# mTSP in Accreditation

Saul Alarcon

September 1, 2019

**Abstract**

This paper will use the Multiple Traveling Salesman Problem (mTSP) to facilitate a common problem in the space of accreditation. Namely, how to schedule inspectors across a wide variety of locations so that the distance traveled is kept to a minimum. This analysis will relax a lot of constraints that are present in the accreditation space, but are currently deemed out of the scope of this paper. These we will be left as future considerations. We will test models for these using MS ASPE and show our results.

**Keywords :** Multiple Traveling Salesmen Problem, Vehicle Routing Problem, Accreditation, Integer Linear Programming, Algorithims

## 1 Introduction

The accreditation space currently faces a dilemma that arises whenever a large amount of facilities need to become accredited within a short time frame. Whenever this happens, inspectors are in great demand, and the supply cannot always meet the demand. To alleviate this, there are several possibilities that can occur. This may include offering incentives to inspectors to perform these inspections, hiring more inspectors, and creating efficient travel routes for these inspectors. This paper will focus on the last option, and use literature surrounding the mTSP problem to schedule inspectors.

mTSP stands for the Multiple Traveling Salesman Problem, and aims to find the shortest tour for $m$ salesmen around a network of locations. In our scenario, inspectors take on the role of of salesmen, and facilities to be inspected take on the role of cities. They will however be modeled in a very similar fashion. In addition to being a tour that minimizes the length, there is another consideration that needs to be addressed.

In this particular scenario, the client is coming to us with existing facilities, however all of their facilities are merging with another company. Therefore, to maintain their accreditation, they will need to be inspected again. Current law dictates that these facilities are not able to bill their clients until they receive accreditation. Therefore the sooner they receive their accreditation, the sooner they are able to begin generating revenue. The client wants to ensure that the facilities that have historically generated the most revenue are prioritized in the accreditation process. So in addition to lessening the distance of the routes taken, we also want to make sure that we are able to prioritize the visits to the facilities that generate more revenue. With these two factors in mind, we can begin to use the mTSP to tackle on this problem.

In addition to being valuable in the accreditation space, the mTSP has been useful in a wide variety of fields, these fields include robotics, electric circuit boards, and ant colonization.

## 2 Literature Review

The majority of the methodology that has been used directly in this work was taken directly from Chapter 10 of "Theory and Practice of Uncertain Programming" (Liu, B, 2009). This book chapter outlines many of the core mathematical foundations that we will use for this problem and slightly modify to attend to our specific needs. There is no other work that we will use to expand upon the mTSP in our specific scenario, however we will keep several options in mind for expanding on this problem in the future.

One of those options include keeping carbon emissions and fuel production to a minimum. Some interesting work at universities in the China area have done just this (Zhang, et al., 2015). The naive approach to this problem would be to simply assign "carbon weights" to each of the distances. However as many authors have pointed out, it cannot be done so simply. One has to taken into consideration the speed (and therefore the fuel expended) at each point in time for the distance traveled to get a more accurate account of the fuel expended. Rather than asssuming that (Carbon Emission) $\alpha$ (Distance Traveled), we will say that there is a non-linear function that relates the two.

Another one of these options that this model can benefit from is the use of occasional drivers (or in our case inspectors). These would be inspectors that are perhaps not willing to embark on an entire tour. Research has been done to utilize these inspectors into an mTSP regardless (Archetti et al., 2016). Perhaps they have another job which they need to attend, or do not desire tours for any other reason. They are only willing to visit one facility/location. This type of inspector does exist within our organization, and incorporating them into our model would yield great benefits. In our current state however, we will assume that every inspector is willing to embark on a tour.

One last option that can benefit the use of mTSP in our current scenario is the use of alternative models to reach our conclusion. In this paper we will only be dealing with a small number of facilities, and only a handful of inspectors. It is safe to say that under most models, we can be confident that they will find an optimal route in a reasonable amount of time. However, as the growth of accreditation continue, it may become increasingly important to use efficient algorithms. In a future scenario that has thousands of facilities to accredit, and only hundreds of inspectors to do so, finding an algorithm with short computation time will be more important. There has been work done to assist in the computation time for larger graphs. One paper does an excellent job at summarizing these different algorithms as well as providing their own (Nuriyeva, Kizilates, 2017).

As mentioned, although these are exciting routes to take, this paper will focus on the simpler scenario that doesn't include these extra benefits due to the time involved in incorporating these solutions simultaneously.

## 3 Methodology

The methodology that is used here is, as has been mentioned, largely inspired from Chapter 10 of "Theory and Practice of Uncertain Programming" (Liu, B, 2009). We will lift many of the common notions from that text and apply them to our scenario. To begin, let us define our graph. This graph will be our set of facilities to visit. Each vertex will represent a facility that needs to be visited. Each edge will represent the distance between those facilities.

