.cfg
```

Where `config_file.cfg` could be replaced with any configuration file in the `config` directory, or omitted entirely. In the case where the configuration file is omitted, `config/default.cfg` would be used as the configuration file.

To directly reproduce experiments, a configuration option to use a provided random number seed has been included. In each log file generated after an experiment run, the seed used as part of that experiment is saved. To use a seed to reproduce an experiment, modify the following configuration file fields like so:

```
use external seed = True
default seed = <insert seed value here>
```

The three varying configurations tested as part of this assignment's deliverables were all configured with the following *shared* parameters:

- 30 runs
- 2000 fitness evaluations
- μ pacman = 20
- λ pacman = 60
- μ ghost = 20
- λ ghost = 60
- comma survival strategy for ghosts and pacmen
- single controller for single pacman
- single controller for three ghosts

Other parameters exist that are not mentioned above that were excluded for brevity. These parameters can be viewed in the configuration files.

The testable differences between deliverable configurations, for each deliverable, are as follows:

Deliverable 1

- Parent Selection: Over-Selection
- Survival Selection: k-Tournament (k pacman = 5, k ghost = 5)

Deliverable 2

- Parent Selection: Fitness Proportional Selection (FPS)
- Survival Selection: k-Tournament (k pacman = 5, k ghost = 5)

Deliverable 3

- Parent Selection: Over-Selection
- Survival Selection: Truncation

Note that the selection methods chosen were applied to both parent and survival selection for pacman and the ghosts.

Because the number of required configuration comparisons for this assignment was relatively small, parent and survival selections were chosen to be compared. These parameters facilitate a coarse-grained exploration that can be performed with a minimal number of comparisons, and their impact on EA performance is of consequence. More fine-tuneable parameters could be explored, such as the parsimony coefficient or `k` in the k-Tournament Selection, and this exploration could be performed for both pacman and the ghosts and their interactions, but after considering the scope of this project, the aforementioned configuration comparisons were chosen instead.

Results

Three configurations of the EA were run to generate experimental results. These configurations, as introduced in the **Experimental Setup** section, were Deliverable 1: Over-Selection Parent Selection and k-Tournament Survival Selection (Figure 1), Deliverable 2: Fitness Proportional Parent Selection and k-Tournament Survival Selection (Figure 2), and Deliverable 3: Over-Selection Parent Selection and Truncation Survival Selection (Figure 3). Pairwise statistical comparisons were performed as follows: Deliverable 1 and Deliverable 2 (Table 1), Deliverable 2 and Deliverable 3 (Table 2), and Deliverable 1 and Deliverable 3 (Table 3).

The statistical analysis performed in these comparisons was done so on the final local best pacman fitness across thirty runs. The analysis consisted of first an f-test which was used to determine the assumption of equal or unequal variances. Then, a t-test (either assuming equal or unequal variances based on the results of the f-test) was performed. This final test produced the result that either one configuration was statistically better than the other for the given problem space or that neither algorithm configuration was statistically better for the given problem space.

Comparing Deliverable 1 with Deliverable 2, the statistical analysis (Table 1) showed that neither algorithm configuration was statistically better. Table 2 and Table 3 each compare Deliverable 3 with Deliverable 1 and Deliverable 2 respectively. In both cases, the algorithm configured with Deliverable 3 was proven to be statistically better for the given problem space. Discussion regarding the results of these experiments can be found in the **Discussion** section.

Note that all experiment results (log files, solution files for the pacman and ghosts, and world files) for the deliverable experiments can be found in the `output` directory in the git repository.

