UNIVERSITY OF WATERLOO

Faculty of Engineering

Department of Management Sciences

MSCI 332: Assignment 3

Topic 3: EV Charger Planning for Long-Distance Travel

Prepared by:

Gunchica Bhalla

Ashwuni Kumar

Qirui Liu

03 December 2021

Table of Contents

Executive summary

This report investigates the dynamical location optimization problem for electric vehicles (henceforth referred to as EV) charging station infrastructure for long-distance trips to solve the obstacle of large-scale adoption of EVs. This report constructs and implements a construction heuristic algorithm and a corresponding simulated annealing metaheuristic in pursuit of feasible solutions, their neighbouring feasible solutions, and the overall minimization of the total cost of building EV charging stations for different origin-destination (OD) pairs.

Problem Description

The lack of EV charging station infrastructure for long-distance trips has become an obstacle in the large-scale adoption of EVs in society and, thus, the future of EV's is highly dependent on the construction of charging stations. As a result, we must develop and implement a construction heuristic to optimize the ability of EV drivers to travel between all available cities in any given OD pair via the shortest path between them.

Description of the Algorithm

The main concern regarding the placement of EV charging stations is ensuring that all journeys within the specified network are completable. In other words, if we are looking at the Greater Toronto Area, regardless of whether a car begins the journey in Burlington or Mississauga and completes it in Brampton or Vaughn, the journey must be completable. A journey is only completable if, throughout the path the car travels, there is always an EV charging station within R of each other, where R is the range the EV driving is able to complete when it begins the journey fully charged. The value used for R was taken from the average range of a Tesla Model Y EV, 303 miles, [4] however, to account for the distance that needs to be driven from the geographical centre of the city to the location of the EV charging station and the vehicle battery that is required to enter a charging stall, the range used is 250 miles or approximately 400 km. As a result, given a network of cities in a specific OD pair, the algorithm must determine in which cities to locate EV charging stations so that every path within the OD pair is completable.

To accomplish this, an algorithm was created. The algorithm iterates through all paths in an OD pair and iteratively builds EV charging stations throughout the path until all paths in the OD pair are completed. Paths are defined as every path m where m starts at city i and ends at city j and i and j are elements of the set of all cities in the OD pair data I and path m is the shortest route possible between the aforementioned i and j. The EV charging stations are built at the city t where the distance between city b and city t is the maximum distance of all cities following city b so long as the distance between the cities is within vehicle range R. City b is initialized as the path origin city i and then iterates through the newly determined EV charging station locations until the distance between all EV charging stations, and the distance between an EV charging stations and the path origin city i and path destination city j are within vehicle range R of each other. The pseudocode for the algorithm is available in Appendix B.

The algorithm provides data regarding all the cities in the OD pair, which cities are connected to each other, and the distances between connected cities. Once given this data, the algorithm iterates through all the possible paths in the OD pair. The path m that is added to a list - paths - is what is returned by the method dijsktra when the respective i and j inputs are passed to it. The dijsktra method works in union with the __init__ and add_edge methods. The pseudocode for these methods in addition to the eventually implemented code was sourced from benalexkeen.com. The exact source for this code can be seen in References: reference [1]. In essence, these methods are used to determine which path can be taken between the inputted origin i and destination j in order to minimize the overall distance travelled (the shortest path problem).

Once all of these shortest path solutions are added to the path list, the algorithm calls the get_Can_list method to determine where to locate EV charging stations on each respective path. The method does this by iterating through the cities in the path in the order they come in and placing stations at the maximum feasible distance from each other. In other words, the algorithm considers an origin point wherein the car will begin its journey fully charged with a range R. The algorithm then iterates through all the cities in the path that come after this origin point, making note of the distance from the origin city to the city at hand. Once the algorithm identifies that the distance from the origin city to the city at hand has exceeded the specified vehicle range R, the algorithm adds the previously looked at city f to the candidates list (the last city whose distance from the origin city was less than the vehicle range R). The algorithm then resets the origin city as city f and repeats this iteration until the distance between all EV charging stations, and the distance between an EV charging station and the path origin city i and path destination city j are within vehicle range R of each other. This method returns the candidates list.

Finally, to prevent double-counting and to make the solution as clear as possible, the algorithm takes the candidates list and adds all the unique stations mentioned to the unique_candidate list. The algorithm will then output the solution to this problem: the locations of where to build the EV charging stations. Furthermore, the algorithm will iterate through the cities in the unique_candidate list, summing up the costs for building a station in each city, and output the total cost for building the aforementioned stations.

The assumptions made to implement this algorithm are located in Appendix F.

In addition to the aforementioned algorithm - the construction heuristic - a metaheuristic was created to explore neighbouring feasible solutions to the initial feasible solution provided by our algorithm. Once a solution for an OD pair was provided by the algorithm, the created simulated annealing metaheuristic would investigate cities beside the EV charging station's city (provided by the algorithm) and, if this neighbouring solution would lessen the cost of building this station, the metaheuristic would accept the neighbouring solution. If the neighbouring solution would lessen the cost of building this station, the metaheuristic would accept the neighbouring solution with some probability p. The pseudocode for the metaheuristic is in Appendix G.

The metaheuristic iterates through all paths in the solution provided by the algorithm and iteratively finds neighbouring feasible solutions for all EV charging stations throughout the path until all paths in the algorithm solution are looked at. In this situation, a neighbouring solution pertains to the relocating of an EV charging station from city n to the city that m that comes prior to n on the path m from path origin point i to path destination point j. In other words, if path m were to be ['Toronto', 'Mississauga', 'Oakville', 'Burlington', 'Hamilton'] and our algorithm provided a solution for this path that an EV charging station were to be built at Burlington, the metaheuristic would consider the neighbouring solution Oakville. A neighbouring solution was said to be feasible if and only if the sum of the distance between the neighbouring solution m and the distance between the initial solution n and city g was less than the EV range R, where, if there is another EV charging station after the EV charging station under review, g is the distance between these two stations, otherwise g is the distance between the EV charging station under review and the path destination j. To continue with the previous example, the neighbouring solution Oakville is feasible if and only if the distance from Oakville to Hamilton is less than the EV range R. If there were another EV charging station in this path solution, say X, the solution would have been feasible if and only if the distance between Oakville and city X was less than R. This neighbouring feasible solution was then faced with two cases: if the cost of building an EV charging station at the neighbouring feasible solution was less than the cost of the EV charging station at the initial solution, the neighbouring feasible solution was accepted. However, if this was not the case, the neighbour feasible solution was accepted with some probability p where (as per the lectures):

$$\text{p(accept candidate solution)} = e^{(Zc - Zn)/T}$$

where Zc is the cost of the EV charging station at the initial solution, Zn is the cost of building an EV charging station at the neighbouring feasible solution, and T is the temperature - a parameter measuring the tendency to accept the neighbouring feasible solution - which was set to 0.4. Once calculated, if the probability p were greater than 0.95, the neighbouring feasible solution was accepted. If not, then the EV charging station remained at the initial solution.

Numerical testing

In order to test the algorithm, data were obtained on three different OD pairs: Toronto, Ontario to Montréal, Québec; Allentown, Pennsylvania to New Haven, Connecticut; and Yreka, California to San Francisco, California. Please refer to Appendix C for this data. The purpose for choosing these pairs are as follows:

- The Toronto to Montréal OD pair has data that is very clustered on one end and then trails off in a very narrow path on the other end. This data was used to test the algorithm against varied path lengths.
- The Allentown to New Haven OD pair has data that falls all over the informal Ivy loop. The loop covers the general area around and between Ivy League schools in the United States (refer to Appendix E). As a very popular road trip for students, education staff, and parents and guardians alike, this interstate trip is a prime candidate for encouraging and affording federal government-funded stations. For instance, this data could apply in a situation such as the one outlined in Grist's article: Biden *wants to build 500,000 EV charging stations. Where will they all go?* [3].
- The Yreka to San Francisco OD pair has data that solely falls within the northern half of the California state. Data like this might be used by the state of California to build federal or state government-funded stations. Alternatively, as the location where Tesla was founded - a leader in EV technology - this location is a prime candidate for the building of privately funded EV charging stations.

A visual representation of the network described by the data for the Yreka, California to San Francisco, California OD pair can be found in Appendix H. The network was determined by paralleling the selected cities in the north California region with the major highway systems for this region.

The output and solutions for the algorithm and the metaheuristic discussed in ‘Description of the Algorithm’ are located in Appendix A.

Let us consider the computational time of the algorithm. As shown in the output, the runtime for the Toronto, Ontario to Montréal, Québec OD pair, the Allentown, Pennsylvania to New Haven, Connecticut OD pair, and the Yreka, California to San Francisco, California OD pair are 0.0433946509983798 seconds, 0.03033331899860059 seconds, and 0.12931593399844132 seconds, respectively. We can see that these runtimes at first glance are quite good however, as a more formal indicator, we can compare how the algorithm fared for the datasets of the greatest difference in magnitudes: the Toronto, Ontario to Montréal, Québec OD pair and the Yreka, California to San Francisco, California OD pair. Let us refer to them as CAN and USA, respectively. As the difference between the average distance between cities for USA and CAN is relatively quite insignificant - 25.3953488 km - we can comfortably compare the impact the number of cities has on the performance of the algorithm. With almost twice the number of cities, the USA's runtime was approximately 2.979997097 that of CAN. As the number of cities added to the dataset shares a quadratic relationship to the number of paths m - m=n*(n-1), we can understand that the runtime doesn’t share a linear relationship with the number of cities in the dataset either. In fact, the aforementioned quadratic relationship indicates that, for the expected quadratic increase in the number of paths to implement EV charging stations on, the algorithm fared quite well: only increasing 2.979997097 fold. As a construction heuristic, the algorithm aims to “decrease the time complexity of problems by giving quick solutions” [6]. Therefore, at its core, the runtime of the algorithm is a good indicator of the quality of our heuristic and its usability. Furthermore, as the potential for our problem is quite limitless, the quality and usability of the heuristic is that much more important. For instance, the data used to test this algorithm is essentially insignificant in comparison to the vast and extensive amounts of data that apply to the topic of locating EV charging stations. For instance, this algorithm could be used on all the potential locations for EV charging stations in the entirety of the United States of America down to the exact street address!

Next, let us compare the performance of the algorithm and the metaheuristic. We see two different situations demonstrated by our three different OD pairs: the metaheuristic reveals that the problem is still feasible and cheaper with less charging stations or the metaheuristic solution is identical to the algorithm solution. In the Toronto, Ontario to Montréal, Québec OD pair, the algorithm returns EV stations located at Toronto, Ajax, and Pickering whereas the metaheuristic simply returns Toronto and Pickering. As Toronto and Ajax are beside one another, this indicates that the metaheuristic identified that path(s) that locate EV charging stations in Ajax are completable and have cheaper costs if we move the station back to Toronto. Toronto was already on the solution list but the metaheuristic identified no need for the addition of Ajax. This is a very good result from the metaheuristic as having to build one less station greatly reduces the total cost for the OD pair, a very good situation in light of our wanting to minimize the objective value. Similarly, the metaheuristic solution for the Yreka, California to San Francisco, California OD pair eliminates the need for two stations - Redwood

City and Markleeville - which shows the potential for the cost reduction possibilities of the metaheuristic. On the other hand, the Allentown, Pennsylvania to New Haven, Connecticut OD pair implies that, either the algorithm returned the only possible feasible solutions for each path, neighbouring solutions were not cheaper options, or that even the relocation of certain path EV charging station solutions could not eliminate the need for any cities. This gives insight into the strength of our algorithm and its ability to return good quality results.

