Assignment 3

Biomimetic Algorithms

Justin Self

AERO 500 Aerospace Modeling and Simulation

CAL POLY
Aerospace Engineering

May 30, 2025

Problem 1

Task 1 (25 points): Implement Ant Colony Optimization (ACO) to solve the 5 node network example starting on slide 16 from class lecture notes. Submit your Python code and any relevant plots or other documents. Find and print the final optimal path your ants found and the value of that path.

Solution.

An ant colony optimization scheme was developed using Python from an object oriented programming approach. The final pheromone matrix, τ, is given in the text output below. The best solution was found using 200 ants over 50 generations. The optimal path between the cities is: $0 > 2 > 1 > 4 > 3$ with a total path cost of 52. The full code is appended to this document in Section 1.

Simulations explored using randomized starting locations for the ants, but starting all the ants from the same starting point (City 0) provided more consistent comparisons. The simulation could start from any city and yield the same results. The prompt asked for solutions using only three ants. The same solution as in the 200-ant over 50 generations solution was found using only three ants occasionally, but re-running the simulation resulted in slightly different path solutions with distances as high as 60. Keeping three ants but increasing to 500 generations still resulted in variable responses with path costs as high as 55 with varying path solutions. The best solution that consistently converged on the true optimal path was using at least 100 ants over 50 generations.

```
------------- After 50 generations, tau is: ------------
[[8.8818e-16 7.6923e+00 6.4756e-10 1.0174e-06 3.9328e-14]
 [1.8565e-09 8.8818e-16 1.6925e-13 2.3291e-14 7.6923e+00]
 [1.2444e-11 1.0155e-06 8.8818e-16 7.6923e+00 1.8415e-09]
 [7.6923e+00 6.5222e-10 1.0174e-06 8.8818e-16 1.5086e-11]
 [1.0162e-06 1.8417e-09 7.6923e+00 7.6085e-12 8.8818e-16]]
Best path is:
[[1.]
 [4.]
 [3.]
 [0.]
 [2.]]
Total path cost is:
52
```

Problem 2

Task 2 (25 points): Implement either the Particle Swarm Optimization (PSO) or the Bees Algorithm (BA) as discussed in class. Use your implementation of PSO or BA to find an estimate of the three dimensional Ackley Function. Submit your Python code and any relevant plots or other documents. Find and print the final minimal value of the Ackley Function found by your algorithm

Solution. A particle swarm optimization code was developed which solves for the global minimum value in the three-dimensional Ackley function, shown in Figure 1. The particle swarm code utilized an object-oriented-programming approach. The solution converged to the known minimum (0,0) in as few as 25 trials (with a close solution of 0.086) and higher precision over more trials (solution at 100 trials is: 1.097e-05). The contour plot with solutions at 50 trials is shown in Figure 2a with an initial best estimate initialization of $g = [5, 5]$. The solution converged quickly for all values of g tested and all trial values higher than 25. Section 2 contains all the Python code for this task.

The smallest value obtained in these tests was 3.997e-15, which was found after running 500 trials, shown in Figure 2b.

