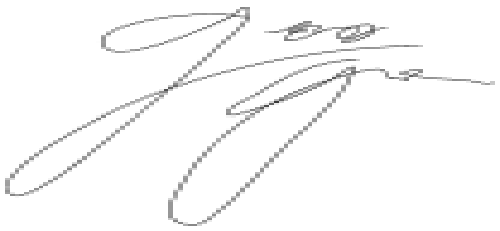
CEE201 Group 9 - Project Analysis and Report

Group Members:
Molly G, Jeff G, Jack G, Gabriela G

Name	Contributions	Signature
Molly	Analyzed the results of the code and wrote analyses based on findings for Tasks 1, 2, and 3. Created the figure for Task 2. Revised and edited report.	
Jeff	Wrote python code for tasks 1,2, worked on task 4 with Jack and annotated .ipynb with comments to ensure clarity.	
Jack	Wrote python code for task 3, worked with Jeff to write code for task 4, added comments for clarity	
Gabriela	Checked over code to ensure it was running properly, cross checked information with online resources, wrote for report, researched methods to complete the tasks.	

Executive Summary

Our team was given a scenario where attendees of a University of Illinois home basketball game leave the stadium by car using the surrounding roadways. Using real location information of the stadium, parking lots, and nearby roads, our team was tasked with analyzing traffic flow in the system, studying both inflow and outflow. The project showed how changes in infrastructure affect overall network performance.

To complete this analysis, our team designed a series of linear programming models in Python. We represented the road network as a series of connected nodes, including parking lots, intersections, gates, and exits, and used Google Maps to estimate the distances between these locations. Each parking lot was assigned an expected number of departing cars specific to our scenario, and decision variables were used to represent how those cars could move through the network. For each task, we adapted the same setup and network structure with a few changes to fit the scenario that each task proposed. Details and results for each task are presented in the technical section of this report.

In Task 1, our optimization results demonstrated that under the baseline assumption of equal flow to each exit, traffic concentrated on a small number of key links while others were lightly used; and when we altered the directional preference so that more trips went to the north and east, total vehicle miles traveled increased, and the main bottleneck shifted to a different internal link. Modifying the parking supply also produced clear effects: opening an additional lot near the core redistributed the flows and reduced the pressure on the previous worst link,

whereas closing a high-impact lot (X2) substantially reduced the total car-miles in the study area by limiting how many vehicles could park and circulate there. In Task 2, increasing the trip demand at one of the largest lots by 20% raised the minimum total car-miles by about 7%, pushed even more traffic onto an already heavily used corridor, and reduced the flows on a more weakly connected link, illustrating how growth at a single major generator may intensify congestion in specific parts of the network. In Task 3, adding lane-based capacity constraints showed that only about 2,500 cars can exit the area in 30 minutes with the current configuration, with a small set of exit links operating at capacity and several links near certain lots carrying relatively low volumes, suggesting that targeted operational measures there - such as traffic coordinators - could help unlock additional throughput. Finally, in Task 4, we found that adding one new lane to a carefully chosen exit link increases 30-minute throughput and yields a positive net economic benefit, but adding a second lane on the same link produces smaller gains that are not cost-effective, illustrating the diminishing return from continued lane expansion and the importance of targeted rather than uniform capacity investments.

From this project, we gained a deeper understanding of the interconnectedness of road systems, and how even small planning decisions can significantly affect traffic efficiency. We learned that traffic flow depends on a network of elements working together. For example, road length and lane count quickly influence how vehicles can move, while car capacity and parking availability affect how often bottlenecks form. We also saw how the placement of intersections, the spacing between roads, and the overall layout of an area play major roles in determining how long it takes for cars to clear a congested zone. This project made us realize that effective traffic

design requires thinking about the system as a whole, not just isolated components, and that thoughtful planning can dramatically improve or worsen real-world travel times.

Technical Section

Below is the breakdown of the foundation of our analysis. **Figure 1** displays how we defined different nodes for our linear programming model. **Figure 2** shows the breakdown of cars per lot specific to our group. Attached at the bottom is the code we used for each task to define the variables and distances of our linear programming model.

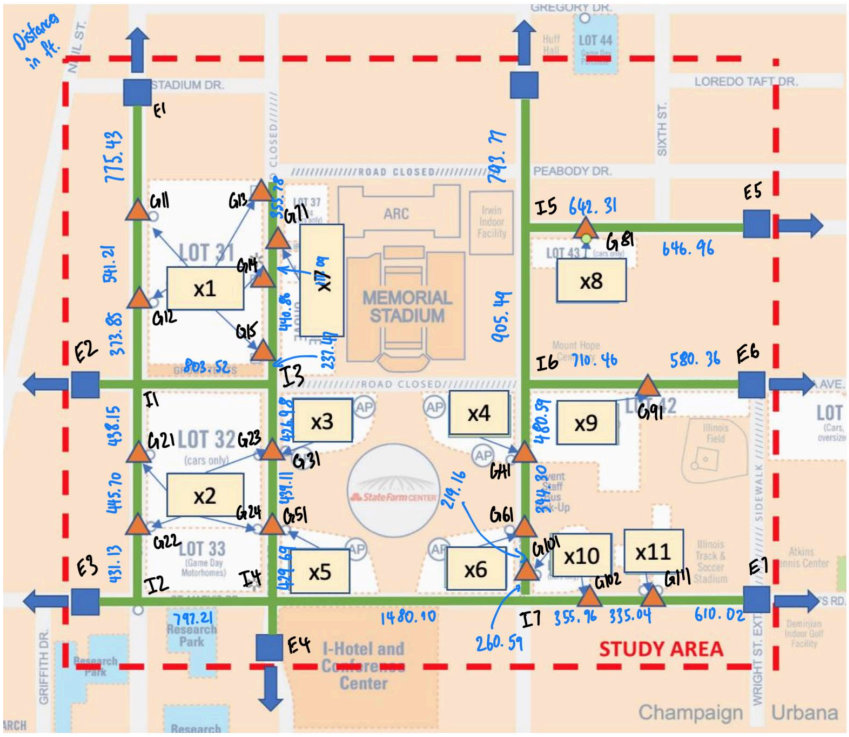


Figure 1: Definition of parking lots, intersections, gates, exits, and lengths between nodes.

	X1	X2	X3	X4	X5	X6	X7	X8	X9	X10	X11
9	2143	2473	434	526	501	437	776	221	601	0	152

Figure 2: Expected number of cars per parking lot

Task 1

Task 1 is made up of four different parts labeled 1A-1D. Each part presents a different case and asks us to compute the minimum total car-miles traveled within the network and find the links with the highest and lowest traffic flows. Task 1A assumes that each outflow node receives the same number of trips. Task 1B assumes that the number of vehicles travelling to the east or north is twice the number of vehicles travelling to the south and west. Task 1C assumes that Parking Lot 40 (x10), which is currently under construction, is operational and contains the same number of cars as Parking Lot 43. Task 1D asks us to find which parking lot closure would cause the greatest increase in overall traffic flow.

To complete Task 1A , we first set up our constraints. We then computed the total number of cars divided by the number of gates to find the number of cars per exit node over the time period. We solved for the minimal distance for all cars combined, and found that under our constraints, it would be 3822.3 total car miles. Below is our highest flow link and lowest flow link that we found in our optimization problem:

```
Status: Optimal
Minimum total car-miles: 3822.2964110795456

Highest Flow Link:
G41 -> I6: flow = 2277 cars, dist = 0.091021 miles, car-miles =
207.254437

Lowest Flow Link:
I1 -> G12: flow = 176 cars, dist = 0.070805 miles, car-miles =
12.447506
```

To complete Task 1B, we adjusted our constraints so that the number of cars travelling to Urbana and Champaign was double that of the cars travelling to Savoy or to Neil Street. We did

this by creating a list of exitWeights, where we gave a value of 2 to the exits that went east and north, and 1 to the other exits, to represent new weights. We used this change to find the following Python output:

```
Minimum total car-miles: 4525.745171198297

Highest Flow Link:
I4 -> I7: flow = 3148.5385 cars, dist = 0.280473 miles, car-miles =
883.081565

Lowest Flow Link:
G21 -> G22: flow = 17.442308 cars, dist = 0.084413 miles, car-miles =
1.472355
```

To complete Task 1C, we changed the capacity of Parking Lot 40 to match the problem and found this result when we ran the Python script:

```
Minimum total car-miles: 4567.602728963446

Highest Flow Link:
G41 -> I6: flow = 3094.1538 cars, dist = 0.091021 miles, car-miles =
281.632457

Lowest Flow Link:
G21 -> G22: flow = 34.442308 cars, dist = 0.084413 miles, car-miles =
2.907374
```

As displayed by the code above, the minimum total distance driven by all the cars with the opening of Lot 40 is about 4568 miles. The link from G41-I6 is the most used path with around 3094 cars taking this route. On the other hand, the link from G21-G22 is the least used path with around 34 cars taking this route. Therefore, after the opening of Lot 40, the flow of traffic has been modified within the network since the highest flow link has changed from I4-I7 to G41-I6. About 3094 cars will use the G41-I6 path as opposed to the previous scenario of around 3149 cars using the G41-I6 path. Since the number of cars on the most used link in the network has decreased, the opening of Lot 40 has successfully reduced congestion.

To complete Task 1D, we manually tested setting each parking lot's number of cars to 0 and compared the new minimum total car-miles to the value we calculated in scenario A. The lot we decided to close ended up being Lot X2 since it had the greatest reduction in congestion for our network. Here is the strip of Python code that allowed us to come to this conclusion:

```
Minimum total car-miles: 2544.1944025568177

Highest Flow Link:
G51 -> I4: flow = 2406.25 cars, dist = 0.081381 miles, car-miles =
195.822266

Lowest Flow Link:
I6 -> G91: flow = 122.875 cars, dist = 0.134557 miles, car-miles =
16.533669
```

As shown by the Python response above, the minimum total distance driven by all the cars is now about 2544 miles which is much less than that of the baseline (4526 miles). The link from G51-I4 is the most used path with around 2406 cars taking this route. The link from I6-G91 is the least used path with around 123 cars taking this route. Therefore, the closing of Lot X2 results in a reduction of miles traveled by cars by almost half since it limits the amount of cars having the ability to park in the area of study. Cars would instead have to park away from the area of study after a game resulting in less congestion and fewer miles traveled. To further increase the traffic flow there could be signs posted to alert drivers to park in lots that are not usually used to capacity, allowing for more spread out parking and less concentration in one area. Additionally, there could be lots that are further away from the stadiums that shuttle buses could be made to regularly drive to and from before and after games; thus further reducing the number of cars parked in the area of study.