Finally, let us consider the quality of our algorithm and metaheuristic. I believe the solutions provided by both our algorithm and our metaheuristic are of high quality. Our heuristic ensures that all paths are completable and does so in the minimum stations required per path. In general, we see that the estimated costs of building EV charging stations are larger than the estimated differences in building different stations. For instance, for our Yreka, California to San Francisco, California OD pair, the estimated cost of building an EV charging station is $126751.04 and the estimated difference in cost of building different stations is $91369.99. Therefore minimizing the number of EV charging stations on a path such that the path is always completable is very significant in the minimization of our objective value: the cost of building said EV charging stations. Furthermore, at its core, the algorithm considers every path within the inputted OD pair. This is very realistic as transportation between cities is a very dynamic network and we have no control over where drivers begin their journeys and where they end them. Furthermore, the algorithm uses the shortest path for each path. Once again, we have no control over drivers and can’t force them to take the more convenient route between their origin and destination for us. Therefore, as taking the shortest path between their origin point and destination is a more realistic expectation, our algorithm once again encourages realism. The use of every shortest path results in the algorithm being set up in a high-quality, realistic way.

Our metaheuristic, on the other hand, investigates neighbouring solutions and obtains better solutions for our data (better objective values). Our metaheuristic is of high quality as it does not impact the completability of any paths and does not implement any additional costly EV charging stations, yet it is able to find neighbouring solutions which can improve our overall objective value. In other words, the metaheuristic maintains the same level of feasibility as our algorithm and does not sacrifice the effectiveness of our algorithm in terms of minimizing the overall cost while locating better solutions for the case at hand. Not to mention, the metaheuristic upholds the realism and quality of the algorithm. Another good quality of the metaheuristic is how useful it is in realistic situations. Realistically, even locations of relatively minimal distance apart can have drastic price differences. For instance, in Toronto, ON the cost of building an EV charging station - as per our previously discussed estimation calculation - in the Wellesley station region is $130,517.58, whereas the cost in the Union station region is $618,879.55. At a mere 3 km apart, in the face of the 400km range value of an EV and average intercity distance of 64.56521739 km (obtained from the Toronto, Ontario to Montréal, Québec OD pair data), this distance is very insignificant yet holds great potential for lowering overall EV charging station costs.

References

[1] Implementing Djikstra’s Shortest Path Algorithm with Python
https://benalexkeen.com/implementing-djikstras-shortest-path-algorithm-with-python/

[2] Average EV Owner Drives Half as Many Miles as Other Drivers—Study, 02-21
https://www.caranddriver.com/news/a35498794/ev-owners-low-mileage-study/#:~:text=PIA's%20study%20of%20EV%20owners,majority%20also%20charge%20in%20public.%22

[3] Biden wants to build 500,000 EV charging stations. Where will they all go?, 04-21
https://grist.org/energy/biden-wants-to-build-500000-ev-charging-stations-where-will-they-all-go/

[4] Longest-range electric vehicles (EVs) you can buy in 2021, 10-21
https://electrek.co/2021/10/15/longest-range-evs-2021/

[5] Zillow
https://www.zillow.com/

[6] Greedy Vs. Heuristic Algorithm, 10-21
https://www.baeldung.com/cs/greedy-vs-heuristic-algorithm

Appendix A: Algorithm and Metaheuristics Results

The output and solutions for the algorithm and the metaheuristic discussed in ‘Description of the Algorithm’ are as follows:

1. Instance 1: Toronto, Ontario to Montréal, Québec OD pair

- **Output:**

  i. Construction Heuristic

  ```
  Run Time:  0.0433946509983798
  Number of shortest paths with a distance greater than Range R:  66
  SOLUTION TO THIS PROBLEM {'I', 'P', 'F'}
  Cost of Building the following station = $531,894
  ```

  ii. Simulated Annealing

  ```
  SOLUTION TO THIS PROBLEM AFTER ANNEALING ARE: {'I',  'P'}
  Cost After Annealing is: $300,548
  ```

- **Solution:**

  i. Construction Heuristic

  The solution is Toronto, Kingston and Ajax. Total cost: $531,894.

  ii. Simulated Annealing

  The solution is Toronto and Kingston. Total cost: $300,548.

2. Instance 2: Allentown, Pennsylvania to New Haven, Connecticut OD pair

- **Output:**

  i. Construction Heuristic

  ```
  Run Time:  0.03033331899860059
  Number of shortest paths with a distance greater than Range R:  23
  SOLUTION TO THIS PROBLEM {'G', 'C', 'K', 'E', 'I'}
  Cost of Building the following station = $559,387
  ```

  ii. Simulated Annealing

  ```
  SOLUTION TO THIS PROBLEM AFTER ANNEALING ARE: {'G', 'C', 'K', 'E', 'I'}
  Cost After Annealing is: $559,387
  ```

- **Solution:**

  i. Construction Heuristic

  The solution is Princeton, New Brunswick, Bethlem, New York City, Allentown. Total cost: $559,387.

  ii. Simulated Annealing

  The solution is Princeton, New Brunswick, Bethlem, New York City, Allentown. Total cost: $559,387.

3. Instance 3: Yreka, California to San Francisco, California OD pair

- **Output:**

i. Construction Heuristic

```
2716810
Run Time:  0.12931593399844132
Number of shortest paths with distance greater than Range R:  334
SOLUTION TO THIS PROBLEM {'W', 'G', 'M', 'J', 'K', 'N', 'V', 'I', 'F', 'B', 'AA', 'H', 'O', 'Y', 'C', 'E', 'L', 'S', 'P', 'R', 'D', 'U'}
Cost of Building the following station = $2,716,810
```

ii. Simulated Annealing

```
SOLUTION TO THIS PROBLEM AFTER ANNEALING ARE: {'W', 'G', 'M', 'J', 'K', 'N', 'V', 'I', 'F', 'B', 'AA', 'H', 'O', 'C', 'E', 'L', 'P', 'D', 'R', 'U'}
Cost After Annealing is: $2,228,989
```

- **Solution:**

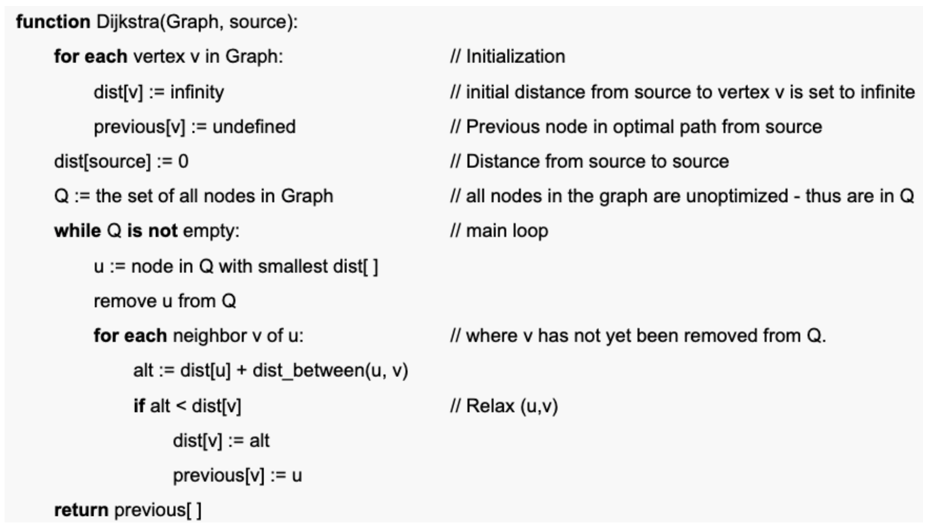
i. Construction Heuristic

The solution is Sonora, Susanville, Nevada City, Oroville, Ukiah, Marysville, Oakland, Willows, Red Bluff, Alturas, Modesto, Quincy, Santa Rosa, Redwood City, Eureka, Redding, Lakeport, Markleeville, Sacramento, Jackson, Weaverville, Stockton. Total cost: $2,716,810.

ii. Simulated Annealing

The solution is Sonora, Susanville, Nevada City, Oroville, Ukiah, Marysville, Oakland, Willows, Red Bluff, Alturas, Modesto, Quincy, Santa Rosa, Eureka, Redding, Lakeport, Sacramento, Jackson, Weaverville, Stockton. Total cost: $2,228,989.

Appendix B: Pseudocode of the Algorithm



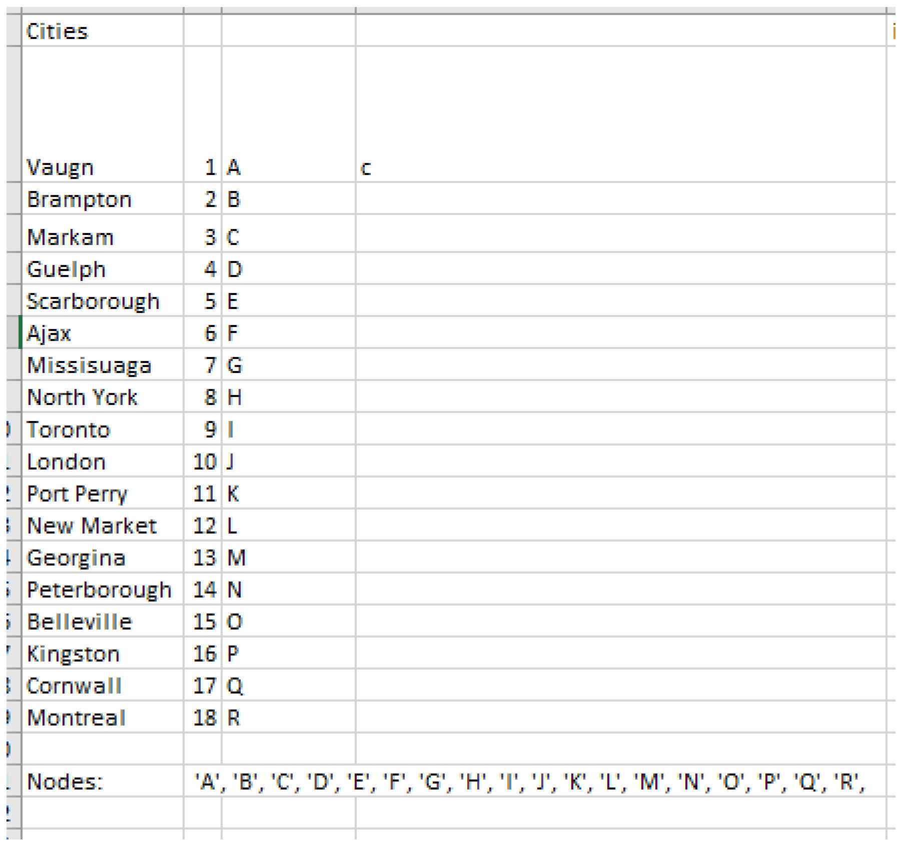
```
function Dijkstra(Graph, source):
    for each vertex v in Graph:                // Initialization
        dist[v] := infinity                    // initial distance from source to vertex v is set to infinite
        previous[v] := undefined               // Previous node in optimal path from source
    dist[source] := 0                          // Distance from source to source
    Q := the set of all nodes in Graph         // all nodes in the graph are unoptimized - thus are in Q
    while Q is not empty:                      // main loop
        u := node in Q with smallest dist[ ]
        remove u from Q
        for each neighbor v of u:              // where v has not yet been removed from Q.
            alt := dist[u] + dist_between(u, v)
            if alt < dist[v]                   // Relax (u,v)
                dist[v] := alt
                previous[v] := u
    return previous[ ]
```

```
function distance(shortest_path):
    For i in range(length of the shortest_path - 1):
            Temp = distance between current node to next node
            Distance = Distance + Temp
    Return Distance

Function get_Can_list (shortest_path):
    While i in range(length of shortest_path - 1)
            arc =  distance between current to next code
            EV_range = EV_range - arc
            If EV_range == 0:
                    Add next node i+1 to the Candidate list
                    Set EV_range = 400 - arc
            If EV_range < 0:
                    Add current node i to the candidate list
                    Set EV_range = 400 - arc
                    i++
            Else:
                    #do nothing
                    i++
    Return Candidate list

    # combining the functions:
     nodes = all nodes in the network
     For i in range (len(nodes))
              Start = nodes[i]
              For i in range (len(nodes))
                if nodes[i] != start:
                  end = nodes[i]
                  shortest = dijsktra(graph, start, end)
                  #check if shortest path is longer than the range of EV then add to list
                  if distance(shortest) > R:
                    Add the path to shortest path array
                    Get station candidates of that shortest path
unique_candidate = set(Candidate)

For s in unique_candidate:
     Sum cost in the unique_candidate list
```