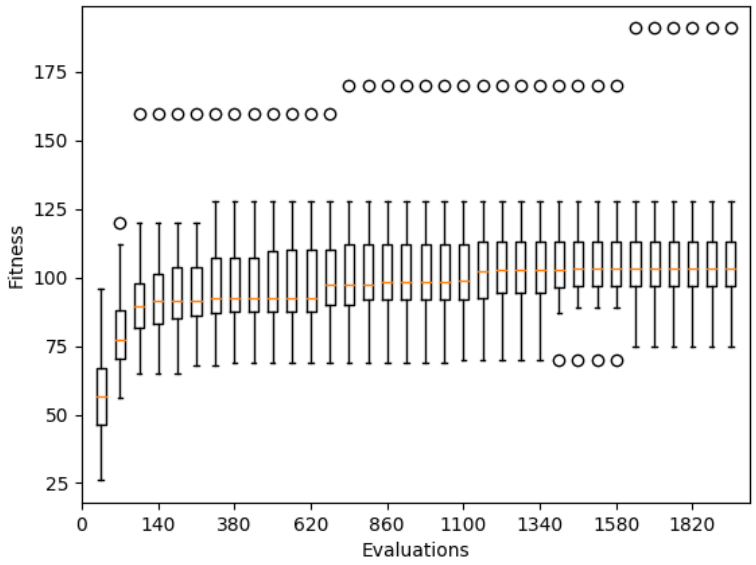

Figure 1: Global Best Fitness versus Fitness Evaluations for the **Over-Selection Parent Selection and k-Tournament Survival Selection (Deliverable 1)**, Randomly Generated Worlds. The figure was generated with data obtained by running the GP with the **deliverable1.cfg** configuration file.

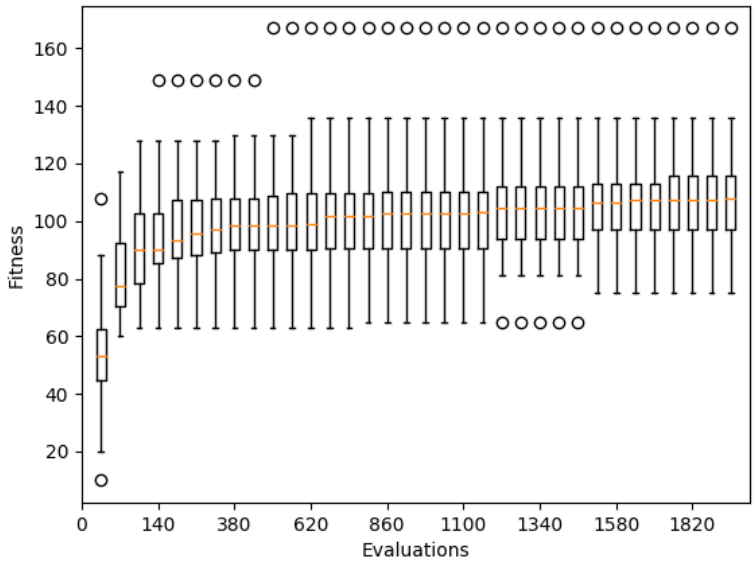

Figure 2: Global Best Fitness versus Fitness Evaluations for the **Fitness Proportional Parent Selection and k-Tournament Survival Selection (Deliverable 2)**, Randomly Generated Worlds. The figure was generated with data obtained by running the GP with the **deliverable2.cfg** configuration file.

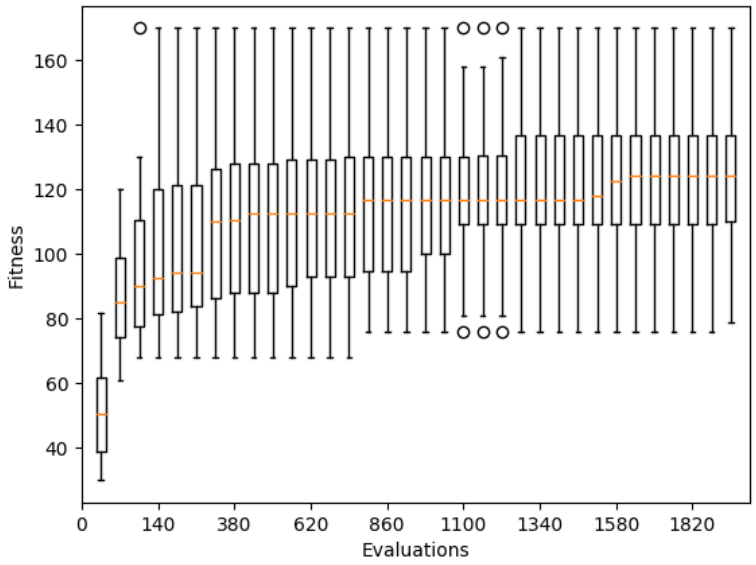

Figure 3: Global Best Fitness versus Fitness Evaluations for the **Over-Selection Parent Selection and Truncation Survival Selection (Deliverable 3)**, Randomly Generated Worlds. The figure was generated with data obtained by running the GP with the **deliverable3.cfg** configuration file.

