Assignment 4

William Li 20720929
Code can also be found here: https://github.com/liwamdaman/ECE457A/tree/main/A4

Q3. NetLogo PSO model

a) Parameter experimentation:

Experimenting with population size (using default Netlogo parameters: landscape smoothness rate of 20, particle inertia of 0.98, particle speed limit of 10, attraction to personal best of 2, and attraction to global best of 1):

	Population = 30	**Population = 80**
Speed to converge	Run #1: Converges to local maximum 0.9921, unable to leave local maximum	Run #1: Seems to converge to a local maximum first, but then manages to break free and the swarm finds the global minimum in 445 ticks.
	Run #2: Converges to global maximum 1 in 20 ticks	Run #2: Very quickly finds global maximum in 25 ticks
	Run #3: Converges to local maximum 0.9384, unable to leave local maximum	Run #3: Quickly finds global maximum in 60 ticks
Ability to find global optima	Run #1: Unable	Run #1: Able
	Run #2: Able	Run #2: Able
	Run #3: Unable	Run #3: Able

From observation, it seems that when using a population size of 30 the ability to converge on the global maximum is more luck based. It seems that if the initial trajectories of the swarm early on in the simulation is close to the global maximum, then it can be found successfully. Otherwise, the population size of 30 is not able to converge to the global maximum if it is not randomized to be particularly close to the swarm.
A population size of 80 is more consistent with its ability to find the local maximum. It seems that the larger swarm is able to cover more area and is more spaced out at the beginning, allowing it to have a higher chance to find the maximum early on. As well, even if the swarm gets stuck in a local maximum, it's able to break free and potentially find the global maximum.

Experimenting with particle speed limits (using default Netlogo parameters: landscape smoothness rate of 20, population size of 50, particle inertia of 0.98, attraction to personal best of 2, and attraction to global best of 1):

	Speed limit = 2	**Speed limit = 6**
Speed to converge	Run #1: Converges to local maximum 0.9449, unable to leave local maximum	Run #1: Finds global maximum in 81 ticks
	Run #2: Converges to local maximum 0.9912, unable to leave local maximum	Run #2: Finds global maximum in 204 ticks
	Run #3: Finds global maximum in 77 ticks	Run #3: Converges to local maximum 0.9603, unable to leave local maximum
Ability to find global optima	Run #1: Unable	Run #1: Able
	Run #2: Unable	Run #2: Able
	Run #3: Able	Run #3: Unable

When using a higher limit, we observe a similar behaviour to when we use a higher population size, i.e. higher consistency, leading to better performance. This is likely because the higher speed limit allows particles to cycle around rapidly, covering space and exploring more of the landscape (and potentially finding the global optima).

Experimenting with particle inertia (using default Netlogo parameters: landscape smoothness rate of 20, population size of 50, particle speed limit of 10, attraction to personal best of 2, and attraction to global best of 1):

	Particle inertia = 0.60	**Particle inertia = 0.729**
Speed to converge	Run #1: Finds global maximum in 33 ticks	Run #1: Converges to local maximum 0.9913, unable to leave local maximum
	Run #2: Converges to local maximum 0.9862, unable to leave local maximum	Run #2: Finds global maximum in 14 ticks
	Run #3: Converges to local maximum 0.9265, unable to leave local maximum	Run #3: Converges to local maximum 0.9642, unable to leave local maximum
Ability to find global optima	Run #1: Able	Run #1: Unable
	Run #2: Unable	Run #2: Able
	Run #3: Unable	Run #3: Unable

When using lower particle inertia (0.6), the particles have a stronger tendency to purely move towards forces of attraction (the personal best and global best results). As such, much less exploration is performed since particles immediately can change direction and move towards good solutions. This exploitation-heavy configuration performs very well when the particles are close to the global optima, but struggles when the global optima is far away.
Using a particle inertia of 0.729 seems to be a better balance of exploitation vs exploration.