Let $V$ be the set of vertices.

Let $A$ be the set of edges.

Let $G = (V, A)$ define the graph of facilities to visit.

Now we will define the distance between each facility. Remember that because the distance from $A$ to $B$ is the same as the distance from $B$, to $A$, our matrix will be symmetrical.

Let $D = (d_{ij})$ be the distance matrix on the set of edges.

Because $d_{ij} = d_{ji}$, D is symmetrical.

We define the decision variable matrix. For each element in this matrix, we will have $x = 1$ if the route is taken in our model and we will have $x = 0$ if that route is not taken.

Let $X = (x_{ij})$ be the decision variable matrix.

Where $x_{ij} \in \{0, 1\} \; \forall i, j$

Finally we define our revenue matrix, which will help us prioritize the locations that generate more revenue (and therefore offer more value to the client).

Let $R = (r_{ij})$ be the revenue matrix.

Suppose there are $n$ vertices and $m > 1$ inspectors available.

Now we can define our objective function for this integer linear programming model.

$$Min\{\sum_{j \in A} \sum_{i \in A: i \neq j} \frac{d_{ij} x_{ij}}{r_{ij}}\}$$

So we will simultaneously minimize the distance that needs to be traveled while placing priority on those that generate more revenue. Note that the terms of the above objective function can also be thought of as $w_{ij} * x_{ij}$, where $w_{ij} = \frac{d_{ij}}{r_{ij}}$. In this case it is a simple weight attributed to each decisin variable.

Now we will identify our constraints. Remember that in contrast to the ordinary TSP, the mTSP allows for multiple inspectors, so we will have $m$ inspectors beginning and ending at the same location. So we have the following constraint to ensure that they all begin in the same location:

$$\sum_{j=2}^{n} x_{1j} = m$$

Notice that this is only true for one location (for one decision variable). Similarly, they must end at the same location as well.

$$\sum_{i=2}^{n} x_{j1} = m$$

For the rest of the locations, we need to ensure that there is only one point of entry and one point of exit. We can do so with the following constraints.

Only one point of entry.

$$\sum_{j=1}^{n} x_{ij} = 1, i = 2, ..., n$$

Only one point of exit.

$$\sum_{i=1}^{n} x_{ij} = 1, j = 2, ..., n$$

Notice that this is not true for $i = 1$ or $j = 1$ because the main location is allowed to have $m$ points of entry and exit as described above.

Finally we will need to eliminate subtours. These are any tours that do not begin or end at the initial location. This can be done by defining the following constraint.

$$\sum_{j \in S} \sum_{i \in S} x_{ij} \leq |S| - 1, \forall S \subseteq V - 1, S \neq \emptyset$$

With all of these tools equipped, we are now ready to dive into the computation of our experiment.

## 4 Computational Experiment and Results

We will begin by outlining all of the different facilities to be visited. These facilities are outlined in map 1. It can be seen that all of these facilities are bordering the eastern side of the United States. It is also worth noting that the beginning and ending point for each inspector will be at a facility in Washington D.C. The programming that was needed to generate this map can be found in figure 2. From here, we recreated all of the matrices and constraints needed for this problem inside of an MS Excel Spreadsheet so that we could use ASPE. The details of this spreadsheeet can be found in the screenshot of figure 3. As a result of this model, the following three paths were produced:

Tour 1 : 1-11-8-1

Tour 2 : 1-9-2-1

Tour 3 : 1-5-4-17-3-10-6-7-13-14-12-15-16-1

The total CPU run time needed to achieve these results was .12 seconds.

Visual paths can be found in figures 4, 5, and 6.

## 5 Discussions and Conclusions

Over the course of this project, there were some difficulties arose. One of these was the creation of all of the subtour constraints. Creating all of these constraints become very burdensome. If implemented with a larger number of facilities, there would need to be a more efficient way of handling these sub tour constraints. The alternative would likely be to automate these subtour constraints. By using a more general program such as Python, we may be able to write code that will capture these subtours without explicitly stating what they were.

Addditionally, we had no benchmark for optimization time. Our program ran in .12 CPU seconds. Without a point of comparison, we cannot really say if that is desirable or undesirable. In the future, we would like to be able to have more control over the optimization time by considering alternative models, as discussed in the introduction.

There is also the limited control we had of the tool that was used to find the solution. ASPE was used due to its convenience. In future developments, it is worth either finding an alternative model, or creating one from scratch. There is a lot of pseudo-code that exists online that can help us do exactly this.

There are so many aspects in which this model can be improved. As previously mentioned, carbon emissions from the travel can be minimized, models can be switched or re-structured, and occasional drivers (inspectors) can be taken advantage of. while those will greatly improve the model, there is even more that can be done and which further research needs to be done, if only to see if any literature exists.

