LAGRANGE COLLEGE
Citations Journal of Undergraduate Research
2021, Volume 18

Traveling Salesman in LaGrange, Georgia

Holston Sebaugh

Faculty Mentor: Stacey L. Ernstberger, PhD
Mathematics

Abstract

In the field of graph theory in mathematics, the Traveling Salesman Problem determines an optimized closed loop through multiple locations. We apply this problem to various destinations throughout LaGrange, Georgia to solve for minimum time and distance paths. This problem has many applications where one needs to determine optimal routes for visiting every location in a predetermined list. Here, present and demonstrate the Traveling Salesman Problem and then we apply this problem to the scenario of visiting the popular locations in LaGrange, Georgia while minimizing distance and time.

When given multiple locations, what is the the most efficient way to visit each of them? This a common scenario, known as a Traveling Salesman Problem (TSP) and can be applied in many situations. For example, when considering LaGrange College, many prospective students may like to see what the city has to offer before making a final decision. We consider the scenic areas in LaGrange, Georgia and construct a Traveling Salesman Problem around visiting those locations with a prospective student. Before we apply this problem to our given scenario, let's first demonstrate a basic example, and then discuss how we have chosen to implement this process using Matlab.

Preliminary Example

We created an example using four nodes with various weights representing the cost to get between nodes. In Figure (1), the arrows represent the weight of the paths to travel between nodes. We transform this diagram into a matrix whose elements represent the distance between the nodes. Here, the (i, j) element represents the weight of the path from node i to node j.

$$A = \begin{bmatrix} 0 & 2 & 1 & 3 \\ 5 & 0 & 2 & 1 \\ 2 & 2 & 0 & 4 \\ 1 & 1 & 3 & 0 \end{bmatrix}$$

Notice that along the main diagonal of A, all the values are zero. Since each element represents the distance from one index to another, it makes sense that the distance from each node to itself would be zero. Note that not all weights are the same between nodes.

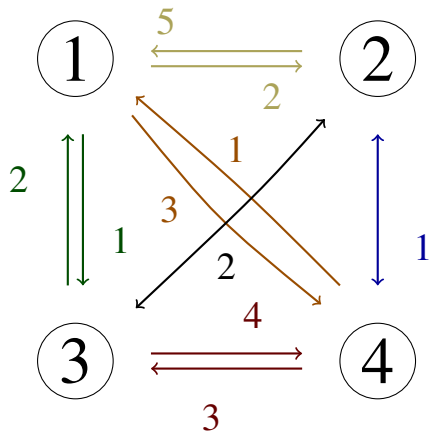

Figure 1: Weighted Diagram of Travel

To find the optimal path with weight P^*, through each node, we begin by looking at all possible paths of our matrix. We use the notation $P(1,\{2,3,4\})$ to represent the weight of the path starting at node one and considering all of the path options that include nodes two, three, and four. We then proceed to break down the paths into their subsequent steps. Here,

$$\begin{aligned} P^*(1,\{2,3,4\}) &= \min(P(1,\{2,3,4\})) \\ &= \min(P((1,2)+P(2,\{3,4\})), \\ &\quad P((1,3)+P(3,\{2,4\})), \\ &\quad P((1,4)+P(4,\{2,3\}))) \end{aligned}$$

where the notation $(1,2)$ represents the weight of the path from

node 1 to node 2. Then,

$$\begin{aligned} P((1,2)+P(2,\{3,4\})) &= 2+\min(P(2,\{3,4\})) \quad (1)\\ P((1,3)+P(3,\{2,4\})) &= 1+\min(P(3,\{2,4\})) \quad (2)\\ P((1,4)+P(4,\{2,3\})) &= 3+\min(P(4,\{2,3\})). \quad (3)\end{aligned}$$

The second terms in (1), (2), and (3) are the minimum weights of the paths that from the second to the third node. Without loss of generality, we will only consider (1) and take the next step. However, the process is the same for (2) and (3).