Table 1: Statistical Analysis performed on Deliverable 1 and Deliverable 2

	deliverable1	deliverable2
mean	107.4	108.13333333333334
variance	395.5733333333333	301.58222222222224
standard deviation	19.88902544956221	17.366122832175932
observations	30	30
df	29	29
F	1.3116599858523932	
F critical	0.5373999648406917	
Equal variances assumed		
observations	30	
df	58	
t Stat	-0.14956692976611996	
P two-tail	0.8816251806191939	
t Critical two-tail	2.0017	
Nether deliverable2 nor deliverable1 is statistically better		

Table 2: Statistical Analysis performed on Deliverable 2 and Deliverable 3

	deliverable2	deliverable3
mean	108.13333333333334	124.56666666666666
variance	301.58222222222224	578.5788888888889
standard deviation	17.366122832175932	24.05366684912903
observations	30	30
df	29	29
F	0.5212465024456475	
F critical	0.5373999648406917	
Unequal variances assumed		
observations	30	
df	31	
t Stat	-2.9829335278061286	
P two-tail	0.004315231188237327	
t Critical two-tail	2.0395	
deliverable3 is statistically better than deliverable2		

Table 3: Statistical Analysis performed on Deliverable 1 and Deliverable 3

	deliverable1	deliverable3
mean	107.4	124.56666666666666
variance	395.5733333333333	578.5788888888889
standard deviation	19.88902544956221	24.05366684912903
observations	30	30
df	29	29
F	0.6836981800234675	
F critical	0.5373999648406917	
Equal variances assumed		
observations	30	
df	58	
t Stat	-2.961907970931727	
P two-tail	0.004425906472211853	
t Critical two-tail	2.0017	
deliverable3 is statistically better than deliverable1		

Discussion

The result that neither Deliverable 1 nor Deliverable 2 is statistically better than each other shows that, all else held constant, switching between Over-Selection and FPS for parent selection does not produce tangible differences in performance. Additionally, both the Deliverable 1 and Deliverable 2 configurations showed reasonably high variance, signaling that the stochastic algorithm's performance in both cases was fairly unpredictable. Both methods use a reasonably high selective pressure, but the selection pressure could still be increased (for instance, by employing Truncation survival selection), which was proven to produce a more optimal algorithm with the Deliverable 3 comparisons.

Since the method of Over-Selection was created for very large GP tree populations, it is possible that the limited size of the GP populations, in the scope of this experiment, rendered the Over-Selection method to be less effective in optimally choosing parents, at least when paired with the other configuration parameters of Deliverable 1. However, this method when paired with Truncation survival selection in Deliverable 3 proved to be the most optimal configuration tested. The efficacy of individual GP configurations on their own is very hard if not impossible to judge. Configurations must be used alongside each other to properly determine how effective they are.

Interestingly, the high variance of Deliverable 3 may have helped prove it as a more optimal configuration. Deliverable 3 was proven to be statistically superior to Deliverable 1 and Deliverable 2. While more sub-optimal solutions were found with Deliverable 3, there were also many consistent, exceedingly optimal solutions found. The optimality of the Deliverable 3 configuration can also be attributed to the fact that it employed a very high selection pressure in its parent and survival selections. Both the Over-Selection parent selection method and the Truncation survival selection method both inherently apply very high selection pressure, and when paired together, the selective pressure of the EA in general becomes quite high. This leads to the conclusion that for this algorithm employing GP, a high selective pressure helps produce optimal controllers.

BONUS1

The first bonus involved experimenting with using a single pacman employing a single controller and multiple ghosts, each employing a different controller. For ease of testing experiment cases such as this, user-configurable options were included which allow the user to choose (1) how many pacmen to test in each world and (2) the multiplicity of units to controllers, i.e. if each unit gets its own controller or if all units of a given type share controllers.