For one, this one include having the option of multiple depots. In our accreditation space, the inspectors do not truly come from a single location, but rather several different locations. This is because the inspectors do not reside within the main location but rather in their own personal spaces, and they are summoned only when there is an inspection that needs to take place.

There is also the challenge of having equal tours. Ideally, we would want the inspectors to have an equal share of work. As it has been shown, this is currently not the case, with one inspector doing most of the heavy lifting. Having the workload (number of facilities) in each tour within approximately 10% of each other would be the best scenario.

Availability throws another wrench into our algorithm. We have assumed that each inspector has complete availability. In reality, many may work a second job or have private matters to attend to. A more effective algorithm would incorporate the schedules of each inspector.

Finally, we need to also consider the credentials of each inspector. There are a wide variety of facilities that need to be accredited. The credentials needed to provide an inspection to facility A may not be the same as the credentials needed to provide an inspection to facility B. Some credentials allow inspectors to visit multiple types of facilities. Some credentials are in more demand than others. Some credentials only allow for a very particular type of inspection. Incorporating credentials into the model would also greatly improve it.

A lot of work has been done for this model. However in order to make it truly effective, a lot more work needs to be done before it can be able to be implemented in real scenario in our accreditation space.

## 6 References

Archetti, C., Savelsbergh, M., Speranza, M. G. (2016). The Vehicle Routing Problem with Occasional Drivers. European Journal of Operational Research, 254(2), 472–480.doi: 10.1016/j.ejor.2016.03.049

Liu, B. (2009). Theory and Practice of Uncertain Programming. Berlin, Heidelberg: Springer Berlin Heidelberg.
Nuriyeva, F., Kizilates, G. (2017). A NEW HEURISTIC ALGORITHM FOR MULTIPLE TRAVELING SALESMAN PROBLEM. Journal of Applied and Engineering Mathematics, 7, 101–109.
Zhang, J., Zhao, Y., Xue, W., Li, J. (2015). Vehicle routing problem with fuel consumption and carbon emission. Int. J. Production Economics, 232–242.

