Kegan Kaiser

CM 3289

Contents

[Basic Structure 2](#_Toc374982225)

[Data Structures Used 2](#_Toc374982226)

[Heuristics 2](#_Toc374982227)

[Closest City 2](#_Toc374982228)

[Random Transition Rule 3](#_Toc374982229)

[Weighted Probability 3](#_Toc374982230)

[Analysis 3](#_Toc374982231)

[Closest City Heuristic 3](#_Toc374982232)

[Random Transition Rule 3](#_Toc374982233)

[Weighted Probability Heuristic – With and Without Distance 4](#_Toc374982234)

[Common Features 4](#_Toc374982235)

[Test Data 5](#_Toc374982236)

# Basic Structure

The program is started by main(), where you can switch between generating fully and sparsely connected graphs. Using the graph, a TSP\_Instance is built. TSP\_Instance takes the generated graph and constructs a number of City objects. City objects are the equivalent of Nodes, I did not notice that we were asked to implement “Node” until after I had set up my solution for fully connected graphs so I replaced “Node” with “City” in the supplied graph generating code. (Sorry for any confusion this causes.) When TSP finishes it reports to main that a TSP instance has been successfully set up, at which point the solve() method is invoked. Solve generates a given number of ant objects and starts them all in random cities. These ants then move through the cities in an attempt to find a complete tour. When determining which city to travel to next, ants call a method called findBestPath(). This method contains three heuristics for solving the problem that I will describe later. When an ant can finds itself at a city with no legal moves remaining it “returns to the anthill”. An anthill object, initialized during setup in TSP\_Instance, receives ants that have completed their tour and judges the path they took. Only the best ant remains in the anthill to report its path at once all the ants have returned.

# Data Structures Used

I used arraylists for most storage. Notably, cities contain a DiPhTable (Distance-Pheromone table) of the following form:

|  |  |  |
| --- | --- | --- |
| City Name | Distance | Pheromone Level |

This is accomplished with a list containing lists of integers. This allows DiPhTable.size() calls to yield the number of neighbors, as well as DiPhTable.get(i) to return all information on a desired connection, with the simple addition of .get(0), .get(1), or .get(2) to determine the name, distance to, or pheromone between the current city and neighboring city.

Many functions use lists of integers containing names of cities instead of lists of city objects. This saves some space and does not harm lookup time for that city in the DiPhTable, since the name is all that is needed. In general, when a class only needs the name of cities a list of integers is used instead of a list of cities.

# Heuristics

## Closest City

This is the first heuristic I chose to implement. I did so first because it is straightforward and was an easy way to test other parts of my code. This heuristic simply picks the city that is closest to the current city each time. While this works out well for fully connected graphs that are not too large, it is not a great heuristic overall. This heuristic works best when each city has at least one ant starting in it.

## Random Transition Rule

This is the AS algorithm described in the book, given by the formula: Pr = [t^A \* n^B] / E[t^A \* n^B]  
As expected, this heuristic worked well overall.

## Weighted Probability

This is an algorithm I wanted to use because it seemed straight-forward to me. First, create a list of each legal path an ant can take. Add extra copies of the list for each city, based on the pheromone level. Additionally, add extra copies of the nearest cities. Then, one city is picked at random from the list. This algorithm was easy to implement and worked well overall. It has the benefits of tuning parameters (n nearest cities to add extra copies of, number of extra copies given for distance, and number of extra copies given for pheromone) and has an inherent noise level, allowing for the randomization that showed to improve performance in our previous studies.

# Analysis

## Closest City Heuristic

The closest city heuristic performed poorly overall. It is only successful on fully connected graphs and even then will only find the shortest path if there is at least one ant per city and the optimal path does not require taking a longer short term path in order to shorten the long term path length. This means that the closest city heuristic was only viable on fully connected graphs with few nodes.

## Random Transition Rule

This heuristic performed well overall. It was highly dependent on the tuning parameters, however. Alpha controls the priority given to pheromone trails, while Beta increases the priority of visibility. In addition, the rate of dissipation of pheromone trails also affects the priority of pheromone trails. The pheromone levels were held constant throughout most experiments; Alpha and Beta were focused on instead.