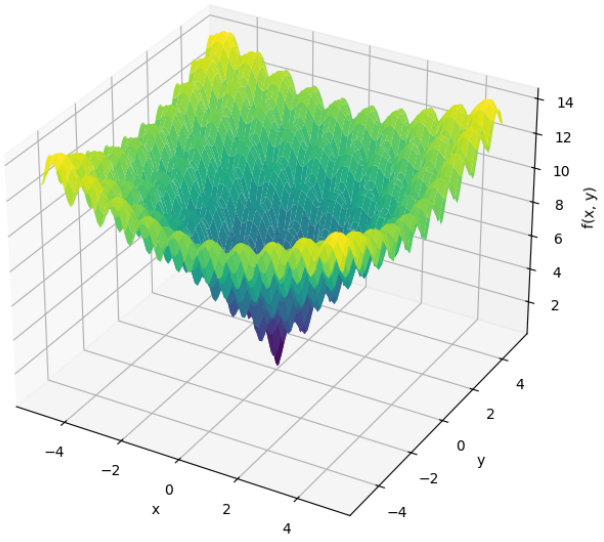
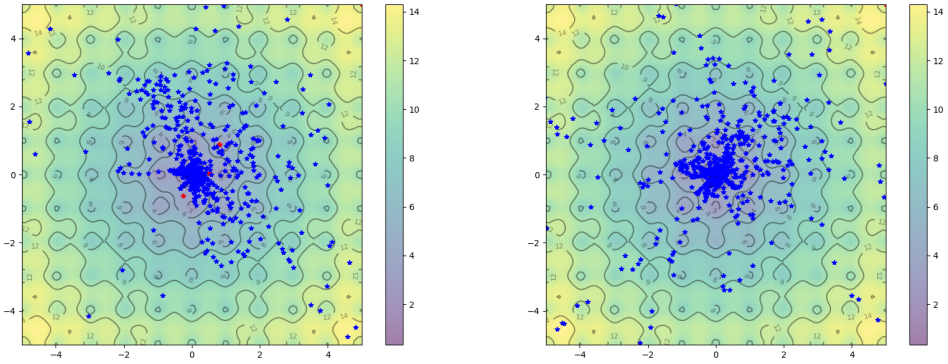


Figure 1: The 3D Ackley Function



(a) 50 trials (b) 500 trials

Figure 2: Particle Swarm Optimization Solving the Ackley Function

1 Appendix A: Ant Colony Optimization Code

```
1
2
3  # Ant Colony Optimization Main
4
5  from antClass import Ant # type:ignore
6  import numpy as np
7
8  # Fix print options for arrays
9  np.set_printoptions(precision=4)
10 # Reset to default settings
11 #np.set_printoptions()
12
13 # Initialize pheromone matrix
14 tau = np.ones((5,5))
15 rho = 0.5 # pheromone dissapation rate; fixed
16
17 for gen in range(50):           # number of generations
18     # Ant loop
19     tau = rho * tau
20
21     for i in range(100):          # number of ants
22         ant = Ant(i,tau) # type:ignore
23         ant.antTravel()
24         #ant.printStatus()
25         tau = tau + ant.pheromone       # update super tau
26
27     # end ant x3 loop
28 # end generation loop
29
30 print(f"------------ After {gen+1} generations, tau is: ------------")
31 print(tau)
32
33 # Find best path
34 bestpathIndex = [ [float((np.argmax(tau[0])))], [float(np.argmax(tau[1]))], [float
       (np.argmax(tau[2]))], [float(np.argmax(tau[3]))], [float(np.argmax(tau[4]))] ]
```

```
bestpathIndex = np.array(bestpathIndex)
print("Best path is: ")
print(bestpathIndex)
print("Total path cost is: ")
print(sum(ant.pathCost))
```