## 7 Appendix

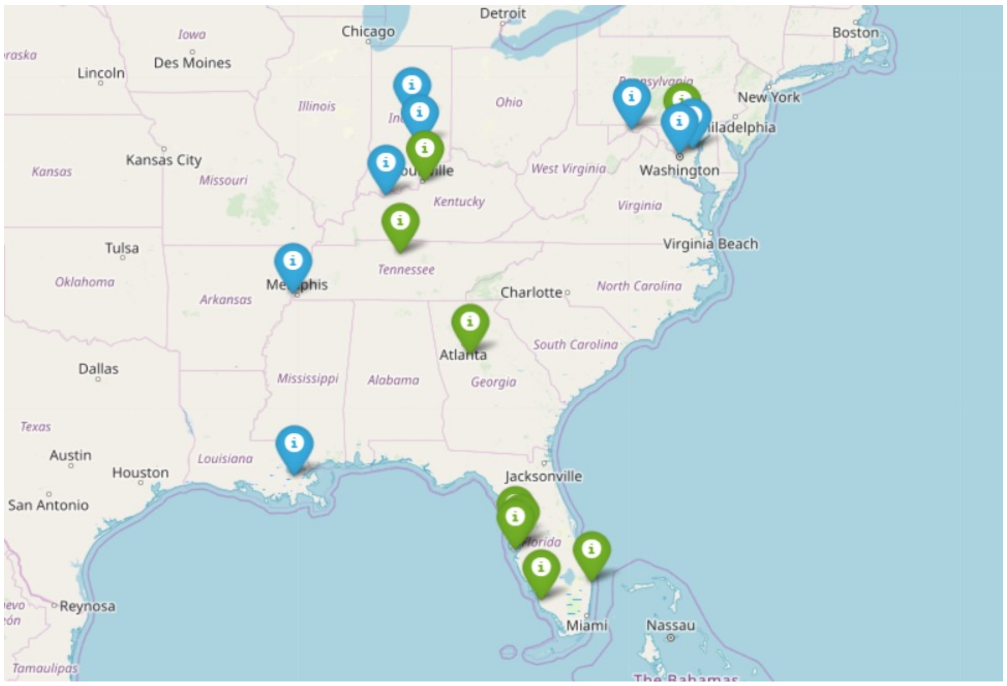
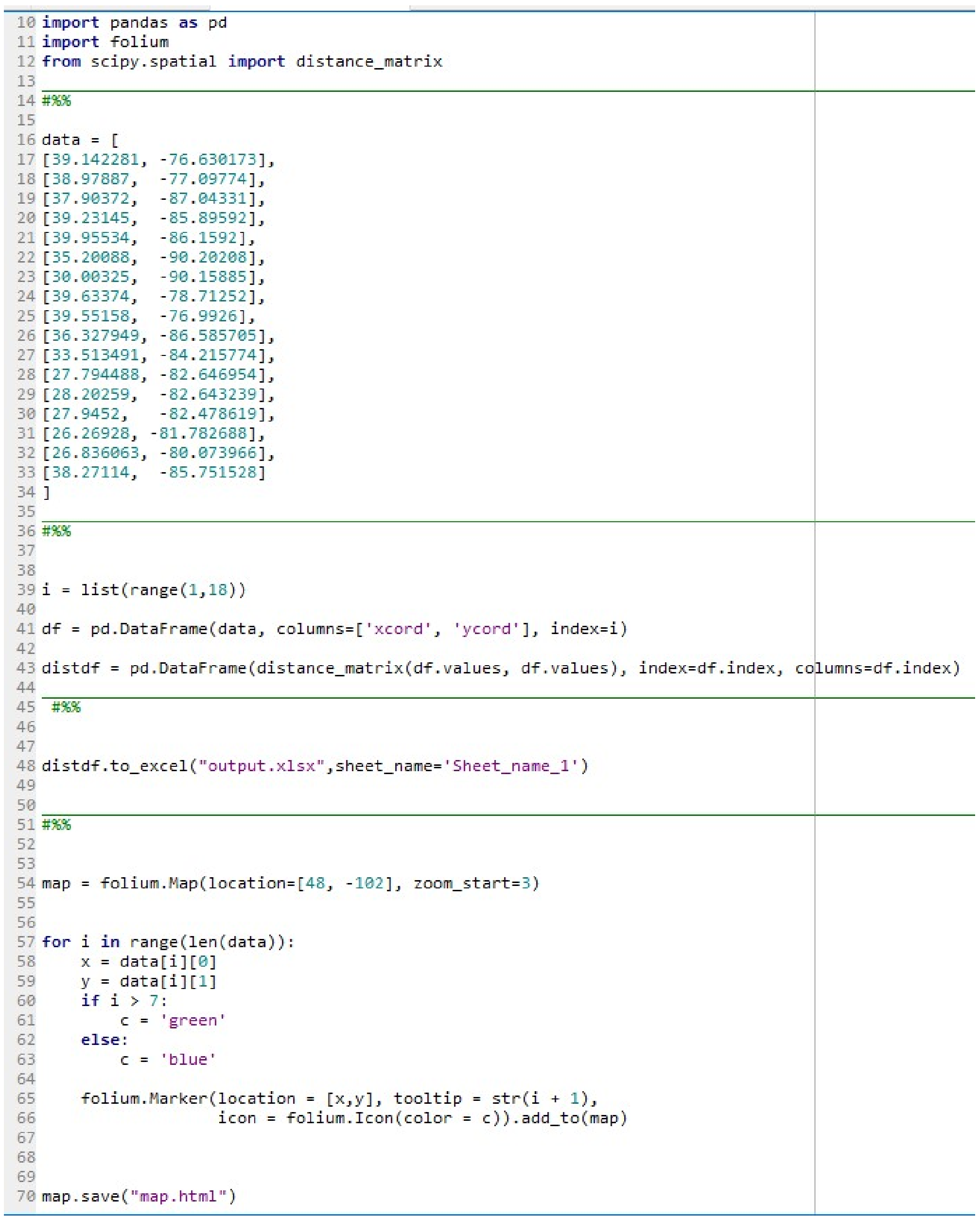
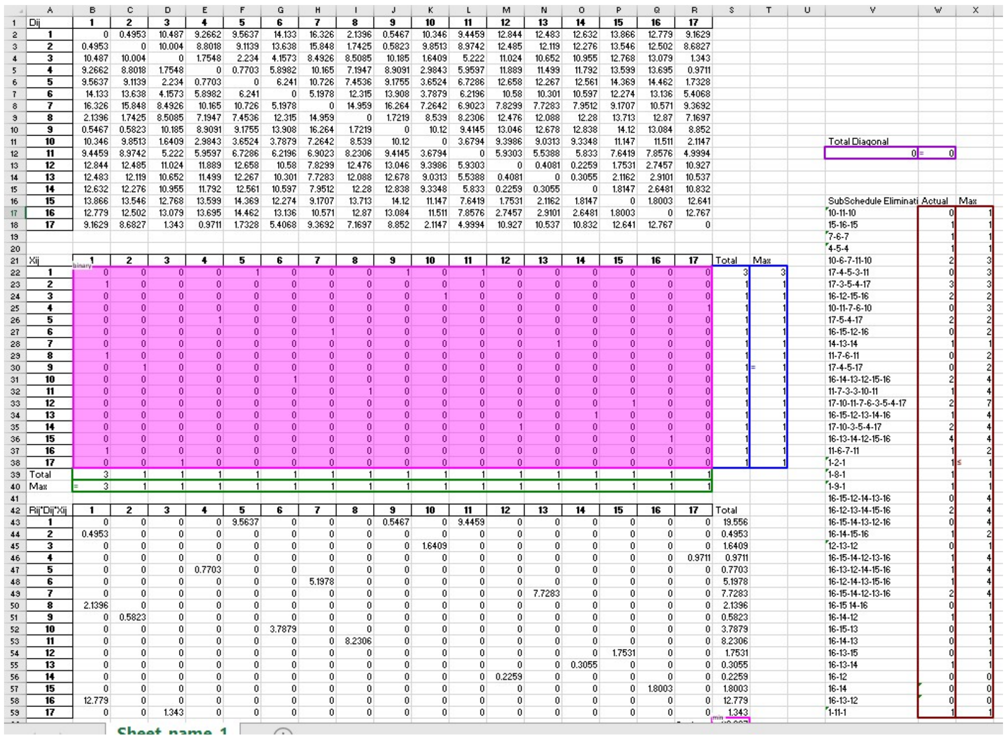