Task 2

The second task is to select a parking lot and increase its trip demand by 20%, using the same assumptions as Task 1A. For this section we increased the trip demand of parking Lot 31 by 20%. **Figure 4** shows the change in values from Task 1A to Task 2. The only thing we had to change from our initial code in Task 1A was a line stating that the demand of Lot 31 was 20% greater than it was previously. The results of our Python code are printed below:

```
ncars[0] = ncars[0] * 1.2
Minimum total car-miles: 4103.457737405303

Highest Flow Link:
G41 -> I6: flow = 2437.5 cars, dist = 0.091021 miles, car-miles =
221.863281

Lowest Flow Link:
I1 -> G12: flow = 58.1 cars, dist = 0.070805 miles, car-miles =
4.113766
```

Observation	Baseline (No change)	20% Increase in Trip Demand to Lot 31
Min Total Car-miles	3822.3	4103.46
Highest Flow Link	G41 → I6 (2277 cars)	G41 → I6 (2437 cars)
Lowest Flow Link	I1 → G12 (176 cars)	I1 → G12 (58 cars)

Figure 4 displays the relationship between the baseline and the new scenario

Our main observation was that the minimum total car-miles increased from Task 1 to 2 when we made this change as it grew by 7.36% as a result of the additional constraint. Likewise the highest and lowest flow links did not change but the amount of cars on each link did. The highest flow link (G41-I6) was used by 2277 cars in the baseline scenario while in the new scenario 2437 cars passed through the link. On the other hand, the lowest flow link (I1-G12) was used by 176 cars in the baseline scenario but only 58 cars used it in the new scenario. The

increase in the highest flow link and decrease in the lowest flow link is due to cars originally passing through I1-G12 switching to G41-I6 in the current scenario. Due to Lot 31 being one of the largest available parking lots in the system and with a 20% increase in trip demand more cars will be exiting from Lot 31. There are more lot-exit nodes on the right side of Lot 31, leading to an increase in cars traveling on the west side of Memorial Stadium and the State Farm Center (Figure 1 perspective). The west side of the area of study, where link I1-G12 is located, contains more intersections, hence more variability in the possible paths the cars can take before reaching link I1-G12 – reducing the number of cars using the I1-G12 link. Therefore, more cars are being funneled toward link G41-I6 as opposed to I1-G12 since there are fewer intersections and fewer possible paths they can take before reaching G41-I6.

Task 3

Given that each lane has a capacity of 450 cars per hour, we used this information to find out how many cars can exit the area in a 30-minute period given our study area. We programmed this problem into Python with the same lots and nodes as we had in the previous two tasks. Using Google Maps, we determined the number of lanes for each road in the network, which varied between one and two excluding turn lanes, and allotted a capacity of 225 cars/30 minutes. Using Python, we then totaled the capacity for each lane and summed them. That allowed us to gather the results as displayed by this Python strip of code:

```
Max cars exiting in 30 min: 2471.0

Highest Flow Link:
I3 -> I1: flow = 450 cars
G22 -> I2: flow = 450 cars
G91 -> E6: flow = 450 cars
I1 -> E2: flow = 450 cars
```

```
I2 -> E3: flow = 450 cars

Lowest Flow Link:
G101 -> I7: flow = 73 cars
I7 -> G102: flow = 73 cars
G102 -> G111: flow = 73 cars
```

As shown by the above code, the lowest flow links are G101-I7, I7-G102, and G102-G111 – all having 73 cars passing through. Therefore, these links, which are a section of the streets surrounding Lots 40 and 41, would benefit the most from the deployment of traffic coordinators to increase capacity and flow.

Task 4

We have been asked to assist the city in making a decision about possibly adding one new lane to an existing city street. It is beneficial to the city to add one additional lane on the specified link. Here is the output from our Python code displaying our findings:

```
Max cars exiting in 30 min: 2696.0

Highest Flow Link:
I1 -> E2: flow = 675 cars
I3 -> I1: flow = 450 cars
G22 -> I2: flow = 450 cars
G91 -> E6: flow = 450 cars
I2 -> E3: flow = 450 cars

Lowest Flow Link:
G101 -> I7: flow = 73 cars
I7 -> G102: flow = 73 cars
G102 -> G111: flow = 73 cars

Cost of adding one lane to specified link: $84054.00
Economic Benefit from additional lane: $135000.00
Expected Revenue: $50946.00
Original benefit: $1482600.00 , New benefit: $1560706.00
Difference in revenue after adding additional lane: $78106.00
```

1. Lane Location and Traffic Impact:

   From our code there were a few options for lanes to choose from as we had initially five links that were at max capacity. Out of the five links at max capacity only three of them were valid choices to increase the number of lanes. Picking a link between intersections would not increase total capacity due to flow constraints requiring the inflow to equal outflow. Only the links that connect intersections or gates to exits would benefit from the additional lane as there is no restriction on flow coming in (considering nobody wants to come back into a basketball game after it ends) and they are the bottlenecks for attempting to leave the network.

2. Economic Benefit Analysis:

   A. After adding one lane the expected revenue increase is given in the code as $50,946.00

   B. The additional benefit is around $78,000 from adding the additional lane

3. Second Lane and Diminishing Returns:

   After increasing the lane count by one on the same link we had this output:

```
Max cars exiting in 30 min: 2921.0

Cost of adding one lane to specified link: $168108.00
Economic Benefit from additional lane: $135000.00
Expected Revenue: $-33108.00
```

From our output the expected revenue is now negative indicating that while the lane count increases and the max number of cars exiting also increases it isn't profitable as the additional economic benefit is outweighed by the cost of adding another lane. Our results imply that the law of diminishing returns applies to transportation infrastructure investments and simply adding more lanes will not necessarily be beneficial.

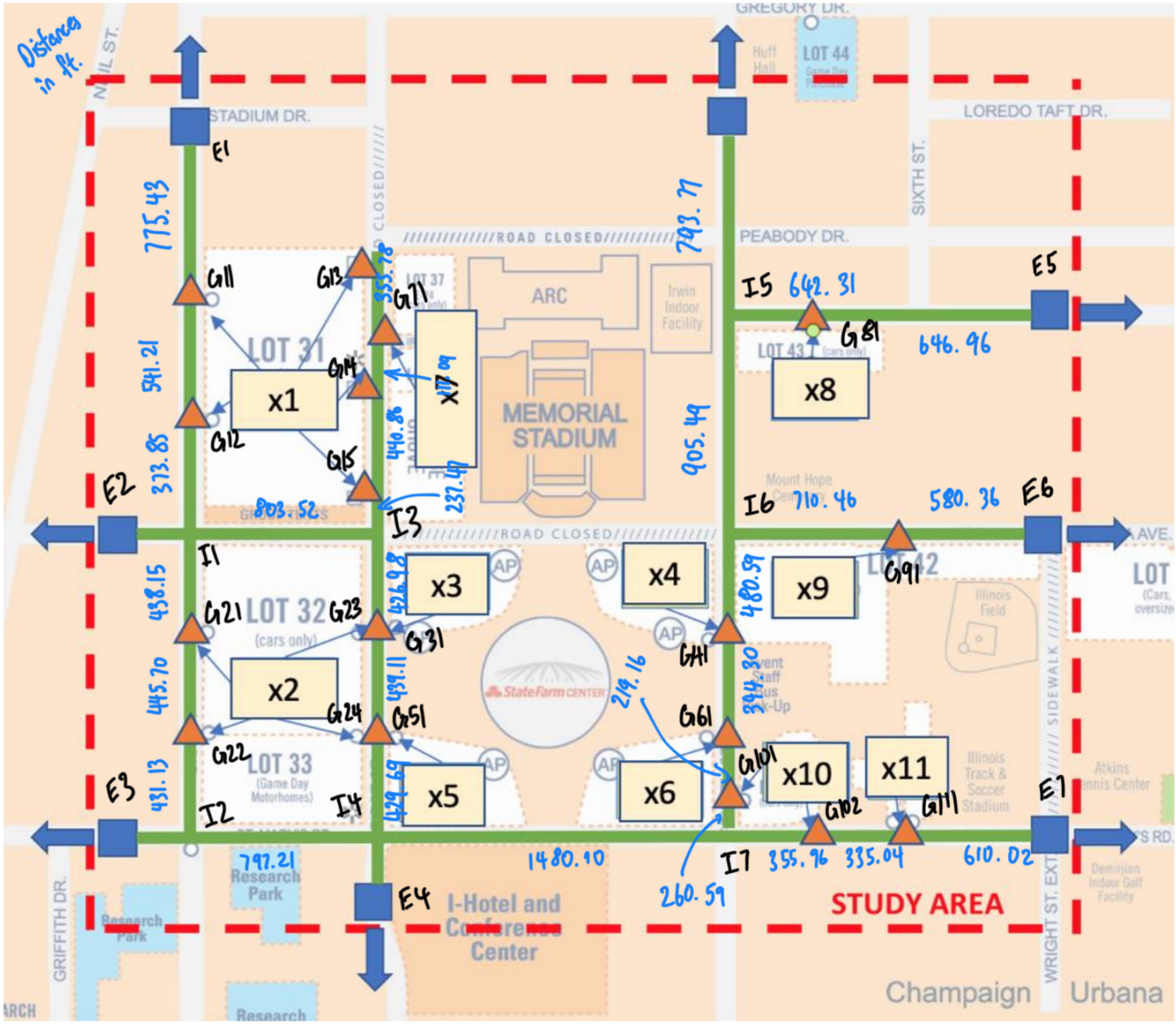
Python Code

In [ ]:
```
a2!pip install pulp
```

```
Requirement already satisfied: pulp in /usr/local/lib/python3.12/dist-packages (3.3.0)
```

In [1]:
```
import numpy as np
import pulp
```

Prepare Network Map



Task 1A

The following chunk of code models scenario A, where all trips are evenly distributed to each exit. Lines are hashtagged with additional comments.

