Assignment #3

Keshav Iyengar

Kevin Kearns

Question 1

1. a.

Each possible solution can be represented by a chromosome (vector) of 3 values representing K_p, T_I, and T_D. For a population of 50, a 3x50 matrix can be used with each column representing a different chromosome in the population.

b.

The fitness function used to evaluate a possible solution is shown below:

```
f = (7/10)*ISE + (1/10)*t_r*100 + (1/10)*t_s*10 + (1/10)*M_p*10;
f = 1/f;
```

First t_r is multiplied by 100, t_s is multiplied by 10, and M_p is multiplied by 10. This brings all these values to around the same order so that they are all seen and minimized in the final solution. Then, the weightings are applied. ISE is judged to be the most important value to minimize with a 0.7 weighting, and the other three values each have a weighting of 0.1 as they are less important to minimize than the ISE value but should still be minimized if possible. Because we want to minimize these values, we take 1/f as our fitness value and then try to minimize this value.

c.

The full code implementation can be seen in the matlab files which have been submitted with this assignment. First, a random 3x50 matrix is initialized with values in the correct range representing the first population of 50 chromosomes. Then a genetic algorithm is used to reproduce the best chromosomes and improve the solution space. First, each of the chromosomes has its fitness value calculated by the fitness.m matlab function. Then, chromosomes are selected in proportion to their fitness so that we have a total of 50 chromosomes to use as parents. These parents are then shuffled using the matlab randperm() function to arrange them in random pairs, and these pairs reproduce together at the crossover probability of 0.6. After crossover, mutation is performed on each gene of every chromosome with a probability of 0.25. After this, due to the elitism survival strategy being used, we check to make sure the best two chromosomes are present in the new population

space. If not, we add them in in random locations to make sure they are present in the next generation. This continues for 150 generations, at the end of which the genetic algorithm should improve on our initial random population with a population with much better fitness values. The fittest chromosome in our last generation is the one which has the best values of K_p, T_I, and T_D for use as controller parameters.

d.

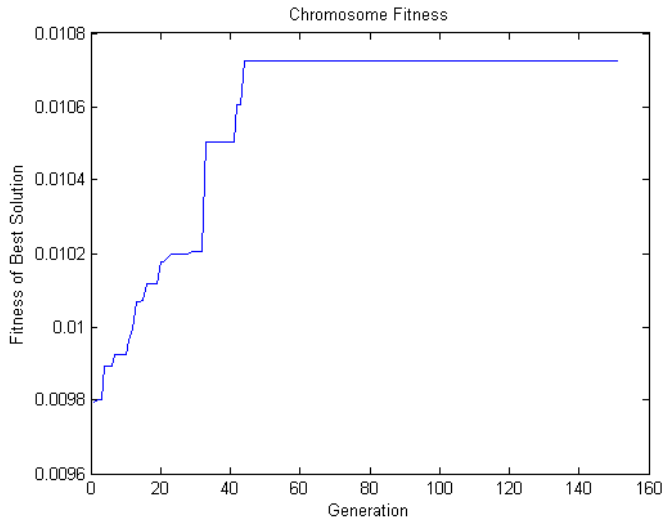

Figure 1:Fitness of best solution for 150 generations trial 1

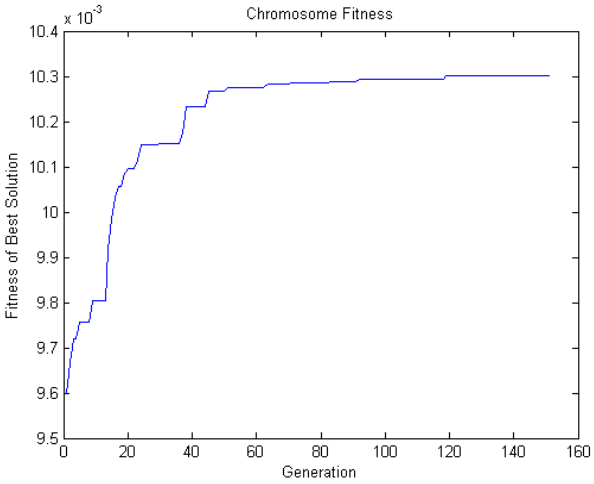

Figure 2:Fitness of best solution for 150 generations trial 2

The fitness of the best solution in each generation is plotted for two trial runs above.

e.