Figure 1: All Locations to be visited, Green Markers indicate facilities that generate more revenue.



```python
import pandas as pd
import folium
from scipy.spatial import distance_matrix

#%%

data = [
[39.142281, -76.630173],
[38.97887,  -77.09774],
[37.90372,  -87.04331],
[39.23145,  -85.89592],
[39.95534,  -86.1592],
[35.20088,  -90.20208],
[30.00325,  -90.15885],
[39.63374,  -78.71252],
[39.55158,  -76.9926],
[36.327949, -86.585705],
[33.513491, -84.215774],
[27.794488, -82.646954],
[28.20259,  -82.643239],
[27.9452,   -82.478619],
[26.26928, -81.782688],
[26.836063, -80.073966],
[38.27114,  -85.751528]
]

#%%


i = list(range(1,18))

df = pd.DataFrame(data, columns=['xcord', 'ycord'], index=i)

distdf = pd.DataFrame(distance_matrix(df.values, df.values), index=df.index, columns=df.index)

 #%%


distdf.to_excel("output.xlsx",sheet_name='Sheet_name_1')


#%%


map = folium.Map(location=[48, -102], zoom_start=3)


for i in range(len(data)):
    x = data[i][0]
    y = data[i][1]
    if i > 7:
        c = 'green'
    else:
        c = 'blue'

    folium.Marker(location = [x,y], tooltip = str(i + 1),
                  icon = folium.Icon(color = c)).add_to(map)



map.save("map.html")
```

Figure 2: Code used to generate excel spreadsheet and maps.