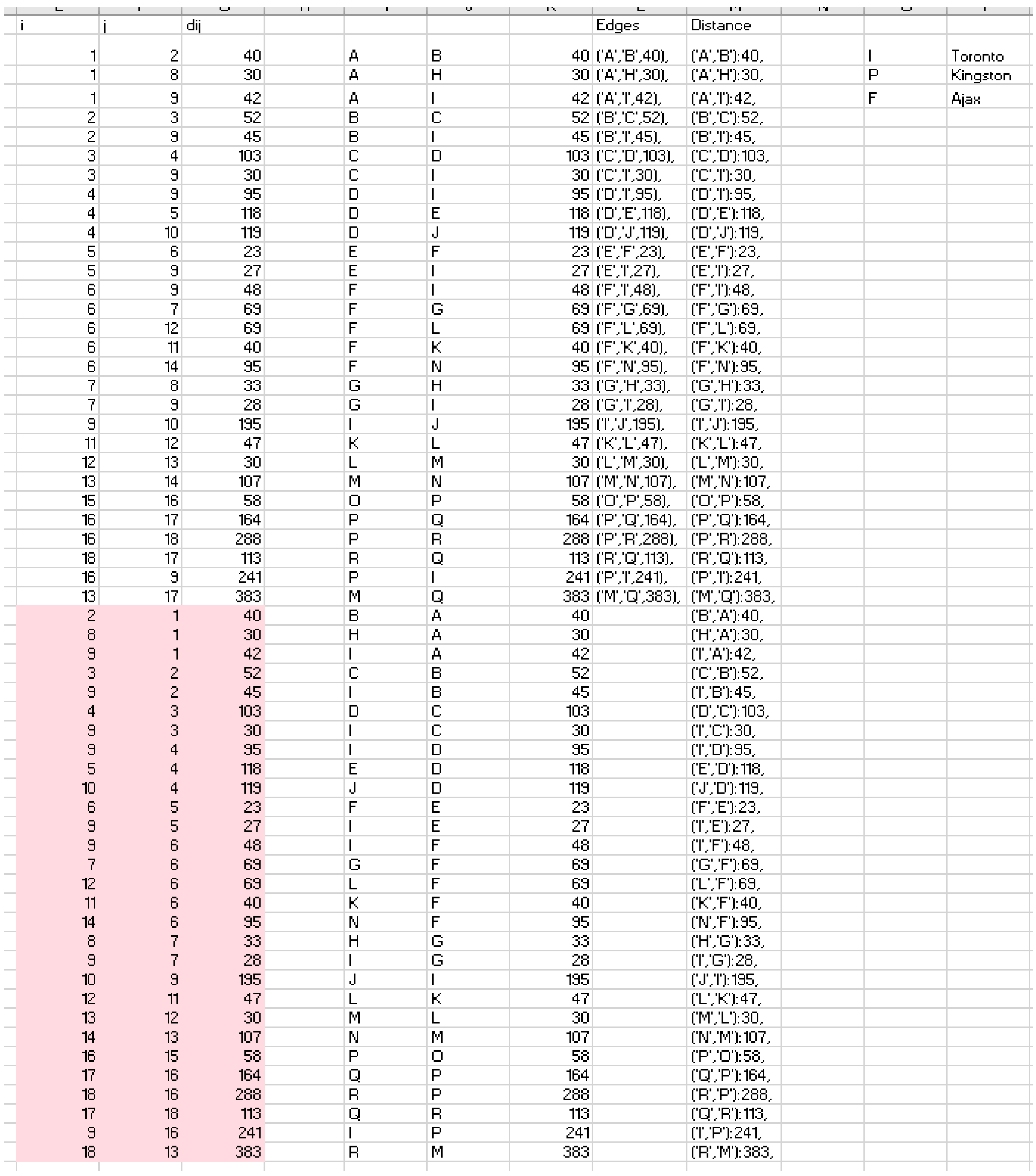
Collect run-time, cost and resulting stations for reporting at the end.

Appendix C: Data

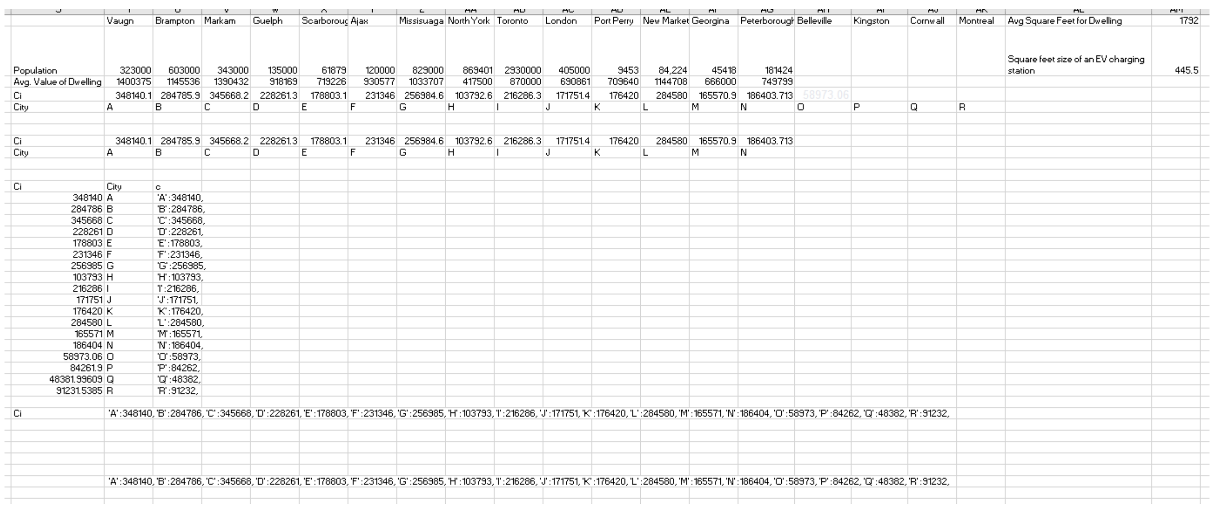
1. Instance 1: Toronto, Ontario to Montréal, Québec OD pair.



Cities			
Vaugn	1	A	c
Brampton	2	B	
Markam	3	C	
Guelph	4	D	
Scarborough	5	E	
Ajax	6	F	
Missisuaga	7	G	
North York	8	H	
Toronto	9	I	
London	10	J	
Port Perry	11	K	
New Market	12	L	
Georgina	13	M	
Peterborough	14	N	
Belleville	15	O	
Kingston	16	P	
Cornwall	17	Q	
Montreal	18	R	
Nodes:	'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R',		



i	j	dij					Edges	Distance			
1	2	40		A	B	40	('A','B',40),	('A','B'):40,		I	Toronto
1	8	30		A	H	30	('A','H',30),	('A','H'):30,		P	Kingston
1	9	42		A	I	42	('A','I',42),	('A','I'):42,		F	Ajax
2	3	52		B	C	52	('B','C',52),	('B','C'):52,			
2	9	45		B	I	45	('B','I',45),	('B','I'):45,			
3	4	103		C	D	103	('C','D',103),	('C','D'):103,			
3	9	30		C	I	30	('C','I',30),	('C','I'):30,			
4	9	95		D	I	95	('D','I',95),	('D','I'):95,			
4	5	118		D	E	118	('D','E',118),	('D','E'):118,			
4	10	119		D	J	119	('D','J',119),	('D','J'):119,			
5	6	23		E	F	23	('E','F',23),	('E','F'):23,			
5	9	27		E	I	27	('E','I',27),	('E','I'):27,			
6	9	48		F	I	48	('F','I',48),	('F','I'):48,			
6	7	69		F	G	69	('F','G',69),	('F','G'):69,			
6	12	69		F	L	69	('F','L',69),	('F','L'):69,			
6	11	40		F	K	40	('F','K',40),	('F','K'):40,			
6	14	95		F	N	95	('F','N',95),	('F','N'):95,			
7	8	33		G	H	33	('G','H',33),	('G','H'):33,			
7	9	28		G	I	28	('G','I',28),	('G','I'):28,			
9	10	195		I	J	195	('I','J',195),	('I','J'):195,			
11	12	47		K	L	47	('K','L',47),	('K','L'):47,			
12	13	30		L	M	30	('L','M',30),	('L','M'):30,			
13	14	107		M	N	107	('M','N',107),	('M','N'):107,			
15	16	58		O	P	58	('O','P',58),	('O','P'):58,			
16	17	164		P	Q	164	('P','Q',164),	('P','Q'):164,			
16	18	288		P	R	288	('P','R',288),	('P','R'):288,			
18	17	113		R	Q	113	('R','Q',113),	('R','Q'):113,			
16	9	241		P	I	241	('P','I',241),	('P','I'):241,			
13	17	383		M	Q	383	('M','Q',383),	('M','Q'):383,			
2	1	40		B	A	40		('B','A'):40,			
8	1	30		H	A	30		('H','A'):30,			
9	1	42		I	A	42		('I','A'):42,			
3	2	52		C	B	52		('C','B'):52,			
9	2	45		I	B	45		('I','B'):45,			
4	3	103		D	C	103		('D','C'):103,			
9	3	30		I	C	30		('I','C'):30,			
9	4	95		I	D	95		('I','D'):95,			
5	4	118		E	D	118		('E','D'):118,			
10	4	119		J	D	119		('J','D'):119,			
6	5	23		F	E	23		('F','E'):23,			
9	5	27		I	E	27		('I','E'):27,			
9	6	48		I	F	48		('I','F'):48,			
7	6	69		G	F	69		('G','F'):69,			
12	6	69		L	F	69		('L','F'):69,			
11	6	40		K	F	40		('K','F'):40,			
14	6	95		N	F	95		('N','F'):95,			
8	7	33		H	G	33		('H','G'):33,			
9	7	28		I	G	28		('I','G'):28,			
10	9	195		J	I	195		('J','I'):195,			
12	11	47		L	K	47		('L','K'):47,			
13	12	30		M	L	30		('M','L'):30,			
14	13	107		N	M	107		('N','M'):107,			
16	15	58		P	O	58		('P','O'):58,			
17	16	164		Q	P	164		('Q','P'):164,			
18	16	288		R	P	288		('R','P'):288,			
17	18	113		Q	R	113		('Q','R'):113,			
9	16	241		I	P	241		('I','P'):241,			
18	13	383		R	M	383		('R','M'):383,			