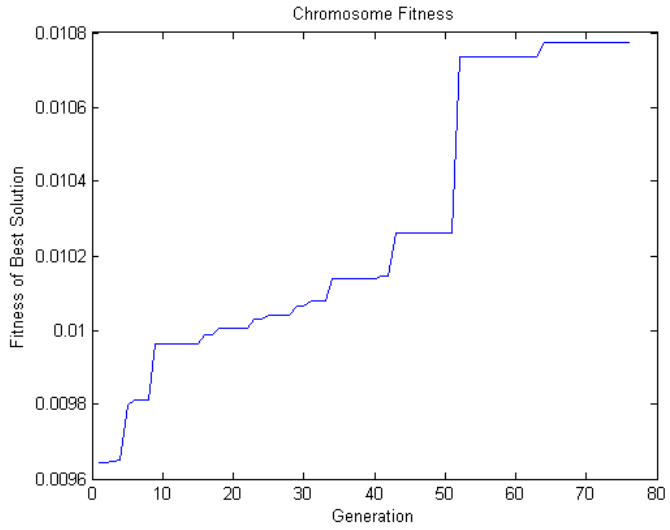

Figure 3:Run for generations = 75

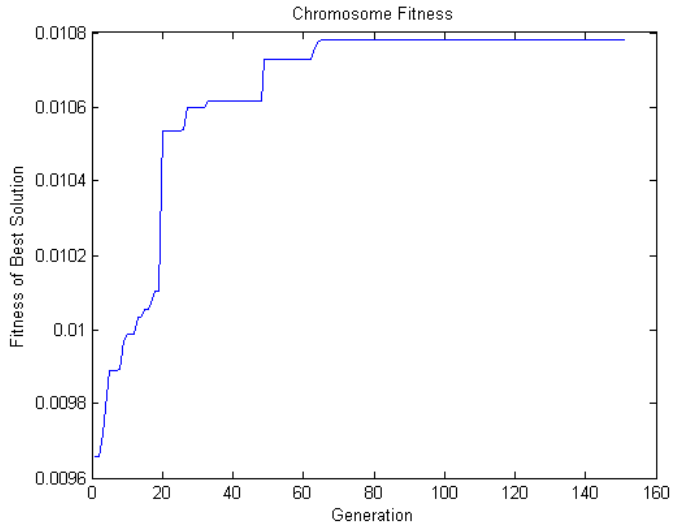

Figure 4:Run for generations = 150

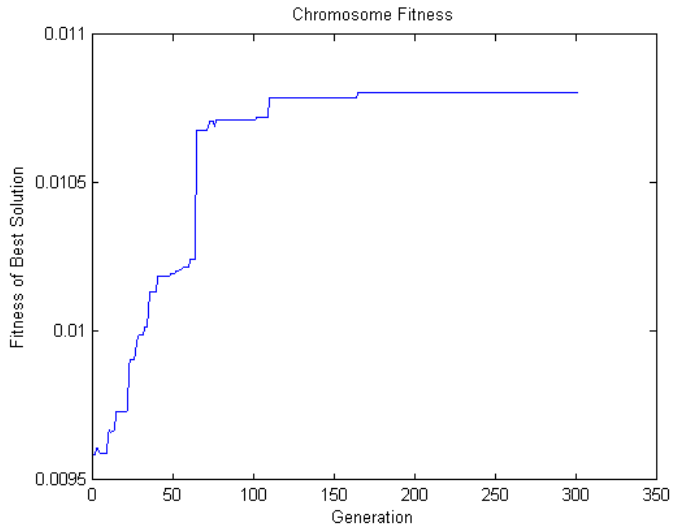

Figure 5: Run for generations = 300

As can be seen in the figures above, the best solution improves quickly in the initial generations and then plateaus. Using 75, 150 or 300 as the number of generations all yield approximately the same final result. Most of the generations after 75 provide only incremental improvements on the best solution generated by generations prior to that.

f.

<u>For population of 10:</u>

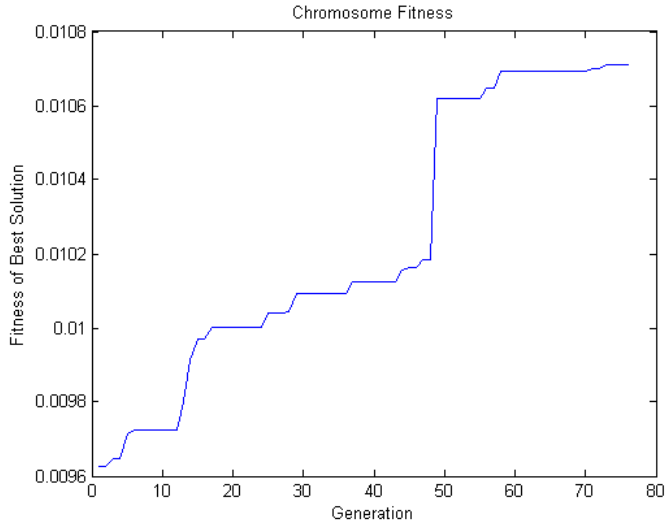

Figure 6: Population = 10 Generations = 75

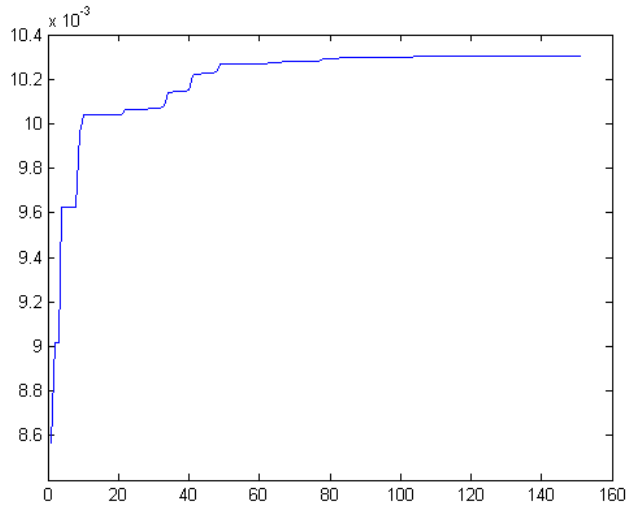

Figure 7:Population = 10 Generations = 150

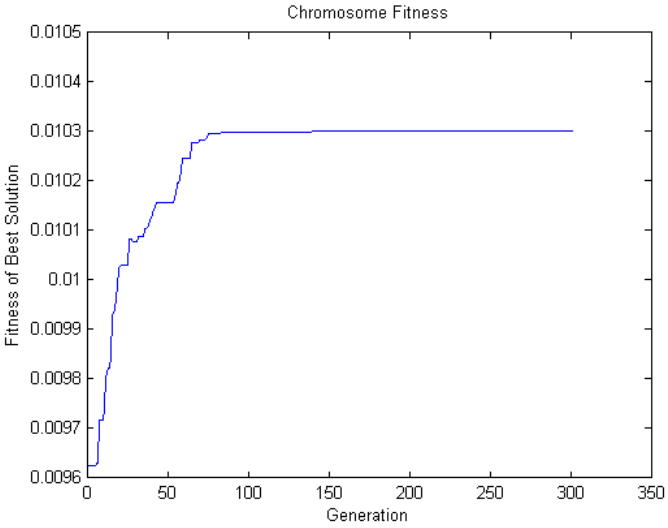

Figure 8: Population = 10 Generations = 300

For a population of 25:

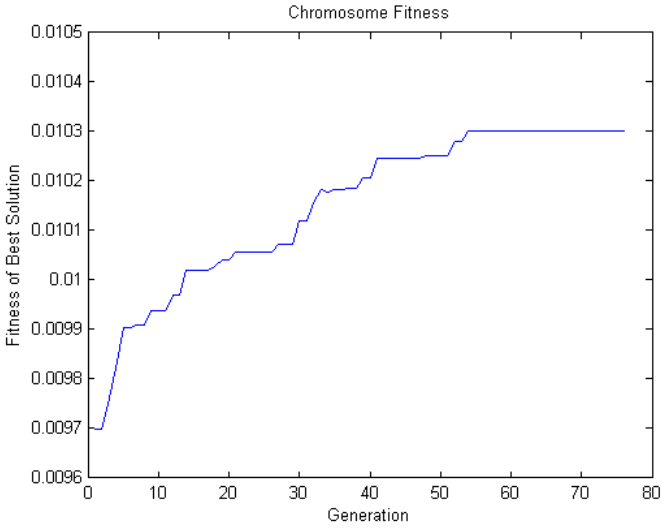

Figure 9:Population = 25 Generations = 75

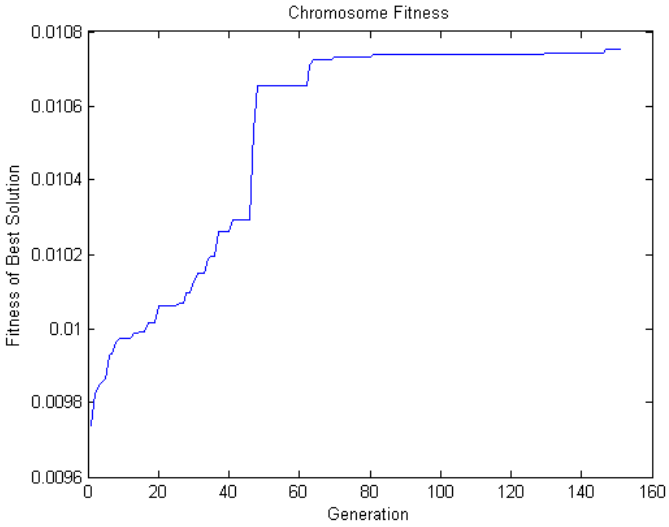

Figure 10:Population = 25 Generations = 150

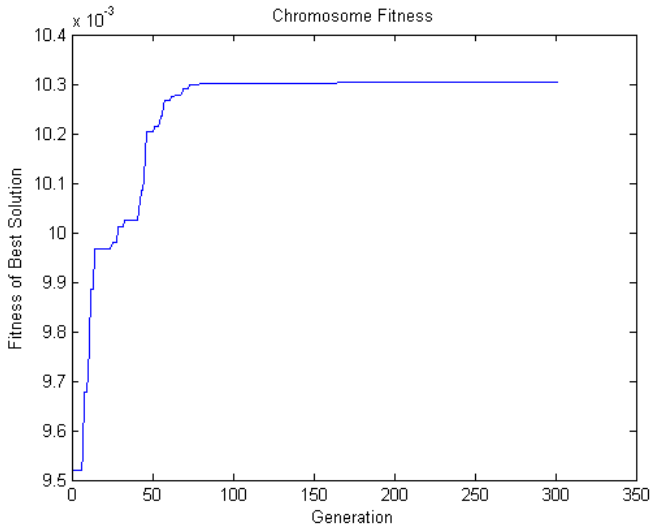

Figure 11:Population = 25 Generations = 300

For a population of 100:

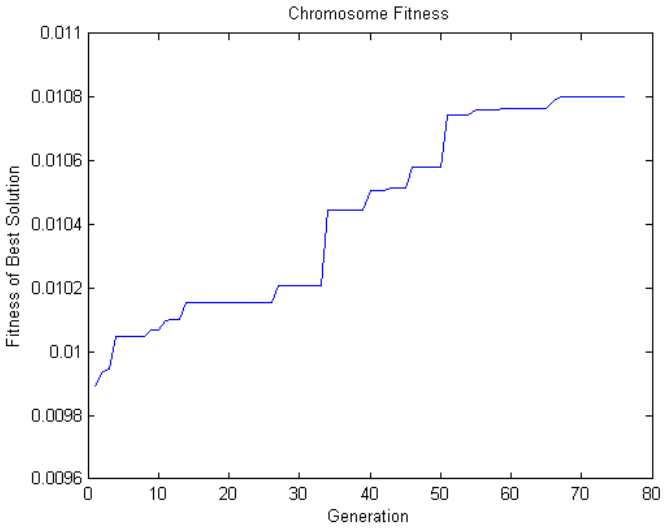

Figure 12:Population = 100 Generations = 75

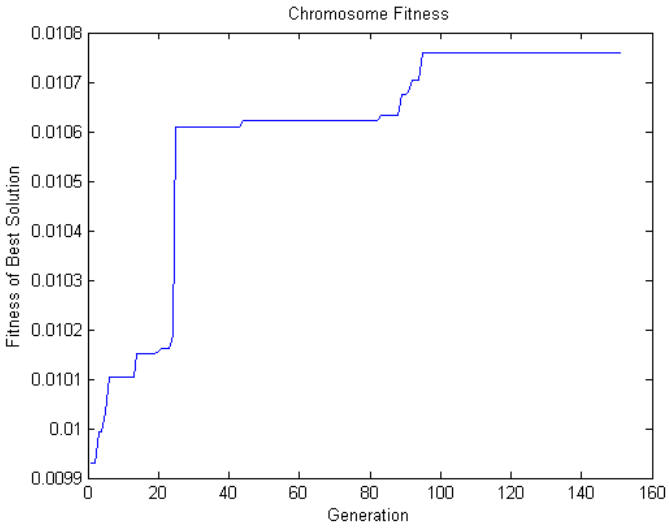

Figure 13:Population = 100 Generations = 150

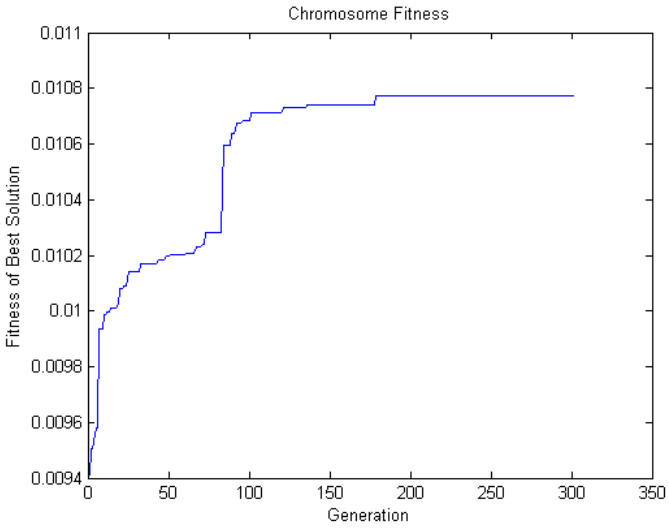

Figure 14:Population = 100 Generations = 300

From changing the population size and generation size and analyzing the results, we can draw two conclusions:

-Increasing the population size results in a better final solution on average.

-The population converges on the solution quickly, in most cases the best solution after 75 generations will be very close to the best solution after 300 generations

-When using the largest population of 100, it takes a bit longer to converge, and converges to around the best value seen at 150 generations

-To get a better solution, you must increase the population size, and with increasing population size it is necessary to have more generations to reach population convergence

-Having a large number of generations with a low population size will not improve the final solution

g.

For this part we use a population of 50 and 150 generations.

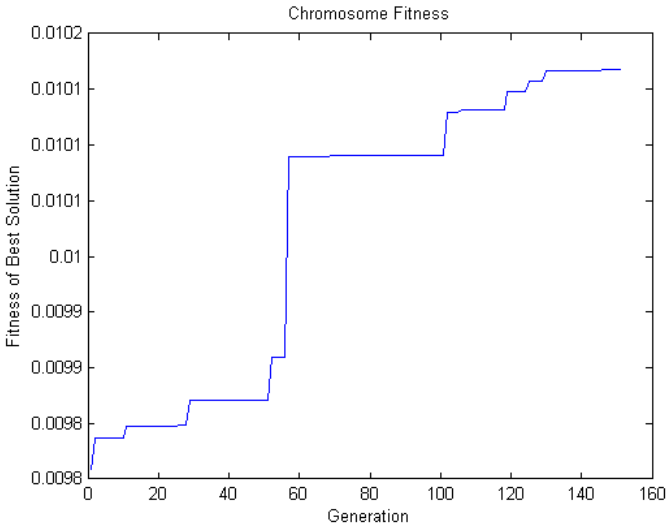

Figure 15: Low crossover probability(0.6) and low mutation probability(0.02)

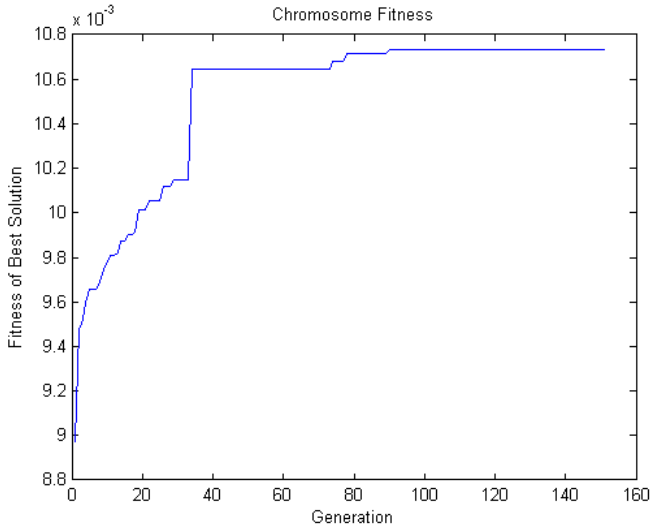

Figure 16:Low crossover probability(0.6) and high mutation probability(0.33)

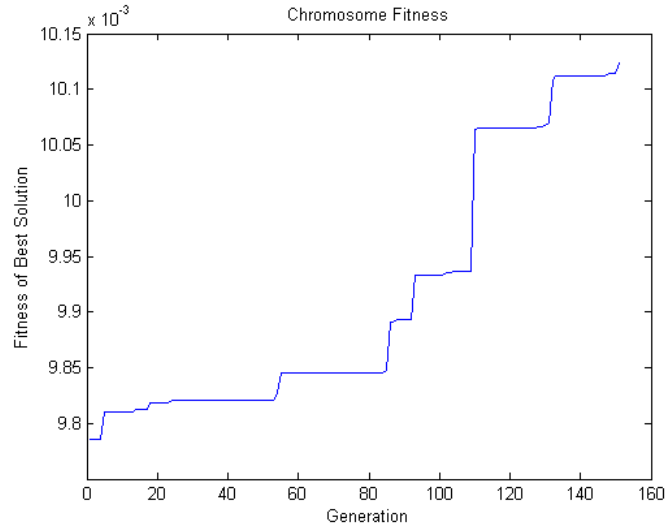

Figure 17:High crossover probability(0.9) and low mutation probability(0.02)