Experimenting with personal-best and global-best factors (using default Netlogo parameters: landscape smoothness rate of 20, population size of 50, particle inertia of 0.98, particle speed limit of 10):

	Attraction to personal best = Attraction to global best = 1.7	**Attraction to personal best = Attraction to global best = 1.494**
Speed to converge	Run #1: Finds global maximum in 13 ticks	Run #1: Converges to local maximum 0.9986, unable to leave local maximum
	Run #2: Converges to local maximum 0.9975, unable to leave local maximum	Run #2: Finds global maximum in 32 ticks
	Run #3: Converges to local maximum 0.9826, unable to leave local maximum	Run #3: Converges to local maximum 0.8798, unable to leave local maximum
Ability to find global optima	Run #1: Able	Run #1: Unable
	Run #2: Unable	Run #2: Able
	Run #3: Unable	Run #3: Unable

dwa

b) <u>The difference between the motion formulation of the NetLogo implementation and classical PSO:</u>
   - One difference is that particle inertia is used in the personal best and global best components of the computed velocity (this is not done in classical PSO):

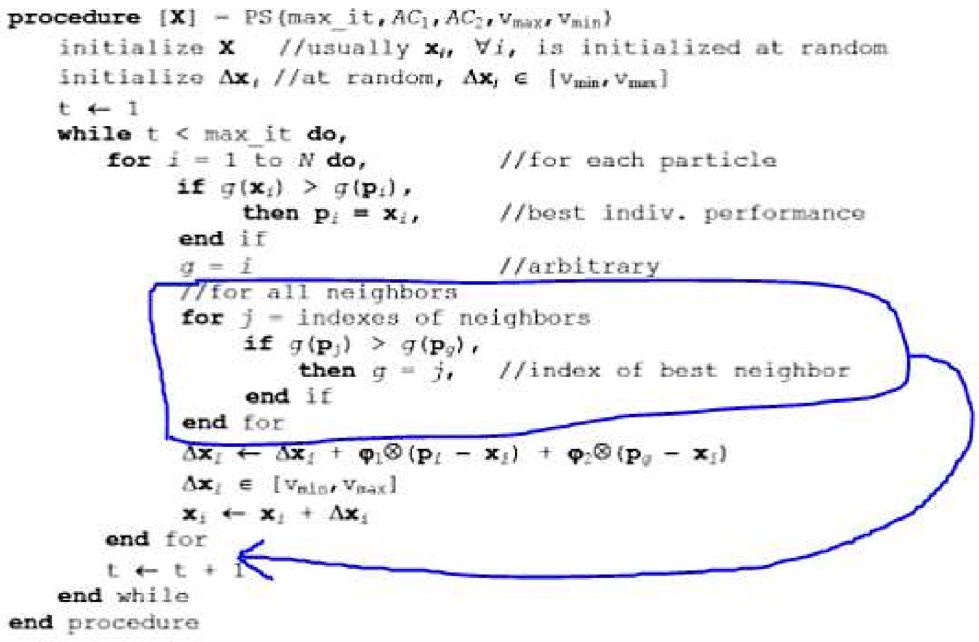
```
; change my velocity by being attracted to the "personal best" value I've found so far
facexy personal-best-x personal-best-y
let dist distancexy personal-best-x personal-best-y
set vx vx + (1 - particle-inertia) * attraction-to-personal-best * (random-float 1.0) * dist * dx
set vy vy + (1 - particle-inertia) * attraction-to-personal-best * (random-float 1.0) * dist * dy

; change my velocity by being attracted to the "global best" value anyone has found so far
facexy global-best-x global-best-y
set dist distancexy global-best-x global-best-y
set vx vx + (1 - particle-inertia) * attraction-to-global-best * (random-float 1.0) * dist * dx
set vy vy + (1 - particle-inertia) * attraction-to-global-best * (random-float 1.0) * dist * dy
```

Q2. PSO implementation analysis

a) Since neighbour's best is checked within the for loop and can be updated each loop, the PSO implementation uses asynchronous updating. This is because the neighbourhood best is updated after each particle computation. Personal best for a particle is also computed immediately as well within the for loop.

b) In order to change the algorithm to work in synchronous mode, we would need to compute the neighbourhood best to be computed outside the for loop. The following move is needed:



```
procedure [X] = PS(max_it, $AC_1$, $AC_2$, $v_{max}$, $v_{min}$)
    initialize X    //usually $x_{ij}$, $\forall i$, is initialized at random
    initialize $\Delta x_i$ //at random, $\Delta x_i \in [v_{min}, v_{max}]$
    t ← 1
    while t < max_it do,
        for i = 1 to N do,           //for each particle
            if $g(x_i) > g(p_i)$,
                then $p_i = x_i$,       //best indiv. performance
            end if
            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
            $\Delta x_i \leftarrow \Delta x_i + \varphi_1 \otimes (p_i - x_i) + \varphi_2 \otimes (p_g - x_i)$
            $\Delta x_i \in [v_{min}, v_{max}]$
            $x_i \leftarrow x_i + \Delta x_i$
        end for
        t ← t + 1
    end while
end procedure
```

c)

i) When C1 is set to zero, then the next iteration of velocity for a particle will not be affected by the personal best of that particle. In other words, velocity will be computed by only the current inertia of the particle and the current neighborhood best of the swarm.

ii) When C2 is set to zero, then the next iteration of velocity for a particle will not be affected by the neighborhood best of the swarm. In other words, velocity will be computed by only the current inertia of the particle and the current personal best of the particle.