When alpha was too high pheromone trails were followed too heavily. This would result in ants converging on a decent, though not necessarily optimal, path without exploring other options. Conversely, when beta was too high ants were far too willing to diverge from paths that were shorter in the long term in favor for paths that were shorter in the shorter term. Values between 1.4 and 2 worked well for alpha when beta was at 2.

We also found that decreasing the opposite parameter was useful for controlling priority. This is logical, but it is worth noting that the affects were not of the same magnitude. The ratio between alpha and beta is what really controls the outcome of the probability. For example, with alpha = 1 and beta = 5, we have a ratio of .2, or 20%. We could increase alpha by 1, giving us a ratio of 40%. If we had instead decreased beta by 1, our ratio would become 25%. This implies that increasing alpha has a greater effect on the priority of pheromone than decreasing beta by the same amount.

## Weighted Probability Heuristic – With and Without Distance

This heuristic also performed well. It is not terribly different from the Random Transition Rule implemented by the book, as both implement an AS approach. I implemented this approach first as a test of the system overall, as it is much simple (at least in my opinion) than the Random Transition Rule. It does follow the same guidelines, however. The path chosen is determined randomly, with increased odds given to edges with higher pheromone levels or lower distances. The key difference is that the Random Transition Rule determines probability by a formula that must be calculated each move, while the Weighted Probability Heuristic follows a “big bag” approach. We can look at it as placing each option into a large bag and pulling one out at random. This gives inherent randomization and allows for the same tuning parameters. We can increase the priority of pheromone by adding extra copies of the city per pheromone level to the bag to increase the odds of that city being chosen (note that, to improve performance, the integer names identifying each city were added to the bag repeatedly, not the entire city object). Similarly, we can add extra copies of the edges with the lowest distance to increase the priority of visibility. This gave this heuristic a similar level of flexibility as the Random Transition Rule, but the performance was slower. Nonetheless both found optimal paths to smaller instances consistently, while losing precision in larger instances. As instances grew larger (the number of cities increased) tuning parameters would often need to be changed to make them more effective in the new instance.

We can see that not incentivizing visibility in the weighted probability heuristic drastically reduced performance. Consider the 100 city fully connected test, where the other algorithms were capable of finding solutions with a total distance less than 3000, while this approach rarely produced results below 4000. This is not at all surprising.

## Common Features

Each approach exhibited certain similarities. For fully connected graphs, distance seemed to be more important in finding the optimal solution. This seems to be due to the fact that there is no way to get stuck, so distance is really all there is to consider. In sparsely connected graphs dead ends exist, so having pheromone to guide ants away from dead ends is helpful. Changing the levels of pheromone added dependent on whether the ant succeeded in making a move or reached a dead end can be very helpful.

Pheromone levels were useful in many more ways than I had anticipated. More than simply guiding ants along shortest paths and avoiding congestion, they were also very helpful in avoiding dead ends, as mentioned above. One interesting possibility that is not mentioned in the book is the possibility of adding pheromone one way instead of two. That is, when an ant travels from city A to city B, add pheromone only to the edge stored in the node representing city A. Laying a pheromone both ways tells ants at either city to travel to the other one, but this may not be desirable. If many more ants are moving away from a node than towards it, it is likely more beneficial to travel elsewhere and approach the node from another direction. We do not want to enforce this behavior to the point that ants refuse to explore new paths of course, but this can be controlled just as before. If nothing else, this adds a new layer of control to the AS approach.

Each heuristic showed that the number of ants only improved performance to a certain extent. While the ideal number of ants varies with the type of heuristic used as well as the size of the problem, it seems a logarithmic relationship exists between the number of ants and performance gains.

While imperfect, this graph shows the idea of a logarithmic relationship between the number of ants and the probability of finding the optimal path. Data is used from the three hundred city fully connected test runs using the weighted probability heuristic.