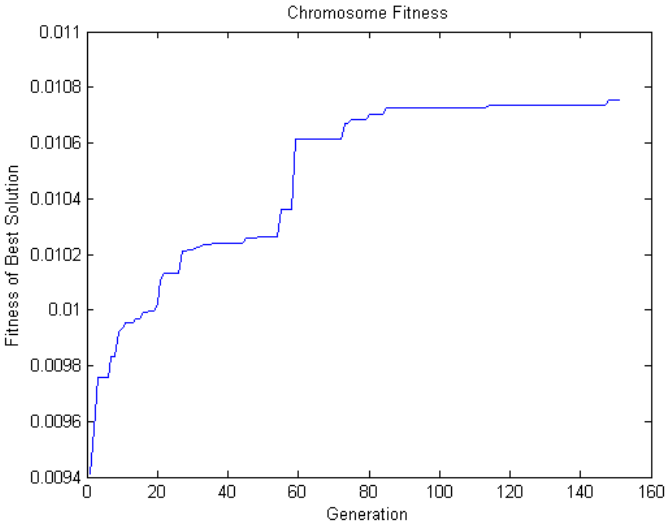

Figure 18:High crossover probability(0.9) and high mutation probability(0.33)

Looking at figures 15-18 above, some patterns become apparent. Graphs with low mutation have almost no incremental improvements, with almost all changes coming in large jumps which are a result of crossover. Graphs with high rates of mutation have numerous incremental improvements as small changes in the number improve the solution. The two test cases with low mutation had very poor final results, of around 10.15E-3, independent of the crossover probability. The other

two cases with high rates of mutation had good results of around 10.80E-3, independent of crossover probability. It is clear that in this problem, high rates of mutation are necessary to exploit the solution space reached by a crossover and tp get a good solution. Low rates of mutation resulted in poor final solutions. The rate of crossover did not have a large effect on the final solution, as the final solution is usually the result of a few crossovers(less important) and numerous small mutations(very important). The best solution had both a high rate of crossover and a high rate of mutation, as both exploration and exploitation was maximized.

Question 2

2. a)

General Observations	-Once any ant has found a food source the other ants quickly converge on it and consume it quickly(with enough ants) -if by change a small piece of food is missed by the ant swarm it is extremely difficult for them to find it again -closer food sources are found and consumed faster, farther away ones are found and consumed later -if an ant finds a food source but there are no ants close by to take advantage of the trail, the trail will dissipate -ant movement is random until a food source is found and trail is made -ants can lose track of a food source even if some ants have found it, especially if there is another food source grabbing their attention -ants focus on one food source at a time			
Population	Diffusion Rate	Evaporation Rate	Finish Time (ticks)	Specific Observations
30	40	10	Extremely long, ended	-ants focus on closest food source first, then the next farthest, then the next farthest -ants cannot form a trail to the farthest food sources
30	40	20	Extremely long, ended	-Faster evaporation rate means the trail disappears faster -Food sources are found and lost multiple times before they are completely consumed -trail is never established to far away food sources -one ant finding a food source does not mean others will immediately find it too -as food sources become smaller and smaller it takes the ants longer and longer to finish them off

30	80	10	10260(extremely long, but let it run)	-Faster dissipation appears to create a less concentrated trail and reduces performance -Takes an extremely long time for the ants to gather far away food sources as a trail is never established
50	40	10	5230	-Increasing the ants from 30 to 50 results in a big performance improvement -Ants establish path easier, although for the farthest food sources still have trouble establishing the path
50	80	10	5730	-higher diffusion rate helps ants converge and create a trail faster -It does not help much for the farthest food source which is why the time is similar to before -still not enough ants to form trails to the farthest away food sources for extended periods of time
100	40	10	1380,1140	-Increasing ants to 100 causes a huge performance increase -trail is able to be formed to even the farthest away food sources
100	40	20	3200	-increased evaporation means trails dissipate quickly and can only be formed to the closest food sources, hindering performance
100	80	10	1670,1710	-increased diffusion rate creates more dispersed trails and seems to confuse the ants, reducing performance time
100	80	20	3800	-increased diffusion and increased evaporation provide the worst performance for this number of ants and prevent trails from being established
Conclusions	-An ant population of 30 is not large enough to establish trails too far away food sources and it takes an extremely long time for them to get all the food -Increasing ant population is the best way to increase performance. 100 ants enables pheromone trails to be made to food sources of all distances -Evaporation rate should be low, that way ants have more time to follow the other ants to the food source. A higher evaporation rate reduces performance and hinders trails from being made			

	-Increased diffusion creates less concentrated trails and decreases performance. It seems to confuse ants. Smaller condensed trail work better. -The best performance came from using 100 ants, a diffusion rate of 40 and an evaporation rate of 10

Question 3

3. i)

Population	Speed Limit	Inertia	Personal-Best	Global-Best	Run Time(ticks)	Best Solution Found	Global optima finds/ trials
30	2	0.6	1.7	1.7	--,--,20,--,28,18,39,--,--,--	0.895,0.9248,1,0.9683,1,1,1,0.8356,0.9397,0.7715	4/10
30	2	0.6	1.494	1.494	162, --,97,--,--,--,8,--,--,--	1.0, 0.9087, 1,0.9409,0.9517,0.9443,1,0.9211,0.897,0.9588	3/10
30	2	0.729	1.7	1.7	--,--,--,33,--	0.9972, 0.8857,0.9507,1,0.8693	1/5
30	2	0.729	1.494	1.494	--,--,15,22,11,103,11,--,--	0.9744,8424,1,1,1,1,1,0.954,0.9557	5/10
30	6	0.6	1.7	1.7	--,--,--,16,--	0.8692, 0.989,0.9322,1,0.9286	1/5
30	6	0.6	1.494	1.494	20, --, --	1, 0.9625, 0.9016	1/3
30	6	0.729	1.7	1.7	--,--,--,--	0.9417,0.9068,0.9637,0.9372	0/4
30	6	0.729	1.494	1.494	28,--,--,--,--	1,0.894,0.9713,0.9758,0.8813	1/5
80	2	0.6	1.7	1.7	--,25,--,33,--,--,12,--,--,--	0.9706,1,0.9503,1,0.9493,0.8255,1,0.9857,0.9929,0.9625	3/10
80	2	0.6	1.494	1.494	--,--,7,--,17,26,--,--,15,--	0.8782,0.9377,1,0.9861,1,1,0.9214,0.8597,1,0.9539	4/10
80	2	0.729	1.7	1.7	--,6,10,48,--	0.9777,1,1,1,0.9963	3/5
80	2	0.7	1.49	1.4	41,--,--,--,	1,0.9904,0.8672,0.7764,1,1,1,1	6/10