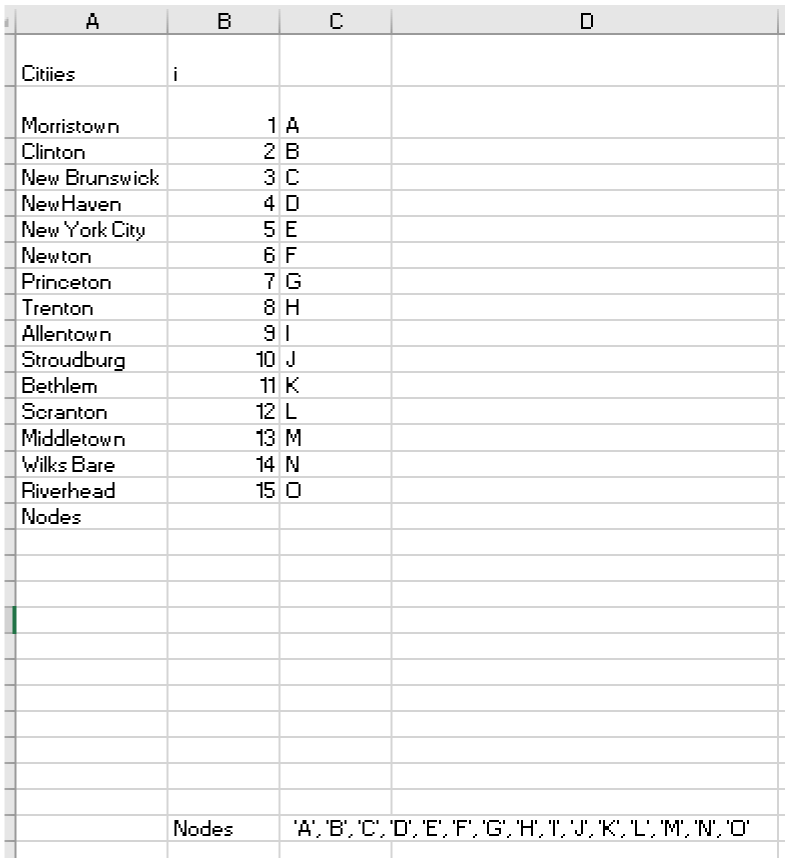


	Vaugn	Brampton	Markam	Guelph	Scarboroug	Ajax	Missisuaga	North York	Toronto	London	Port Perry	New Market	Georgina	Peterborough	Belleville	Kingston	Cornwall	Montreal	Avg Square Feet for Dwelling	1792
Population	323000	603000	343000	135000	61879	120000	829000	869401	2930000	405000	9453	84,224	45418	181424					Square feet size of an EV charging station	445.5
Avg. Value of Dwelling	1400375	1145536	1390432	918169	719226	930577	1033707	417500	870000	690861	709640	1144708	666000	749799						
Ci	348140.1	284785.9	345668.2	228261.3	178803.1	231346	256984.6	103792.6	216286.3	171751.4	176420	284580	165570.9	186403.713	58973.06					
City	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R		
Ci	348140.1	284785.9	345668.2	228261.3	178803.1	231346	256984.6	103792.6	216286.3	171751.4	176420	284580	165570.9	186403.713						
City	A	B	C	D	E	F	G	H	I	J	K	L	M	N						

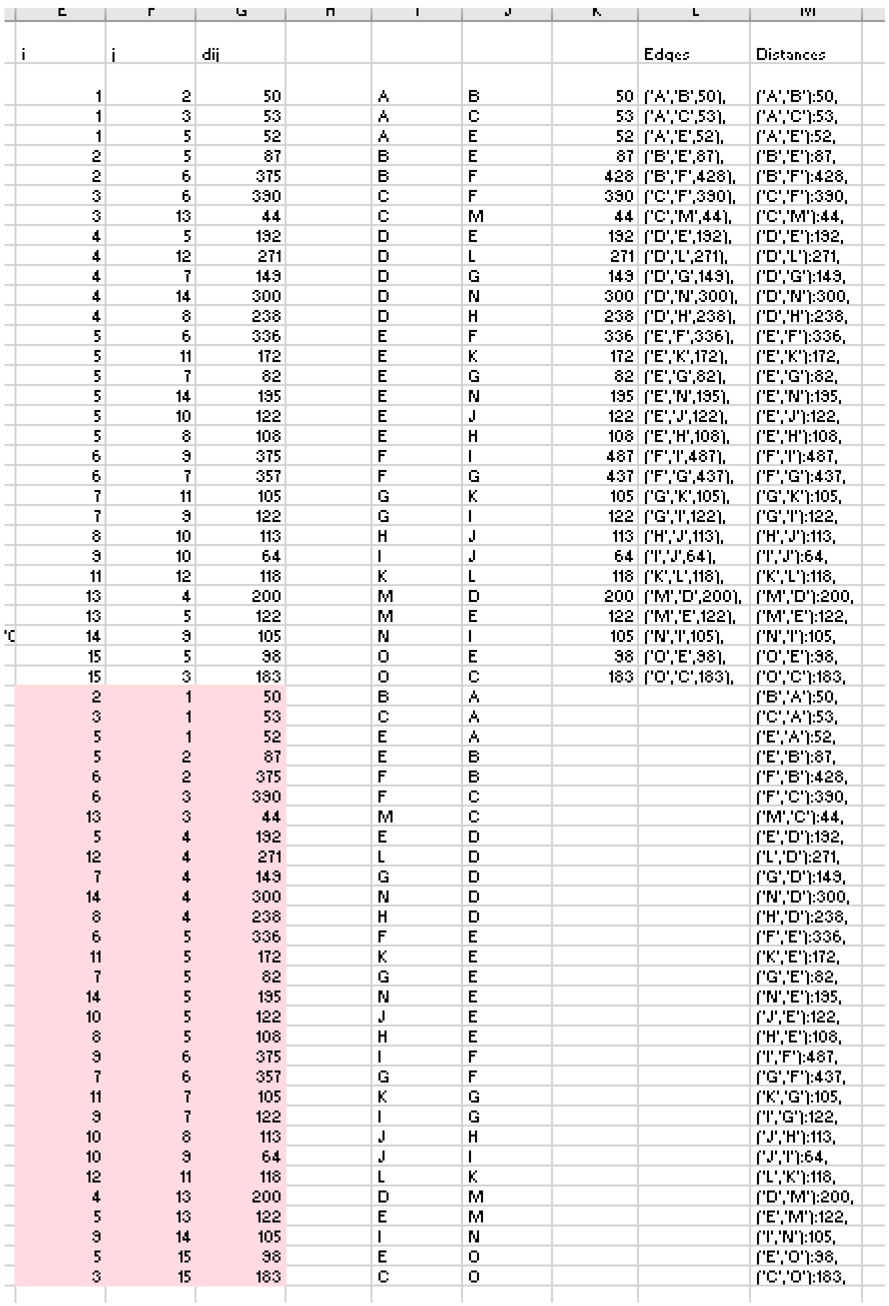
Ci	City	c
348140	A	'A':348140,
284786	B	'B':284786,
345668	C	'C':345668,
228261	D	'D':228261,
178803	E	'E':178803,
231346	F	'F':231346,
256985	G	'G':256985,
103793	H	'H':103793,
216286	I	'I':216286,
171751	J	'J':171751,
176420	K	'K':176420,
284580	L	'L':284580,
165571	M	'M':165571,
186404	N	'N':186404,
58973.06	O	'O':58973,
84261.9	P	'P':84262,
48381.99609	Q	'Q':48382,
91231.5385	R	'R':91232,

Ci	'A':348140, 'B':284786, 'C':345668, 'D':228261, 'E':178803, 'F':231346, 'G':256985, 'H':103793, 'I':216286, 'J':171751, 'K':176420, 'L':284580, 'M':165571, 'N':186404, 'O':58973, 'P':84262, 'Q':48382, 'R':91232,
	'A':348140, 'B':284786, 'C':345668, 'D':228261, 'E':178803, 'F':231346, 'G':256985, 'H':103793, 'I':216286, 'J':171751, 'K':176420, 'L':284580, 'M':165571, 'N':186404, 'O':58973, 'P':84262, 'Q':48382, 'R':91232,

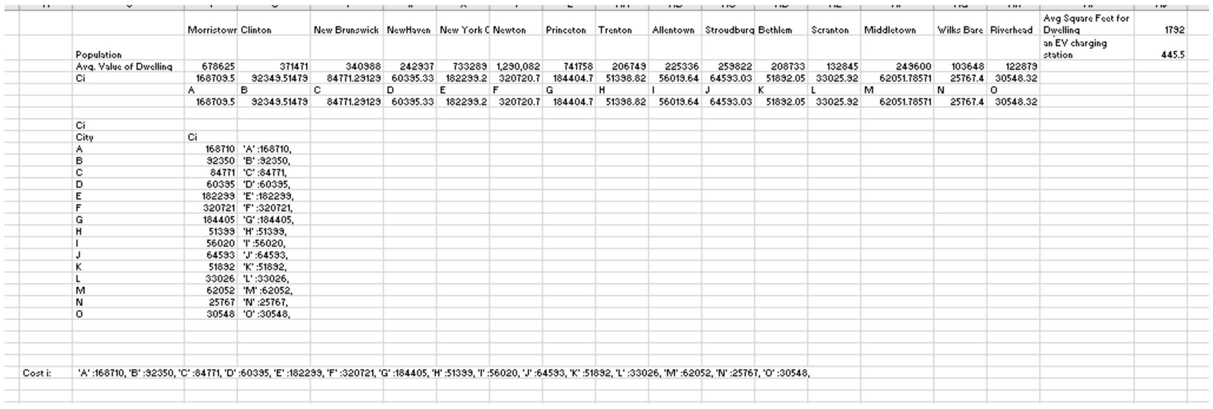
2. Instance 2: Allentown, Pennsylvania to New Haven, Connecticut OD pair.