The BONUS1 experiment plot (Figure 4) showed exceptionally high outlying fitness values with averages much lower than the outlying values, as if the multiple ghost controllers led to higher pacman fitness values. It was expected that multiple ghost controllers would lead to better ghosts with the capacity to work together, leading to worse pacman fitnesses. After visual inspection of the plots, it appeared as if the BONUS1 configuration would be optimal when compared to the configuration with one controller for all ghosts.

Statistical analysis was performed on the BONUS1 configuration and the deliverable1 configuration, as the only difference between the two configurations was the ghost, controller multiplicity, which led to a fair and narrow comparison. The analysis (Table 4) proved that neither configuration was statistically optimal. This demonstrated that despite the optimal outlying values in BONUS1, an optimal algorithm is not guaranteed because of it.

Note that all experiment results (log files, solution files for the pacman and ghosts, and world files) for BONUS1 can be found in the `output` directory in the git repository.

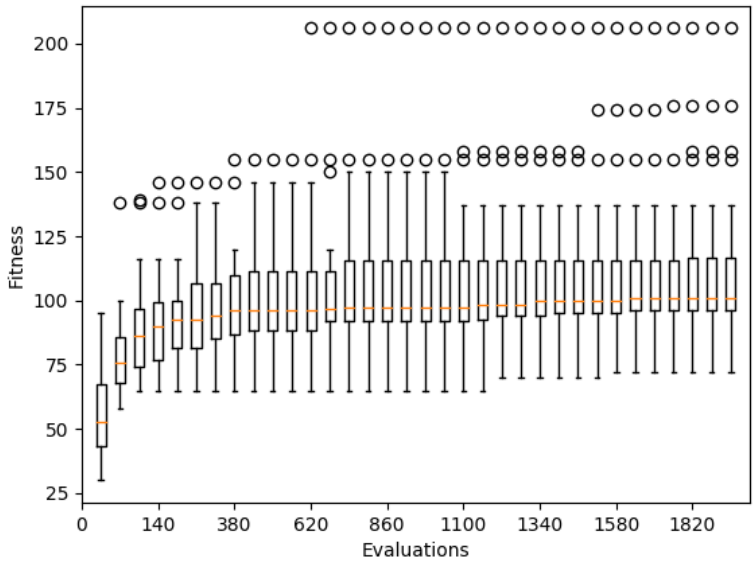

Figure 4: Global Best Fitness versus Fitness Evaluations for the **BONUS1: Single pacman, multiple ghosts employing different controllers**, Randomly Generated Worlds. The figure was generated with data obtained by running the GP with the **BONUS1.cfg** configuration file.

Table 4: Statistical Analysis performed on BONUS1 and Deliverable 1

	BONUS1	deliverable1
mean	111.4	107.4
variance	862.0400000000001	395.5733333333333
standard deviation	29.360517706607286	19.88902544956221
observations	30	30
df	29	29
F	2.179216664419577	
F critical	0.5373999648406917	
Unequal variances assumed		
observations	30	
df	31	
t Stat	0.6074148757553575	
P two-tail	0.5462715238455983	
t Critical two-tail	2.0395	
Nether deliverable1 nor BONUS1 is statistically better		

BONUS2a, BONUS2b, BONUS2c, and BONUS2d

The following BONUS2 assignments were completed as part of this report:

- BONUS2a: Multiple pacmen employing the same controller, multiple ghosts employing the same controller
- BONUS2b: Multiple pacmen employing different controllers, multiple ghosts employing the same controller
- BONUS2c: Multiple pacmen employing the same controller, multiple ghosts employing different controllers
- BONUS2d: Multiple pacmen employing different controllers, multiple ghosts employing different controllers

As discussed in the BONUS1 section, user-configurable options were included which allow the user to choose how many pacmen to test in each world and the multiplicity of units to controllers, making the process of testing multiple configurations of varying pacman count and unit/controller multiplicity more manageable.