| Dij | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **1** | 0 | 0.4953 | 10.487 | 9.2662 | 9.5637 | 14.133 | 16.326 | 2.1396 | 0.5467 | 10.346 | 9.4459 | 12.844 | 12.483 | 12.632 | 13.866 | 12.779 | 9.1629 |
| **2** | 0.4953 | 0 | 10.004 | 8.8018 | 9.1139 | 13.638 | 15.848 | 1.7425 | 0.5823 | 9.8513 | 8.9742 | 12.485 | 12.119 | 12.276 | 13.546 | 12.502 | 8.6827 |
| **3** | 10.487 | 10.004 | 0 | 1.7548 | 2.234 | 4.1573 | 8.4926 | 8.5085 | 10.185 | 1.6409 | 5.222 | 11.024 | 10.652 | 10.955 | 12.768 | 13.079 | 1.343 |
| **4** | 9.2662 | 8.8018 | 1.7548 | 0 | 0.7703 | 5.8982 | 10.165 | 7.1947 | 8.9091 | 2.9843 | 5.9597 | 11.889 | 11.499 | 11.792 | 13.599 | 13.695 | 0.9711 |
| **5** | 9.5637 | 9.1139 | 2.234 | 0.7703 | 0 | 6.241 | 10.726 | 7.4536 | 9.1755 | 3.6524 | 6.7286 | 12.658 | 12.267 | 12.561 | 14.369 | 14.462 | 1.7328 |
| **6** | 14.133 | 13.638 | 4.1573 | 5.8982 | 6.241 | 0 | 5.1978 | 12.315 | 13.908 | 3.7879 | 6.2196 | 10.58 | 10.301 | 10.597 | 12.274 | 13.136 | 5.4068 |
| **7** | 16.326 | 15.848 | 8.4926 | 10.165 | 10.726 | 5.1978 | 0 | 14.959 | 16.264 | 7.2642 | 6.9023 | 7.8299 | 7.7283 | 7.9512 | 9.1707 | 10.571 | 9.3692 |
| **8** | 2.1396 | 1.7425 | 8.5085 | 7.1947 | 7.4536 | 12.315 | 14.959 | 0 | 1.7219 | 8.539 | 8.2306 | 12.476 | 12.088 | 12.28 | 13.713 | 12.87 | 7.1697 |
| **9** | 0.5467 | 0.5823 | 10.185 | 8.9091 | 9.1755 | 13.908 | 16.264 | 1.7219 | 0 | 10.12 | 9.4145 | 13.046 | 12.678 | 12.838 | 14.12 | 13.084 | 8.852 |
| **10** | 10.346 | 9.8513 | 1.6409 | 2.9843 | 3.6524 | 3.7879 | 7.2642 | 8.539 | 10.12 | 0 | 3.6794 | 9.3986 | 9.0313 | 9.3348 | 11.147 | 11.511 | 2.1147 |
| **11** | 9.4459 | 8.9742 | 5.222 | 5.9597 | 6.7286 | 6.2196 | 6.9023 | 8.2306 | 9.4145 | 3.6794 | 0 | 5.9303 | 5.5388 | 5.833 | 7.6419 | 7.8576 | 4.9994 |
| **12** | 12.844 | 12.485 | 11.024 | 11.889 | 12.658 | 10.58 | 7.8299 | 12.476 | 13.046 | 9.3986 | 5.9303 | 0 | 0.4081 | 0.2259 | 1.7531 | 2.7457 | 10.927 |
| **13** | 12.483 | 12.119 | 10.652 | 11.499 | 12.267 | 10.301 | 7.7283 | 12.088 | 12.678 | 9.0313 | 5.5388 | 0.4081 | 0 | 0.3055 | 2.1162 | 2.9101 | 10.537 |
| **14** | 12.632 | 12.276 | 10.955 | 11.792 | 12.561 | 10.597 | 7.9512 | 12.28 | 12.838 | 9.3348 | 5.833 | 0.2259 | 0.3055 | 0 | 1.8147 | 2.6481 | 10.832 |
| **15** | 13.866 | 13.546 | 12.768 | 13.599 | 14.369 | 12.274 | 9.1707 | 13.713 | 14.12 | 11.147 | 7.6419 | 1.7531 | 2.1162 | 1.8147 | 0 | 1.8003 | 12.641 |
| **16** | 12.779 | 12.502 | 13.079 | 13.695 | 14.462 | 13.136 | 10.571 | 12.87 | 13.084 | 11.511 | 7.8576 | 2.7457 | 2.9101 | 2.6481 | 1.8003 | 0 | 12.767 |
| **17** | 9.1629 | 8.6827 | 1.343 | 0.9711 | 1.7328 | 5.4068 | 9.3692 | 7.1697 | 8.852 | 2.1147 | 4.9994 | 10.927 | 10.537 | 10.832 | 12.641 | 12.767 | 0 |

| Xij | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | Total | Max |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **1** | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 3 | 3 |
| **2** | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |
| **3** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |
| **4** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 |
| **5** | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |
| **6** | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |
| **7** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 |
| **8** | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |
| **9** | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |
| **10** | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |
| **11** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |
| **12** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 1 |
| **13** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 1 |
| **14** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |
| **15** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 |
| **16** | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |
| **17** | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |
| Total | 3 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | | |
| Max | 3 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | | |

| Rij*Dij*Xij | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | Total |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **1** | 0 | 0 | 0 | 0 | 9.5637 | 0 | 0 | 0 | 0.5467 | 0 | 9.4459 | 0 | 0 | 0 | 0 | 0 | 0 | 19.556 |
| **2** | 0.4953 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0.4953 |
| **3** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1.6409 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1.6409 |
| **4** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0.9711 | 0.9711 |
| **5** | 0 | 0 | 0 | 0.7703 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0.7703 |
| **6** | 0 | 0 | 0 | 0 | 0 | 0 | 5.1978 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 5.1978 |
| **7** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 7.7283 | 0 | 0 | 0 | 0 | 7.7283 |
| **8** | 2.1396 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 2.1396 |
| **9** | 0 | 0.5823 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0.5823 |
| **10** | 0 | 0 | 0 | 0 | 0 | 3.7879 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 3.7879 |
| **11** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 8.2306 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 8.2306 |
| **12** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1.7531 | 0 | 0 | 1.7531 |
| **13** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0.3055 | 0 | 0 | 0 | 0.3055 |
| **14** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0.2259 | 0 | 0 | 0 | 0 | 0 | 0.2259 |
| **15** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1.8003 | 0 | 1.8003 |
| **16** | 12.779 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 12.779 |
| **17** | 0 | 0 | 1.343 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1.343 |

