CS 3510

Traveling Salesmen Project

Heather Mikan and Gage Carr

April 17th, 2020

The Ant Colony Problem

How the Algorithm Works

First, we must consider the list of destination coordinates as a graph G=(V,E) where each destination corresponds to a vertex in V, and G is complete (ie you can travel from any node to any other).

The ant colony algorithm works a lot like actual ants trying to find the shortest path somewhere. Ants can leave behind pheromone deposits as they travel. The way this algorithm works is that any path traveled by an ant gets an additional pheromone deposited (per edge) directly related to how SHORT the path is. Therefore, as more ants travel through the graph, the shorter paths get much higher pheromone levels added along their edges than the longer paths.

This trend is apparent in our algorithm by the eta matrix for each ant (create_eta method). The eta is equivalent to 1/the total path cost to that node.

Additionally, ants determine their path with probabilities based on the pheromone levels and cost of each edge. With this additional piece of information, we have two values associated with each edge, cost and pheromone levels. Therefore, the shorter paths become even more popular as ants are more likely to take them as the algorithm continues.

This trend is apparent in our algorithm in the get_next method. Here, the probability of visiting each unvisited node is calculated based on the following formula:

For all e not in visited nodes, P_e = $\text{cost}_e^{\alpha}\ \text{eta}_e^{\beta} / \sum_e cost_e^{\alpha}\ eta_e^{\beta}$

How Ant Colony Algorithm Translates to our Traveling Salesmen Project

Structure: **class** method

Graph{read_coordinates, create_cost_matrix, create_pheromone}

Colony{pheromone_update, travel, opt_path_finder}

Ant{create_eta, get_next, delta_update}

main

Method Descriptions:

Graph

read_coordinates -> This method takes in the input file and maps out the nodes into structures we can use in our algorithm (nodeMapping, size, nodes)

create_cost_matrix -> This method uses scipy.spatial.distance to fill in the cost matrix for the graph. This cost matrix outlines the distance between every node (since G is complete).

create_pheromone -> This method initializes the pheromone matrix with each edge having the initial pheromone level of 1.

Colony

pheromone_update -> This method updates the pheromone matrix of G for each ant in the antList and adjusts the matrix according to the pheromone evaporation coefficient (rho)., which controls how fast pheromone decreases. Each ant has a delta matrix that updates in the delta_update method.

travel -> This method creates ants based on how many you pass in in the main method. Then for each ant, this method calls create_eta and get_next for each node in the ants path (aka the whole graph). Then it updates the appropriate colony attributes (totalCost, visited_nodes).

opt_path_finder -> This method iterates through every generation and calls travel, tracks the ant with the shortest path, and updates the pheromone matrix. Basically, this method keeps the ants moving for as many generations as you specify.

Ant

create_eta -> This method creates an eta matrix for each edge equal to one over the cost of that edge. This eta is used in get_next to calculate probability.

get_next -> This method looks at each of the unvisited nodes and calculates the probability of traveling to each node. This calculation can be altered with parameters passed in in the main function, giving certain variables more weight than others. Then it uses a random number generator with a cumulative distribution method to pick the next node. Finally, it updates all the appropriate ant attributes to move it to the next vertex.

delta_update -> This method updates all visited edges' pheromone delta in path order for the ant. The delta matrix is updated in one of three ways depending on how the method is called in the main method. Then, the delta matrix is used to update the colony's pheromone matrix.

main -> This method is obviously the main method where we call our other supporting classes and methods. First, this method looks at the command line argument and uses the filename to create a Graph Object and set it up using read_coordinates, create_cost_matrix, and create_pheromone_matrix. Then, this method initializes a colony with the following parameters:

graph (graph object): structure for mapping out graph from input file

ants (int): number of ants you want

generations(int): number of times you want to run the ants through

alpha (float): determines how much cost effects the probabilities of edges

beta(float): determine how much eta effects the probabilities of edges

rho(float): the decay or how much pheromone is lost each cycle

q(int): denominator in determining the pheromone delta update

strategy(int): either 1, 2, or other determine how to specifically use q.

These parameters are important for controlling the behavior of the ant colony and how quickly the algorithm runs. The more ants and generations, the more accurate the path information but the longer the algorithm takes.

With colony.opt_path_finder we can get the optimal path from our colony object given these parameters.