Assignment 4

Q1

Part A

Note that in this part, alot of PSO runs converged at a value that wasn't the 'true best', as such, some particles jitter around a local optimum and the simulation continues. Other times, the 'true best' is found, at which time the simulation stops. When the simulation doesn't stop, it is difficult to determine the exact point of convergence so it will be estimated by seeing when the particles are visually "stuck" around a local optimum. The simulations were run with the default `landscape-smoothness = 20`.

Population	Speed Limit	Particle Inertia	Attraction to Personal & Global Best	Convergence Speed (# of ticks)	Notes
30	2	0.6	1.7	within 50 ticks	
30	2	0.6	1.494	within 70 ticks	
30	2	0.729	1.7	within 50 ticks	
30	2	0.729	1.494	within 50 ticks	
30	6	0.6	1.7	within 30 ticks	seems to find better slightly solutions
30	6	0.6	1.494	within 30 ticks	
30	6	0.729	1.7	within 30 ticks	
30	6	0.729	1.494	within 30 ticks	
80	2	0.6	1.7	true best: ~13 ticks, non best within 40 ticks	finds the best solution more often than smaller population
80	2	0.6	1.494	true best: 26 ticks, non best within 50 ticks	
80	2	0.729	1.7	within 40 ticks	
80	2	0.729	1.494	within 40 ticks	
80	6	0.6	1.7	within 40 ticks	
80	6	0.6	1.494	within 20 ticks	

Population	Speed Limit	Particle Inertia	Attraction to Personal & Global Best	Convergence Speed (# of ticks)	Notes
80	6	0.729	1.7	true best solution: ~10 ticks, within 20 ticks	
80	6	0.729	1.494	true best solution: ~10 ticks, within 20 ticks	

Population Effects

In the simulation, larger `populations` (i.e. number of particles) appear to cause convergence speed as well as likelihood of finding the "true best" solution. This makes sense as with more particles, more solutions in the search space are being explored.

Speed Limit Effects

The `speed limit` describes the maximum component velocity of an individual particle. Increasing the limit allows particles to move at a higher velocity, and in the simulation, this appeared to be increase the convergence speed and seemed to slightly increase the solution fitness. Increasing the maximum particle velocity will allow for higher exploration as particles may escape a local optimum and can move farther in between iterations.

Particle Inertia Effects

In the simulation, no significant discernable effect on solution fitness and convergence speed was found by varying the `particle inertia`. This could be due to the discrepancy between the values `[0.6, 0.729]` being too small. In theory, increasing a particle's inertia makes it more resistant to directional changes. Higher inertia will result in less effect from the cognitive and social components when calculating velocity; the particle will tend to continue its original course instead of being influenced by a promising solution, leading to more exploration. Lower inertia will have an opposite effect and the particle will be more sensitive to promising points and tend to change directions more to reach these points, increasing exploitation.

Attraction to Personal & Global Best

In the simulation, no significant discernable effect on solution fitness and convergence speed was found by varying the `attraction to personal/global best`. This could be due to the discrepancy between the values `[1.7, 1.494]` being too small. In theory, because both the particle and global bests are being set to a singular value, the social and cognition components have equal weighting. Increasing the attraction value will have a similar effect to decreasing the `particle inertia`. The attraction increases the effect of social and cognition components, causing the particles to more readily change directions towards the most promising solution in the vicinity.

Part B

The following code from the NetLogo implementation of PSO and is the cognitive component in the x direction.

```
    set vx vx + (1 - particle-inertia) * attraction-to-personal-best * (random
```

Notice the is a `(1 - particle-intertia)` term in velocity equation that is not present in the canonical PSO algorithm. This allows a configuration where the particles are purely inertia driven. Consider when the parameter `particle-inertia` is set to `1.0`; the cognitive component will go to `0`. The `(1 - particle-inertia)` term is also present in the cognitive-y and social-x,y velocity equations, meaning when `particle-inertia` is set to `1.0`, all influence from the cognitive and social components will be 0. As such, the motion of the particles will be purely inertia driven; they will continuously move in the directions they were initialized with.

Q2

Several assumptions are made for this question:

1. The function `g(x)` gives the fitness value for the solution x
2. The variable `phi_1` and `phi_2` give the inertia adjusted coefficients of the cognitive and social components.
3. The operator `epsilon` denoting `belongs to` will clip values so that they fall within a given range.
4. The variable p is initialized elsewhere and denotes the set of all particles, where `p_i` is the ith particle.