| Total Diagonal | | |
|---|---|---|
| 0 | = | 0 |

| SubSchedule Eliminati | Actual | Max |
|---|---|---|
| 10-11-10 | 0 | 1 |
| 15-16-15 | 1 | 1 |
| 7-6-7 | 1 | 1 |
| 4-5-4 | 1 | 1 |
| 10-6-7-11-10 | 2 | 3 |
| 17-4-5-3-11 | 0 | 3 |
| 17-3-5-4-17 | 3 | 3 |
| 16-12-15-16 | 2 | 2 |
| 10-11-7-6-10 | 0 | 3 |
| 17-5-4-17 | 2 | 2 |
| 16-15-12-16 | 0 | 2 |
| 14-13-14 | 1 | 1 |
| 11-7-6-11 | 0 | 2 |
| 17-4-5-17 | 0 | 2 |
| 16-14-13-12-15-16 | 2 | 4 |
| 11-7-3-3-10-11 | 1 | 4 |
| 17-10-11-7-6-3-5-4-17 | 2 | 7 |
| 16-15-12-13-14-16 | 1 | 4 |
| 17-10-3-5-4-17 | 2 | 4 |
| 16-13-14-12-15-16 | 4 | 4 |
| 11-6-7-11 | 1 | 2 |
| 1-2-1 | 1 | 1 |
| 1-8-1 | 1 | 1 |
| 1-9-1 | 1 | 1 |
| 16-15-12-14-13-16 | 0 | 4 |
| 16-12-13-14-15-16 | 2 | 4 |
| 16-15-14-13-12-16 | 0 | 4 |
| 16-14-15-16 | 1 | 2 |
| 12-13-12 | 0 | 1 |
| 16-15-14-12-13-16 | 1 | 4 |
| 16-13-12-14-15-16 | 1 | 4 |
| 16-12-14-13-15-16 | 1 | 4 |
| 16-15-14-12-13-16 | 2 | 4 |
| 16-15 14-16 | 0 | 1 |
| 16-14-12 | 1 | 1 |
| 16-15-13 | 0 | 1 |
| 16-14-13 | 0 | 1 |
| 16-13-15 | 0 | 1 |
| 16-13-14 | 1 | 1 |
| 16-12 | 0 | 0 |
| 16-14 | 0 | 0 |
| 16-13-12 | 0 | 0 |
| 1-11-1 | 1 | 1 |

Sheet name 1

Figure 3: Screenshot of all values and constraints in MS ASPE.