In [5]:
```
#Task1A
ncars = np.array([2143, 2473, 434, 526, 501, 437, 776, 221, 601, 0, 152])
#expected number of cars in the parking lots in order x1-x11
tcars = np.sum(ncars)
outflow1a = tcars / 8 #equal distribution among each 8 exits

parkingLots = ['x1', 'x2', 'x3', 'x4', 'x5', 'x6', 'x7', 'x8', 'x9', 'x10', 'x11']   #parking lots
exits = ['E1', 'E2', 'E3', 'E4', 'E5', 'E6', 'E7', 'E8'] #exits
exitGates = ['G11','G12','G13','G14','G15','G21','G22','G23','G24','G31','G41','G51','G61','G71','G81','G91','G101',
             'G102','G111'] #gij i <- parking lot j <- exit gate number
intersections = ['I1','I2','I3','I4','I5','I6','I7'] #intersection between roads as labeled on the figure
```

```
totalNodes = parkingLots + exits + exitGates + intersections #total number of nodes

gatesToLots = { #sets up a dictionary corresponding each exit gate to their respective parking lots
    'x1' : ['G11','G12','G13','G14','G15'],
    'x2' : ['G21','G22','G23','G24'],
    'x3' : ['G31'],
    'x4' : ['G41'],
    'x5' : ['G51'],
    'x6' : ['G61'],
    'x7' : ['G71'],
    'x8' : ['G81'],
    'x9' : ['G91'],
    'x10' : ['G101', 'G102'],
    'x11' : ['G111']
}

links = { #all these values are in ft
    ('G11', 'E1'): 775.43,
    ('G11', 'G12'): 541.21, ('G12', 'G11'): 541.21,
    ('G12', 'I1'): 373.85, ('I1', 'G12'): 373.85,
    ('I1', 'I3'): 803.52, ('I3', 'I1'): 803.52,
    ('G21', 'I1'): 438.15, ('I1', 'G21'): 438.15,
    ('G21', 'G22'): 445.70, ('G22', 'G21'): 445.70,
    ('G22', 'I2'): 431.13, ('I2', 'G22'): 431.13,
    ('I4', 'I2'): 797.21, ('I2', 'I4'): 797.21,
    ('G13', 'G71'): 355.78, ('G71', 'G13'): 355.78,
    ('G71', 'G14'): 111.09, ('G14', 'G71'): 111.09,
    ('G14', 'G15'): 440.80, ('G15', 'G14'): 440.80,
    ('G15', 'I3'): 237.47, ('I3', 'G15'): 237.47,
    ('I3', 'G23'): 426.98, ('G23', 'I3'): 426.98, ('I3', 'G31'): 426.98, ('G31', 'I3'): 426.98,
    ('G23', 'G24'): 439.11, ('G24', 'G23'): 439.11, ('G31', 'G51'): 439.11, ('G51', 'G31'): 439.11,
    ('G24', 'I4') : 429.69, ('I4', 'G24') : 429.69, ('G51', 'I4') : 429.69, ('I4', 'G51') : 429.69,
    ('I4', 'I7') : 1480.90, ('I7', 'I4') : 1480.90,
    ('I5', 'E8') : 793.77,
    ('I5', 'G81') : 642.31, ('G81', 'I5') : 642.31,
    ('G81', 'E5') : 646.96,
    ('I5', 'I6') : 905.49, ('I6', 'I5') : 905.49,
    ('I6', 'G91') : 710.46, ('G91', 'I6') : 710.46,
    ('G91', 'E6') : 580.36,
    ('I6', 'G41') : 480.59, ('G41', 'I6') : 480.59,
    ('G41', 'G61') : 344.30, ('G61', 'G41') : 344.30,
    ('G61', 'G101') : 219.16, ('G101', 'G61') : 219.16,
    ('G101', 'I7') : 260.59, ('I7', 'G101') : 260.59,
    ('I7', 'G102') : 355.96, ('G102', 'I7') : 355.96,
    ('G102', 'G111') : 355.04, ('G111', 'G102') : 355.04,
    ('G111', 'E7') : 610.02,
    ('I1', 'E2') : 700.45,
    ('I2', 'E3') : 693.98,
    ('I4', 'E4') : 327.93 #all the links between nodes with distances based off figure
}

for i in links:
    links[i] /= 5280 #converts all the ft into miles

decisionVariables = {} #sets all the decision variables as links between each nodes as Cij where i is the
                       #start node and j is the finish node
for (i,j) in links:
    decisionVariables[(i,j)] = pulp.LpVariable(f"c{i}{j}", lowBound=0)
    #lowbound so that no links can be less than 0

prob = pulp.LpProblem("MinimizeTotalDistance", pulp.LpMinimize) #establish the minimization problem
prob += pulp.lpSum(decisionVariables[(i,j)] * links[(i,j)] for (i,j) in links)
#adds all the decision variables to the problem for PULP to solve

gateInflow = {g: 0.0 for g in exitGates}
#sets the inflow for all gates as 0 before adding inflows from each parking lot

for xx, lot in enumerate(parkingLots):
#constraint 1: sets the number of cars from each parking lot equal to each # of gates
    gates = gatesToLots.get(lot, [])
    perGate = float(ncars[xx]) / len(gates)
    for g in gates:
        gateInflow[g] += perGate

for node in exitGates + intersections:
#constraint 2: for gates outflow-inflow = source, for intersections outflow = inflow
    inflows = [decisionVariables[(i,j)] for (i,j) in decisionVariables if j == node]
    outflows = [decisionVariables[(i,j)] for (i,j) in decisionVariables if i == node]
    rhs = gateInflow.get(node, 0)  # for gates this is positive; for intersections it's 0
```

```
    prob += (pulp.lpSum(outflows) - pulp.lpSum(inflows) == rhs) #outflow = inflow

carsPerExit = float(np.sum(ncars)) / len(exits) #equal distribution of cars per exit

for e in exits: #exit constraint so that each exit receives carsPerExit
    inflows = [decisionVariables[(i,j)] for (i,j) in decisionVariables if j == e]
    prob += (pulp.lpSum(inflows) == carsPerExit)

prob.solve(pulp.PULP_CBC_CMD(msg=0))
print("Status:", pulp.LpStatus[prob.status])
print("Minimum total car-miles:", pulp.value(prob.objective))

nonzero = {k: v.varValue for k, v in decisionVariables.items() if v.varValue > 0}
sortedDescending = sorted(nonzero.items(), key=lambda x: x[1], reverse=True)
#sorts flow of links from highest to lowest
sortedAscending = sorted(nonzero.items(), key=lambda x: x[1])
#sorts flow of links from lowest to highest

print("\nHighest Flow Link:") #this block and block below print out lowest and highest flow links
for (i,j), flow in sortedDescending[:1]:
    print(f"{i} -> {j}: flow = {flow:.0f} cars, dist = {links[(i,j)]:.6f} miles, car-miles = {flow*links[(i,j)]:.6f}")

print("\nLowest Flow Link:")
for (i,j), flow in sortedAscending[:1]:
    print(f"{i} -> {j}: flow = {flow:.0f} cars, dist = {links[(i,j)]:.6f} miles, car-miles = {flow*links[(i,j)]:.6f}")
```

```
Status: Optimal
Minimum total car-miles: 3822.2964110795456

Highest Flow Link:
G41 -> I6: flow = 2277 cars, dist = 0.091021 miles, car-miles = 207.254437

Lowest Flow Link:
I1 -> G12: flow = 176 cars, dist = 0.070805 miles, car-miles = 12.447506
```

Task 1B

The following chunk of code models scenario B, where trips are weighted so that twice as many cars go to the north and east exits. Lines are hashtagged with additional comments. The majority of this code is copied from task 1A with the main difference being the weighted exits portion.

In [7]:
```
#Task1B
ncars = np.array([2143, 2473, 434, 526, 501, 437, 776, 221, 601, 0, 152])
#expected number of cars in the parking lots in order x1-x11
tcars = np.sum(ncars)
outflow1a = tcars / 8 #equal distribution among each 8 exits

parkingLots = ['x1', 'x2', 'x3', 'x4', 'x5', 'x6', 'x7', 'x8', 'x9', 'x10', 'x11']   #parking lots
exits = ['E1', 'E2', 'E3', 'E4', 'E5', 'E6', 'E7', 'E8'] #exits
exitGates = ['G11','G12','G13','G14','G15','G21','G22','G23','G24','G31','G41','G51','G61','G71','G81','G91','G101',
             'G102','G111'] #gij i <- parking lot j <- exit gate number
intersections = ['I1','I2','I3','I4','I5','I6','I7'] #intersection between roads as labeled on the figure
totalNodes = parkingLots + exits + exitGates + intersections #total number of nodes

gatesToLots = { #sets up a dictionary corresponding each exit gate to their respective parking lots
    'x1' : ['G11','G12','G13','G14','G15'],
    'x2' : ['G21','G22','G23','G24'],
    'x3' : ['G31'],
    'x4' : ['G41'],
    'x5' : ['G51'],
    'x6' : ['G61'],
    'x7' : ['G71'],
    'x8' : ['G81'],
    'x9' : ['G91'],
    'x10' : ['G101', 'G102'],
    'x11' : ['G111']
}

links = { #all these values are in ft
    ('G11', 'E1'): 775.43,
    ('G11', 'G12'): 541.21, ('G12', 'G11'): 541.21,
    ('G12', 'I1'): 373.85, ('I1', 'G12'): 373.85,
    ('I1', 'I3'): 803.52, ('I3', 'I1'): 803.52,
    ('G21', 'I1'): 438.15, ('I1', 'G21'): 438.15,
    ('G21', 'G22'): 445.70, ('G22', 'G21'): 445.70,
    ('G22', 'I2'): 431.13, ('I2', 'G22'): 431.13,
    ('I4', 'I2'): 797.21, ('I2', 'I4'): 797.21,
```

```
    ('G13', 'G71'): 355.78, ('G71', 'G13'): 355.78,
    ('G71', 'G14'): 111.09, ('G14', 'G71'): 111.09,
    ('G14', 'G15'): 440.80, ('G15', 'G14'): 440.80,
    ('G15', 'I3'): 237.47, ('I3', 'G15'): 237.47,
    ('I3', 'G23'): 426.98, ('G23', 'I3'): 426.98, ('I3', 'G31'): 426.98, ('G31', 'I3'): 426.98,
    ('G23', 'G24'): 439.11, ('G24', 'G23'): 439.11, ('G31', 'G51'): 439.11, ('G51', 'G31'): 439.11,
    ('G24', 'I4') : 429.69, ('I4', 'G24') : 429.69, ('G51', 'I4') : 429.69, ('I4', 'G51') : 429.69,
    ('I4', 'I7') : 1480.90, ('I7', 'I4') : 1480.90,
    ('I5', 'E8') : 793.77,
    ('I5', 'G81') : 642.31, ('G81', 'I5') : 642.31,
    ('G81', 'E5') : 646.96,
    ('I5', 'I6') : 905.49, ('I6', 'I5') : 905.49,
    ('I6', 'G91') : 710.46, ('G91', 'I6') : 710.46,
    ('G91', 'E6') : 580.36,
    ('I6', 'G41') : 480.59, ('G41', 'I6') : 480.59,
    ('G41', 'G61') : 344.30, ('G61', 'G41') : 344.30,
    ('G61', 'G101') : 219.16, ('G101', 'G61') : 219.16,
    ('G101', 'I7') : 260.59, ('I7', 'G101') : 260.59,
    ('I7', 'G102') : 355.96, ('G102', 'I7') : 355.96,
    ('G102', 'G111') : 355.04, ('G111', 'G102') : 355.04,
    ('G111', 'E7') : 610.02,
    ('I1', 'E2') : 700.45,
    ('I2', 'E3') : 693.98,
    ('I4', 'E4') : 327.93 #all the links between nodes with distances based off figure
}

for i in links:
    links[i] /= 5280 #converts all the ft into miles

decisionVariables = {} #sets all the decision variables as links between each nodes as Cij where i is the
                       #start node and j is the finish node
for (i,j) in links:
    decisionVariables[(i,j)] = pulp.LpVariable(f"c{i}{j}", lowBound=0)

prob = pulp.LpProblem("MinimizeTotalDistance", pulp.LpMinimize) #establish the minimization problem
prob += pulp.lpSum(decisionVariables[(i,j)] * links[(i,j)] for (i,j) in links)
#adds all the decision variables to the problem for PULP to solve

gateInflow = {g: 0.0 for g in exitGates}
#sets the inflow for all gates as 0 before adding inflows from each parking lot

for xx, lot in enumerate(parkingLots):
#constraint 1: equalizes the number of cars from each parking lot to each # of gates
    gates = gatesToLots.get(lot, [])
    perGate = float(ncars[xx]) / len(gates)
    for g in gates:
        gateInflow[g] += perGate

for node in exitGates + intersections:
#constraint 2: for gates outflow-inflow = source, for intersections outflow = inflow
    inflows = [decisionVariables[(i,j)] for (i,j) in decisionVariables if j == node]
    outflows = [decisionVariables[(i,j)] for (i,j) in decisionVariables if i == node]
    rhs = gateInflow.get(node, 0)  # for gates this is positive; for intersections it's 0
    prob += (pulp.lpSum(outflows) - pulp.lpSum(inflows) == rhs) #outflow = inflow

exitWeights = {'E1': 2, 'E2': 1, 'E3': 1, 'E4': 1, 'E5': 2, 'E6': 2, 'E7': 2, 'E8': 2}
totalWeight = sum(exitWeights.values())
carsPerExit = {e: tcars * w / totalWeight for e, w in exitWeights.items()}

for e in exits: #exit constraint so that each exit receives carsPerExit
    inflows = [decisionVariables[(i,j)] for (i,j) in decisionVariables if j == e]
    prob += (pulp.lpSum(inflows) == carsPerExit[e])

prob.solve(pulp.PULP_CBC_CMD(msg=0))
print("Status:", pulp.LpStatus[prob.status])
print("Minimum total car-miles:", pulp.value(prob.objective))

nonzero = {k: v.varValue for k, v in decisionVariables.items() if v.varValue > 0}
sortedDescending = sorted(nonzero.items(), key=lambda x: x[1], reverse=True)
#sorts flow of links from highest to lowest
sortedAscending = sorted(nonzero.items(), key=lambda x: x[1]) #sorts flow of links from lowest to highest

print("\nHighest Flow Link:") #this block and block below print out lowest and highest flow links
for (i,j), flow in sortedDescending[:1]:
    print(f"{i} -> {j}: flow = {flow:.0f} cars, dist = {links[(i,j)]:.6f} miles, car-miles = {flow*links[(i,j)]:.6f}")

print("\nLowest Flow Link:")
for (i,j), flow in sortedAscending[:1]:
    print(f"{i} -> {j}: flow = {flow:.0f} cars, dist = {links[(i,j)]:.6f} miles, car-miles = {flow*links[(i,j)]:.6f}")
```

```
Status: Optimal
Minimum total car-miles: 4525.745171198297

Highest Flow Link:
I4 -> I7: flow = 3149 cars, dist = 0.280473 miles, car-miles = 883.081565

Lowest Flow Link:
G21 -> G22: flow = 17 cars, dist = 0.084413 miles, car-miles = 1.472355
```