$$\begin{aligned} P((2,3)+P(3,\{4\})) &= 2+\min(P(3,\{4\})) \quad (4)\\ P((2,4)+P(4,\{3\})) &= 1+\min(P(4,\{3\})) \quad (5)\end{aligned}$$

Here we seperately consider each remaining node as the nest step. We continue this process until the final step. We will now include the distance it takes from the fourth and final node back to node one, and display it as the value in parenthesis. For (4) and (5) this becomes

$$\begin{aligned} P(3,\{4\}) = P((3,4)+(4,1)) &= 4+(1)=5\\ P(4,\{3\}) = P((4,3)+(3,1)) &= 3+(2)=5.\end{aligned}$$

Since we are solving for the minimum weight, we must first solve for the minima of (4) and (5) as follows:

$$\begin{aligned} P((2,3)+P(3,\{4\})) &= 2+5=7\\ P((2,4)+P(4,\{3\})) &= 1+5=\mathbf{6}.\end{aligned}$$

We repeat this process for (2) and (3) to obtain:

$$\begin{aligned} P((3,2)+P(2,\{4\})) &= 2+2=\mathbf{4}\\ P((3,4)+P(4,\{2\})) &= 3+6=9\end{aligned}$$

and

$$\begin{aligned} P((4,2)+P(2,\{3\})) &= 1+4=\mathbf{5}\\ P((4,3)+P(3,\{2\})) &= 3+7=10.\end{aligned}$$

The bold face values represent the minimum weight path of the two options from each second node. We evaluate (1), (2), and (3) to find

$$\begin{aligned} P((1,2)+P(2,\{3,4\})) &= 2+6=8\\ P((1,3)+P(3,\{2,4\})) &= 1+4=\mathbf{5}\\ P((1,4)+P(4,\{2,3\})) &= 3+5=8.\end{aligned}$$

Now we can assemble our path of minimum weight as follows:

$$\begin{aligned} P^*(1,\{2,3,4\}) &= P((1,3)+P(3,\{2,4\}))\\ &= P((1,3)+(3,2)+P(2,\{4\}))\\ &= P((1,3)+(3,2)+(2,4)+(4,1))\end{aligned}$$

and hence

$$P^*(1,\{2,3,4\})=5.$$

Thus the optimal path through the numbered nodes in Figure 1 is given by

$$1\to 3\to 2\to 4\to 1. \quad (6)$$

If we were to begin the path at a different node, our optimal path would appear in the same order as in (6). For example, if we again considered the system depicted in Figure 1 and chose node 2 as our initial location, our optimal path would then become

$$2\to 4\to 1\to 3\to 2.$$

Now that we have a basic understanding of how to solve the traveling salesman problem, we can address other methods and practices used in addressing this problem.

Methods of Implementing Traveling Salesman

While our described process was just one way of solving the Traveling Salesman Problem, this method will become an issue when the number of nodes increases beyond just a few. The resulting number of permutations that must be considered when looking at every optimal path requires a large amount of runtime. As this is a common problem in graph theory, many methods exist to solve this problem in a different way:

- the Miller, Tucker, and Zemlin method (MTZ)
- the Dantzig, Fulkerson, and Johnson method (DFJ)
- Ant Colony System (ACS)
- Genetic Algorithm (GA).

The last two methods, ACS and GA, originate from biology and embrace machine learning through "families" of iteration. The DFJ is a robust method that handles calculating very quickly, and the MTZ ultimately acts as a good starting guide for the TSP [1]. However, these methods tend to involve subroutines where the overall optimal path may not be connected.

Implementing the Method using MATLAB

In our method, which is the direct method involving permutations, we began by loading a square matrix of dimension $n\times n$ consisting of weights to travel between nodes. For each path through all the nodes, there is a cost to travel, and this cost is determined by adding the appropriate weights in the matrix.