		29	4	94	15,22,28,42,--,21	,0.945,1	
80	6	0.6	1.7	1.7	--,16,--,--,19,750,10,--,5,--	0.9699,1,0.9467,0.8868,1,1,1,0.8873,1,0.9629	5/10
80	6	0.6	1.494	1.494	--,--,12,--,--,269,5,--,9,9,	0.8993,0.9444,1,0.9988,0.9831,1,1,0.9694,1,1,	5/10
80	6	0.729	1.7	1.7	--,19,--,11,10,13,--,7,--,--,	0.9888,1,0.9477,1,1,1,0.9926,1,0.9977,0.96	5/10
80	6	0.729	1.494	1.494	--,9,15,--,10,20,9,--,17,9	0.9717,1,1,0.9792,1,1,1,0.9816,1,1	7/10

REPORT OBSERVATIONS:

-Higher population with more particles is better at finding global optima and provides the best improvement

-1.494 attraction provides a small improvement on 1.7 attraction in the success rate of finding global optima, being less attracted to personal and global bests will allow for more exploration and reduce getting stuck in local optima

-6 speed limit instead of 2 has a similar rate of convergence on the global maximum, however, it converges to the global maximum much faster

-0.729 inertia provides better performance than an inertia of 0.6 and has a higher success rate of finding the global optima. Higher inertia will reduce the change in motion of the particle and allow it to shoot past local optima without getting stuck.

ii) The NetLogo PSO implementation is different from the classical implementation in that the NetLogo library works to optimize the function made up of a discrete grid of values. In the classical PSO, the solution space is continuous. This difference can be seen from the following piece of code**, where each discrete patch of the solution space will have its own value:

```
patches-own
[
  val  ; each patch has a "fitness" value associated with it
       ; the goal of the particle swarm is to find the patch with the best fitness value
]
```

***Code sample is from: Stonedahl, F. and Wilensky, U. (2008). NetLogo Particle Swarm Optimization model. http://ccl.northwestern.edu/netlogo/models/ParticleSwarmOptimization. Center for Connected Learning and Computer-Based Modeling, Northwestern University, Evanston, IL.*

b)

Algorithm Description:

The Particle Swarm Optimization algorithm is a simple one to implement. In steps, below, the algorithm is described. Also look at the octave (matlab script file) code for comments on each line implemented.

1. Define the problem
    1. Cost function definition
    2. Variable size and min and max constraints on optimization variables
2. Parameters of PSO
    1. Number of iterations (or other stopping condition)
    2. Population count
    3. W, C1, C2 constants
3. Initialization
    1. Define the particle (initialization the particle struct)
        1. Position
        2. Velocity
        3. Cost
        4. Best.Cost
        5. Best.Position
    2. Create an array of empty particles with random values within constraints for position
    3. Set global best (or neighborhood best) to infinity since this is a minimization problem

4. Calculate costs for each particle set global best (or neighborhood best)

4. Main Loop

    1. Two for loops, one for number of iterations and another for each particle
    2. Apply velocity equation
    3. Update particle position
    4. Calculate cost
    5. Update global (or neighborhood) best

Part 1 Simple PSO:

For the simple, we set W = 0 so there is no inertial component in the velocity equation. In Figure 1, the result is a quick convergence.

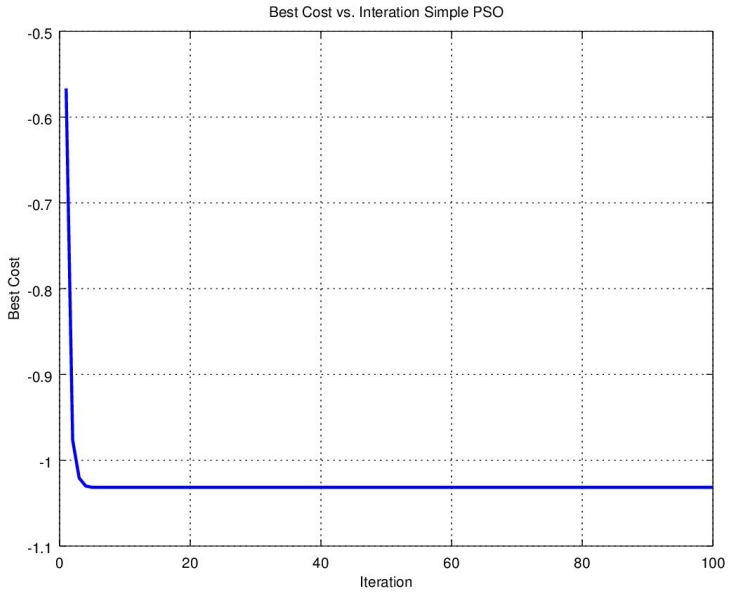

Figure 19: Simple PSO Best Cost Optimization

W = 0

C1 = 2

C2 = 2

With an Internal Implementation:

For this PSO, an inertial parameter is set. As seen in Figure 2, convergence is slower.