Part A

For synchronous update modes, the entire population is updated before the `neighborhood best`. In the asynchronous mode, the `neighborhood best` is updated after every particle. From the following snippet,

```
for i = 1 to N do,                  //for each particle
  ...
    g = i                           //arbitrary
    //for all neighbors
    for j = indexes of neighbors
      if g(p_j) > g(p_g)
        then g = j                  //index of best neighbor
      end if
    end for
  ...
```

the best neighbor is found and its index is stored in g. As this is occuring within the loop iterating over each particle, the `neighborhood best` is being updated for each particle individually. Assuming the value of p contains pointers to values of x, then the PSO implementation under consideration has an asynchronous update mode. This is because the statement:

```
    x_i <- x_i + delta_x_i
```

will update `x_i` will be seen by when accessing `p_i` immediately afterwards in the next iteration of loop increment variable `j`. Meaning, the `neighborhood best` is being updated for each particle.

Part B

In order to change the algorithm from asynchronous to synchronous, the inner for loop, `J`, that finds the best neighbor index should be moved outside the loop iterating over each particle, `I`. Instead of just the best neighbor's index, a copy of the point's data should be copied, perhaps as `p_g'`. Then, for each particle, instead of using `p_g`, which would be a reference to the point with index `g`, the copy `p_g'` should be used. In this sense, the `neighborhood best` is being held constant for all particles within an iteration. Upon further consideration, this may only work if a global neighborhood was being used.

Alternatively, a simpler way could be to store copies of the new values of `x_i` and actually set them when `i > N` or at the end of each iteration when `t` is being incremented. This would mean having the set of all particles and their velocities, `x` and `delta_x` stay constant within loop `I`. Instead, the new position and velocity values are being stored in `x'` and `delta_x'`. Once loop `I` is complete, `x = x'` and `delta_x = delta_x'` will update all the new positions and velocities. Although the `J` loop is still within the `I` loop, because the position and velocities being used to calculate the neighborhood is being held constant across `i`, the `neighborhood best` is not being changed with information generated within loop `I` and is thus not asynchronous.

Part C

The given PDF appears to be missing two substraction, –, operators within the cognitive and social components. The following section assumes this to be a typographic or file formatting error and that the PSO model is that which is shown in lecture 9–2.

I) `C_1 = 0`

Setting `C_1 = 0` will eliminate the `cognitive` component from the velocity equation. This will result in the model becoming highly exploitative and the particles will quickly converge at the neighborhood best.

II) `C_2 = 0`

Setting `C_1 = 0` will eliminate the `social` component from the velocity equation. The particles will stay in the narrowvicinity of their initial locations and rapidly cease movement as they find the best value in that narrow vicinity. This is arguably no longer a `meta-heuristic` algorithm as there is no heuristic updating the individual particle's heuristic. I.e. the `social component` is effectively the `meta-heuristic` for individual particle `heuristics`. With low enough inertia, the particles may just sit in their initial positions within single digit iterations. This is no more effective than bozosort as it now depends on the random generation of initial particle positions.

III) `W`

The W parameter is inertia, which describes a particle's resistance to change. A high inertia means that it will be more difficult to change a particle's velocity, meaning it is less affected by both social and cognitive components. The converse is true; lower inertia means that social and cognitive components are more dominant in a particle's velocity. As such, higher inertias may promote higher exploration as a partcle is less likely to be drawn towards local optima and have a greater tendency to continue on their original path.

Q3

The implementation of Q3 is designed to be run standalone in a web browser. Simply open the `Q3.html` file in a browser with Javascript enabled (preferrably Chrome). The code runs on pageload so to run again with a different randomly generated initial population, simply refresh the page. To view a tree representation of the best program in any iteration, click on the corresponding data point on the plot. Note that you may to zoom out if the tree is too large. To change the parameters, the code will have to be modified directly. Unfortunately due to time constraints and the effort-to-mark ratio for this assignment, no UI was developed to change the GP parameters.

The logic pertaining to Genetic Programming is sandwiched between two comments, `BEGIN MARKING HERE` and `END MARKING HERE`.