Algorithm 1: Ant Colony Main Script

```
# ant class

import numpy as np
import random

class Ant:

    def __init__(self,antIndex,tau):
        self.antIndex = antIndex
        self.alpha = 1
        self.beta = 2
        self.numCities = 5
        self.distMatrix = np.array([
                                    [0, 10, 12, 11, 14],
                                    [10, 0, 13, 15, 8],
                                    [12, 13, 0, 9, 14],
                                    [11, 15, 9, 0, 16],
                                    [14, 8, 14, 16, 0]
                                    ])
        self.tau = tau # new

    def printStatus(self):
        # Desired outputs
        print(f"----- ANT {self.antIndex} -----")
        print(f"Ant {self.antIndex} started at: ",self.startCity)
        print(f"Ant {self.antIndex} finished at: ",self.lastCity)
        print(f"Ant {self.antIndex} path is (by index): ",self.path)
        print(f"Ant {self.antIndex} Path Cost is: ",self.pathCost)
        print(f"Total Ant {self.antIndex} path cost is: ",self.sumPathCost)
        print(f"Ant {self.antIndex} Pheromone matrix is: ")
        #print(self.pheromone)


    def antTravel(self):
        # initialize loop
        self.startCity = []
        self.lastCity = []
        self.path = []
        self.pathCost = []
        self.sumPathCost = []
        self.pheromone = np.zeros((5,5))

        # Build new eta everytime
        self.eta = np.divide(1,self.distMatrix)
        np.fill_diagonal(self.eta, np.zeros((5,5)))

        # Start travel
```

```
        #self.startCity = random.randint(0,4)
        self.startCity = 0 # try this
        self.path = [self.startCity]
        self.currentCity = self.startCity
        p = []
        self.eta[:,self.currentCity] = 0
        self.pathCost = []

        for i in range(4):
            cumP_all = self.findProbability()
            p = cumP_all[self.currentCity]
            #p = p[0]
            #print("p[2] is: ",p[2])
            # good parsed out

            # Step 3: Decide where ant (k) should go next
    ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
            # generate random number between 0,1
            nextOption = random.uniform(0,1)

            if nextOption < p[0]:
                nextCity = 0 # go to city 1
            elif nextOption < p[1]:
                nextCity = 1 # go to city 2
            elif nextOption < p[2]:
                nextCity = 2 # go to city 3
            elif nextOption < p[3]:
                nextCity = 3 # go to city 4
            elif nextOption < p[4]:
                nextCity = 4 # go to city 5

            # Set visibility of city just visited to 0
            alreadyVisited = nextCity

            # Get distance cost from distance matrix
            self.pathCost.append(int( self.distMatrix[self.currentCity,nextCity])
    )

            self.addSinglePherLoc(nextCity)

            # Track ant path history
            self.path.append(alreadyVisited)

            self.eta[:,alreadyVisited] = 0 # prefill the first one
            self.currentCity = alreadyVisited # update
            # end loop


        # # # Outside of loop # # # #
        # Now, return ant to its starting point.
        self.path.append(self.startCity)
        self.lastCity = self.path[-1]
        self.pathCost.append(int( self.distMatrix[self.currentCity,self.lastCity])
     )
```

```
        self.sumPathCost = np.sum(self.pathCost)

        # Update pheromone matrix with last entry
        self.addSinglePherLoc(self.lastCity)

        # Compute actual pheromone value and place inside pher. matrix
        self.pheromone = self.pheromone / self.sumPathCost

        # # end function
        return self.startCity, self.lastCity, self.path, self.pathCost, self.sumPathCost, self.eta, self.pheromone



    def addSinglePherLoc(self,nextCity):
        self.pheromone[self.currentCity,nextCity] = 1


    def buildPheromone(self):
        # ALL OF STEP 4 in this method.
        # call this after the path is complete
        # compute value
        # set tau to all zeros
        pass


    def findProbability(self):
        # Compute tau and eta
        #self.tau = np.ones((5,5))

        # city probabilities
        cp1 = []
        cp2 = []
        cp3 = []
        cp4 = []
        cp5 = []

        for i in range(self.numCities):
            cp1.append( float(  (self.tau[0,i])**self.alpha * (self.eta[0,i])**self.beta   ) )
            cp2.append( float(  (self.tau[1,i])**self.alpha * (self.eta[1,i])**self.beta   ) )
            cp3.append( float(  (self.tau[2,i])**self.alpha * (self.eta[2,i])**self.beta   ) )
            cp4.append( float(  (self.tau[3,i])**self.alpha * (self.eta[3,i])**self.beta   ) )
            cp5.append( float(  (self.tau[4,i])**self.alpha * (self.eta[4,i])**self.beta   ) )

            # compute scalar sum value
            sums1 = float(np.sum(cp1)) # scalar value
            sums2 = float(np.sum(cp2))
            sums3 = float(np.sum(cp3))
            sums4 = float(np.sum(cp4))
```

```
            sums5 = float(np.sum(cp5))

        sums = np.array([ [sums1],[sums2],[sums3],[sums4],[sums5]])

        #print("Probabilities by city [each row]")
        #print(sums)


        # Now, compute local decision and cumulative probabilities
        # Step 2: Calculate local decision and cumulative probabilities
    ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

        # local DECISION prob
        dp1 = []
        dp2 = []
        dp3 = []
        dp4 = []
        dp5 = []

        for x in range(self.numCities):
            dp1.append( 0 if sums1 == 0 else float( cp1[x] / sums1) )
            dp2.append( 0 if sums2 == 0 else float( cp2[x] / sums2) )
            dp3.append( 0 if sums3 == 0 else float( cp3[x] / sums3) )
            dp4.append( 0 if sums4 == 0 else float( cp4[x] / sums4) )
            dp5.append( 0 if sums5 == 0 else float( cp5[x] / sums5) )

        #print("Local Decision Probabilities")
        localDecProb = np.array([ [dp1], [dp2], [dp3], [dp4] ,[dp5] ])
        #print(localDecProb)

        #print("Sums of each prob:")
        #dpSums = np.array( [ [np.sum(dp1)], [np.sum(dp2)], [np.sum(dp3)], [np.sum
    (dp4)], [np.sum(dp5)]      ] )
        #print(dpSums)
        # # # should all be 1, good.

        # Cumulative Probabilities (could loop this)

        pc1 = dp1[0]              # P(go to city 1)
        pc2 = dp1[1] + pc1        # P(go to city 2)
        pc3 = dp1[2] + pc2        # P(go to city 3)
        pc4 = dp1[3] + pc3        # P(go to city 4)
        pc5 = dp1[4] + pc4        # P(go to city 5)

        cumP1 = [pc1,pc2,pc3,pc4,pc5]

        pc1 = dp2[0]              # P(go to city 1)
        pc2 = dp2[1] + pc1        # P(go to city 2)
        pc3 = dp2[2] + pc2        # P(go to city 3)
        pc4 = dp2[3] + pc3        # P(go to city 4)
        pc5 = dp2[4] + pc4        # P(go to city 5)

        cumP2 = [pc1,pc2,pc3,pc4,pc5]