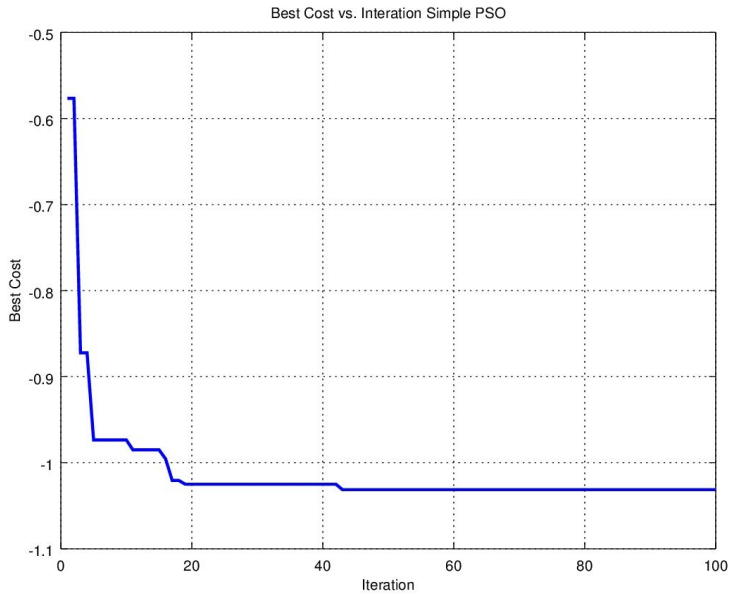

Figure 20: Inertial PSO Best Cost Optimization

W = 1

C1 = 2

C2 = 2

Part 2 (Vmax Implementation):

For this PSO, a maximum velocity was set and checked by checking the maximum velocity. If it was greater than the maximum velocity, a velocity with the same direction but a magnitude of the maximum velocity was set as the new velocity.

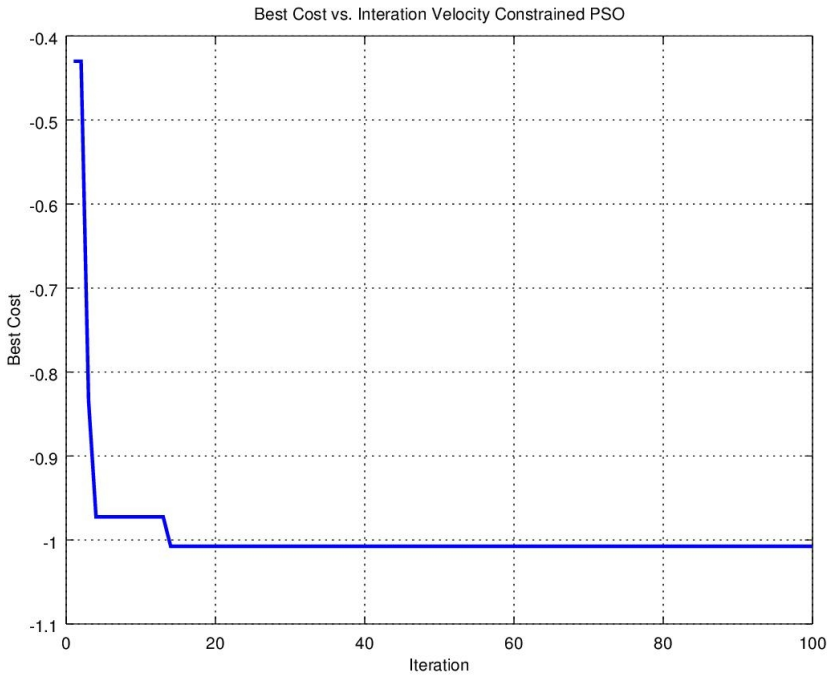

Figure 21: Velocity Constrained PSO

W = 1

C1 = 2

C2 = 2

VMax = 100

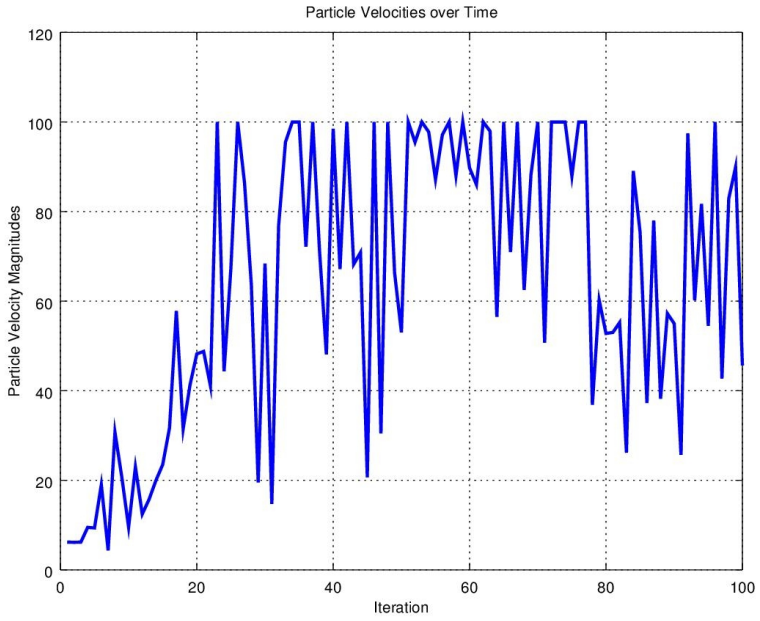

Figure 22: Particle Velocities in Constrained Velocity PSO

W = 1

C1 = ½

C2 = ½

VMax = 100

Part 3: Neighborhood Solution for Part 1 and Part 2:

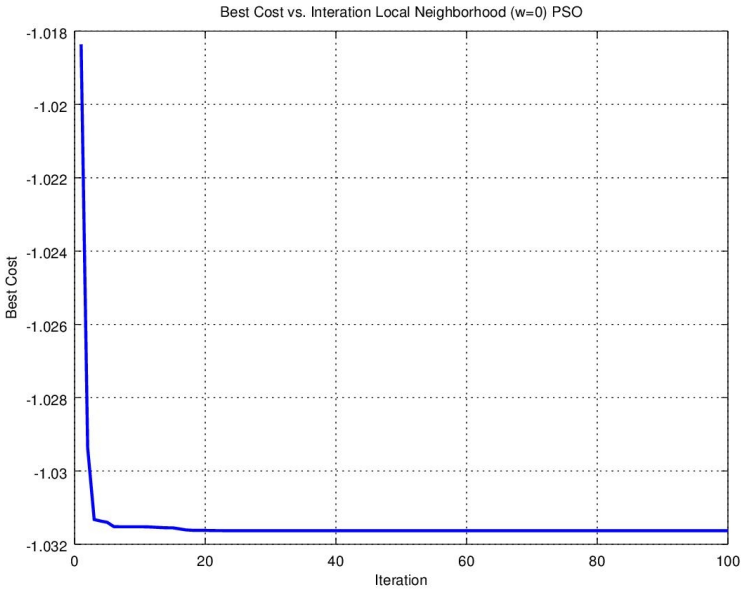

Figure 23: No inertia Neighborhood PSO

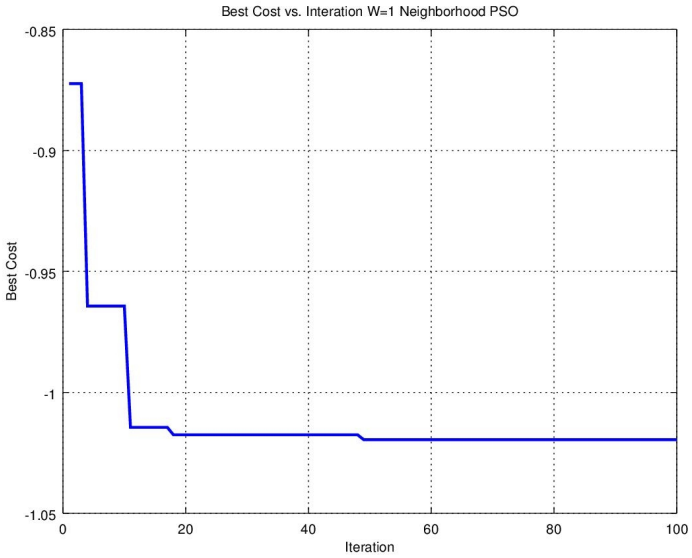

Figure 24: Inertial Neighborhood PSO

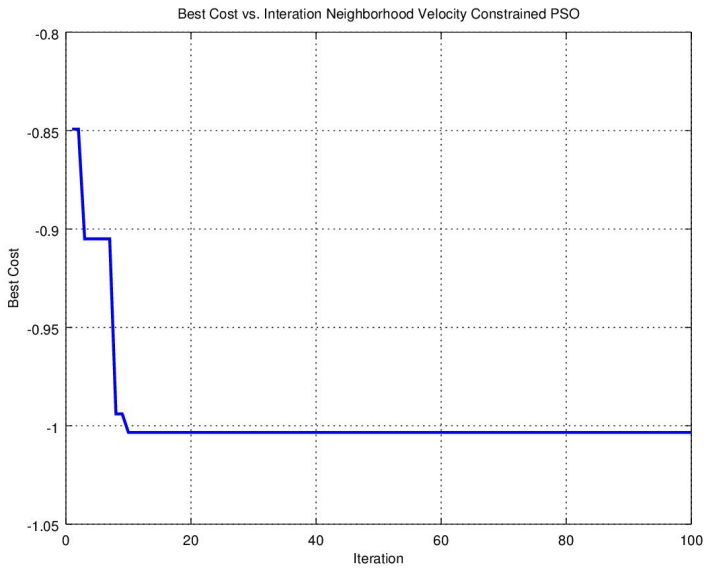

Figure 25: Neighborhood Velocity Constrained PSO

Part 4: Guaranteed Convergence

From the notes in class, we can guarantee converge with the appropriate values for C1, C2, W.

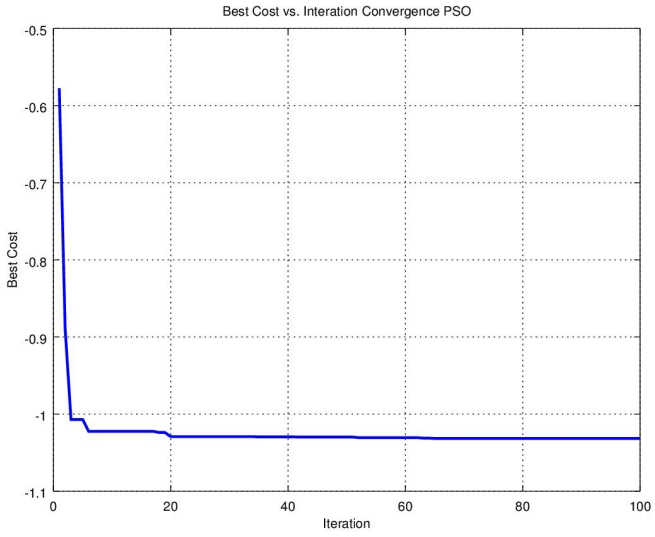

Figure 26: Convergence PSO

Part 5: Simple PSO with 5 Different Initial Seeds

In this algorithm, the seed initially randomized. Running the algorithm five times, there are the best found cost.

Run Number	Best Found Cost (z value)
1	-1.0010
2	-0.97319
3	-1.0231
4	-0.94769
5	-1.029