Part A: Parameter Selection

The following parameters were found using trial, error and some intuition.

```
maxIterations = 15
maxDepth = 5
populationSize = 200
leafNodeProbability = 0.1
mutationProbability = 0.1
```

- `maxIterations`: the number of iterations / generations that GP is run.
- `maxDepth`: max depth for random tree generations. Will be ignored for subtree swaps.
- `populationSize`: number of programs/trees/candidate solutions in a generation. Stays constant accross generations.
- `leafNodeProbability`: the probability of a leaf node being chosen when randomly generating trees and the max depth has not been reached.
- `mutationProbability`: the probability of a mutation occuring instead of crossover.

Some observations

Note that the final best fitness is not necessarily the best fitness ever seen. This is because the survivor selection used was generational replacement with no elitism. As such, there is no guarantee that the solution fitness will be monotonically increasing.

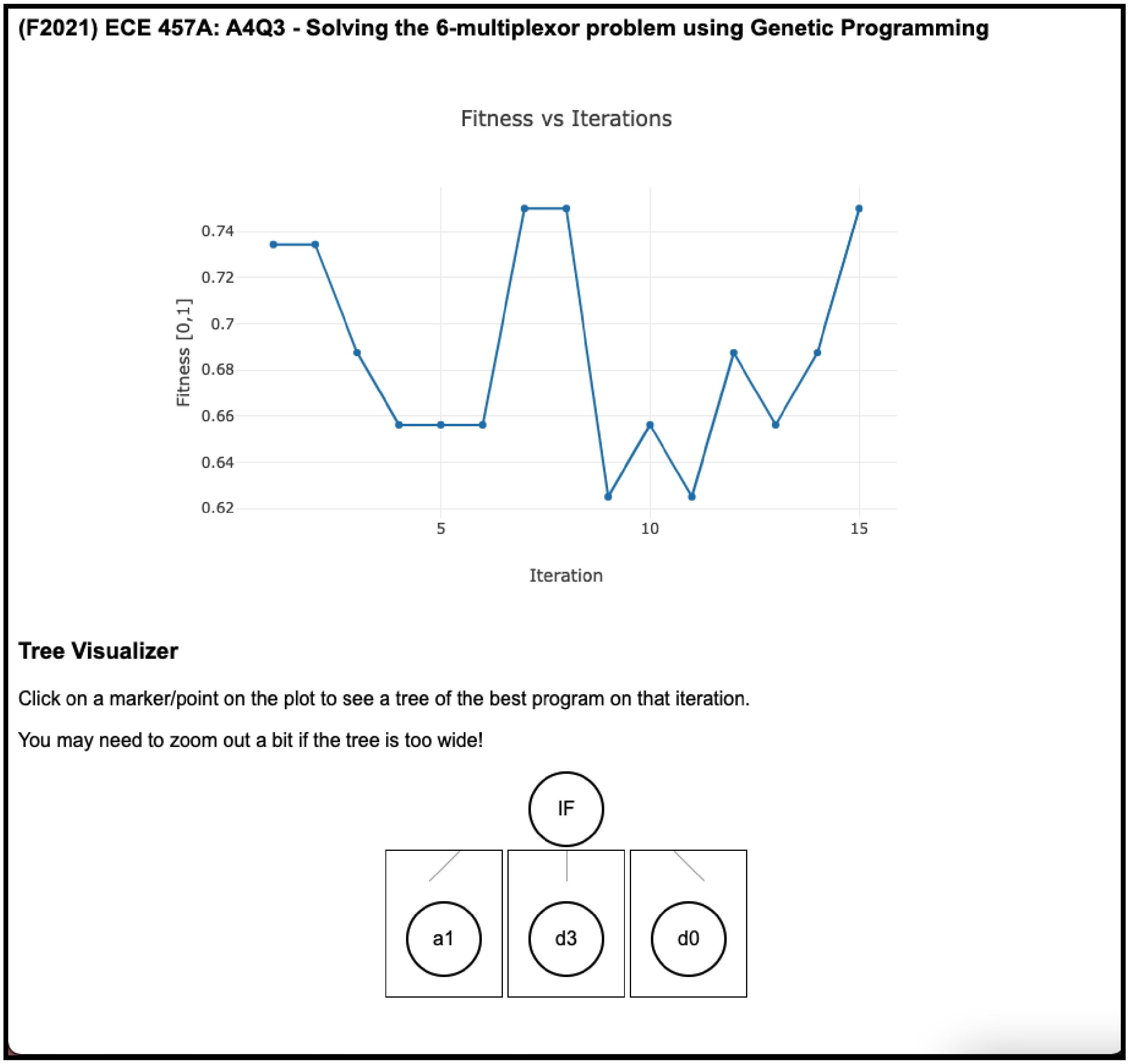
A relatively low iterations, 15 was used as it was found that sometimes the values would converge to a local optimum with a single-node-tree, only the term d3 would yield a fitness value of `0.625`. This single-node-tree would quickly take over a large proportion of the population, at which point crossover between two of these trees would create identical offspring. Mutation being explotative did not serve to escape this local optimum empirically.