Task 1C

The following chunk of code models scenario C, where trips are weighted so that twice as many cars go to the north and east exits and parking lot 40 is open with the same number of cars as lot 43. Lines are hashtagged with additional comments. The majority of this code is copied from task 1B with the main difference being the array 'ncars' being adjusted.

In [9]:
```
#Task1C; modeled same way as 1B where cars are weighted to go east-north and not equally distributed
ncars = np.array([2143, 2473, 434, 526, 501, 437, 776, 221, 601, 221, 152])
#expected number of cars in the parking lots in order x1-x11; x10 now has same number of cars as x8
tcars = np.sum(ncars)
outflow1a = tcars / 8 #equal distribution among each 8 exits

parkingLots = ['x1', 'x2', 'x3', 'x4', 'x5', 'x6', 'x7', 'x8', 'x9', 'x10', 'x11']   #parking lots
exits = ['E1', 'E2', 'E3', 'E4', 'E5', 'E6', 'E7', 'E8'] #exits
exitGates = ['G11','G12','G13','G14','G15','G21','G22','G23','G24','G31','G41','G51','G61','G71','G81','G91','G101',
             'G102','G111'] #gij i <- parking lot j <- exit gate number
intersections = ['I1','I2','I3','I4','I5','I6','I7'] #intersection between roads as labeled on the figure
totalNodes = parkingLots + exits + exitGates + intersections #total number of nodes

gatesToLots = { #sets up a dictionary corresponding each exit gate to their respective parking lots
    'x1' : ['G11','G12','G13','G14','G15'],
    'x2' : ['G21','G22','G23','G24'],
    'x3' : ['G31'],
    'x4' : ['G41'],
    'x5' : ['G51'],
    'x6' : ['G61'],
    'x7' : ['G71'],
    'x8' : ['G81'],
    'x9' : ['G91'],
    'x10' : ['G101', 'G102'],
    'x11' : ['G111']
}

links = { #all these values are in ft
    ('G11', 'E1'): 775.43,
    ('G11', 'G12'): 541.21, ('G12', 'G11'): 541.21,
    ('G12', 'I1'): 373.85, ('I1', 'G12'): 373.85,
    ('I1', 'I3'): 803.52, ('I3', 'I1'): 803.52,
    ('G21', 'I1'): 438.15, ('I1', 'G21'): 438.15,
    ('G21', 'G22'): 445.70, ('G22', 'G21'): 445.70,
    ('G22', 'I2'): 431.13, ('I2', 'G22'): 431.13,
    ('I4', 'I2'): 797.21, ('I2', 'I4'): 797.21,
    ('G13', 'G71'): 355.78, ('G71', 'G13'): 355.78,
    ('G71', 'G14'): 111.09, ('G14', 'G71'): 111.09,
    ('G14', 'G15'): 440.80, ('G15', 'G14'): 440.80,
    ('G15', 'I3'): 237.47, ('I3', 'G15'): 237.47,
    ('I3', 'G23'): 426.98, ('G23', 'I3'): 426.98, ('I3', 'G31'): 426.98, ('G31', 'I3'): 426.98,
    ('G23', 'G24'): 439.11, ('G24', 'G23'): 439.11, ('G31', 'G51'): 439.11, ('G51', 'G31'): 439.11,
    ('G24', 'I4') : 429.69, ('I4', 'G24') : 429.69, ('G51', 'I4') : 429.69, ('I4', 'G51') : 429.69,
    ('I4', 'I7') : 1480.90, ('I7', 'I4') : 1480.90,
    ('I5', 'E8') : 793.77,
    ('I5', 'G81') : 642.31, ('G81', 'I5') : 642.31,
    ('G81', 'E5') : 646.96,
    ('I5', 'I6') : 905.49, ('I6', 'I5') : 905.49,
    ('I6', 'G91') : 710.46, ('G91', 'I6') : 710.46,
    ('G91', 'E6') : 580.36,
    ('I6', 'G41') : 480.59, ('G41', 'I6') : 480.59,
    ('G41', 'G61') : 344.30, ('G61', 'G41') : 344.30,
    ('G61', 'G101') : 219.16, ('G101', 'G61') : 219.16,
    ('G101', 'I7') : 260.59, ('I7', 'G101') : 260.59,
    ('I7', 'G102') : 355.96, ('G102', 'I7') : 355.96,
    ('G102', 'G111') : 355.04, ('G111', 'G102') : 355.04,
    ('G111', 'E7') : 610.02,
    ('I1', 'E2') : 700.45,
    ('I2', 'E3') : 693.98,
    ('I4', 'E4') : 327.93 #all the links between nodes with distances based off figure
}
```

```
for i in links:
    links[i] /= 5280 #converts all the ft into miles

decisionVariables = {} #sets all the decision variables as links between each nodes as Cij where i is the
                       #start node and j is the finish node
for (i,j) in links:
    decisionVariables[(i,j)] = pulp.LpVariable(f"c{i}{j}", lowBound=0)

prob = pulp.LpProblem("MinimizeTotalDistance", pulp.LpMinimize) #establish the minimization problem
prob += pulp.lpSum(decisionVariables[(i,j)] * links[(i,j)] for (i,j) in links)
#adds all the decision variables to the problem for PULP to solve

gateInflow = {g: 0.0 for g in exitGates}
#sets the inflow for all gates as 0 before adding inflows from each parking lot

for xx, lot in enumerate(parkingLots):
#constraint 1: equalizes the number of cars from each parking lot to each # of gates
    gates = gatesToLots.get(lot, [])
    perGate = float(ncars[xx]) / len(gates)
    for g in gates:
        gateInflow[g] += perGate

for node in exitGates + intersections:
#constraint 2: for gates outflow-inflow = source, for intersections outflow = inflow
    inflows = [decisionVariables[(i,j)] for (i,j) in decisionVariables if j == node]
    outflows = [decisionVariables[(i,j)] for (i,j) in decisionVariables if i == node]
    rhs = gateInflow.get(node, 0)  # for gates this is positive; for intersections it's 0
    prob += (pulp.lpSum(outflows) - pulp.lpSum(inflows) == rhs) #outflow = inflow

exitWeights = {'E1': 2, 'E2': 1, 'E3': 1, 'E4': 1, 'E5': 2, 'E6': 2, 'E7': 2, 'E8': 2}
totalWeight = sum(exitWeights.values())
carsPerExit = {e: tcars * w / totalWeight for e, w in exitWeights.items()}

for e in exits: #exit constraint so that each exit receives carsPerExit
    inflows = [decisionVariables[(i,j)] for (i,j) in decisionVariables if j == e]
    prob += (pulp.lpSum(inflows) == carsPerExit[e])

prob.solve(pulp.PULP_CBC_CMD(msg=0))
print("Status:", pulp.LpStatus[prob.status])
print("Minimum total car-miles:", pulp.value(prob.objective))

nonzero = {k: v.varValue for k, v in decisionVariables.items() if v.varValue > 0}
sortedDescending = sorted(nonzero.items(), key=lambda x: x[1], reverse=True)
#sorts flow of links from highest to lowest
sortedAscending = sorted(nonzero.items(), key=lambda x: x[1]) #sorts flow of links from lowest to highest

print("\nHighest Flow Link:") #this block and block below print out lowest and highest flow links
for (i,j), flow in sortedDescending[:1]:
    print(f"{i} -> {j}: flow = {flow:.0f} cars, dist = {links[(i,j)]:.6f} miles, car-miles = {flow*links[(i,j)]:.6f}")

print("\nLowest Flow Link:")
for (i,j), flow in sortedAscending[:1]:
    print(f"{i} -> {j}: flow = {flow:.0f} cars, dist = {links[(i,j)]:.6f} miles, car-miles = {flow*links[(i,j)]:.6f}")
```

```
Status: Optimal
Minimum total car-miles: 4567.602728963446

Highest Flow Link:
G41 -> I6: flow = 3094 cars, dist = 0.091021 miles, car-miles = 281.632457

Lowest Flow Link:
G21 -> G22: flow = 34 cars, dist = 0.084413 miles, car-miles = 2.907374
```

Task 1D

The following chunk of code models scenario D, where all trips are evenly distributed to each exit. Lines are hashtagged with additional comments. We manually tested setting each parking lot's number of cars to 0 and comparing the new minimum total car-miles to the value we calculated in scenario A.