A	B	C	D
Citiies	i		
Morristown	1	A	
Clinton	2	B	
New Brunswick	3	C	
NewHaven	4	D	
New York City	5	E	
Newton	6	F	
Princeton	7	G	
Trenton	8	H	
Allentown	9	I	
Stroudburg	10	J	
Bethlem	11	K	
Scranton	12	L	
Middletown	13	M	
Wilks Bare	14	N	
Riverhead	15	O	
Nodes			
	Nodes	'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N', 'O'	



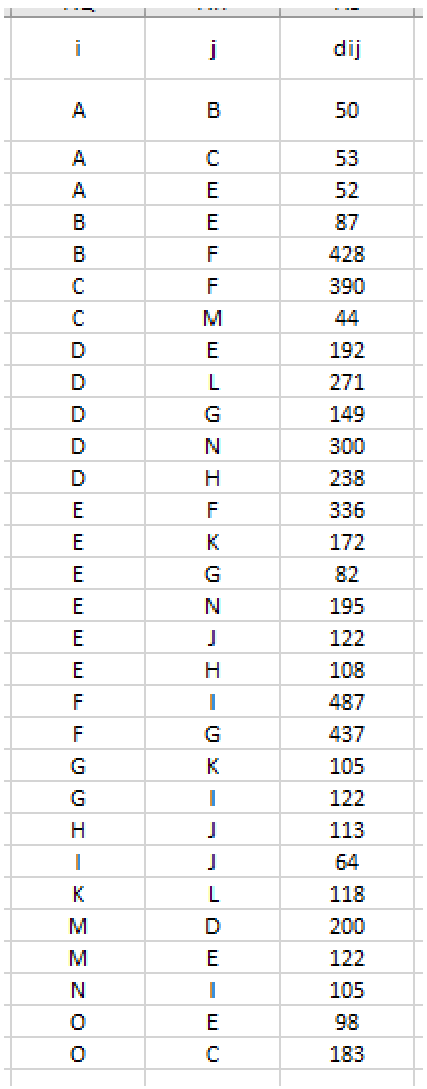
i	j	dij				Edges	Distances
1	2	50	A	B	50	('A','B',50),	('A','B'):50,
1	3	53	A	C	53	('A','C',53),	('A','C'):53,
1	5	52	A	E	52	('A','E',52),	('A','E'):52,
2	5	87	B	E	87	('B','E',87),	('B','E'):87,
2	6	375	B	F	428	('B','F',428),	('B','F'):428,
3	6	390	C	F	390	('C','F',390),	('C','F'):390,
3	13	44	C	M	44	('C','M',44),	('C','M'):44,
4	5	192	D	E	192	('D','E',192),	('D','E'):192,
4	12	271	D	L	271	('D','L',271),	('D','L'):271,
4	7	149	D	G	149	('D','G',149),	('D','G'):149,
4	14	300	D	N	300	('D','N',300),	('D','N'):300,
4	8	238	D	H	238	('D','H',238),	('D','H'):238,
5	6	336	E	F	336	('E','F',336),	('E','F'):336,
5	11	172	E	K	172	('E','K',172),	('E','K'):172,
5	7	82	E	G	82	('E','G',82),	('E','G'):82,
5	14	195	E	N	195	('E','N',195),	('E','N'):195,
5	10	122	E	J	122	('E','J',122),	('E','J'):122,
5	8	108	E	H	108	('E','H',108),	('E','H'):108,
6	9	375	F	I	487	('F','I',487),	('F','I'):487,
6	7	357	F	G	437	('F','G',437),	('F','G'):437,
7	11	105	G	K	105	('G','K',105),	('G','K'):105,
7	9	122	G	I	122	('G','I',122),	('G','I'):122,
8	10	113	H	J	113	('H','J',113),	('H','J'):113,
9	10	64	I	J	64	('I','J',64),	('I','J'):64,
11	12	118	K	L	118	('K','L',118),	('K','L'):118,
13	4	200	M	D	200	('M','D',200),	('M','D'):200,
13	5	122	M	E	122	('M','E',122),	('M','E'):122,
14	9	105	N	I	105	('N','I',105),	('N','I'):105,
15	5	98	O	E	98	('O','E',98),	('O','E'):98,
15	3	183	O	C	183	('O','C',183),	('O','C'):183,
2	1	50	B	A			('B','A'):50,
3	1	53	C	A			('C','A'):53,
5	1	52	E	A			('E','A'):52,
5	2	87	E	B			('E','B'):87,
6	2	375	F	B			('F','B'):428,
6	3	390	F	C			('F','C'):390,
13	3	44	M	C			('M','C'):44,
5	4	192	E	D			('E','D'):192,
12	4	271	L	D			('L','D'):271,
7	4	149	G	D			('G','D'):149,
14	4	300	N	D			('N','D'):300,
8	4	238	H	D			('H','D'):238,
6	5	336	F	E			('F','E'):336,
11	5	172	K	E			('K','E'):172,
7	5	82	G	E			('G','E'):82,
14	5	195	N	E			('N','E'):195,
10	5	122	J	E			('J','E'):122,
8	5	108	H	E			('H','E'):108,
9	6	375	I	F			('I','F'):487,
7	6	357	G	F			('G','F'):437,
11	7	105	K	G			('K','G'):105,
9	7	122	I	G			('I','G'):122,
10	8	113	J	H			('J','H'):113,
10	9	64	J	I			('J','I'):64,
12	11	118	L	K			('L','K'):118,
4	13	200	D	M			('D','M'):200,
5	13	122	E	M			('E','M'):122,
9	14	105	I	N			('I','N'):105,
5	15	98	E	O			('E','O'):98,
3	15	183	C	O			('C','O'):183,



	Morristowr	Clinton	New Brunswick	NewHaven	New York C	Newton	Princeton	Trenton	Allentown	Stroudburg	Bethlem	Scranton	Middletown	Wilks Bare	Riverhead	Avg Square Feet for Dwelling	1792
Population																an EV charging station	445.5
Avg. Value of Dwelling	678625	371471	340988	242937	733289	1,290,082	741758	206749	225336	259822	208733	132845	249600	103648	122879		
Ci	168709.5	92349.51479	84771.29129	60395.33	182299.2	320720.7	184404.7	51398.82	56019.64	64593.03	51892.05	33025.92	62051.78571	25767.4	30548.32		
	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O		
	168709.5	92349.51479	84771.29129	60395.33	182299.2	320720.7	184404.7	51398.82	56019.64	64593.03	51892.05	33025.92	62051.78571	25767.4	30548.32		

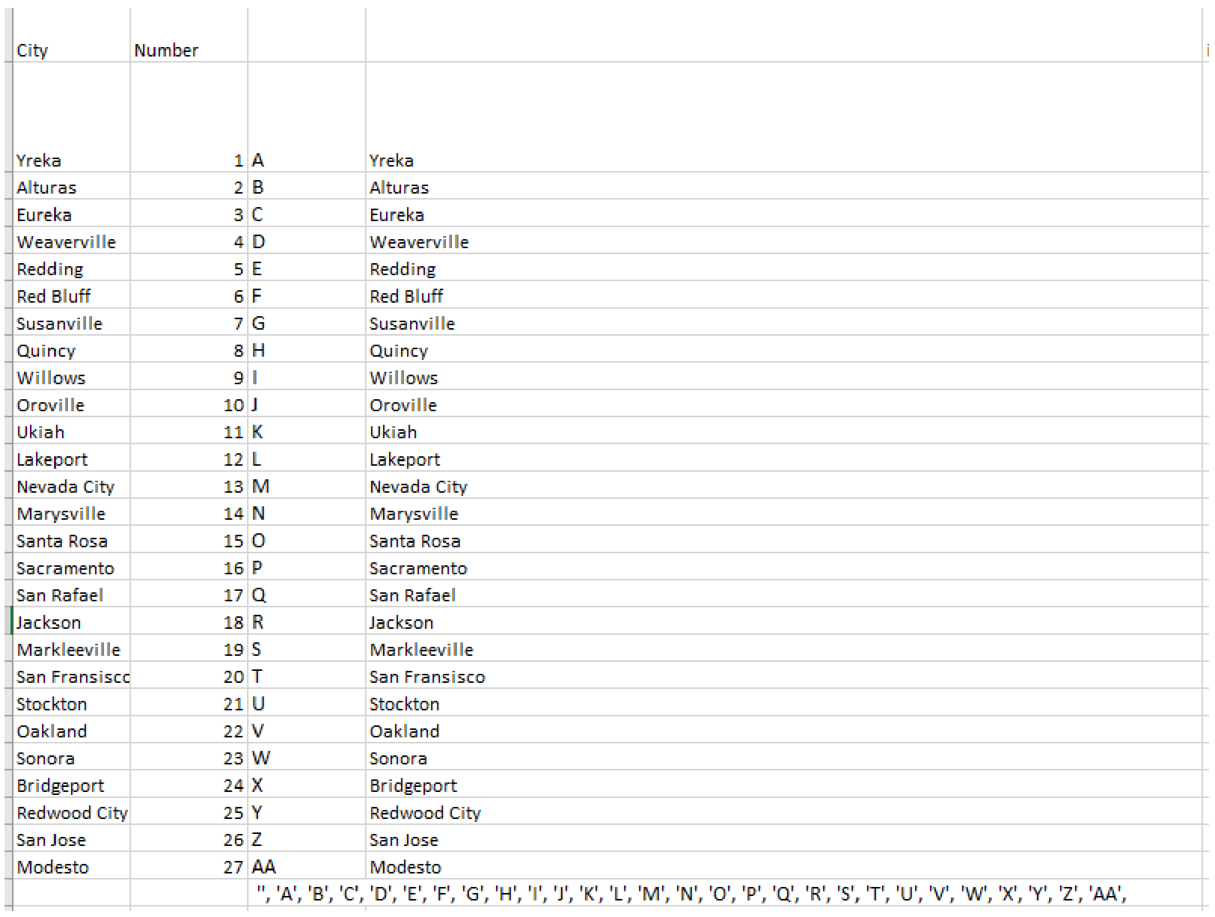
Ci		
City	Ci	
A	168710	'A' :168710,
B	92350	'B' :92350,
C	84771	'C' :84771,
D	60395	'D' :60395,
E	182299	'E' :182299,
F	320721	'F' :320721,
G	184405	'G' :184405,
H	51399	'H' :51399,
I	56020	'I' :56020,
J	64593	'J' :64593,
K	51892	'K' :51892,
L	33026	'L' :33026,
M	62052	'M' :62052,
N	25767	'N' :25767,
O	30548	'O' :30548,

Cost i: 'A' :168710, 'B' :92350, 'C' :84771, 'D' :60395, 'E' :182299, 'F' :320721, 'G' :184405, 'H' :51399, 'I' :56020, 'J' :64593, 'K' :51892, 'L' :33026, 'M' :62052, 'N' :25767, 'O' :30548,