```

```
        pc1 = dp3[0]              # P(go to city 1)
        pc2 = dp3[1] + pc1        # P(go to city 2)
        pc3 = dp3[2] + pc2        # P(go to city 3)
        pc4 = dp3[3] + pc3        # P(go to city 4)
        pc5 = dp3[4] + pc4        # P(go to city 5)

        cumP3 = [pc1,pc2,pc3,pc4,pc5]

        pc1 = dp4[0]              # P(go to city 1)
        pc2 = dp4[1] + pc1        # P(go to city 2)
        pc3 = dp4[2] + pc2        # P(go to city 3)
        pc4 = dp4[3] + pc3        # P(go to city 4)
        pc5 = dp4[4] + pc4        # P(go to city 5)

        cumP4 = [pc1,pc2,pc3,pc4,pc5]

        pc1 = dp5[0]              # P(go to city 1)
        pc2 = dp5[1] + pc1        # P(go to city 2)
        pc3 = dp5[2] + pc2        # P(go to city 3)
        pc4 = dp5[3] + pc3        # P(go to city 4)
        pc5 = dp5[4] + pc4        # P(go to city 5)

        cumP5 = [pc1,pc2,pc3,pc4,pc5]

        # combine
        cumP_all = [ cumP1, cumP2, cumP3, cumP4, cumP5 ]

        #print()
        #print("All cumulative Probabilities are: ")
        #print(cumP_all)

        return cumP_all
```

Algorithm 2: Ant Class

2 Appendix B: Particle Swarm Optimization Code

```
# find global minimum of Ackley's function using PSO or Bees

import PSO # type:ignore
import numpy as np
import matplotlib.pyplot as plt

# initialize
S = 20
particles = []
w = 0.8
phi_p = 0.1
phi_g = 0.1

# create all the particles
for s in range(S):
    x = np.random.uniform(0,5,2)
    v = np.random.uniform(-np.abs(5-0), np.abs(5-0),2)
    # call the particle constructor and append the list
    particles.append(PSO.Particle(x,v,w,phi_p,phi_g))

particle: PSO.Particle
g = [5,5] # set best known value position to anywhere

# Setup contour plot stuff (but don't plt.show() yet
PSO.setPlot()

print("Global best is: ")
print("----------------------")

numTrials = 50
for iteration in range(numTrials):

    # Plot best known position
    plt.plot(g[0],g[1],'r*')

    for particle in particles:
        particle.updateVelocity(g)
        particle.updatePosition()
        particle.updateValue()

        particleBest = PSO.f(particle.p)
        globalBest = PSO.f(g)

        if particleBest < globalBest:
            g = [particle.p[0], particle.p[1]]
            print(globalBest)
            #print(f'g0 is: {g[0]} and g[1] is: {g[1]} ')

        # plot this particle
        plt.plot(particle.x[0],particle.x[1],'b*')