In [14]:
```
#Task1D
ncars = np.array([2143, 0, 434, 526, 501, 437, 776, 221, 601, 0, 152])
#expected number of cars in the parking lots in order x1-x11; replacing random lots with 0 to test which one makes
#biggest difference
tcars = np.sum(ncars) #[2143, 2473, 434, 526, 501, 437, 776, 221, 601, 0, 152]
outflow1a = tcars / 8 #equal distribution among each 8 exits

parkingLots = ['x1', 'x2', 'x3', 'x4', 'x5', 'x6', 'x7', 'x8', 'x9', 'x10', 'x11']   #parking lots
```

```
exits = ['E1', 'E2', 'E3', 'E4', 'E5', 'E6', 'E7', 'E8'] #exits
exitGates = ['G11','G12','G13','G14','G15','G21','G22','G23','G24','G31','G41','G51','G61','G71','G81','G91','G101',
             'G102','G111'] #gij i <- parking lot j <- exit gate number
intersections = ['I1','I2','I3','I4','I5','I6','I7'] #intersection between roads as labeled on the figure
totalNodes = parkingLots + exits + exitGates + intersections #total number of nodes

gatesToLots = { #sets up a dictionary corresponding each exit gate to their respective parking lots
    'x1' : ['G11','G12','G13','G14','G15'],
    'x2' : ['G21','G22','G23','G24'],
    'x3' : ['G31'],
    'x4' : ['G41'],
    'x5' : ['G51'],
    'x6' : ['G61'],
    'x7' : ['G71'],
    'x8' : ['G81'],
    'x9' : ['G91'],
    'x10' : ['G101', 'G102'],
    'x11' : ['G111']
}

links = { #all these values are in ft
    ('G11', 'E1'): 775.43,
    ('G11', 'G12'): 541.21, ('G12', 'G11'): 541.21,
    ('G12', 'I1'): 373.85, ('I1', 'G12'): 373.85,
    ('I1', 'I3'): 803.52, ('I3', 'I1'): 803.52,
    ('G21', 'I1'): 438.15, ('I1', 'G21'): 438.15,
    ('G21', 'G22'): 445.70, ('G22', 'G21'): 445.70,
    ('G22', 'I2'): 431.13, ('I2', 'G22'): 431.13,
    ('I4', 'I2'): 797.21, ('I2', 'I4'): 797.21,
    ('G13', 'G71'): 355.78, ('G71', 'G13'): 355.78,
    ('G71', 'G14'): 111.09, ('G14', 'G71'): 111.09,
    ('G14', 'G15'): 440.80, ('G15', 'G14'): 440.80,
    ('G15', 'I3'): 237.47, ('I3', 'G15'): 237.47,
    ('I3', 'G23'): 426.98, ('G23', 'I3'): 426.98, ('I3', 'G31'): 426.98, ('G31', 'I3'): 426.98,
    ('G23', 'G24'): 439.11, ('G24', 'G23'): 439.11, ('G31', 'G51'): 439.11, ('G51', 'G31'): 439.11,
    ('G24', 'I4') : 429.69, ('I4', 'G24') : 429.69, ('G51', 'I4') : 429.69, ('I4', 'G51') : 429.69,
    ('I4', 'I7') : 1480.90, ('I7', 'I4') : 1480.90,
    ('I5', 'E8') : 793.77,
    ('I5', 'G81') : 642.31, ('G81', 'I5') : 642.31,
    ('G81', 'E5') : 646.96,
    ('I5', 'I6') : 905.49, ('I6', 'I5') : 905.49,
    ('I6', 'G91') : 710.46, ('G91', 'I6') : 710.46,
    ('G91', 'E6') : 580.36,
    ('I6', 'G41') : 480.59, ('G41', 'I6') : 480.59,
    ('G41', 'G61') : 344.30, ('G61', 'G41') : 344.30,
    ('G61', 'G101') : 219.16, ('G101', 'G61') : 219.16,
    ('G101', 'I7') : 260.59, ('I7', 'G101') : 260.59,
    ('I7', 'G102') : 355.96, ('G102', 'I7') : 355.96,
    ('G102', 'G111') : 355.04, ('G111', 'G102') : 355.04,
    ('G111', 'E7') : 610.02,
    ('I1', 'E2') : 700.45,
    ('I2', 'E3') : 693.98,
    ('I4', 'E4') : 327.93 #all the links between nodes with distances based off figure
}

for i in links:
    links[i] /= 5280 #converts all the ft into miles

decisionVariables = {}
#sets all the decision variables as links between each nodes as Cij where i is the start node and j is the finish node
for (i,j) in links:
    decisionVariables[(i,j)] = pulp.LpVariable(f"c{i}{j}", lowBound=0) #lowbound so that no links can be less than 0

prob = pulp.LpProblem("MinimizeTotalDistance", pulp.LpMinimize) #establish the minimization problem
prob += pulp.lpSum(decisionVariables[(i,j)] * links[(i,j)] for (i,j) in links)
#adds all the decision variables to the problem for PULP to solve

gateInflow = {g: 0.0 for g in exitGates}
#sets the inflow for all gates as 0 before adding inflows from each parking lot

for xx, lot in enumerate(parkingLots):
#constraint 1: equalizes the number of cars from each parking lot to each # of gates
    gates = gatesToLots.get(lot, [])
    perGate = float(ncars[xx]) / len(gates)
    for g in gates:
        gateInflow[g] += perGate

for node in exitGates + intersections:
#constraint 2: for gates outflow-inflow = source, for intersections outflow = inflow
```

```
    inflows = [decisionVariables[(i,j)] for (i,j) in decisionVariables if j == node]
    outflows = [decisionVariables[(i,j)] for (i,j) in decisionVariables if i == node]
    rhs = gateInflow.get(node, 0)  # for gates this is positive; for intersections it's 0
    prob += (pulp.lpSum(outflows) - pulp.lpSum(inflows) == rhs) #outflow = inflow

carsPerExit = float(np.sum(ncars)) / len(exits) #equal distribution of cars per exit

for e in exits: #exit constraint so that each exit receives carsPerExit
    inflows = [decisionVariables[(i,j)] for (i,j) in decisionVariables if j == e]
    prob += (pulp.lpSum(inflows) == carsPerExit)

prob.solve(pulp.PULP_CBC_CMD(msg=0))
print("Status:", pulp.LpStatus[prob.status])
print("Minimum total car-miles:", pulp.value(prob.objective))
print("Minimum total car-miles (1A): 3822.2964110795456")
#closing down parking lot x2 causes the lowest total car-miles compared to any other parking lot
#this also makes sense because cutting down the biggest parking lot
#means you have less cars in the network trying to leave

nonzero = {k: v.varValue for k, v in decisionVariables.items() if v.varValue > 0}
sortedDescending = sorted(nonzero.items(), key=lambda x: x[1], reverse=True)
#sorts flow of links from highest to lowest
sortedAscending = sorted(nonzero.items(), key=lambda x: x[1]) #sorts flow of links from lowest to highest

print("\nHighest Flow Link:") #this block and block below print out lowest and highest flow links
for (i,j), flow in sortedDescending[:1]:
    print(f"{i} -> {j}: flow = {flow:.0f} cars, dist = {links[(i,j)]:.6f} miles, car-miles = {flow*links[(i,j)]:.6f}")

print("\nLowest Flow Link:")
for (i,j), flow in sortedAscending[:1]:
    print(f"{i} -> {j}: flow = {flow:.0f} cars, dist = {links[(i,j)]:.6f} miles, car-miles = {flow*links[(i,j)]:.6f}")
```

```
Status: Optimal
Minimum total car-miles: 2544.1944025568177
Minimum total car-miles (1A): 3822.2964110795456

Highest Flow Link:
G51 -> I4: flow = 2406 cars, dist = 0.081381 miles, car-miles = 195.822266

Lowest Flow Link:
I6 -> G91: flow = 123 cars, dist = 0.134557 miles, car-miles = 16.533669
```

Task 2

The following chunk of code models task 2, based off of the code from Task 1A. For this section we increased the trip demand out of parking lot 31 by 20%.

In [17]:

```
#Task2
ncars = np.array([2143, 2473, 434, 526, 501, 437, 776, 221, 601, 0, 152])
#expected number of cars in the parking lots in order x1-x11
ncars[0] = ncars[0] * 1.2 #increasing parking lot 31's trip demand by 20%
tcars = np.sum(ncars)
outflow1a = tcars / 8 #equal distribution among each 8 exits

parkingLots = ['x1', 'x2', 'x3', 'x4', 'x5', 'x6', 'x7', 'x8', 'x9', 'x10', 'x11']   #parking lots
exits = ['E1', 'E2', 'E3', 'E4', 'E5', 'E6', 'E7', 'E8'] #exits
exitGates = ['G11','G12','G13','G14','G15','G21','G22','G23','G24','G31','G41','G51','G61','G71','G81','G91',
             'G101','G102','G111'] #gij i <- parking lot j <- exit gate number
intersections = ['I1','I2','I3','I4','I5','I6','I7'] #intersection between roads as labeled on the figure
totalNodes = parkingLots + exits + exitGates + intersections #total number of nodes

gatesToLots = { #sets up a dictionary corresponding each exit gate to their respective parking lots
    'x1' : ['G11','G12','G13','G14','G15'],
    'x2' : ['G21','G22','G23','G24'],
    'x3' : ['G31'],
    'x4' : ['G41'],
    'x5' : ['G51'],
    'x6' : ['G61'],
    'x7' : ['G71'],
    'x8' : ['G81'],
    'x9' : ['G91'],
    'x10' : ['G101', 'G102'],
    'x11' : ['G111']
}

links = { #all these values are in ft
    ('G11', 'E1'): 775.43,
```

```
    ('G11', 'G12'): 541.21, ('G12', 'G11'): 541.21,
    ('G12', 'I1'): 373.85, ('I1', 'G12'): 373.85,
    ('I1', 'I3'): 803.52, ('I3', 'I1'): 803.52,
    ('G21', 'I1'): 438.15, ('I1', 'G21'): 438.15,
    ('G21', 'G22'): 445.70, ('G22', 'G21'): 445.70,
    ('G22', 'I2'): 431.13, ('I2', 'G22'): 431.13,
    ('I4', 'I2'): 797.21, ('I2', 'I4'): 797.21,
    ('G13', 'G71'): 355.78, ('G71', 'G13'): 355.78,
    ('G71', 'G14'): 111.09, ('G14', 'G71'): 111.09,
    ('G14', 'G15'): 440.80, ('G15', 'G14'): 440.80,
    ('G15', 'I3'): 237.47, ('I3', 'G15'): 237.47,
    ('I3', 'G23'): 426.98, ('G23', 'I3'): 426.98, ('I3', 'G31'): 426.98, ('G31', 'I3'): 426.98,
    ('G23', 'G24'): 439.11, ('G24', 'G23'): 439.11, ('G31', 'G51'): 439.11, ('G51', 'G31'): 439.11,
    ('G24', 'I4') : 429.69, ('I4', 'G24') : 429.69, ('G51', 'I4') : 429.69, ('I4', 'G51') : 429.69,
    ('I4', 'I7') : 1480.90, ('I7', 'I4') : 1480.90,
    ('I5', 'E8') : 793.77,
    ('I5', 'G81') : 642.31, ('G81', 'I5') : 642.31,
    ('G81', 'E5') : 646.96,
    ('I5', 'I6') : 905.49, ('I6', 'I5') : 905.49,
    ('I6', 'G91') : 710.46, ('G91', 'I6') : 710.46,
    ('G91', 'E6') : 580.36,
    ('I6', 'G41') : 480.59, ('G41', 'I6') : 480.59,
    ('G41', 'G61') : 344.30, ('G61', 'G41') : 344.30,
    ('G61', 'G101') : 219.16, ('G101', 'G61') : 219.16,
    ('G101', 'I7') : 260.59, ('I7', 'G101') : 260.59,
    ('I7', 'G102') : 355.96, ('G102', 'I7') : 355.96,
    ('G102', 'G111') : 355.04, ('G111', 'G102') : 355.04,
    ('G111', 'E7') : 610.02,
    ('I1', 'E2') : 700.45,
    ('I2', 'E3') : 693.98,
    ('I4', 'E4') : 327.93 #all the links between nodes with distances based off figure
}

for i in links:
    links[i] /= 5280 #converts all the ft into miles

decisionVariables = {}
#sets all the decision variables as links between each nodes as Cij where i is the start node
#and j is the finish node
for (i,j) in links:
    decisionVariables[(i,j)] = pulp.LpVariable(f"c{i}{j}", lowBound=0)
    #lowbound so that no links can be less than 0

prob = pulp.LpProblem("MinimizeTotalDistance", pulp.LpMinimize) #establish the minimization problem
prob += pulp.lpSum(decisionVariables[(i,j)] * links[(i,j)] for (i,j) in links)
#adds all the decision variables to the problem for PULP to solve

gateInflow = {g: 0.0 for g in exitGates} #sets the inflow for all gates as 0 before adding inflows from each parking lot

for xx, lot in enumerate(parkingLots):
#constraint 1: equalizes the number of cars from each parking lot to each # of gates
    gates = gatesToLots.get(lot, [])
    perGate = ncars[xx] / len(gates)
    for i in gates:
        gateInflow[i] += perGate

for node in exitGates + intersections:
#constraint 2: for gates outflow-inflow = source, for intersections outflow = inflow
    inflows = [decisionVariables[(i,j)] for (i,j) in decisionVariables if j == node]
    outflows = [decisionVariables[(i,j)] for (i,j) in decisionVariables if i == node]
    rhs = gateInflow.get(node, 0)  # for gates this is positive; for intersections it's 0
    prob += (pulp.lpSum(outflows) - pulp.lpSum(inflows) == rhs) #outflow = inflow

carsPerExit = tcars / len(exits) #equal distribution of cars per exit

for e in exits: #exit constraint so that each exit receives carsPerExit
    inflows = [decisionVariables[(i,j)] for (i,j) in decisionVariables if j == e]
    prob += (pulp.lpSum(inflows) == carsPerExit)

prob.solve(pulp.PULP_CBC_CMD(msg=0))
print("Status:", pulp.LpStatus[prob.status])
print("Minimum total car-miles:", pulp.value(prob.objective))

nonzero = {k: v.varValue for k, v in decisionVariables.items() if v.varValue > 0}
sortedDescending = sorted(nonzero.items(), key=lambda x: x[1], reverse=True)
#sorts flow of links from highest to lowest
sortedAscending = sorted(nonzero.items(), key=lambda x: x[1]) #sorts flow of links from lowest to highest

print("\nHighest Flow Link:") #this block and block below print out lowest and highest flow links
```

```
for (i,j), flow in sortedDescending[:1]:
    print(f"{i} -> {j}: flow = {flow:0f} cars, dist = {links[(i,j)]:.6f} miles, car-miles = {flow*links[(i,j)]:.6f}")

print("\nLowest Flow Link:")
for (i,j), flow in sortedAscending[:1]:
    print(f"{i} -> {j}: flow = {flow:.0f} cars, dist = {links[(i,j)]:.6f} miles, car-miles = {flow*links[(i,j)]:.6f}")
```

```
Status: Optimal
Minimum total car-miles: 4103.457737405303

Highest Flow Link:
G41 -> I6: flow = 2437.500000 cars, dist = 0.091021 miles, car-miles = 221.863281

Lowest Flow Link:
I1 -> G12: flow = 58 cars, dist = 0.070805 miles, car-miles = 4.113766
```