`leafNodeProbability` was also lowered in order to prevent the early generation of these single-node-trees. `mutationProbability` was decreased in order to increase exploration using the subtree swapping crossover. Decreasing mutation probability increases crossover probability since in GP, mutation and crossover are mutually exclusive.

PART B,C,D: Results

Several runs were made, the UI created is able to show the plot of fitness vs iterations (Part B) and a tree representation of the finalist program (Part D). The fitness of the finalist (Part C) will be shown in the title.

Run 1: `Fitness: 0.75`



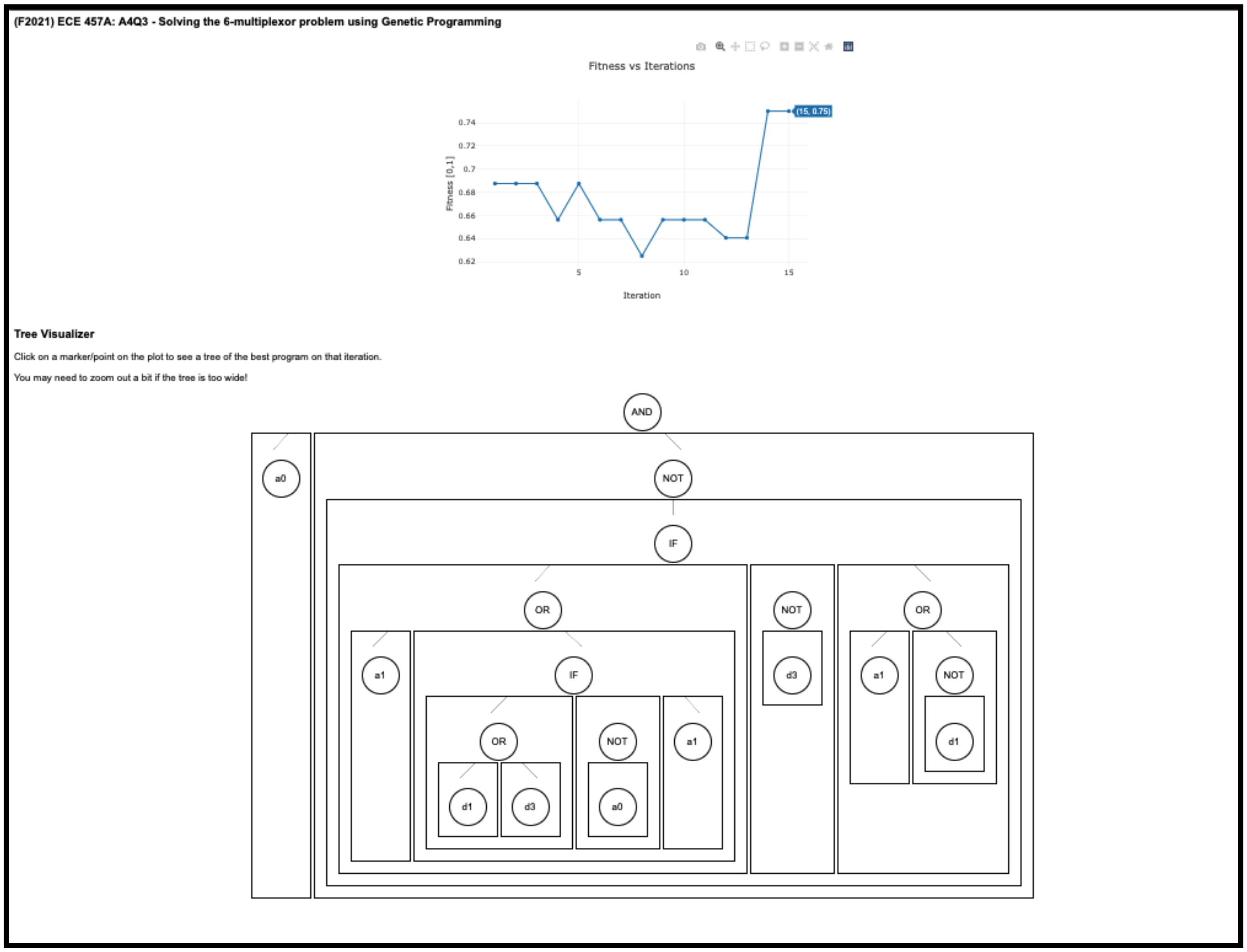
(F2021) ECE 457A: A4Q3 - Solving the 6-multiplexor problem using Genetic Programming

Tree Visualizer

Click on a marker/point on the plot to see a tree of the best program on that iteration.