For a matrix of size $n\times n$, we have n nodes and we first list the permutations of those nodes, excluding our starting and

finishing node. We then assemble a full array of all possible paths $(n-1)$ into an $(n-1)! \times (n-1)$ matrix and reinsert our starting and finishing node, thus creating a $(n-1)! \times (n+1)$ matrix. Then, to calculate the total cost of each path, we implement the following chunk of code:

```
for j = 1:(n-1)!
    for i = 1:(n-1)
        cost(i,j) = A(path(i,j),path(i+1,j));
    end
    cost(n+1,j) = A(path(n,j),path(n+1,j));
    total_cost(j) = sum(cost(:,j));
end
```

In this code, the `A` matrix is our initial weight matrix, `cost` contains the distance weight between nodes within each path, and `path` is a matrix of our possible paths. Once we run this within MATLAB, we can then calculate a simple `min` operation of our variable `total_cost` which contains the total weight of each path and find our shortest distance along with its corresponding path. Executing it this way has one drawback of taking $O(n!)$ operations which can slow down runtime for larger values of n.

To speed up this process and reduce the number of operations, we implement a path comparison conditional. To begin a comparison conditional, we create a variable called `best_dist` and prepare a placeholder for our value of shortest path. We let this initial placeholder be the total sum of all the elements within our cost matrix, which quickly gets replaced as soon as we start iterating through our possible paths. Once we have calculated the total cost of the first path, we compare it to `best_dist`. If it is less, we replace the value of `best_dist` with the new value and save the index of the path being used. From here, we repeat the process. As we compute `total_cost(j)`, we compare it to our `best_path`:

- If `total_cost(j)` is larger than `best_path`, it is ignored.
- If `total_cost(j)` is equal to `best_path` then the indices `j` is stored.
- If `total_cost(j)` is less than `best_dist`, the value of `best_dist` would be replaced and the previous index stored in `best_path` would be removed and replaced by the current index.

We then repeat this process through all iterations to gain our final minimum distance and path(s).

Application

We now apply this method to our initial LaGrange College prospective students scenario. Recall that our goal is to show a prospective LaGrange College student the highlights of the town, beginning and ending at LaGrange College. The following locations were chosen based on their historical and cultural significance to the city, with consideration given to the intended age group:

- LaGrange College
- Callaway Stadium
- LaGrange Mall
- Great Wolf Lodge
- Pyne Road Park
- Granger Park
- Charlie Joseph's
- Lafayette Fountain
- Callaway Clocktower
- Wild Leap Brewing
- LaGrange Art Museum
- Sweetland Amphitheater

We collect data consisting of the distance and time between each of these locations prior to implementing the code for the Traveling Salesman Problem in MATLAB.

DISTANCE(MI)	LaGrange College	Cal. Stadium	Mall	Great Wolf	Pyne Road Park	Granger Park	Charlie Joseph's	Fountain	Clocktower	Wild Leap	Art Museum	Sweetland
LaGrange College	0	1.1	3.8	4.1	6.9	1.2	0.9	0.7	1.8	1.1	0.9	1.3
Cal. Stadium	1.1	0	4.7	3.4	7.6	2.2	1.6	1.7	1	1.4	1.7	2.3
Mall	3.8	4.7	0	9	10.6	4	3.2	3.3	4.7	3.5	3	3.6
Great Wolf	4.1	3.4	4.8	0	10.2	4.3	3.3	3.4	2.7	3	3.3	3.9
Pyne Road Park	6.9	7.6	10.6	10.7	0	8	7.5	7.5	8.5	7.8	7.6	8.1
Granger Park	1.2	2.2	4	4.3	8	0	1	0.9	2	1.2	1	0.6
Charlie Joseph's	0.8	1.6	3.2	3.5	7.5	1	0	0.07	1.5	0.4	0.2	0.7
Fountain	0.7	1.6	3.3	3.4	7.5	0.9	0.2	0	1.4	0.3	0.1	0.6
Clocktower	1.8	1	4.7	2.7	8.7	2	1.4	1.4	0	1.3	1.6	2.3
Wild Leap	1	1.4	3.5	3	7.8	1.2	0.2	0.3	1.3	0	0.4	1.3
Art Museum	0.9	1.7	3	3.3	7.6	1	0.2	0.1	1.6	0.3	0	0.7
Sweetland	1.4	2.3	4	4.3	8.2	0.4	1.1	1	2.2	1.2	0.7	0