Task 2 Results

The minimum total car-miles now increases by 7.36% due to the additional constraint put on the system. Likewise the highest and lowest flow links change as cars flow through different links in order to satisfy the additional constraint.

Task 3

Like with Task 2 and Task 1 subtasks, the initial code was copied over to initialize the necessary information.

In [21]:
```
# Task 3
# Trip demand
import numpy as np
import pulp

ncars = np.array([2143, 2473, 434, 526, 501, 437, 776, 221, 601, 0, 152])

parkingLots = ['x1', 'x2', 'x3', 'x4', 'x5', 'x6', 'x7', 'x8', 'x9', 'x10', 'x11']
exits = ['E1', 'E2', 'E3', 'E4', 'E5', 'E6', 'E7', 'E8']
exitGates = ['G11','G12','G13','G14','G15',
             'G21','G22','G23','G24',
             'G31','G41','G51','G61','G71',
             'G81','G91','G101','G102','G111']
intersections = ['I1','I2','I3','I4','I5','I6','I7']

gatesToLots = {
    'x1' : ['G11','G12','G13','G14','G15'],
    'x2' : ['G21','G22','G23','G24'],
    'x3' : ['G31'],
    'x4' : ['G41'],
    'x5' : ['G51'],
    'x6' : ['G61'],
    'x7' : ['G71'],
    'x8' : ['G81'],
    'x9' : ['G91'],
    'x10' : ['G101', 'G102'],
    'x11' : ['G111']
}

links = {  # distances in ft, same as Task 2
    ('G11', 'E1'): 775.43,
    ('G11', 'G12'): 541.21, ('G12', 'G11'): 541.21,
    ('G12', 'I1'): 373.85, ('I1', 'G12'): 373.85,
    ('I1', 'I3'): 803.52, ('I3', 'I1'): 803.52,
    ('G21', 'I1'): 438.15, ('I1', 'G21'): 438.15,
    ('G21', 'G22'): 445.70, ('G22', 'G21'): 445.70,
    ('G22', 'I2'): 431.13, ('I2', 'G22'): 431.13,
    ('I4', 'I2'): 797.21, ('I2', 'I4'): 797.21,
    ('G13', 'G71'): 355.78, ('G71', 'G13'): 355.78,
    ('G71', 'G14'): 111.09, ('G14', 'G71'): 111.09,
    ('G14', 'G15'): 440.80, ('G15', 'G14'): 440.80,
    ('G15', 'I3'): 237.47, ('I3', 'G15'): 237.47,
    ('I3', 'G23'): 426.98, ('G23', 'I3'): 426.98,
    ('I3', 'G31'): 426.98, ('G31', 'I3'): 426.98,
    ('G23', 'G24'): 439.11, ('G24', 'G23'): 439.11,
    ('G31', 'G51'): 439.11, ('G51', 'G31'): 439.11,
    ('G24', 'I4') : 429.69, ('I4', 'G24') : 429.69,
    ('G51', 'I4') : 429.69, ('I4', 'G51') : 429.69,
    ('I4', 'I7') : 1480.90, ('I7', 'I4') : 1480.90,
    ('I5', 'E8') : 793.77,
    ('I5', 'G81') : 642.31, ('G81', 'I5') : 642.31,
```

```
    ('G81', 'E5') : 646.96,
    ('I5', 'I6') : 905.49, ('I6', 'I5') : 905.49,
    ('I6', 'G91') : 710.46, ('G91', 'I6'): 710.46,
    ('G91', 'E6') : 580.36,
    ('I6', 'G41') : 480.59, ('G41', 'I6') : 480.59,
    ('G41', 'G61') : 344.30, ('G61', 'G41') : 344.30,
    ('G61', 'G101') : 219.16, ('G101', 'G61') : 219.16,
    ('G101', 'I7') : 260.59, ('I7', 'G101') : 260.59,
    ('I7', 'G102') : 355.96, ('G102', 'I7') : 355.96,
    ('G102', 'G111') : 355.04, ('G111', 'G102') : 355.04,
    ('G111', 'E7') : 610.02,
    ('I1', 'E2') : 700.45,
    ('I2', 'E3') : 693.98,
    ('I4', 'E4') : 327.93
}

# Lane capacity constant: Assignment says 450 cars an hour per lane
laneCap30 = 450 * 0.5  # 225 cars per lane per 30 minutes

#number of lanes in each direction
lanes = {
         ('G11', 'E1'): 1,
    ('G11', 'G12'): 1, ('G12', 'G11'): 1,
    ('G12', 'I1'): 1, ('I1', 'G12'): 1,
    ('I1', 'I3'): 2, ('I3', 'I1'): 2,
    ('G21', 'I1'): 2, ('I1', 'G21'): 2,
    ('G21', 'G22'): 2, ('G22', 'G21'): 2,
    ('G22', 'I2'): 2, ('I2', 'G22'): 2,
    ('I4', 'I2'): 2, ('I2', 'I4'): 2,
    ('G13', 'G71'): 1, ('G71', 'G13'): 1,
    ('G71', 'G14'): 1, ('G14', 'G71'): 1,
    ('G14', 'G15'): 1, ('G15', 'G14'): 1,
    ('G15', 'I3'): 1, ('I3', 'G15'): 1,
    ('I3', 'G23'): 1, ('G23', 'I3'): 1,
    ('I3', 'G31'): 1, ('G31', 'I3'): 1,
    ('G23', 'G24'): 1, ('G24', 'G23'): 1,
    ('G31', 'G51'): 1, ('G51', 'G31'): 1,
    ('G24', 'I4'): 1, ('I4', 'G24'): 1,
    ('G51', 'I4'): 1, ('I4', 'G51'): 1,
    ('I4', 'I7'): 2, ('I7', 'I4'): 2,
    ('I5', 'E8'): 1,
    ('I5', 'G81'): 1, ('G81', 'I5'): 1,
    ('G81', 'E5'): 1,
    ('I5', 'I6'): 1, ('I6', 'I5'): 1,
    ('I6', 'G91'): 2, ('G91', 'I6'): 2,
    ('G91', 'E6'): 2,
    ('I6', 'G41'): 1, ('G41', 'I6'): 1,
    ('G41', 'G61'): 1, ('G61', 'G41'): 1,
    ('G61', 'G101'): 1, ('G101', 'G61'): 1,
    ('G101', 'I7'): 1, ('I7', 'G101'): 1,
    ('I7', 'G102'): 1, ('G102', 'I7'): 1,
    ('G102', 'G111'): 1, ('G111', 'G102'): 1,
    ('G111', 'E7'): 1,
    ('I1', 'E2'): 2,
    ('I2', 'E3'): 2,
    ('I4', 'E4'): 1
}
#Creates a dictionary of all the link capacities
link_capacity = {}
for node in links.keys(): # Corrected from link_lengths_ft.keys()
    link_capacity[node] = lanes[node] * laneCap30 # Corrected from lanes_per_link[node]

#Creates a dictionary of all the decision variables (In this case the links).
decision_variables = {}
for (i, j) in links:
    decision_variables[(i, j)] = pulp.LpVariable(f"c{i}{j}", lowBound=0)

#Creates our linear programming problem
prob = pulp.LpProblem("MaximizeOutflow", pulp.LpMaximize)

prob += pulp.lpSum(var for (i, j), var in decision_variables.items() if j in exits)

# Source supply at gates, which is the amount of cars divided by the amount of gates lot has.
gateSupply = {
    'G11': 428.6,
    'G12': 428.6,
    'G13': 428.6,
    'G14': 428.6,
    'G15': 428.6,
```

```
    'G21': 618.25,
    'G22': 618.25,
    'G23': 618.25,
    'G24': 618.25,
    'G31': 434.0,
    'G41': 526.0,
    'G51': 501.0,
    'G61': 437.0,
    'G71': 776.0,
    'G81': 221.0,
    'G91': 601.0,
    'G101': 0.0,
    'G102': 0.0,
    'G111': 152.0
}

# Flow conservation:
# At intersections: inflow = outflow
# At gates: outflow - inflow <= supply (can't send more than parked cars)
for node in exitGates + intersections:
    inflows = [var for (i, j), var in decision_variables.items() if j == node]
    outflows = [var for (i, j), var in decision_variables.items() if i == node]

    if node in intersections:
        prob += (pulp.lpSum(inflows) == pulp.lpSum(outflows))
    else:
        prob += (pulp.lpSum(outflows) - pulp.lpSum(inflows) <= gateSupply[node])

# Capacity constraints on each link
for (i, j), var in decision_variables.items():
    prob += var <= link_capacity[(i, j)], f"cap_{i}_{j}"

prob.solve(pulp.PULP_CBC_CMD(msg=0))
print("Max cars exiting in 30 min:", pulp.value(prob.objective))

nonzero = {k: v.varValue for k, v in decision_variables.items() if v.varValue and v.varValue > 0}
sortedDescending = sorted(nonzero.items(), key=lambda x: x[1], reverse=True)
sortedAscending = sorted(nonzero.items(), key=lambda x: x[1])

print("\nHighest Flow Link:")
for (i, j), flow in sortedDescending[:5]:
    print(f"{i} -> {j}: flow = {flow:.0f} cars")

print("\nLowest Flow Link:")
for (i, j), flow in sortedAscending[:3]:
    print(f"{i} -> {j}: flow = {flow:.0f} cars")
```

```
Max cars exiting in 30 min: 2471.0

Highest Flow Link:
I3 -> I1: flow = 450 cars
G22 -> I2: flow = 450 cars
G91 -> E6: flow = 450 cars
I1 -> E2: flow = 450 cars
I2 -> E3: flow = 450 cars

Lowest Flow Link:
G101 -> I7: flow = 73 cars
I7 -> G102: flow = 73 cars
G102 -> G111: flow = 73 cars
```