You may need to zoom out a bit if the tree is too wide!

Run 2: `Fitness: 0.75`

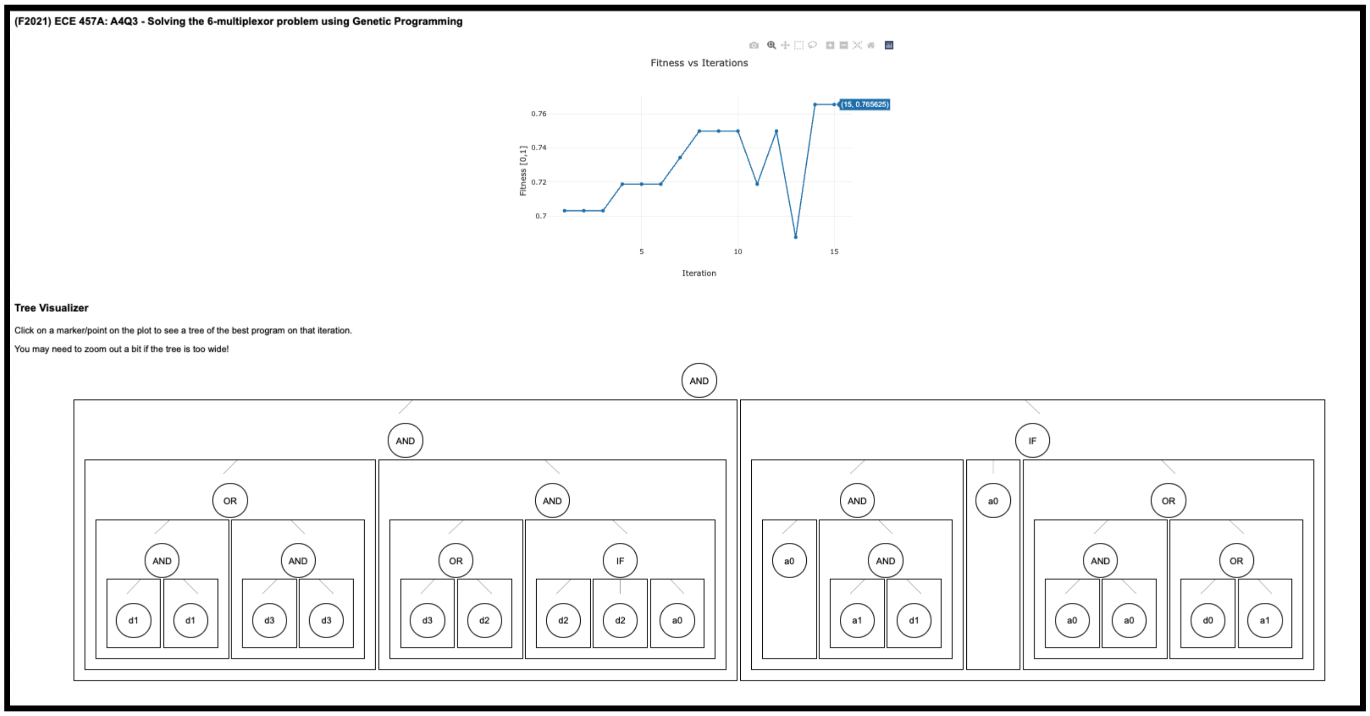

(F2021) ECE 457A: A4Q3 - Solving the 6-multiplexor problem using Genetic Programming

Tree Visualizer

Click on a marker/point on the plot to see a tree of the best program on that iteration.

You may need to zoom out a bit if the tree is too wide!

Run 3: `Fitness: 0.765625`


(F2021) ECE 457A: A4Q3 - Solving the 6-multiplexor problem using Genetic Programming

Tree Visualizer

Click on a marker/point on the plot to see a tree of the best program on that iteration.

You may need to zoom out a bit if the tree is too wide!