Figure 2: Cost Matrix for Distance in Miles

Optimal Distance Path

We obtain the minimal distance between each location using Google Maps [2] as seen in Figure (2). The starting location is listed along the left hand side of the table and the ending location is listed along the top of the table.

We allow LaGrange College to represent the beginning and end nodes of our path, and we then derive the optimal path through all locations excluding Great Wolf Lodge and Pyne Road Park. We choose to omit these to reduce the wait due to the processing time needed for each additional location. For example, as we can see in the Table 1, the processing time can grow exponentially with the addition of more locations, hence we perform our computations using a reduced number of locations for our scenario.

# of Locations	Time Taken to Calculate
6	0.395s
7	0.3725s
8	0.476s
9	3.45s
10	563.84s

Table 1: Time taken to Process Locations

By implementing our code using the cost matrix in Figure 3 with the aforementioned modifications, we obtain an optimized route which begins and ends at LaGrange College. The route is given as follows:

LaGrange College → Callaway Stadium
→ Clocktower → Wild Leap → Charlie Joseph's
→ Fountain → Art Museum → Mall → Sweetland
→ Granger Park → LaGrange College.

The distance for the optimal path is 11.97 miles. In Figure (4) we see a map overlay of our optimal distance path in LaGrange for our given locations.

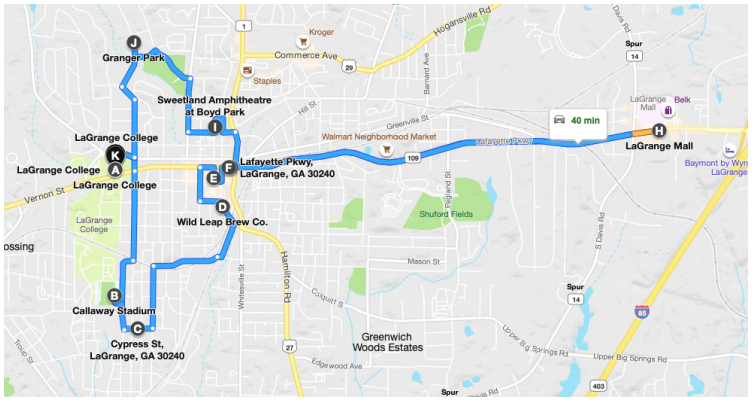

Figure 4: Distance Path Map Overlay [2]

Although the path of optimal distance seems like it would be a good choice for a prospective student, when driving we typically choose the path of minimum time without regard to the distance. In order to determine whether these routes are the same, we need to obtain timed data in addition to our distance data.

Optimal Time Path

We follow the same process to find our optimal (minimum) timed route, beginning by collecting the timed data in minutes using Google Maps [2]. We see the results of the data collection in Figure (3). We should note that traffic was not taken into consideration with our timed paths, i.e., if we had obtained the data at 5:00 PM we likely would have had longer time estimates corresponding to higher levels of traffic. The data was