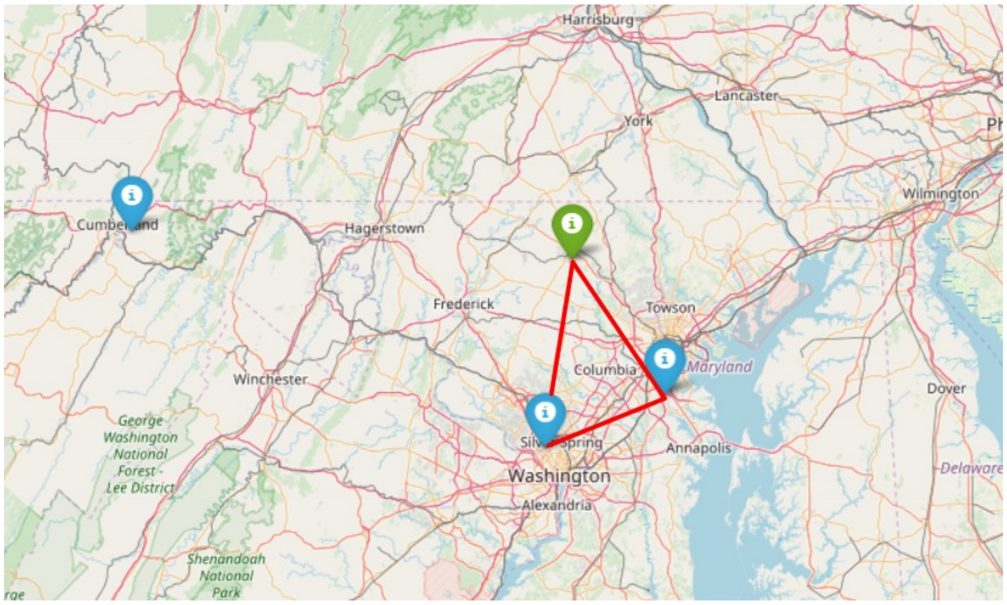


Figure 4: Tour One

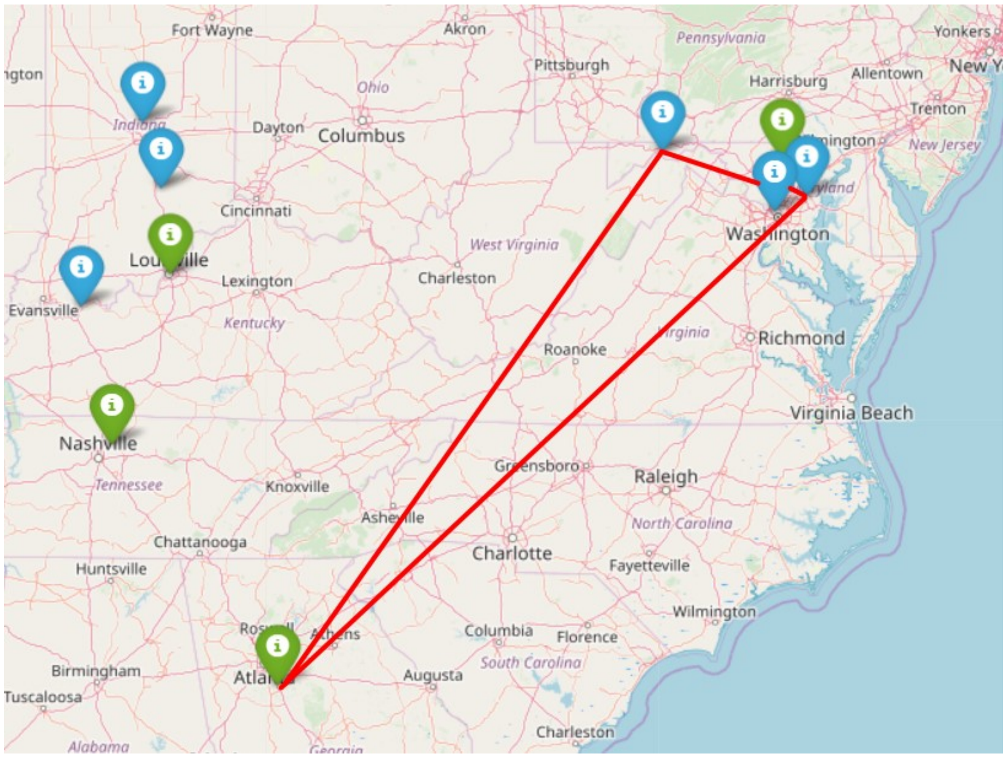


Figure 5: Tour Two

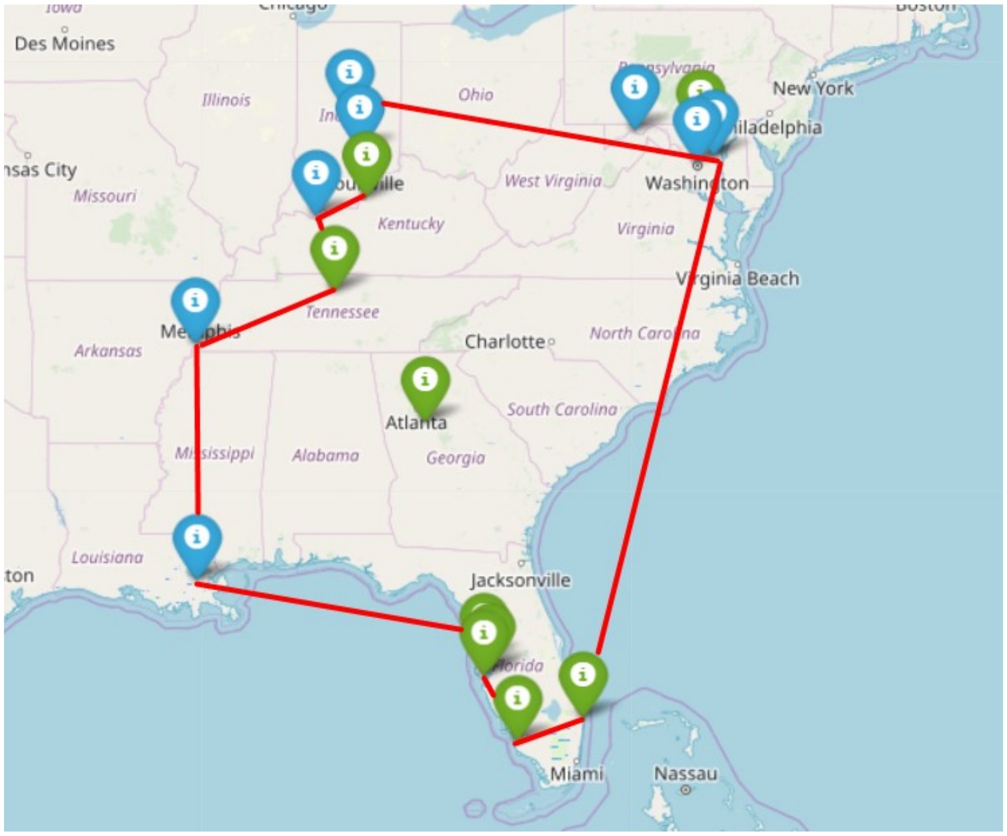


Figure 6: Tour Three