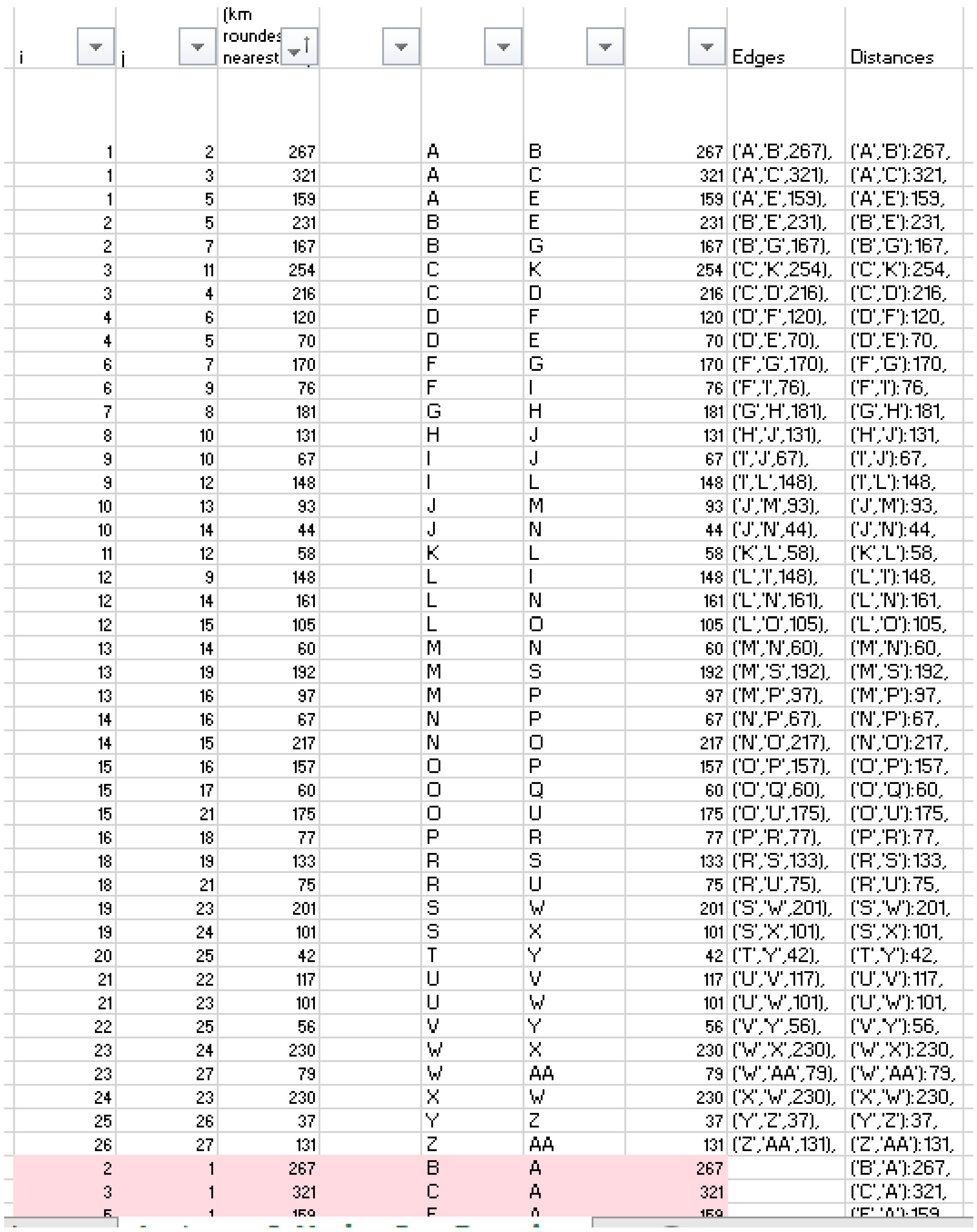


i	j	dij
A	B	50
A	C	53
A	E	52
B	E	87
B	F	428
C	F	390
C	M	44
D	E	192
D	L	271
D	G	149
D	N	300
D	H	238
E	F	336
E	K	172
E	G	82
E	N	195
E	J	122
E	H	108
F	I	487
F	G	437
G	K	105
G	I	122
H	J	113
I	J	64
K	L	118
M	D	200
M	E	122
N	I	105
O	E	98
O	C	183

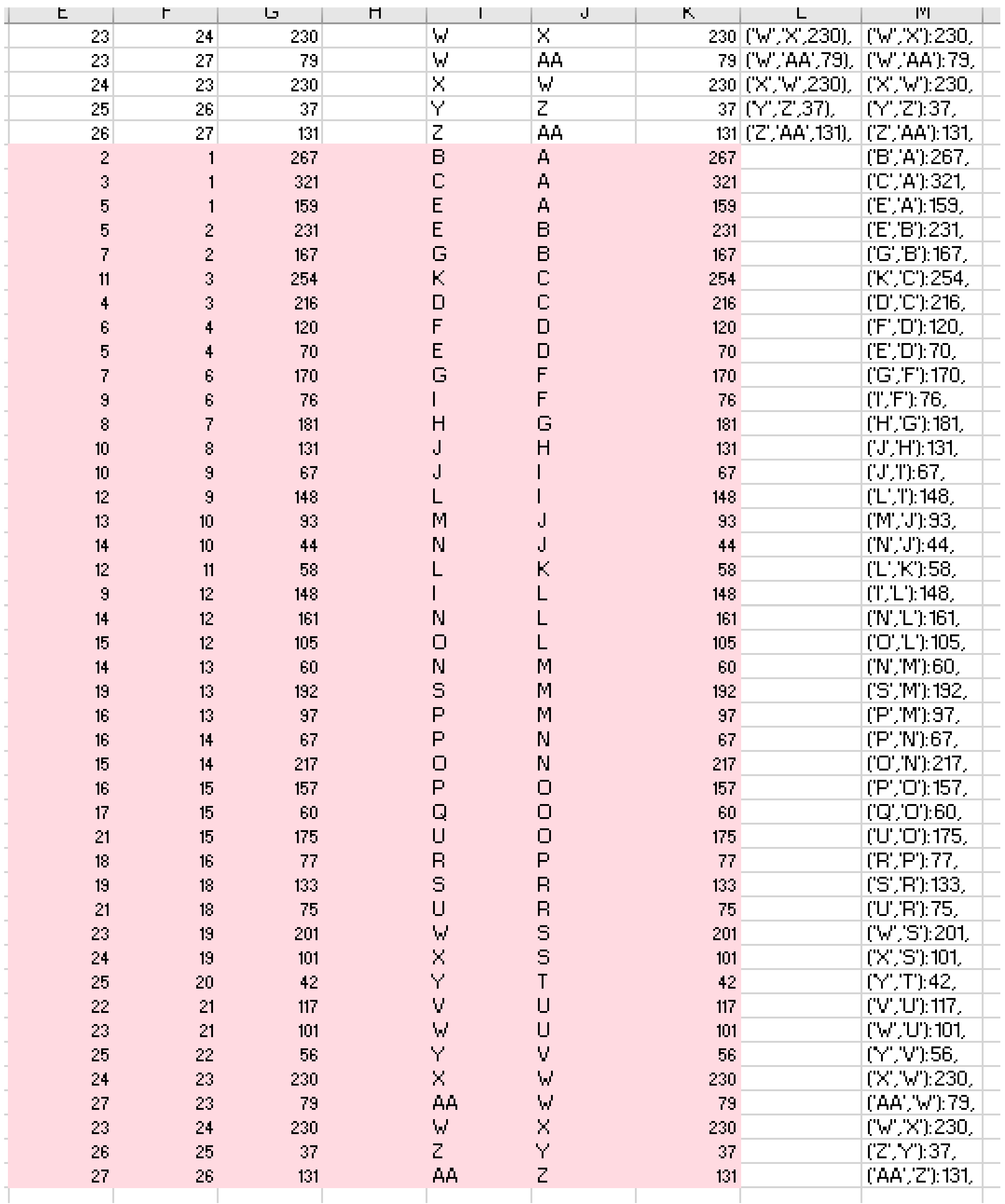
3. Instance 3: Yreka, California to San Francisco, California OD pair.



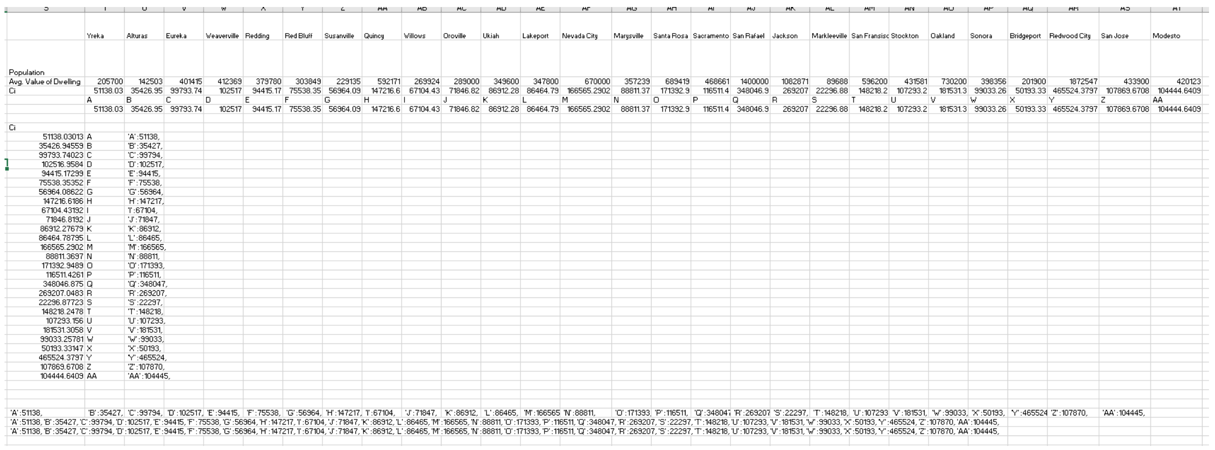
City	Number		
Yreka	1	A	Yreka
Alturas	2	B	Alturas
Eureka	3	C	Eureka
Weaverville	4	D	Weaverville
Redding	5	E	Redding
Red Bluff	6	F	Red Bluff
Susanville	7	G	Susanville
Quincy	8	H	Quincy
Willows	9	I	Willows
Oroville	10	J	Oroville
Ukiah	11	K	Ukiah
Lakeport	12	L	Lakeport
Nevada City	13	M	Nevada City
Marysville	14	N	Marysville
Santa Rosa	15	O	Santa Rosa
Sacramento	16	P	Sacramento
San Rafael	17	Q	San Rafael
Jackson	18	R	Jackson
Markleeville	19	S	Markleeville
San Fransiscc	20	T	San Fransisco
Stockton	21	U	Stockton
Oakland	22	V	Oakland
Sonora	23	W	Sonora
Bridgeport	24	X	Bridgeport
Redwood City	25	Y	Redwood City
San Jose	26	Z	San Jose
Modesto	27	AA	Modesto
		'', 'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X', 'Y', 'Z', 'AA',	