TIMES(MIN)	LaGrange College	Cal. Stadium	Mall	Great Wolf	Pyne Road Park	Granger Park	Charlie Joseph's	Fountain	Clocktower	Wild Leap	Art Museum	Sweetland
Lagrange College	0	4	8	9	10	3	3	2	5	4	3	4
Callaway Stadium	3	0	10	8	12	5	5	4	4	4	5	7
Mall	9	10	0	9	17	8	7	5	10	7	6	8
Great Wolf	9	8	9	0	15	9	6	6	4	5	5	8
Pyne Road Park	10	12	17	15	0	14	13	11	14	13	13	15
Granger Park	3	6	8	9	14	0	5	3	6	5	4	3
Charlie Joseph's	3	5	7	8	13	4	0	1	5	3	2	4
Fountain	2	4	7	7	12	4	2	0	4	2	1	4
Callaway Clocktower	5	3	10	6	14	5	4	4	0	3	4	7
Wild Leap	4	4	7	6	13	5	1	2	3	0	2	4
Lagrange Art Museum	3	5	6	6	13	5	3	1	4	2	0	3
Sweetland	4	7	9	10	15	2	6	4	7	6	3	0

Figure 3: Cost Matrix for Time in Minutes

collected at a time when we anticipated very few cars on the road.

After running the timed data through our Traveling Salesman code, we obtained two optimal routes. One route is exactly the same path as our optimal distance path. However, we have an alternate route of optimal time as follows:

LaGrange College → Callaway Stadium
→ Clocktower → Wild Leap → Charlie Joseph's
→ Fountain → Sweetland → Art Museum → Mall
→ Granger Park → LaGrange College

Both routes require the minimum time of thirty-three minutes to traverse through each location, beginning and ending at LaGrange College. It should be noted that when we implement a different number of nodes or a different assortment of locations, we occasionally obtain multiple paths as has occurred in this case. This is a common result in Traveling Salesman Problems. Depending on the application, often only one of the optimal solutions is provided.

In Figure (5), we see a map overlay of our optimal distance path in LaGrange for our given locations. In our application, the optimal time path would be the path that we would recommend for a prospective LaGrange College student to take in order to have a better idea of some of the highlights of the town.

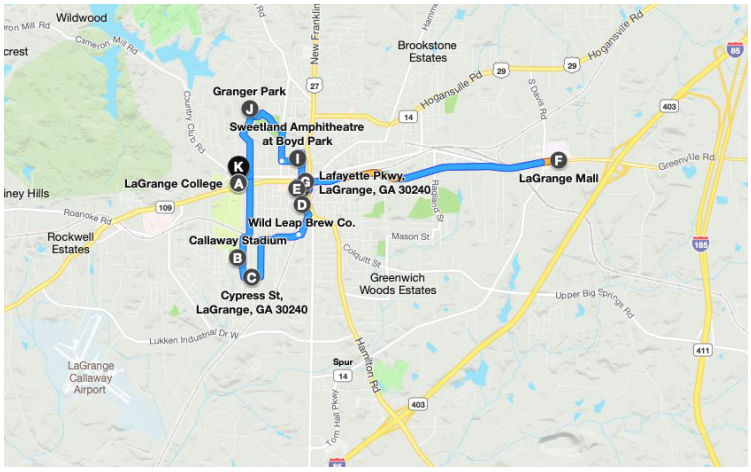

Figure 5: Time Path Map Overlay [2]

Conclusions

The Traveling Salesman Problem has many real world applications and many methods for implementation to best handle each of those applications. It seems straightforward to implement in theory, but in practice is rigorous in formulation, and there are various drawbacks to each method. The direct method that we implemented is thorough but lacks in speed, and is rendered unusable with large numbers of nodes. With more development, our goal would be to reach more locations and have a faster processing time by implementing one of the methods that does not require a thorough search through the permutations of each node. We would like to reduce computational time while increasing the amount of nodes handled per path, and this would likely involve the incorporation of machine learning to achieve this goal.

References

[1] Baobab. "Three Different Methods to Solve the Traveling Salesman Problem." 02 Oct. 2020.

[2] Google Maps. LaGrange, Georgia. https://goo.gl/maps. Web. 9 April 2021.

[3] Mishra, Neeraj. "Travelling Salesman Problem in C and C++." The Crazy Programmer. 25 May 2017.

[4] Singh, Nishant. "Traveling Salesman Problem (TSP) Implementation." GeeksforGeeks. 04 Nov. 2020.