CE191: Civil and Environmental Engineering Systems Analysis

Lab 3

The Travelling Salesman Problem

Due: March 26, 11:59pm

The Travelling Salesman Problem

You are the manager of a construction materials delivery company. Each day, a driver must deliver materials from the distribution center to each of the customer locations before returning to the distribution center, without travelling to any of the customer locations more than once. Your goal is to determine the order in which to travel to the customer locations such that the deliveries are completed as quickly as possible.

Figure 1 shows a graph representing the road network connecting the distribution center to the customer locations. Vertex 1 represents the distribution center, and the remaining vertices in the graph represent the customer locations. The edges in the graph represent the roads, and the costs associated with each edge represent the time it takes to drive that road. The graph is directed, meaning it may take different amounts time to travel a road in each direction.

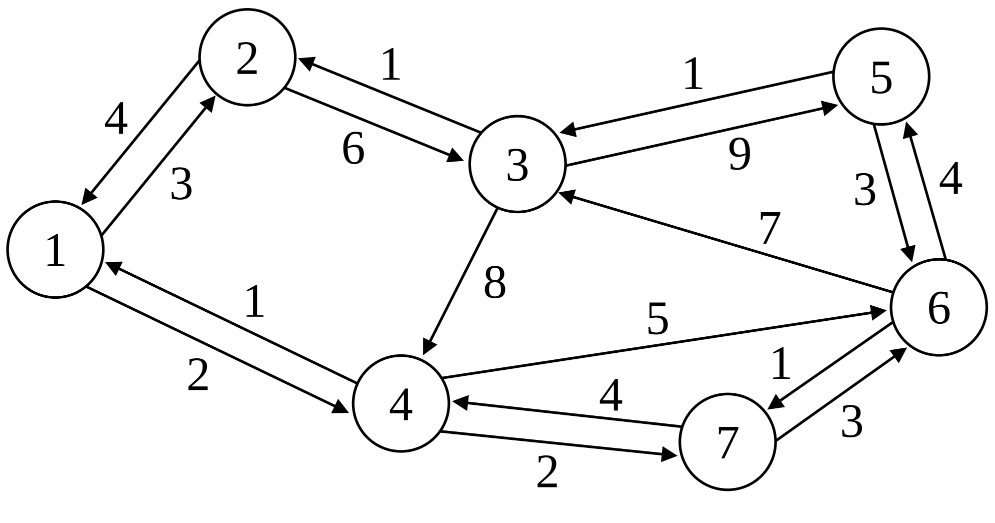


Figure 1: A directed graph representing a road network with a distribution center and 6 customer locations.

Part 1 (6 points)

Q1.a Visualize the Graph

Write a function `visGraph(edges)` to visualize the road network given its edge list. The input to the function is an edge list `edges`. The variable `edges` is a matrix representing the edges in the graph. If the graph has an edge from i to j with cost c, the corresponding row in `edges` is [i j c]. The order of the rows in `edges` is arbitrary. Load the graphs from `graph1.mat`, `graph2.mat`, `graph3.mat`, `graph4.mat`, `graph5.mat` and `graph6.mat`, plot these 6 networks using `visGraph`.

Hint: Use the Matlab function `biograph(CMatrix)`. The input to `biograph` is a connection matrix `CMatrix`. `CMatrix(i,j)` is 0 if there is no link from i to j. `CMatrix(i,j)` is c if there exists an edge from i to j with cost c.
The figure for `graph_ex.mat` should be similar to the figure below:

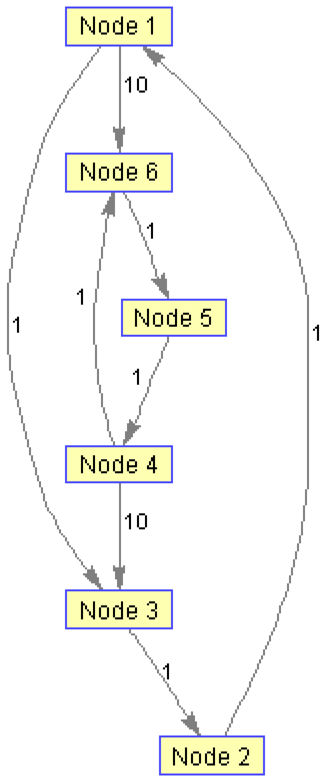


Figure 2

Q1.b Formulate the linear relaxation of subtour relaxation of TSP

Write a function `prob = formLP(graph)` that formulates a linear program that can be used to solve the subtour relaxation of the asymmetric travelling salesman problem (TSP). The input to the function is a structure `graph` with an edge list field `graph.edges`. The output of the function is a structure `prob` that must have the fields `f`, `Aeq`, `beq`, `lb`, and `ub`, which are the inputs to `linprog`, and can have additional fields that are not listed here. You must use one decision variable per edge in the graph, and they must be in the same order as in `graph.edges`. For efficiency, pass lower and upper bound constraints to `linprog` using `lb` and `ub`, not using `A` and `b`. The function should be written to work in the general case, for an arbitrary directed graph.