Each BONUS2 sub-part was experimentally run and the pacman fitness plots were each individually graphed. They can be found in Figure 5 (BONUS2a), Figure 6 (BONUS2b), Figure 7 (BONUS2c), and Figure 8 (BONUS2d). Visually inspecting the fitness plots, they all showed fairly high outlying values, signifying good algorithm performance, at least on the surface.

To scientifically compare these configurations, pairwise statistical analysis was performed on all possible pairings of BONUS2 configurations. The pairings are as follows: BONUS2a vs BONUS2b (Table 5), BONUS2a vs BONUS2c (Table 6), BONUS2a vs BONUS2d (Table 7), BONUS2b vs BONUS2c (Table 8), BONUS2b vs BONUS2d (Table 9), and BONUS2c vs BONUS2d (Table 10).

Interesting findings from this analysis included the relative superiority of BONUS2a. It was found to be superior to BONUS2b (Table 5) and BONUS2d (Table 7), dominating more configurations than any other configuration in the group. The only other configuration statistically proven to be better than another was BONUS2c when compared to BONUS2b (Table 8).

The finding that BONUS2a was the most optimal configuration when compared to the other BONUS2 comparisons implies that when the controller multiplicity is maximum, i.e. when each unit shares a controller with all other units of the same type, algorithm performance is optimal. One would hypothesize, however, that allowing units to employ different controllers would promote cooperation between units of a type. This behavior was by and large not observed as part of this experimentation. It is possible that more evaluations and a much larger population could produce this behavior more consistently.

Note that all experiment results (log files, solution files for the pacman and ghosts, and world files) for BONUS2 can be found in the `output` directory in the git repository.

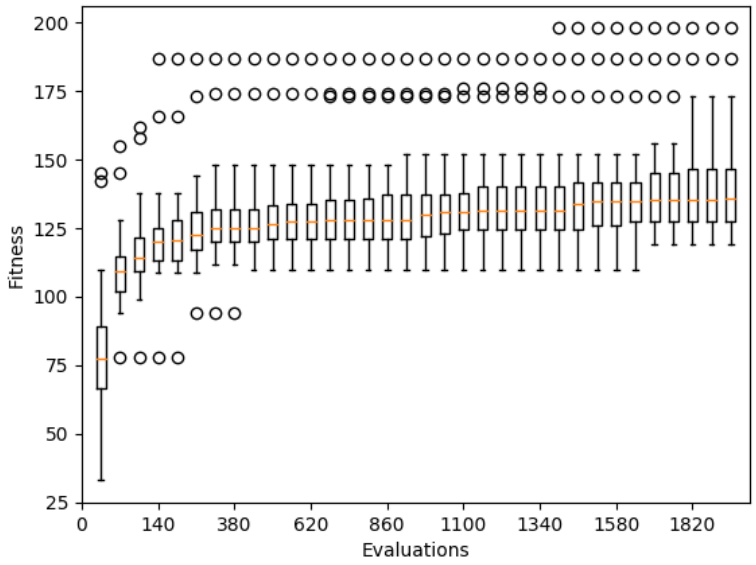

Figure 5: Global Best Fitness versus Fitness Evaluations for the **BONUS2a: Multiple pacmen employing the same controller, multiple ghosts employing the same controller**, Randomly Generated Worlds. The figure was generated with data obtained by running the GP with the **BONUS2a.cfg** configuration file.

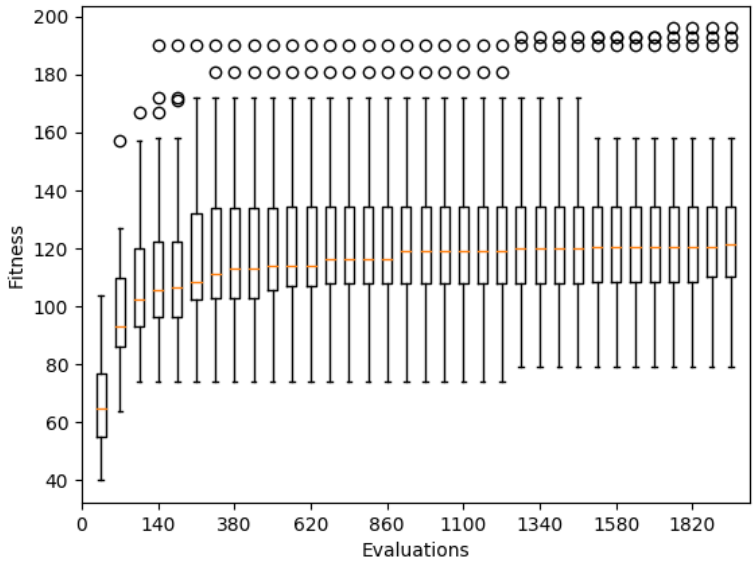

Figure 6: Global Best Fitness versus Fitness Evaluations for the **BONUS2b: Multiple pacmen employing different controllers, multiple ghosts employing the same controller**, Randomly Generated Worlds. The figure was generated with data obtained by running the GP with the **BONUS2b.cfg** configuration file.