i	j	(km rounded nearest					Edges	Distances
1	2	267		A	B	267	('A','B',267),	('A','B'):267,
1	3	321		A	C	321	('A','C',321),	('A','C'):321,
1	5	159		A	E	159	('A','E',159),	('A','E'):159,
2	5	231		B	E	231	('B','E',231),	('B','E'):231,
2	7	167		B	G	167	('B','G',167),	('B','G'):167,
3	11	254		C	K	254	('C','K',254),	('C','K'):254,
3	4	216		C	D	216	('C','D',216),	('C','D'):216,
4	6	120		D	F	120	('D','F',120),	('D','F'):120,
4	5	70		D	E	70	('D','E',70),	('D','E'):70,
6	7	170		F	G	170	('F','G',170),	('F','G'):170,
6	9	76		F	I	76	('F','I',76),	('F','I'):76,
7	8	181		G	H	181	('G','H',181),	('G','H'):181,
8	10	131		H	J	131	('H','J',131),	('H','J'):131,
9	10	67		I	J	67	('I','J',67),	('I','J'):67,
9	12	148		I	L	148	('I','L',148),	('I','L'):148,
10	13	93		J	M	93	('J','M',93),	('J','M'):93,
10	14	44		J	N	44	('J','N',44),	('J','N'):44,
11	12	58		K	L	58	('K','L',58),	('K','L'):58,
12	9	148		L	I	148	('L','I',148),	('L','I'):148,
12	14	161		L	N	161	('L','N',161),	('L','N'):161,
12	15	105		L	O	105	('L','O',105),	('L','O'):105,
13	14	60		M	N	60	('M','N',60),	('M','N'):60,
13	19	192		M	S	192	('M','S',192),	('M','S'):192,
13	16	97		M	P	97	('M','P',97),	('M','P'):97,
14	16	67		N	P	67	('N','P',67),	('N','P'):67,
14	15	217		N	O	217	('N','O',217),	('N','O'):217,
15	16	157		O	P	157	('O','P',157),	('O','P'):157,
15	17	60		O	Q	60	('O','Q',60),	('O','Q'):60,
15	21	175		O	U	175	('O','U',175),	('O','U'):175,
16	18	77		P	R	77	('P','R',77),	('P','R'):77,
18	19	133		R	S	133	('R','S',133),	('R','S'):133,
18	21	75		R	U	75	('R','U',75),	('R','U'):75,
19	23	201		S	W	201	('S','W',201),	('S','W'):201,
19	24	101		S	X	101	('S','X',101),	('S','X'):101,
20	25	42		T	Y	42	('T','Y',42),	('T','Y'):42,
21	22	117		U	V	117	('U','V',117),	('U','V'):117,
21	23	101		U	W	101	('U','W',101),	('U','W'):101,
22	25	56		V	Y	56	('V','Y',56),	('V','Y'):56,
23	24	230		W	X	230	('W','X',230),	('W','X'):230,
23	27	79		W	AA	79	('W','AA',79),	('W','AA'):79,
24	23	230		X	W	230	('X','W',230),	('X','W'):230,
25	26	37		Y	Z	37	('Y','Z',37),	('Y','Z'):37,
26	27	131		Z	AA	131	('Z','AA',131),	('Z','AA'):131,
2	1	267		B	A	267		('B','A'):267,
3	1	321		C	A	321		('C','A'):321,
5	1	159		E	A	159		('E','A'):159,



E	F	G	H	I	J	K	L	M
23	24	230		W	X	230	('W','X',230),	('W','X'):230,
23	27	79		W	AA	79	('W','AA',79),	('W','AA'):79,
24	23	230		X	W	230	('X','W',230),	('X','W'):230,
25	26	37		Y	Z	37	('Y','Z',37),	('Y','Z'):37,
26	27	131		Z	AA	131	('Z','AA',131),	('Z','AA'):131,
2	1	267		B	A	267		('B','A'):267,
3	1	321		C	A	321		('C','A'):321,
5	1	159		E	A	159		('E','A'):159,
5	2	231		E	B	231		('E','B'):231,
7	2	167		G	B	167		('G','B'):167,
11	3	254		K	C	254		('K','C'):254,
4	3	216		D	C	216		('D','C'):216,
6	4	120		F	D	120		('F','D'):120,
5	4	70		E	D	70		('E','D'):70,
7	6	170		G	F	170		('G','F'):170,
9	6	76		I	F	76		('I','F'):76,
8	7	181		H	G	181		('H','G'):181,
10	8	131		J	H	131		('J','H'):131,
10	9	67		J	I	67		('J','I'):67,
12	9	148		L	I	148		('L','I'):148,
13	10	93		M	J	93		('M','J'):93,
14	10	44		N	J	44		('N','J'):44,
12	11	58		L	K	58		('L','K'):58,
9	12	148		I	L	148		('I','L'):148,
14	12	161		N	L	161		('N','L'):161,
15	12	105		O	L	105		('O','L'):105,
14	13	60		N	M	60		('N','M'):60,
19	13	192		S	M	192		('S','M'):192,
16	13	97		P	M	97		('P','M'):97,
16	14	67		P	N	67		('P','N'):67,
15	14	217		O	N	217		('O','N'):217,
16	15	157		P	O	157		('P','O'):157,
17	15	60		Q	O	60		('Q','O'):60,
21	15	175		U	O	175		('U','O'):175,
18	16	77		R	P	77		('R','P'):77,
19	18	133		S	R	133		('S','R'):133,
21	18	75		U	R	75		('U','R'):75,
23	19	201		W	S	201		('W','S'):201,
24	19	101		X	S	101		('X','S'):101,
25	20	42		Y	T	42		('Y','T'):42,
22	21	117		V	U	117		('V','U'):117,
23	21	101		W	U	101		('W','U'):101,
25	22	56		Y	V	56		('Y','V'):56,
24	23	230		X	W	230		('X','W'):230,
27	23	79		AA	W	79		('AA','W'):79,
23	24	230		W	X	230		('W','X'):230,
26	25	37		Z	Y	37		('Z','Y'):37,
27	26	131		AA	Z	131		('AA','Z'):131,



S	T	U	V	W	X	Y	Z	AA	AB	AC	AD	AE	AF
	Yreka	Alturas	Eureka	Weaverville	Redding	Red Bluff	Susanville	Quincy	Willows	Oroville	Ukiah	Lakeport	Nevada City
Population													
Avg. Value of Dwelling	205700	142503	401415	412369	379780	303849	229135	592171	269924	289000	349600	347800	670000
Ci	51138.03	35426.95	99793.74	102517	94415.17	75538.35	56964.09	147216.6	67104.43	71846.82	86912.28	86464.79	166565.2902
	A	B	C	D	E	F	G	H	I	J	K	L	M
	51138.03	35426.95	99793.74	102517	94415.17	75538.35	56964.09	147216.6	67104.43	71846.82	86912.28	86464.79	166565.2902

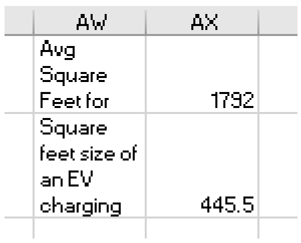
S	AG	AH	AI	AJ	AK	AL	AM	AN	AO	AP	AQ	AR	AS	AT
	Marysville	Santa Rosa	Sacramento	San Rafael	Jackson	Markleeville	San Fransisc	Stockton	Oakland	Sonora	Bridgeport	Redwood City	San Jose	Modesto
Population														
Avg. Value of Dwelling	357239	689419	468661	1400000	1082871	89688	596200	431581	730200	398356	201900	1872547	433900	420123
Ci	88811.37	171392.9	116511.4	348046.9	269207	22296.88	148218.2	107293.2	181531.3	99033.26	50193.33	465524.3797	107869.6708	104444.6409
	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	AA
	88811.37	171392.9	116511.4	348046.9	269207	22296.88	148218.2	107293.2	181531.3	99033.26	50193.33	465524.3797	107869.6708	104444.6409

S	T	U
Ci		
51138.03013	A	'A':51138,
35426.94559	B	'B':35427,
99793.74023	C	'C':99794,
102516.9584	D	'D':102517,
94415.17299	E	'E':94415,
75538.35352	F	'F':75538,
56964.08622	G	'G':56964,
147216.6186	H	'H':147217,
67104.43192	I	'I':67104,
71846.8192	J	'J':71847,
86912.27679	K	'K':86912,
86464.78795	L	'L':86465,
166565.2902	M	'M':166565,
88811.3697	N	'N':88811,
171392.9489	O	'O':171393,
116511.4261	P	'P':116511,
348046.875	Q	'Q':348047,
269207.0483	R	'R':269207,
22296.87723	S	'S':22297,
148218.2478	T	'T':148218,
107293.156	U	'U':107293,
181531.3058	V	'V':181531,
99033.25781	W	'W':99033,
50193.33147	X	'X':50193,
465524.3797	Y	'Y':465524,
107869.6708	Z	'Z':107870,
104444.6409	AA	'AA':104445,

'A':51138, 'B':35427, 'C':99794, 'D':102517, 'E':94415, 'F':75538, 'G':56964, 'H':147217, 'I':67104, 'J':71847, 'K':86912, 'L':86465, 'M':166565 'N':88811, 'O':171393 'P':116511, 'Q':348047 'R':269207 'S':22297, 'T':148218, 'U':107293 'V':181531, 'W':99033, 'X':50193, 'Y':465524 'Z':107870, 'AA':104445,

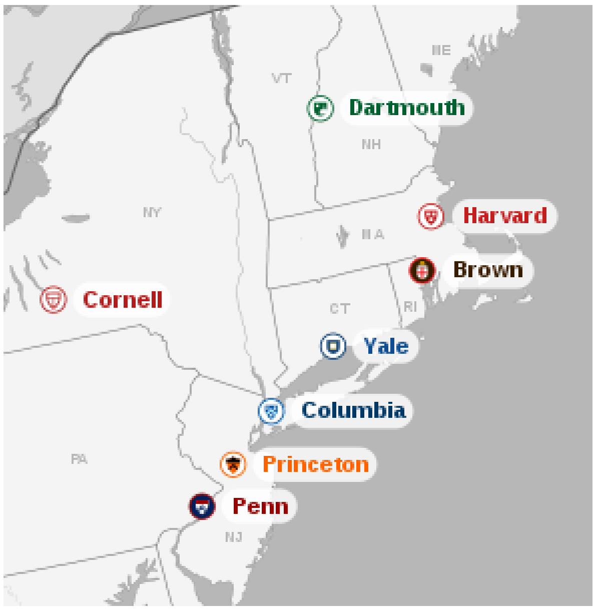
'A':51138, 'B':35427, 'C':99794, 'D':102517, 'E':94415, 'F':75538, 'G':56964, 'H':147217, 'I':67104, 'J':71847, 'K':86912, 'L':86465, 'M':166565, 'N':88811, 'O':171393, 'P':116511, 'Q':348047, 'R':269207, 'S':22297, 'T':148218, 'U':107293, 'V':181531, 'W':99033, 'X':50193, 'Y':465524, 'Z':107870, 'AA':104445,

'A':51138, 'B':35427, 'C':99794, 'D':102517, 'E':94415, 'F':75538, 'G':56964, 'H':147217, 'I':67104, 'J':71847, 'K':86912, 'L':86465, 'M':166565, 'N':88811, 'O':171393, 'P':116511, 'Q':348047, 'R':269207, 'S':22297, 'T':148218, 'U':107293, 'V':181531, 'W':99033, 'X':50193, 'Y':465524, 'Z':107870, 'AA':104445,



AW	AX
Avg Square Feet for	1792
Square feet size of an EV charging	445.5

Appendix E: Ivy Loop

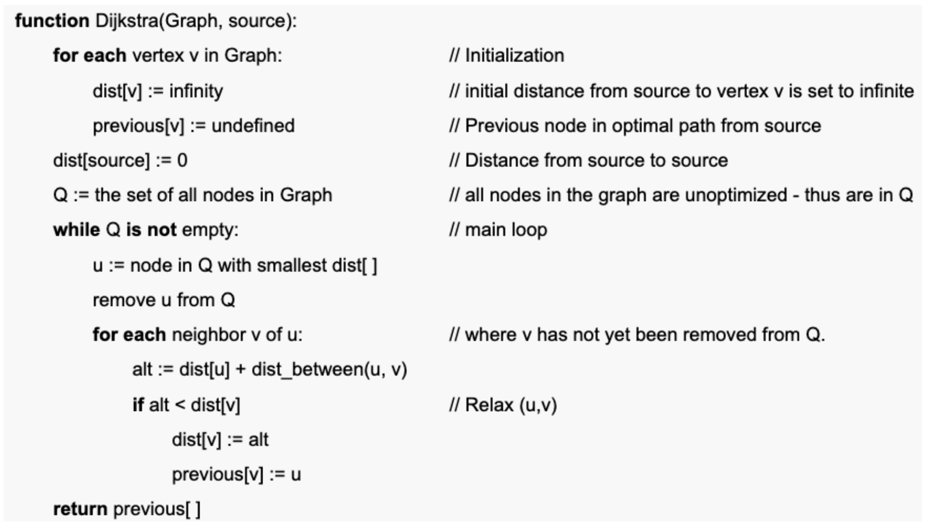


Appendix F: Assumptions

The assumptions made to implement this algorithm are as follows:

- We assume drivers travelling from city i to city j will always take the shortest route available. Realistically, drivers may prefer cheaper routes, more scenic routes, etc.
- We assume a linear relationship between battery consumption. Realistically, the nature of the route taken would impact the battery consumed differently.
- We assume all vehicles start their journeys with a fully charged vehicle. This is a very realistic assumption as over 90% of EV owners charge their vehicles at home [2]. However, as the problem is centred around maximizing journey completability, EV owners can forget to charge their car, not let their car charge for a sufficient amount of time, etc.
- We assume drivers will always charge their vehicles fully. Realistically, drivers may not do this for financial reasons, time constraints or other circumstantial reasons.
- We assume that all EV charging will have unlimited capacity and will be able to service all visitors. Realistically, only a limited number of EV charging stalls can be built in an EV charging station due to budget constraints, local laws, size constraints, etc. Furthermore, it is not realistic to expect that drivers will wait for extended periods of time in a queue to charge their car.
- We assume that all stations will be of the same size. Realistically, stations are built in accordance with local demand, budgets, etc.
- We assume all cities in the OD pair are less than the EV range R apart.
- We assume that the average range of a Tesla Model Y EV is an accurate estimator for the range of the average EV.
- We assume that the value of the average value of dwellings in a city obtained from the website Zillow.com, please refer to reference [5], divided by the average size of a dwelling is an accurate estimate for the cost per square foot in the city. Realistically, the legislation is different in different locations in addition to building preferences, living preferences of residences, etc. Furthermore, locations within cities have different costs per square feet and therefore, it is not realistic to generalize the cost per square foot as such.
- We assume that the distance between the geographical centres of the two cities is a good estimate of the distance between said cities.

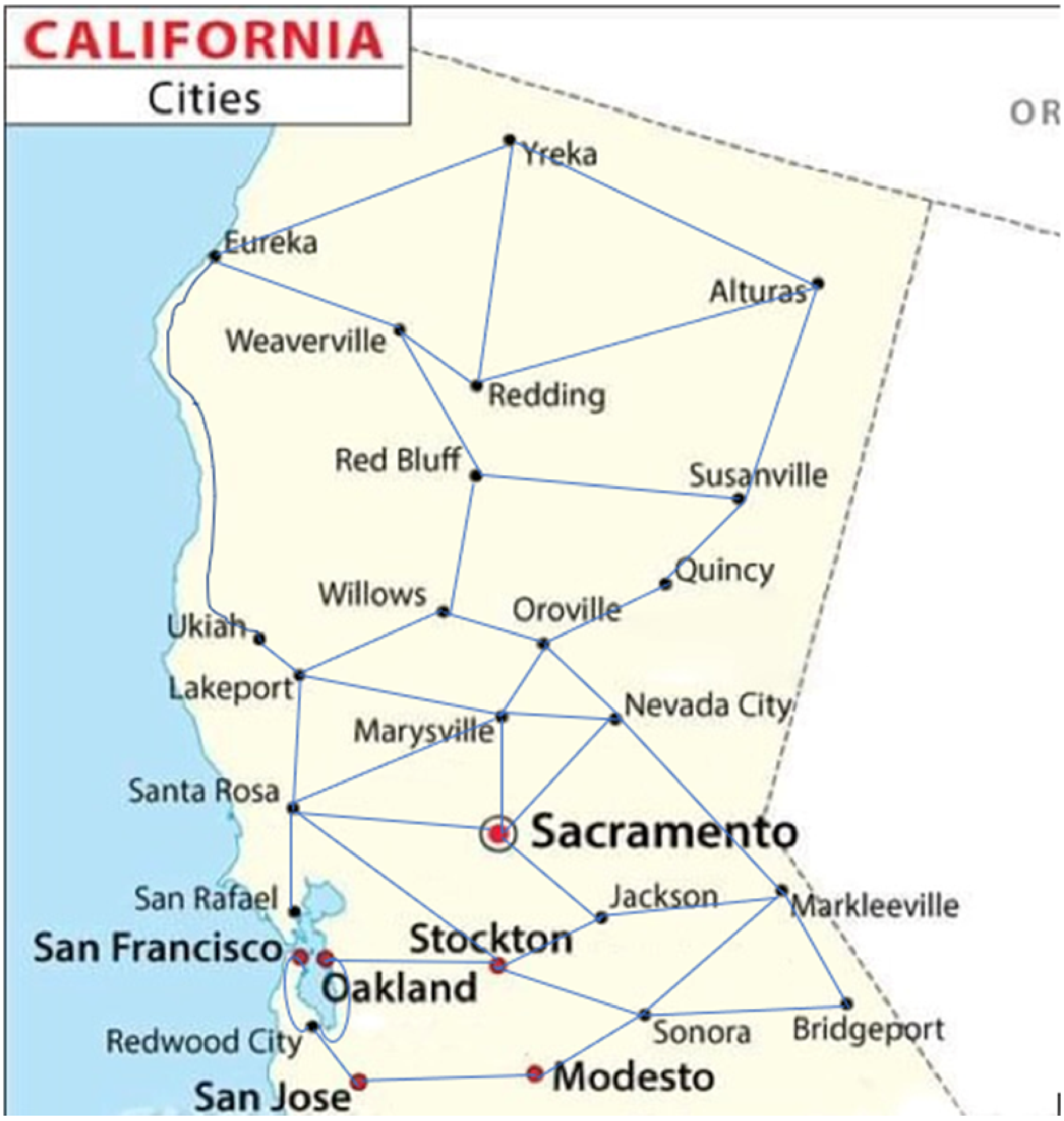
Appendix G: Pseudocode of the Metaheuristic



```
function Dijkstra(Graph, source):
    for each vertex v in Graph:                 // Initialization
        dist[v] := infinity                     // initial distance from source to vertex v is set to infinite
        previous[v] := undefined                // Previous node in optimal path from source
    dist[source] := 0                           // Distance from source to source
    Q := the set of all nodes in Graph          // all nodes in the graph are unoptimized - thus are in Q
    while Q is not empty:                       // main loop
        u := node in Q with smallest dist[ ]
        remove u from Q
        for each neighbor v of u:               // where v has not yet been removed from Q.
            alt := dist[u] + dist_between(u, v)
            if alt < dist[v]                    // Relax (u,v)
                dist[v] := alt
                previous[v] := u
    return previous[ ]
```

```
function distance(shortest_path):
    For i in range(length of the shortest_path - 1):
            Temp = distance between current node to next node
            Distance = Distance + Temp
    Return Distance


Function get_Can_list (shortest_path):
    While i in range(length of shortest_path - 1)
            arc =  distance between current to next code
            EV_range = EV_range - arc
            If EV_range == 0:
                    Add next node i+1 to the solution list
                    Set EV_range = 400 - arc
            If EV_range < 0:
                    Add current node i to the solution list
                    Set EV_range = 400 - arc
                    i++
            Else:
                    #do nothing
                    i++
    Return Solution list


    # combining the functions:
     nodes = all nodes in the network
     For i in range (len(nodes))
                  Start = nodes[i]
                  For i in range (len(nodes))
                     if nodes[i] != start:
                        end = nodes[i]
                        shortest = dijsktra(graph, start, end)
                        #check if shortest path is longer than the range of EV then add to list
                        if distance(shortest) > R:
                          Add the path to shortest path array
                          Get station candidates of that shortest path
unique_candidate = set(Candidate)

For s in unique_candidate:
     Collect all the solution lists in a list and then convert it to a set.
      Sum cost in the unique_candidate list

Collect run-time, cost and resulting stations for reporting at the end.

Function simulated_annealing(path list, solution list)
For s in solution dictionary
     ind = index of s in path list
     ind_sol = index of s in solution list
distance= distance between s and the element in the index before s in path list (prev)
If s is the last element in solution list:
     Way1  = dijsktra map between s and prev
     For h = 0 to length of Way1
                  distance= distance + distance between s and prev
     Else
     Way  = dijsktra map between s and next element in solution list(next)
     For h = 0 to length of Way1
                  distance= distance + distance between s and next
     If distance < EV Range and cost of prev < cost of s:
                  Add to anneal_candidate list
     If distance < EV Range and cost of prev > cost of s:
        Zn = Cost of prev
        Zc = Cost of s
        T = 0.4 Zc
        X = Zn-Zc/T
        Probability = exp ^ X
        If probability > 0.95
                Add prev to anneal_ candidate list
        Else
                Add s to anneal_candidate list
Else
        Add s to anneal_candidate list
Return the anneal_candidate list

For g in 0 - length of  paths dictionary
        Add candidates to annealing candidates by calling the simulated_annealing(path
dictionary.get(g),solutions dictionary.get(g))
For each element in the annealing_candidates add them to a set to remove duplicates
Print the unique solution
Calculate cost by getting the cost from the cost dictionary for key = each element of the set and adding
them together.

Print out the solution and the total cost.
```

Appendix H: North California Network


CALIFORNIA
Cities
OR
Yreka
Eureka
Alturas
Weaverville
Redding
Red Bluff
Susanville
Quincy
Willows
Oroville
Ukiah
Lakeport
Nevada City
Marysville
Santa Rosa
Sacramento
San Rafael
Jackson
Markleeville
San Francisco
Stockton
Oakland
Redwood City
Sonora
Bridgeport
San Jose
Modesto