To test your function, load the graph from `graph1.mat`, run your function, then compare your results to the test case results in `prob1test1.mat`. To match up with prob1.mat exactly you need to write the all the outgoing constraints before all the incoming constraints and if i < j you need to write the constraint for node i before the one for j.

Part 2 (6 points)

Q2.a Solve the linear relaxation of subtour relaxation TSP
Write a function `prob = solveLP(prob)` that solves a linear program relaxation of the asymmetric TSP. The input `prob` is the output of `formLP`. The output of the function is `prob`, a structure with the same fields as the input `prob`, and in addition, the fields `sol`, `cost`, `sol_edges, isFeasible`, and `hasSubtours`. As in Problem 1, the output structure can have additional fields that are not listed.

The field `prob.sol` is the solution to the linear program, `prob.sol_edges` is the edge list for the solution, and `prob.cost` is the corresponding cost. The field `prob.isFeasible` is true if the linear program is feasible and false otherwise. In the case where `prob.isFeasible` is true, `prob.hasSubtours` is true if the solution contains more than one tour, and false otherwise. If `prob.isFeasible` is false, `prob.hasSubtours`, `prob.sol`, `prob.sol_edges` and `prob.cost` are empty. The function should be written to work in the general case, for an arbitrary directed graph.

Hint: To test your function, load the graph from `graph1.mat`, formulate the linear program using `formLP`, then solve the linear program using `solveLP`, then compare your results to the test case results in `prob2test1.mat`. Similarly, the test case results for `graph2.mat` are in `prob2test2.mat`, and the test case results for `graph3.mat` are in `prob2test3.mat`.

Q2.b Visualize your solution to the linear relaxation of subtour relaxation TSP
Use the function `visGraph` to visualize the solution to the linear relaxation of subtour relaxation TSP. Load the graph from `graph1.mat`, `graph2.mat`, `graph3.mat`, `graph4.mat`, `graph5.mat` and `graph6.mat`, formulate the linear program using `formLP`, then solve the linear program using `solveLP`, then visualize your solution with `visGraph`.

For example, the graph `graph_ex.mat`, processed through `formLP`, and `solveLP`, produces the figure below:

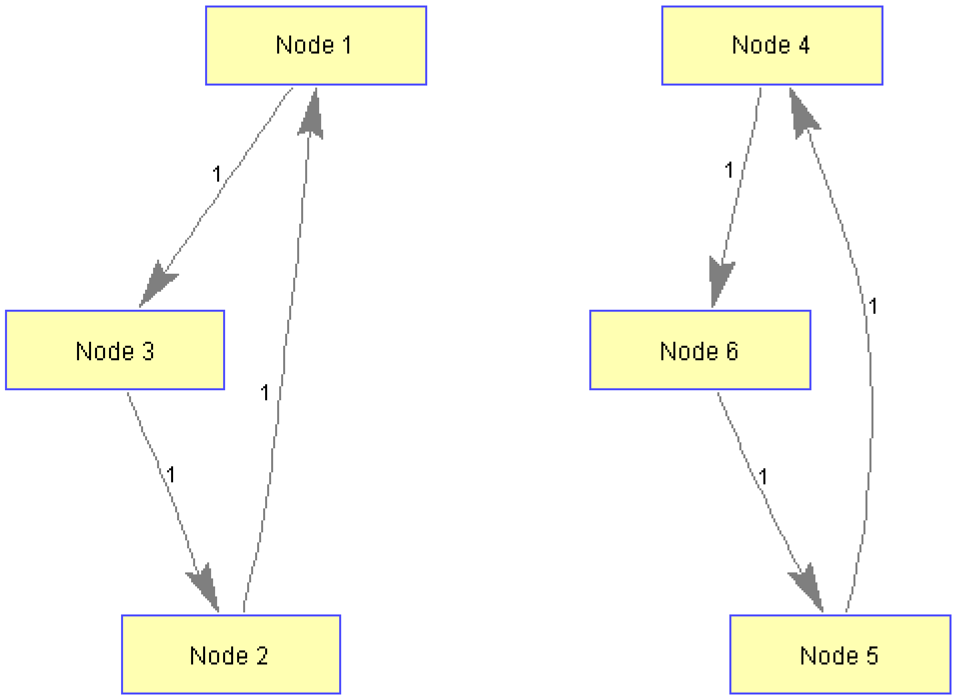


Figure 3

Part 3 (8 points)

Q3. Solve TSP with branch and bound algorithm
Write a function `[tour,cost] = solveTSP(graph)` that solves the asymmetric TSP using a branch and bound algorithm. The input `graph` is the same as in Problem 1. The outputs of the function are `tour`, a vector containing the vertices in the optimal tour in the order they should be travelled, and `cost`, the corresponding optimal cost. The function should be written to work in the general case, for an arbitrary directed graph. You can assume that `graph` contains the appropriate edges such that there exists a solution to the asymmetric TSP.

Load the graph from `graph1.mat`, `graph2.mat`, `graph3.mat`, `graph4.mat`, `graph5.mat` and `graph6.mat`, then solve the TSP using `solveTSP`, then visualize your solution with `visGraph`.

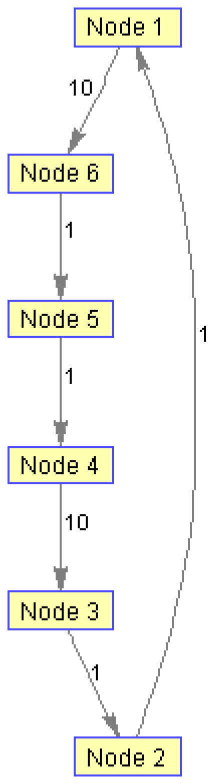
Hint1: Read section 5.2.1 of the following article:
http://web.tecnico.ulisboa.pt/~mcasquilho/acad/or/TSP/LUTch5_2.pdf

Hint2: Before write `solveTSP`, perform the following example:

Load the graph from `graph_ex.mat`. First solve the linear relaxation of subtour relaxation TSP using `prog0=solveLP(prog,graph)`. *Figure 3* shows `prog0` contains subtours. Thus, we apply branch and bound to `prog0`. We can generate 3 branches:

- Generate `prog1` by adding the new constraint x13=0,
- Generate `prog2` by adding the new constraint x32=0
- Generate `prog3` by adding the new constraint x21=0.

Solve `prog1`, `prog2` and `prog3` separately using `solveLP`. You will see that `prog2` and `prog3` are infeasible, and `prog1` leads to the optimal solution. Visualize your solution `prog1` with `visGraph`. It should be similar to the figure below:



Hint3: Test Cases:

```
>> load('graph1.mat')
>> [tour,cost] = solveTSP(graph)
tour =
     1     4     7     6     5     3     2     1
cost =
     17
>> load('graph4.mat')
>> [tour,cost] = solveTSP(graph)
tour =
     1     4     2     5     3     1
cost =
     261
```

Program Design

This assignment is an exercise in the design of a complicated computer program. To keep your code organized, you will probably want to write other functions that are not listed in the assignment. If you do, you must submit them along with the m-files listed in the Deliverables section. The following suggestions are meant to help you design your code, but you are not required follow them.

- Overall, the application of branch and bound in this problem is similar to examples discussed in class. The TSP has its own branching and bounding strategies.
- There are many possible branching strategies. You may use any strategy you choose, as long as it yields the correct result. ***You should choose a branching strategy that forces at least one edge in at least one subtour to not be included in the solution***. How you choose the edge(s) and subtour(s) is up to you.
- The branch and bound portion of your code can be written using the recursive approach illustrated in lecture, or using an iterative approach. If you use the recursive approach, a function other than `solveTSP` may need to be the recursive function.
- You may want to write a function that takes a problem to be branched upon, adds the necessary constraints, then returns the resulting subproblems.
- You may want to write a function that takes a solved problem and converts the solution into the corresponding list of vertices along the tour(s).

Extra credit

You can earn extra credit by submitting early.

- 1 point: submit report and code for problems 1 and 2 by March 5, 11:59pm
- 1 point: submit report and code for all problems by March 12, 11:59pm

Deliverables

Submit your report on bCourses:

FIRSTNAME_LASTNAME_LAB3.pdf

Submit the following m-files on bCourse. Be sure that the filenames and function declarations are exactly as specified, including spelling and case. If you wrote any other functions that are used by these m-files, you must submit them as well.

```
visGraph.m
formLP.m
solveLP.m
solveTSP.m
```