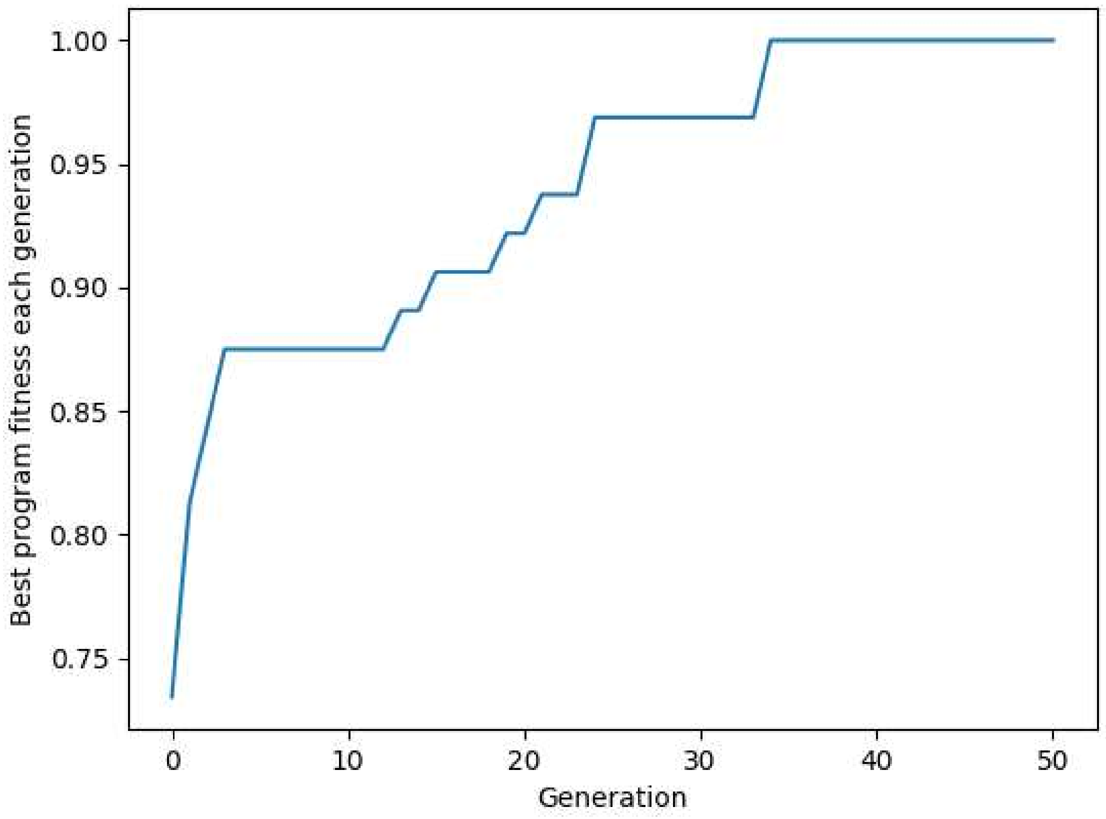
iii) The importance of the W (inertial) parameter is that it allows a particle to continue moving slightly in its current trajectory. This means that the velocity of a particle is not purely dictated by personal bests and neighborhood bests. This promotes exploration of the solution space. Without the W parameter, the PSO would act more like hill-climbing, and be susceptible to local optima.

Q3. Genetic programming implementation

a) Parameter Choices:

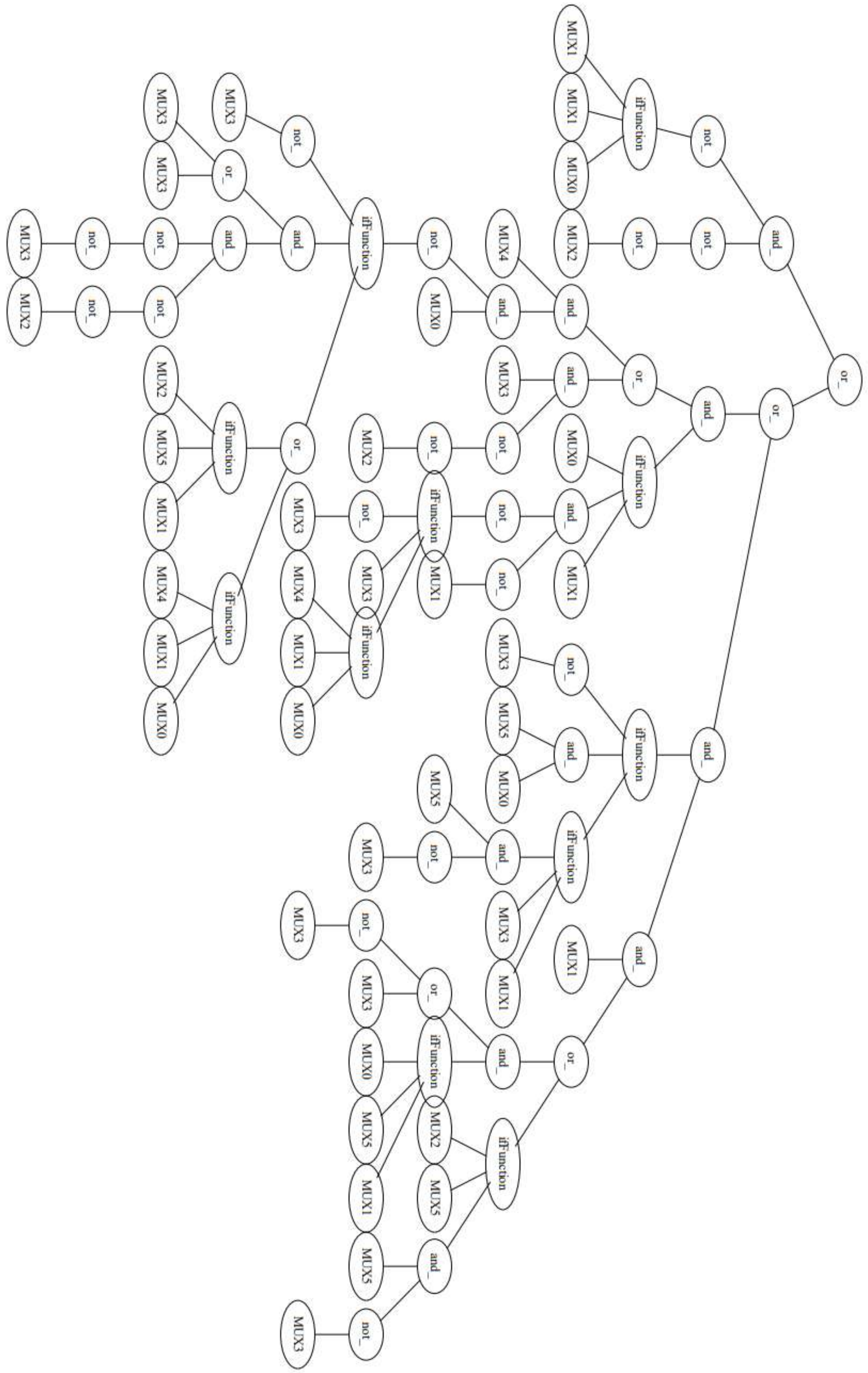
Population size	500
Number of generations	50
Selection method	Tournament
Tournament size	50
Crossover probability	0.8
Mutation probability	0.1
During mutation, generated tree height	Height in range of [2, 4]

b) Progress of the best program fitness in each iteration:



c) Fitness of the finalist program = 1 (success on 64/64 possible cases)

d) Tree of the finalist program:



MUX3
MUX2
not_
not_
MUX3
MUX3
not_
not_
MUX2
MUX5
MUX1
MUX4
MUX1
MUX0
MUX3
or_
and_
ifFunction
ifFunction
not_
and_
or_
MUX3
MUX4
MUX1
MUX0
MUX3
MUX3
ifFunction
MUX2
not_
MUX3
ifFunction
MUX3
not_
MUX3
MUX0
MUX5
MUX1
MUX5
not_
not_
MUX0
not_
ifFunction
MUX1
MUX5
not_
or_
ifFunction
MUX2
MUX5
and_
MUX4
and_
MUX3
not_
not_
not_
MUX3
MUX5
MUX0
and_
MUX3
MUX1
and_
ifFunction
MUX1
MUX1
MUX0
MUX2
and_
and_
MUX0
and_
MUX1
not_
and_
ifFunction
MUX1
or_
ifFunction
not_
or_
ifFunction
ifFunction
and_
not_
not_
and_
and_
and_
or_
or_