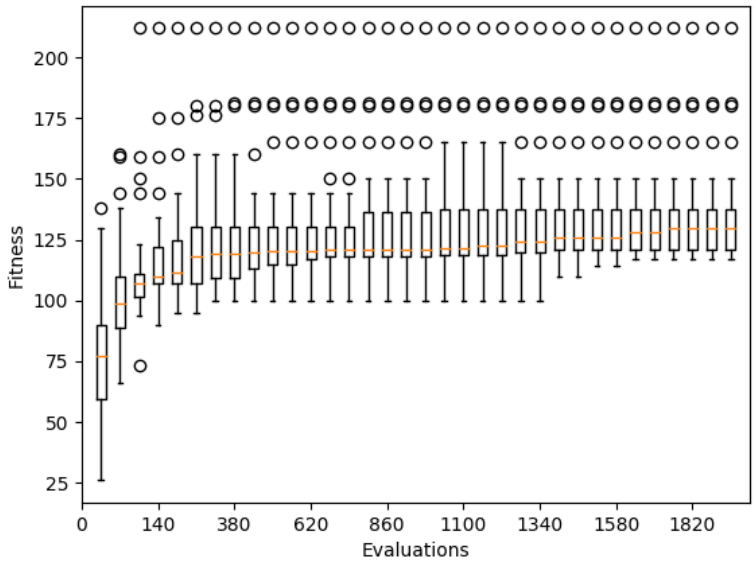

Figure 7: Global Best Fitness versus Fitness Evaluations for the **BONUS2c: Multiple pacmen employing the same controller, multiple ghosts employing different controllers**, Randomly Generated Worlds. The figure was generated with data obtained by running the GP with the **BONUS2c.cfg** configuration file.

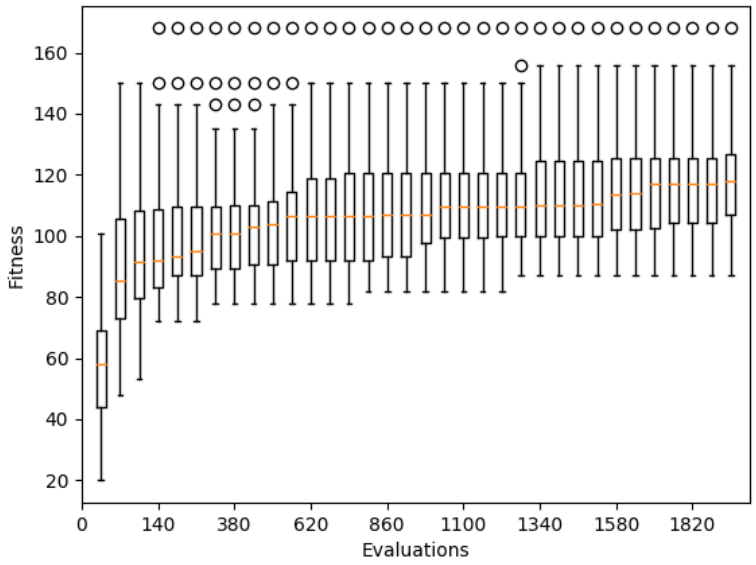

Figure 8: Global Best Fitness versus Fitness Evaluations for the **BONUS2d: Multiple pacmen employing different controllers, multiple ghosts employing different controllers**, Randomly Generated Worlds. The figure was generated with data obtained by running the GP with the **BONUS2d.cfg** configuration file.

Table 5: Statistical Analysis performed on BONUS2a and BONUS2b

	BONUS2a	BONUS2b
mean	140.7	127.6
variance	342.47666666666663	784.9733333333332
standard deviation	18.506125112153182	28.017375561128727
observations	30	30
df	29	29
F	0.4362908294124641	
F critical	0.5373999648406917	
Equal variances assumed		
observations	30	
df	58	
t Stat	2.1009786982574874	
P two-tail	0.0399926723492717	
t Critical two-tail	2.0017	
BONUS2a is statistically better than BONUS2b		

Table 6: Statistical Analysis performed on BONUS2a and BONUS2c

	BONUS2a	BONUS2c
mean	140.7	135.9
variance	342.47666666666663	478.02333333333337
standard deviation	18.506125112153182	21.8637447234762
observations	30	30
df	29	29
F	0.7164434093175367	
F critical	0.5373999648406917	
Unequal variances assumed		
observations	30	
df	31	
t Stat	0.9024038585626759	
P two-tail	0.3706762522069167	
t Critical two-tail	2.0395	
Nether BONUS2c nor BONUS2a is statistically better		

Table 7: Statistical Analysis performed on BONUS2a and BONUS2d

	BONUS2a	BONUS2d
mean	140.7	119.36666666666666
variance	342.47666666666663	354.3655555555555
standard deviation	18.506125112153182	18.82459974489645
observations	30	30
df	29	29
F	0.9664502130568247	
F critical	0.5373999648406917	
Unequal variances assumed		
observations	30	
df	31	
t Stat	4.352016063471005	
P two-tail	5.5455771508607214e-05	
t Critical two-tail	2.0395	
BONUS2a is statistically better than BONUS2d		

Table 8: Statistical Analysis performed on BONUS2b and BONUS2c

	BONUS2b	BONUS2c
mean	127.6	135.9
variance	784.9733333333332	478.02333333333337
standard deviation	28.017375561128727	21.8637447234762
observations	30	30
df	29	29
F	1.6421234667763773	
F critical	0.5373999648406917	
Equal variances assumed		
observations	30	
df	58	
t Stat	-1.2576968962023538	
P two-tail	0.2135417162246614	
t Critical two-tail	2.0017	
Nether BONUS2c nor BONUS2b is statistically better		

Table 9: Statistical Analysis performed on BONUS2b and BONUS2d

	BONUS2b	BONUS2d
mean	127.6	119.36666666666666
variance	784.9733333333332	354.3655555555555
standard deviation	28.017375561128727	18.82459974489645
observations	30	30
df	29	29
F	2.2151513346230667	
F critical	0.5373999648406917	
Unequal variances assumed		
observations	30	
df	31	
t Stat	1.313554924158955	
P two-tail	0.1949035354931222	
t Critical two-tail	2.0395	
Nether BONUS2d nor BONUS2b is statistically better		

Table 10: Statistical Analysis performed on BONUS2c and BONUS2d

	BONUS2c	BONUS2d
mean	135.9	119.36666666666666
variance	478.02333333333337	354.3655555555555
standard deviation	21.8637447234762	18.82459974489645
observations	30	30
df	29	29
F	1.348955410138307	
F critical	0.5373999648406917	
Unequal variances assumed		
observations	30	
df	31	
t Stat	3.086002570767297	
P two-tail	0.0031343644838580814	
t Critical two-tail	2.0395	
BONUS2c is statistically better than BONUS2d		

Conclusion

After completing this assignment and the related experimentation and analysis, it is apparent that the most optimal EA configuration implemented as part of the main assignment deliverables is optimal due to high selective pressure for its parent and survival selections. This

ensures that cruft is cut away from the medium-small population of individuals and that the most fit individual(s) in each generation are guaranteed to survive, driving up the success of discovering and keeping optimally fit individuals as the algorithm progresses. Additionally, it was observed that Genetic Programs can balloon both in time elapsed and in space required as problems scale. It is evident that in order to use GP algorithms in a meaningful way, appropriate optimizations, the use of parallelization, and proper machine selection and upgrades must all be accounted for.