Task 4

For task 4 we reused the code from task 3 and adjusted it to account for adding a lane in each direction on the road(link) we felt could most benefit from the additional flow.

In [24]:
```
#Task 4
import numpy as np
import pulp

ncars = np.array([2143, 2473, 434, 526, 501, 437, 776, 221, 601, 0, 152])

parkingLots = ['x1', 'x2', 'x3', 'x4', 'x5', 'x6', 'x7', 'x8', 'x9', 'x10', 'x11']
exits = ['E1', 'E2', 'E3', 'E4', 'E5', 'E6', 'E7', 'E8']
exitGates = ['G11','G12','G13','G14','G15',
             'G21','G22','G23','G24',
             'G31','G41','G51','G61','G71',
```

```
            'G81','G91','G101','G102','G111']
intersections = ['I1','I2','I3','I4','I5','I6','I7']

gatesToLots = {
    'x1' : ['G11','G12','G13','G14','G15'],
    'x2' : ['G21','G22','G23','G24'],
    'x3' : ['G31'],
    'x4' : ['G41'],
    'x5' : ['G51'],
    'x6' : ['G61'],
    'x7' : ['G71'],
    'x8' : ['G81'],
    'x9' : ['G91'],
    'x10' : ['G101', 'G102'],
    'x11' : ['G111']
}

links = {  # distances in ft, same as Task 2
    ('G11', 'E1'): 775.43,
    ('G11', 'G12'): 541.21, ('G12', 'G11'): 541.21,
    ('G12', 'I1'): 373.85, ('I1', 'G12'): 373.85,
    ('I1', 'I3'): 803.52, ('I3', 'I1'): 803.52,
    ('G21', 'I1'): 438.15, ('I1', 'G21'): 438.15,
    ('G21', 'G22'): 445.70, ('G22', 'G21'): 445.70,
    ('G22', 'I2'): 431.13, ('I2', 'G22'): 431.13,
    ('I4', 'I2'): 797.21, ('I2', 'I4'): 797.21,
    ('G13', 'G71'): 355.78, ('G71', 'G13'): 355.78,
    ('G71', 'G14'): 111.09, ('G14', 'G71'): 111.09,
    ('G14', 'G15'): 440.80, ('G15', 'G14'): 440.80,
    ('G15', 'I3'): 237.47, ('I3', 'G15'): 237.47,
    ('I3', 'G23'): 426.98, ('G23', 'I3'): 426.98,
    ('I3', 'G31'): 426.98, ('G31', 'I3'): 426.98,
    ('G23', 'G24'): 439.11, ('G24', 'G23'): 439.11,
    ('G31', 'G51'): 439.11, ('G51', 'G31'): 439.11,
    ('G24', 'I4') : 429.69, ('I4', 'G24') : 429.69,
    ('G51', 'I4') : 429.69, ('I4', 'G51') : 429.69,
    ('I4', 'I7') : 1480.90, ('I7', 'I4') : 1480.90,
    ('I5', 'E8') : 793.77,
    ('I5', 'G81') : 642.31, ('G81', 'I5') : 642.31,
    ('G81', 'E5') : 646.96,
    ('I5', 'I6') : 905.49, ('I6', 'I5') : 905.49,
    ('I6', 'G91') : 710.46, ('G91', 'I6'): 710.46,
    ('G91', 'E6') : 580.36,
    ('I6', 'G41') : 480.59, ('G41', 'I6') : 480.59,
    ('G41', 'G61') : 344.30, ('G61', 'G41') : 344.30,
    ('G61', 'G101') : 219.16, ('G101', 'G61') : 219.16,
    ('G101', 'I7') : 260.59, ('I7', 'G101') : 260.59,
    ('I7', 'G102') : 355.96, ('G102', 'I7') : 355.96,
    ('G102', 'G111') : 355.04, ('G111', 'G102') : 355.04,
    ('G111', 'E7') : 610.02,
    ('I1', 'E2') : 700.45,
    ('I2', 'E3') : 693.98,
    ('I4', 'E4') : 327.93
}

# Lane capacity constant: Assignment says 450 cars an hour per lane
laneCap30 = 450 * 0.5  # 225 cars per lane per 30 minutes

#number of lanes in each direction
lanes = {
         ('G11', 'E1'): 1,
    ('G11', 'G12'): 1, ('G12', 'G11'): 1,
    ('G12', 'I1'): 1, ('I1', 'G12'): 1,
    ('I1', 'I3'): 2, ('I3', 'I1'): 2,
    ('G21', 'I1'): 2, ('I1', 'G21'): 2,
    ('G21', 'G22'): 2, ('G22', 'G21'): 2,
    ('G22', 'I2'): 2, ('I2', 'G22'): 2,
    ('I4', 'I2'): 2, ('I2', 'I4'): 2,
    ('G13', 'G71'): 1, ('G71', 'G13'): 1,
    ('G71', 'G14'): 1, ('G14', 'G71'): 1,
    ('G14', 'G15'): 1, ('G15', 'G14'): 1,
    ('G15', 'I3'): 1, ('I3', 'G15'): 1,
    ('I3', 'G23'): 1, ('G23', 'I3'): 1,
    ('I3', 'G31'): 1, ('G31', 'I3'): 1,
    ('G23', 'G24'): 1, ('G24', 'G23'): 1,
    ('G31', 'G51'): 1, ('G51', 'G31'): 1,
    ('G24', 'I4'): 1, ('I4', 'G24'): 1,
    ('G51', 'I4'): 1, ('I4', 'G51'): 1,
    ('I4', 'I7'): 2, ('I7', 'I4'): 2,
```

```
    ('I5', 'E8'): 1,
    ('I5', 'G81'): 1, ('G81', 'I5'): 1,
    ('G81', 'E5'): 1,
    ('I5', 'I6'): 1, ('I6', 'I5'): 1,
    ('I6', 'G91'): 2, ('G91', 'I6'): 2,
    ('G91', 'E6'): 2,
    ('I6', 'G41'): 1, ('G41', 'I6'): 1,
    ('G41', 'G61'): 1, ('G61', 'G41'): 1,
    ('G61', 'G101'): 1, ('G101', 'G61'): 1,
    ('G101', 'I7'): 1, ('I7', 'G101'): 1,
    ('I7', 'G102'): 1, ('G102', 'I7'): 1,
    ('G102', 'G111'): 1, ('G111', 'G102'): 1,
    ('G111', 'E7'): 1,
    ('I1', 'E2'): 3, #increase to 3 lanes instead of 2
    ('I2', 'E3'): 2,
    ('I4', 'E4'): 1
}
#Creates a dictionary of all the link capacities
link_capacity = {}
for node in links.keys(): # Corrected from link_lengths_ft.keys()
    link_capacity[node] = lanes[node] * laneCap30 # Corrected from lanes_per_link[node]

#Creates a dictionary of all the decision variables (In this case the links).
decision_variables = {}
for (i, j) in links:
    decision_variables[(i, j)] = pulp.LpVariable(f"c{i}{j}", lowBound=0)

#Creates our linear programming problem
prob = pulp.LpProblem("MaximizeOutflow", pulp.LpMaximize)

prob += pulp.lpSum(var for (i, j), var in decision_variables.items() if j in exits)

# Source supply at gates, which is the amount of cars divided by the amount of gates lot has.
gateSupply = {
    'G11': 428.6,
    'G12': 428.6,
    'G13': 428.6,
    'G14': 428.6,
    'G15': 428.6,
    'G21': 618.25,
    'G22': 618.25,
    'G23': 618.25,
    'G24': 618.25,
    'G31': 434.0,
    'G41': 526.0,
    'G51': 501.0,
    'G61': 437.0,
    'G71': 776.0,
    'G81': 221.0,
    'G91': 601.0,
    'G101': 0.0,
    'G102': 0.0,
    'G111': 152.0
}

# Flow conservation:
# At intersections: inflow = outflow
# At gates: outflow - inflow <= supply (can't send more than parked cars)
for node in exitGates + intersections:
    inflows = [var for (i, j), var in decision_variables.items() if j == node]
    outflows = [var for (i, j), var in decision_variables.items() if i == node]

    if node in intersections:
        prob += (pulp.lpSum(inflows) == pulp.lpSum(outflows))
    else:
        prob += (pulp.lpSum(outflows) - pulp.lpSum(inflows) <= gateSupply[node])

# Capacity constraints on each link
for (i, j), var in decision_variables.items():
    prob += var <= link_capacity[(i, j)], f"cap_{i}_{j}"

prob.solve(pulp.PULP_CBC_CMD(msg=0)) #solve LP
print("Max cars exiting in 30 min:", pulp.value(prob.objective)) #print out objective value from LP

nonzero = {k: v.varValue for k, v in decision_variables.items() if v.varValue and v.varValue > 0}
#loops through decision variables dictionary and puts nonzero values into new dictionary `nonzero`
sortedDescending = sorted(nonzero.items(), key=lambda x: x[1], reverse=True)
#sorts values in nonzero in descending order hence `reverse=True`
sortedAscending = sorted(nonzero.items(), key=lambda x: x[1]) #sorts values in nonzero in ascending order
```

```
print("\nHighest Flow Link:")
for (i, j), flow in sortedDescending[:5]: #for loop to show highest flow links; top 5 included as they all
                                          #have same value except for the one we added a lane too
    print(f"{i} -> {j}: flow = {flow:.0f} cars")
#print statement to show output also rounds to nearest car since we cannot have decimal values of cars

print("\nLowest Flow Link:")
for (i, j), flow in sortedAscending[:3]:
#for loop to organize lowest flow links; bottom 3 included as they all have the same value
    print(f"{i} -> {j}: flow = {flow:.0f} cars") #same as above for highest flow link

print(f"\nCost of adding one lane to specified link: ${links['I1', 'E2'] * 120:.2f}")
#cost of adding an additional lane; all costs rounded to 2 decimals
print(f"Economic Benefit from additional lane: ${(2696-2471)*4*150:.2f}")
#difference in expected revenue for different values of cars leaving in 30 minutes
print(f"Expected Revenue: ${(2696-2471)*4*150 - links['I1', 'E2'] * 120:.2f}")
#revenue from number of cars exiting in 30 min minus cost of additional lane
print(f"Original benefit: ${2471*4*150:.2f}, New benefit: ${2696*4*140 + ((2696-2471)*4*150 - links['I1', 'E2'] * 120):.2
#original and new benefits as calculated using max number of cars leaving in a 30 minute period
print(f"Difference in revenue after adding additional lane: ${1560706.00-1482600.00:.2f}")
#calculating the difference in benefits and rounding to 2 decimals
```

```
Max cars exiting in 30 min: 2696.0

Highest Flow Link:
I1 -> E2: flow = 675 cars
I3 -> I1: flow = 450 cars
G22 -> I2: flow = 450 cars
G91 -> E6: flow = 450 cars
I2 -> E3: flow = 450 cars

Lowest Flow Link:
G101 -> I7: flow = 73 cars
I7 -> G102: flow = 73 cars
G102 -> G111: flow = 73 cars

Cost of adding one lane to specified link: $84054.00
Economic Benefit from additional lane: $135000.00
Expected Revenue: $50946.00
Original benefit: $1482600.00, New benefit: $1560706.00
Difference in revenue after adding additional lane: $78106.00
```