```

```
plt.show()
```

Algorithm 3: Particle Swarm Main Script

```
# Particle Swarm Optimization Class

import numpy as np
import matplotlib.pyplot as plt
#import ackleyPlotTest # type: ignore
import math


class Particle:
    def __init__(self,x,v,w,phi_p,phi_g):
        # save vars as self.var
        self.x = x      # particle position
        self.p = self.x # best known position = p
        self.v = v      # particle veloc
        self.w = w
        self.phi_p = phi_p
        self.phi_g = phi_g

    def updatePosition(self):
        # implement x = x + v on self
        self.x = self.x + self.v
        return self.x

    def updateVelocity(self,g):
        self.r_p = np.random.uniform(0,1)
        self.r_g = np.random.uniform(0,1)
        # random r_p and r_g
        # then update self.v = ...
        self.v = self.w * self.v + self.phi_p * self.r_p * (self.p - self.x) + self.phi_g * self.r_g * (g - self.x)
        return self.v

    def updateValue(self):
        # calculate self.value based on f(x)
        # update best known position of this particle
        # that is, update p
        currValue = f(self.x)

        if  currValue < f(self.p):
            self.p = self.x


def f(x):
    # objective function
    # test function for now before going into Ackley's

    # Actual Ackley's function

    sum1 = 0
    sum2 = 0
    c = 2*math.pi
```

```
    d = len(x)
    b = 0.2
    a = 20

    for ind in range(d):
        xi = x[ind]
        sum1 = sum1 + xi**2
        sum2 = sum2 + np.cos(c*xi)

    term1 = -a * np.exp(-b*np.sqrt(sum1/d))
    term2 = -np.exp(sum2/d)

    value  = term1 + term2 + a + np.exp(1)
    return value

    #return (x[0] - 3.14)**2 + (x[1] - 2.72) **2 + np.sin(3*x[0] + 1.41) + np.sin(4*x[1] - 1.73)
    # good, works.


def setPlot():
    # adapted from [1]
    x, y = np.array(np.meshgrid(np.linspace(-5,5,100), np.linspace(-5,5,100)))
    z = f([x, y])
    fig, ax = plt.subplots(figsize=(8,6))
    fig.set_tight_layout(True)
    img = ax.imshow(z, extent=[-5, 5, -5, 5], origin='lower', cmap='viridis', alpha=0.5)
    fig.colorbar(img, ax=ax)
    contours = ax.contour(x, y, z, 10, colors='black', alpha=0.4)
    ax.clabel(contours, inline=True, fontsize=8, fmt="%.0f")
    ax.set_xlim([-5,5])
    ax.set_ylim([-5,5])

'''
References
[1]: https://machinelearningmastery.com/a-gentle-introduction-to-particle-swarm-optimization/
'''
```

Algorithm 4: PSO Class

```
# Plot 3D Ackley function

import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

def ackley(x, y, a = 20, b = 0.2, c = 2*np.pi):
    return -a * np.exp(-b * np.sqrt((x**2 + y**2) / 2)) - np.exp((np.cos(c * x) + np.cos(c * y)) / 2) + a + np.exp(1)

x = np.linspace(-5, 5, 100)
y = np.linspace(-5, 5, 100)
X, Y = np.meshgrid(x, y)
```

```
Z = ackley(X, Y)

# Plot it
fig = plt.figure(figsize=(10, 8))
ax = fig.add_subplot(111, projection='3d')
ax.plot_surface(X, Y, Z, cmap='viridis')
ax.set_xlabel('x')
ax.set_ylabel('y')
ax.set_zlabel('f(x, y)')
ax.set_title('3D Ackley Function Plot')
plt.show()

# ref: matplotlib documentation help. Nice!
```

Algorithm 5: 3D Ackley Function