Task 4 part 3

For part 3 we added an additional lane to the same link and then recalculated the economic benefits.

In [27]:
```
#Task 4
import numpy as np
import pulp

ncars = np.array([2143, 2473, 434, 526, 501, 437, 776, 221, 601, 0, 152])

parkingLots = ['x1', 'x2', 'x3', 'x4', 'x5', 'x6', 'x7', 'x8', 'x9', 'x10', 'x11']
exits = ['E1', 'E2', 'E3', 'E4', 'E5', 'E6', 'E7', 'E8']
exitGates = ['G11','G12','G13','G14','G15',
             'G21','G22','G23','G24',
             'G31','G41','G51','G61','G71',
             'G81','G91','G101','G102','G111']
intersections = ['I1','I2','I3','I4','I5','I6','I7']

gatesToLots = {
    'x1' : ['G11','G12','G13','G14','G15'],
    'x2' : ['G21','G22','G23','G24'],
    'x3' : ['G31'],
    'x4' : ['G41'],
    'x5' : ['G51'],
    'x6' : ['G61'],
    'x7' : ['G71'],
    'x8' : ['G81'],
    'x9' : ['G91'],
    'x10' : ['G101', 'G102'],
    'x11' : ['G111']
}

links = {  # distances in ft, same as Task 2
    ('G11', 'E1'): 775.43,
    ('G11', 'G12'): 541.21, ('G12', 'G11'): 541.21,
```

```
    ('G12', 'I1'): 373.85, ('I1', 'G12'): 373.85,
    ('I1', 'I3'): 803.52, ('I3', 'I1'): 803.52,
    ('G21', 'I1'): 438.15, ('I1', 'G21'): 438.15,
    ('G21', 'G22'): 445.70, ('G22', 'G21'): 445.70,
    ('G22', 'I2'): 431.13, ('I2', 'G22'): 431.13,
    ('I4', 'I2'): 797.21, ('I2', 'I4'): 797.21,
    ('G13', 'G71'): 355.78, ('G71', 'G13'): 355.78,
    ('G71', 'G14'): 111.09, ('G14', 'G71'): 111.09,
    ('G14', 'G15'): 440.80, ('G15', 'G14'): 440.80,
    ('G15', 'I3'): 237.47, ('I3', 'G15'): 237.47,
    ('I3', 'G23'): 426.98, ('G23', 'I3'): 426.98,
    ('I3', 'G31'): 426.98, ('G31', 'I3'): 426.98,
    ('G23', 'G24'): 439.11, ('G24', 'G23'): 439.11,
    ('G31', 'G51'): 439.11, ('G51', 'G31'): 439.11,
    ('G24', 'I4') : 429.69, ('I4', 'G24') : 429.69,
    ('G51', 'I4') : 429.69, ('I4', 'G51') : 429.69,
    ('I4', 'I7') : 1480.90, ('I7', 'I4') : 1480.90,
    ('I5', 'E8') : 793.77,
    ('I5', 'G81') : 642.31, ('G81', 'I5') : 642.31,
    ('G81', 'E5') : 646.96,
    ('I5', 'I6') : 905.49, ('I6', 'I5') : 905.49,
    ('I6', 'G91') : 710.46, ('G91', 'I6'): 710.46,
    ('G91', 'E6') : 580.36,
    ('I6', 'G41') : 480.59, ('G41', 'I6') : 480.59,
    ('G41', 'G61') : 344.30, ('G61', 'G41') : 344.30,
    ('G61', 'G101') : 219.16, ('G101', 'G61') : 219.16,
    ('G101', 'I7') : 260.59, ('I7', 'G101') : 260.59,
    ('I7', 'G102') : 355.96, ('G102', 'I7') : 355.96,
    ('G102', 'G111') : 355.04, ('G111', 'G102') : 355.04,
    ('G111', 'E7') : 610.02,
    ('I1', 'E2') : 700.45,
    ('I2', 'E3') : 693.98,
    ('I4', 'E4') : 327.93
}

# Lane capacity constant: Assignment says 450 cars an hour per lane
laneCap30 = 450 * 0.5  # 225 cars per lane per 30 minutes

#number of lanes in each direction
lanes = {
        ('G11', 'E1'): 1,
    ('G11', 'G12'): 1, ('G12', 'G11'): 1,
    ('G12', 'I1'): 1, ('I1', 'G12'): 1,
    ('I1', 'I3'): 2, ('I3', 'I1'): 2,
    ('G21', 'I1'): 2, ('I1', 'G21'): 2,
    ('G21', 'G22'): 2, ('G22', 'G21'): 2,
    ('G22', 'I2'): 2, ('I2', 'G22'): 2,
    ('I4', 'I2'): 2, ('I2', 'I4'): 2,
    ('G13', 'G71'): 1, ('G71', 'G13'): 1,
    ('G71', 'G14'): 1, ('G14', 'G71'): 1,
    ('G14', 'G15'): 1, ('G15', 'G14'): 1,
    ('G15', 'I3'): 1, ('I3', 'G15'): 1,
    ('I3', 'G23'): 1, ('G23', 'I3'): 1,
    ('I3', 'G31'): 1, ('G31', 'I3'): 1,
    ('G23', 'G24'): 1, ('G24', 'G23'): 1,
    ('G31', 'G51'): 1, ('G51', 'G31'): 1,
    ('G24', 'I4'): 1, ('I4', 'G24'): 1,
    ('G51', 'I4'): 1, ('I4', 'G51'): 1,
    ('I4', 'I7'): 2, ('I7', 'I4'): 2,
    ('I5', 'E8'): 1,
    ('I5', 'G81'): 1, ('G81', 'I5'): 1,
    ('G81', 'E5'): 1,
    ('I5', 'I6'): 1, ('I6', 'I5'): 1,
    ('I6', 'G91'): 2, ('G91', 'I6'): 2,
    ('G91', 'E6'): 2,
    ('I6', 'G41'): 1, ('G41', 'I6'): 1,
    ('G41', 'G61'): 1, ('G61', 'G41'): 1,
    ('G61', 'G101'): 1, ('G101', 'G61'): 1,
    ('G101', 'I7'): 1, ('I7', 'G101'): 1,
    ('I7', 'G102'): 1, ('G102', 'I7'): 1,
    ('G102', 'G111'): 1, ('G111', 'G102'): 1,
    ('G111', 'E7'): 1,
    ('I1', 'E2'): 4, #increase to 4 lanes instead of 3 from task 4 parts 1 and 2
    ('I2', 'E3'): 2,
    ('I4', 'E4'): 1
}
#Creates a dictionary of all the link capacities
link_capacity = {}
for node in links.keys(): # Corrected from link_lengths_ft.keys()
```

```
    link_capacity[node] = lanes[node] * laneCap30 # Corrected from lanes_per_link[node]

#Creates a dictionary of all the decision variables (In this case the links).
decision_variables = {}
for (i, j) in links:
    decision_variables[(i, j)] = pulp.LpVariable(f"c{i}{j}", lowBound=0)

#Creates our linear programming problem
prob = pulp.LpProblem("MaximizeOutflow", pulp.LpMaximize)

prob += pulp.lpSum(var for (i, j), var in decision_variables.items() if j in exits)

# Source supply at gates, which is the amount of cars divided by the amount of gates lot has.
gateSupply = {
    'G11': 428.6,'G12': 428.6,'G13': 428.6,'G14': 428.6,'G15': 428.6,'G21': 618.25,'G22': 618.25,'G23': 618.25,
    'G24': 618.25,'G31': 434.0,'G41': 526.0,'G51': 501.0,'G61': 437.0,'G71': 776.0,'G81': 221.0,'G91': 601.0,
    'G101': 0.0,'G102': 0.0,'G111': 152.0
}

# Flow conservation:
# At intersections: inflow = outflow
# At gates: outflow - inflow <= supply (can't send more than parked cars)
for node in exitGates + intersections:
    inflows = [var for (i, j), var in decision_variables.items() if j == node]
    outflows = [var for (i, j), var in decision_variables.items() if i == node]

    if node in intersections:
        prob += (pulp.lpSum(inflows) == pulp.lpSum(outflows))
    else:
        prob += (pulp.lpSum(outflows) - pulp.lpSum(inflows) <= gateSupply[node])

# Capacity constraints on each link
for (i, j), var in decision_variables.items():
    prob += var <= link_capacity[(i, j)], f"cap_{i}_{j}"

prob.solve(pulp.PULP_CBC_CMD(msg=0))
print("Max cars exiting in 30 min:", pulp.value(prob.objective))

print(f"\nCost of adding one lane to specified link: ${links['I1', 'E2'] * 2 * 120:.2f}")
print(f"Economic Benefit from additional lane: ${(2921-2696)*4*150:.2f}")
print(f"Expected Revenue: ${(2696-2471)*4*150 - links['I1', 'E2'] * 2 * 120:.2f}")
```

```
Max cars exiting in 30 min: 2921.0

Cost of adding one lane to specified link: $168108.00
Economic Benefit from additional lane: $135000.00
Expected Revenue: $-33108.00
```

In [ ]: