Homework Assignment 2

Fall 2022

IE 532 Analysis of Network Data Due on Friday 10/14 (at 11:59 pm)

There are 4 questions in this assignment.

Please answer every question to the best of your knowledge and make sure to show your work for partial credit.

Make sure to cite your sources, if you decide to use more material to help you solve the exercises.

Remember that collaboration between students is allowed and encouraged, but please give your collaborators proper reference by letting me know who you worked with and what they contributed.

Only electronic submissions are accepted for this assignment.

You may either type your answers using LaTeX or other word processing software, or scan your handwritten answers and submit them. If the latter, please make sure the scanned copy is legible.

If submitting multiple files, create a folder and compress it (in either .zip, .rar, or .7z format, among others) before submitting.

Question 1: Scheduling and shortest paths

(a) When timetabling (scheduling shifts), we are interested in having at least one available resource at each of the operating hours. For example, in a bus company, we may be interested in having at least one available driver at each hour of the operations for a specific bus route/line. As an example, we offer Table 1, which shows all possible shifts for a bus company. Say, we need to ensure that at least one driver is on duty at each of the shifts (from 9am to 5pm) at the total lowest cost.

Design this scheduling problem as a shortest path problem and solve it with an appropriate algorithm. Explain why you chose the algorithm you did. The example is offered to help you check your approach; that said, please try to offer how to create a shortest path problem for a general case with multiple available shifts.

Hours	9am-1pm	9am-11am	noon-3pm	noon-5pm	2pm-5pm	1pm-4pm	4pm-5pm
Cost	30	18	21	38	20	22	9

Table 1: Duty hours for a bus company.

(b) A company needs to schedule the following tasks (shown in Table 2), numbered from 1 to 7, each of which has a duration and a list of prerequisite tasks that need to have been completed before they are started. For example, Task 4 will take 10 hours and needs Tasks 2 and 3 to be over before it is taken up.

Task	Duration	Prerequisites
Task 1	8 hrs	None
Task 2	4 hrs	Task 1
Task 3	3 hrs	Task 1
Task 4	10 hrs	Tasks 2, 3
Task 5	5 hrs	Task 2
Task 6	4 hrs	Task 3
Task 7	3 hrs	Tasks 4, 5, 6

Table 2: The tasks and their details.

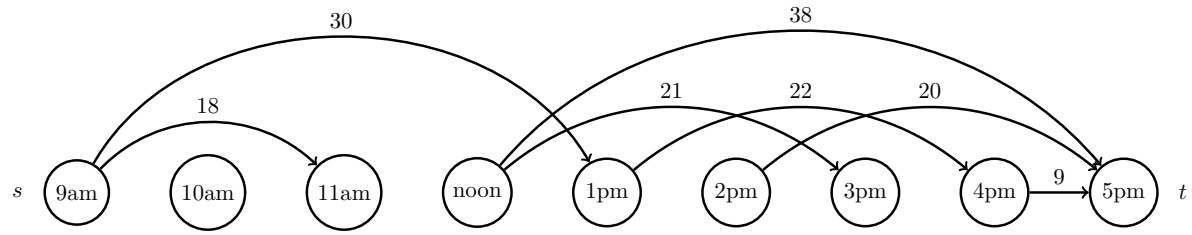
Let t_i be a decision variable that signals the *starting time* of Task i, and assume that there is a fake "starting task", Task 0, with a duration of 0 hours, that starts at time 0, and a fake "ending task", Task 8, with a duration of 0 hours, and a starting time of t_8. Clearly, the smallest the time t_8 the earliest the ending of the overall project, so our problem would have the goal of min t_8. Formulate the problem of minimizing the duration of the whole project, while satisfying all the prerequisite requirements. Then, using the theory discussed during Lecture 10, formulate its dual. Can it be written as a shortest path problem? And, if so, recommend a method that solves the problem.

Answer (a) Let us create one node for every hour in our schedule. That is, we have

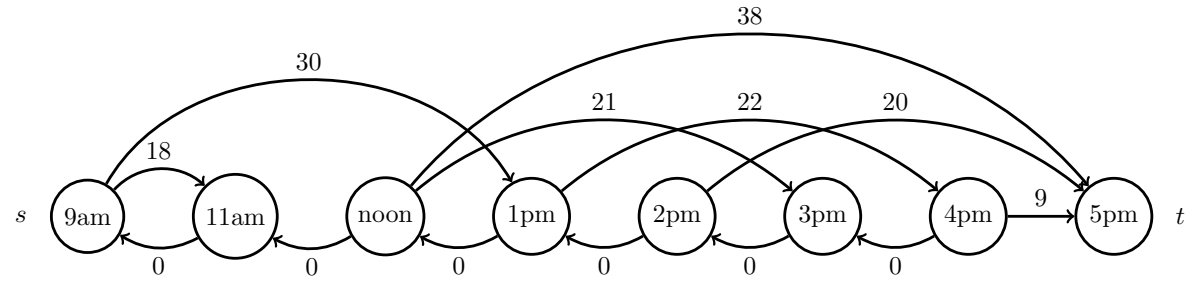
$$\{\text{9am, 10am, 11am, noon, 1pm, 2pm, 3pm, 4pm, 5pm}\}.$$

Our source node would be $9am$ and our terminal node would be $5pm$.

For the arcs, consider one arc for every shift: hence we will have an arc (9am, 1pm) with a cost of 30, an arc (9am, 11am) with a cost of 18, and so on. This gives a total of 7 arcs.



First, let us remove the unnecessary nodes (for example, 10am). In addition to that, we will connect every node in the schedule to all nodes that are earlier than it. For example, we may connect 11am to 9am, noon to 9am, 1pm to noon, 11am, and 9am, etc. All in all, our network becomes as follows:



(b) Let $t_i \geq 0$ be the only (and continuous) decision variable of our model. Additionally, there is only one type of constraint: namely that task j cannot begin until after task i has been completed, whenever i is a prerequisite of j. In mathematical terms, we have:

$$t_j \geq t_i + d_i,$$

where d_i is the duration of Task i. Overall, assuming we have n tasks to complete, we will have:

$$\begin{aligned} \min \quad & t_{n+1} - t_0 \\ s.t. \quad & t_j - t_i \geq d_i, \qquad \forall\,(i,j) : j \text{ is a prerequisite for } i, \forall i = 0, 1, \ldots, n+1. \end{aligned}$$

This should happen for all prerequisite requirements, hence in our given example, we will have 9 constraints (for the regular tasks) and 2 more constraints (for the two fake tasks). The formulation for the toy example would be:

$$
\begin{array}{llllllllllll}
\min & -t_0 & & & & & & & & & t_8 & \\
s.t. & -t_0 & +t_1 & & & & & & & & & \geq 0 \\
 & & -t_1 & +t_2 & & & & & & & & \geq 8 \\
 & & -t_1 & & +t_3 & & & & & & & \geq 8 \\
 & & & -t_2 & & +t_4 & & & & & & \geq 4 \\
 & & & & -t_3 & +t_4 & & & & & & \geq 3 \\
 & & & -t_2 & & & +t_5 & & & & & \geq 4 \\
 & & & & -t_3 & & & +t_6 & & & & \geq 3 \\
 & & & & & -t_4 & & & +t_7 & & & \geq 10 \\
 & & & & & & -t_5 & & +t_7 & & & \geq 5 \\
 & & & & & & & -t_6 & +t_7 & & & \geq 4 \\
 & & & & & & & & -t_7 & +t_8 & & \geq 3
\end{array}
$$

We observe in the formulation before that the primal problem is **not** easily transformed to a network problem; for that to be the case, we would expect the constraint matrix to have an interesting form, for example have each column contain exactly one "+1" and one "-1" terms. This is not true for the columns, but it is true for the rows!

Assign to each constraint a dual variable v_{ij}, i referring the negative term in the constraint and j to the positive term. For example, the first constraint would be associated with a dual variable of v_{01}, the second constraint with a dual variable of v_{12}, and so on. We are now ready to formulate the dual problem, as in formulation (3).

$$
\begin{array}{llllllllllll}
\max & & 8v_{12} & +8v_{13} & +4v_{24} & +3v_{34} & +4v_{25} & +3v_{36} & +10v_{47} & +5v_{57} & +4v_{67} & +3v_{78} & & (3a) \\
s.t. & -v_{01} & & & & & & & & & & & = -1 & (3b) \\
 & v_{01} & -v_{12} & -v_{13} & & & & & & & & & = 0 & (3c) \\
 & & v_{12} & & -v_{24} & & -v_{25} & & & & & & = 0 & (3d) \\
 & & & v_{13} & & -v_{34} & & -v_{36} & & & & & = 0 & (3e) \\
 & & & & v_{24} & +v_{34} & & & -v_{47} & & & & = 0 & (3f) \\
 & & & & & & v_{25} & & & -v_{57} & & & = 0 & (3g) \\
 & & & & & & & v_{36} & & & -v_{67} & & = 0 & (3h) \\
 & & & & & & & & v_{47} & +v_{57} & +v_{67} & -v_{78} & = 0 & (3i) \\
 & & & & & & & & & & & v_{78} & = 1 & (3j)
\end{array}
$$

The above constraint matrix does indeed represent a network, where every column represents exactly one arc and every row a node. Letting the nodes be numbered $0, \ldots, 8$, we get the network of Figure 1.

Finally, let us go back to the (dual) formulation and its associated network. It states that 1 unit exits node 0 to go to node 1, and 1 unit enters node 8 from node 7: overall we are interested in choosing the *longest* path! As stated in class, shortest paths are easy and solved in polynomial time. The longest path, on the other hand, is notoriously hard in general graphs.

Observing the graph, we think of a nice way to turn the longest path at hand to a shortest path problem: why not turn all arcs into negative costs and solve it as a shortest path problem? Unfortunately, negative costs may not always work: Dijkstra's cannot be used with negative costs, label-correcting algorithms cannot be used with negative cycles.. That said, under the assumption that no cycle exists, we have options!

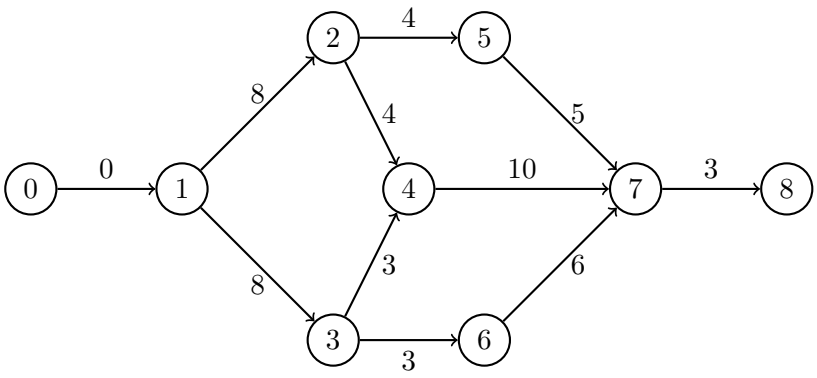


Figure 1: The network of the exercise.

In our case, let us consider what a cycle $(i_1, i_2, \ldots, i_k, i_1)$ would mean. It would imply that Task i_1 serves as a prerequisite for Task i_2, which in turn serves as a prerequisite for the next node in the cycle, all the way to Task i_k which would serve as a prerequisite for Task i_1. This implies that Task i_1 would be a prerequisite for Task i_1, which is clearly impossible in the context of task management!

Overall, the network is **acyclic** and as such turning all costs negative would still allow us to solve a shortest path problem using a label-correcting algorithm. Even better, we can employ topological sorting, and solve this faster!

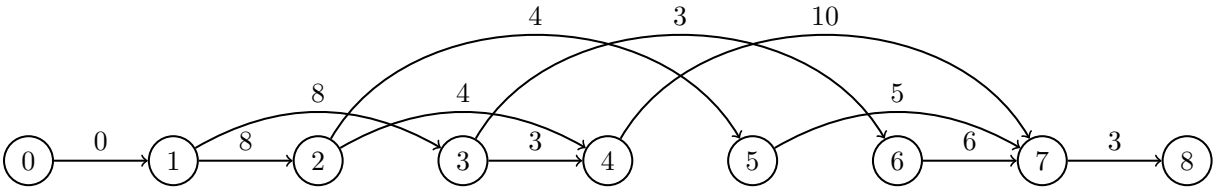


Figure 2: The topological sorting.

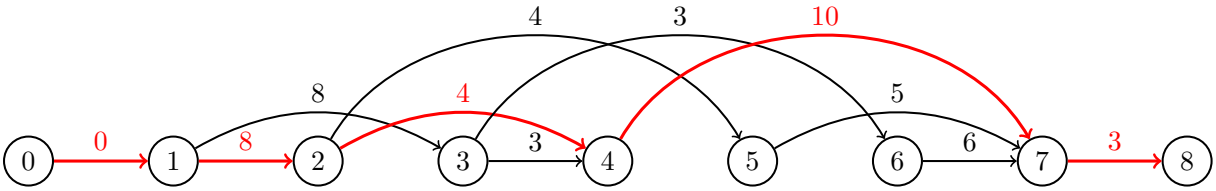


Figure 3: The "longest" path after topologically sorting.

The difference is that now we are looking for a longest path; so instead of picking the smallest among all incoming arcs, we pick the biggest. This leads to the path (in red) of Figure 3.

Question 2: Shortest path deeper cuts

(a) Consider the following statements and answer whether they are true or false. Justify your answer by providing a proof to the statement (if true) or by showing a counter example (when false).

> **Statement 1**. A network where all arc costs are different has a unique shortest path connecting two nodes.
>
> **Statement 2**. In a directed network, if we drop all directions (that is, every arc can be traversed in every direction no matter its original sense), the shortest path will not change.
>
> **Statement 3**. Assume we solved the shortest path problem but we underestimated each arc cost by $k > 0$ units (that is, instead of c_{ij} we should have used $\hat{c}_{ij} = c_{ij} + k$. Then, the solution to the two problems should be the same.
>
> **Statement 4**. Assume we solved the shortest path problem but we underestimated each arc cost by a factor of $k > 0$ (that is, instead of c_{ij} we should have used $\hat{c}_{ij} = c_{ij} \cdot k$). Then, the solution to the two problems should be the same.

(b) In some instances, we may be interested in finding **two shortest paths**. Why? Well, in cases where we are focused on the robustness of the proposed solution, we may want to focus on having a "Plan B". Specifically, assume that we are interested in solving the **2-disjointed shortest paths problem** for the same source and terminal nodes. We call two paths disjointed if they share no arcs (they are allowed to share nodes, though).

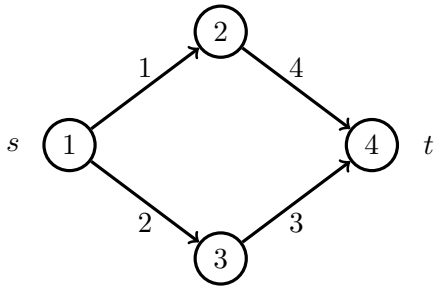
An algorithm to tackle this problem is the ingenious Suurballe's algorithm. A big-picture description is offered here:

1. Compute a shortest path tree from the source s to every other node (including the terminal t). Let d_{si} be the distance from s to i in the shortest path tree.
2. Modify all arc costs to be $c'_{ij} = c_{ij} + d_{si} - d_{sj}$. Note how this modification turns all arcs that are in the shortest path tree to have a cost of 0.
3. Construct a residual network by reversing all arcs in the current shortest path from s to t as well as removing all arcs that are opposite to the shortest path from s to t (in the reverse direction than the shortest path from s to t).
4. Find a shortest path from s to t in the residual network.
5. Discard any arc that appears in both shortest paths in opposite directions. The remaining arcs form two paths from s to t that share no arcs.

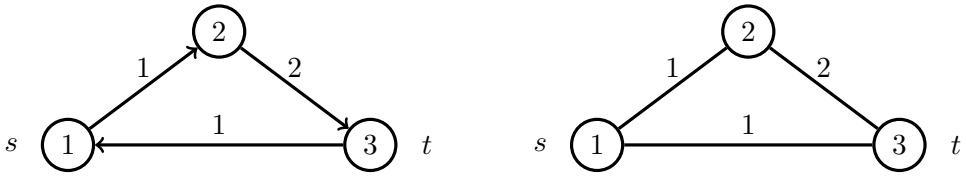
Implement this algorithm using networkx and then solve the network that appears in Lecture 6 Slide 21.

Answer (a) Indicative proofs and counterexamples follow:

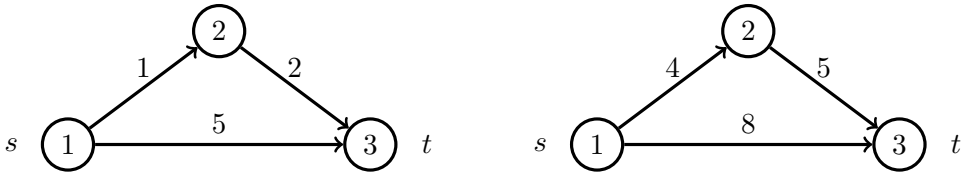
> **Statement 1**. This is False. Consider the following network, where there are two shortest paths of length 5 connecting the source node to the terminal node.



Statement 2. This is False. Consider the following counterexample where the original (directed) network has a total cost of 3, whereas the second one (undirected) has a total cost of 1.



Statement 3. This is False. Once more consider the following counterexample, where the original network has a shortest path of 3 ($1 \to 2 \to 3$). If all the edges were underestimated by 3 units, then that path now has a cost of 9, compared to the other path ($1 \to 3$) which has a cost of 8 and is now shorter.



Statement 4. This is true. For a contradiction, assume that the shortest path in the original network (let it be P^* and consisting of arcs A^*) is not the shortest path in the new network where all costs are multiplied by k; instead the new shortest path now is $\hat{P}$ consisting of arcs $\hat{A}$. Hence, we have that:

$$\sum_{(i,j)\in\hat{A}} k \cdot c_{ij} \leq \sum_{(i,j)\in A^*} k \cdot c_{ij} \implies k \sum_{(i,j)\in\hat{A}} c_{ij} \leq k \sum_{(i,j)\in A^*} c_{ij} \implies \sum_{(i,j)\in\hat{A}} c_{ij} \leq \sum_{(i,j)\in A^*} c_{ij}.$$

This last inequality is a contradiction, though, seeing as the shortest path in the original network was P^* and hence $\sum_{(i,j)\in\hat{A}} c_{ij} > \sum_{(i,j)\in A^*} c_{ij}$

Question 3: Spanning and Steiner trees

(a) Prove or provide a counterexample to the following statement: *If all arcs of a network have distinct costs (that is, no two arc costs are equal), then the network has a unique minimum spanning tree.*

(b) Consider the problem of the *minimum cost spanning forest*, in which the goal is to identify k trees that span all nodes of a network in the minimum cost. Recommend an approach to solve this problem when k is given in advance. Then, use networkx to code this approach and solve the minimum cost spanning forest for $k = 2, 3$ in the network provided in the file *phylo.csv*.

(c) Write code in networkx that reads the illinois.csv zip codes, randomly selects k of them (you may try $k \in \{5, 10, 15\}$) and then returns the minimum cost **Steiner tree** connecting them. You may assume the distances between two zip codes are the Haversine distance based on their longitudes and latitudes. You may also assume that we may only connect two zip codes by an edge if the distance between them is below 40 units.

Answer (a) The statement is **true**. Consider the path optimality: it states that any non-tree arc has to be at least equal to the tree arcs in the cycle it creates. Assume for a contradiction that there is an alternative solution (i.e., two different minimum spanning trees). Consider the non-tree arc that when added to the first tree produces the second tree. The tree arc it replaces has to be of equal cost. But, this is a contradiction as our assumption is that all costs are distinct.

(b) Consider Kruskal's algorithm. It states that we keep on adding the smallest cost arc as long as it does not create a cycle. The algorithm terminates when $n-1$ arcs have been added and one tree has been constructed.

In our case, we can stop prior to that! Let our graph have n nodes. Assume we are interested in building $k = 2$ trees. Further assume that the two trees will end up with n_1 and n_2 nodes respectively, for a total of $n_1 + n_2 = n$ nodes in total. The two trees, then, will have $n_1 - 1$ and $n_2 - 1$ arcs, for a total of $n_1 - 1 + n_2 - 1 = n - 2$ arcs. Hence, we can use Kruskal's and terminate when $n - 2$ arcs have been added. In general, when wanting to construct $k > 1$ trees, we can stop when $n - k - 1$ arcs have been added.

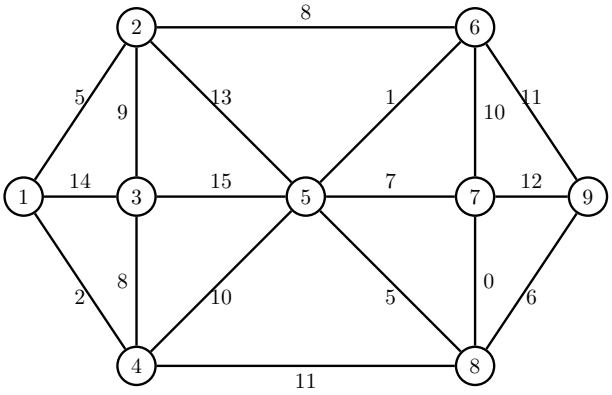


Figure 4: A toy example to showcase the algorithm.

Let us now go ahead and show how this works in a toy example. Consider the network of Figure 4. Then, for $k = 2$ we would have the forest of Figure 5 and for $k = 3$ the forest of

Figure 6.

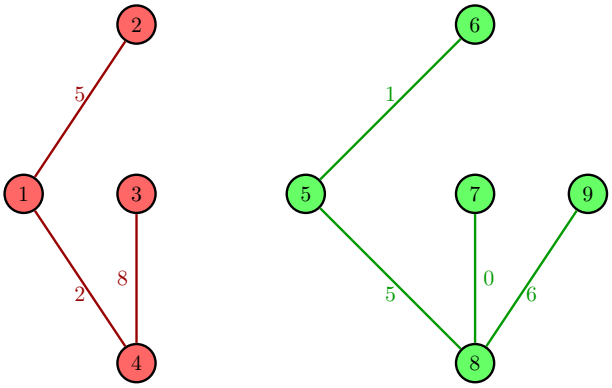


Figure 5: The forest we would get for $k = 2$.

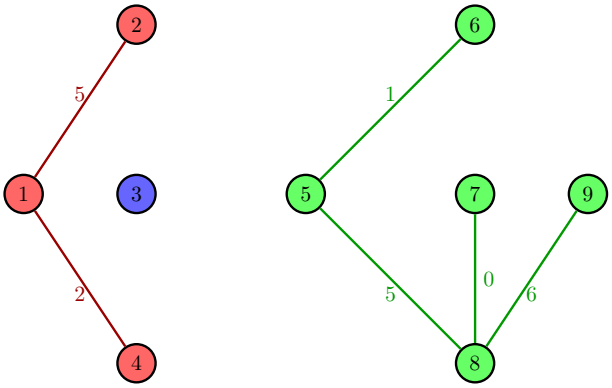
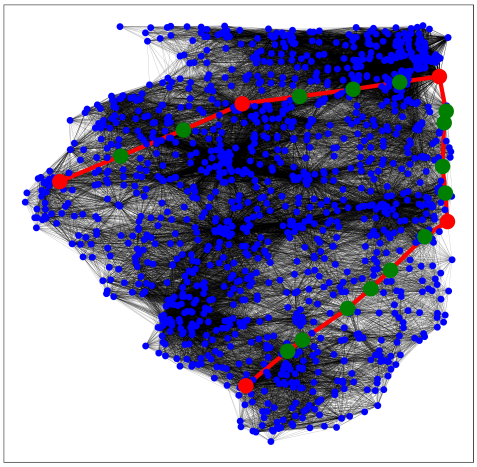


Figure 6: Similarly, for $k = 3$ we would simply stop even earlier getting the forest shown here.

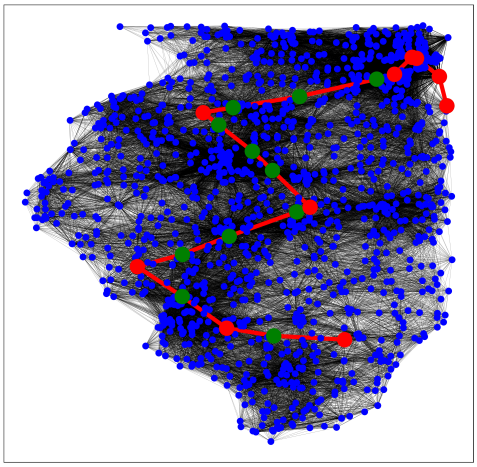
The code and the solution to the phylogenetic problem is given in the Jupyter notebook of `phylo.ipynb`.

(c) Check the code at `Steiner.ipynb`. It produces the following Steiner trees. In the figures, in red we show the Steiner nodes and the Steiner tree edges; in green we show the non-Steiner nodes that are in the Steiner tree. Finally, in blue and black are the original nodes and arcs (that do not participate in the Steiner trees produced).

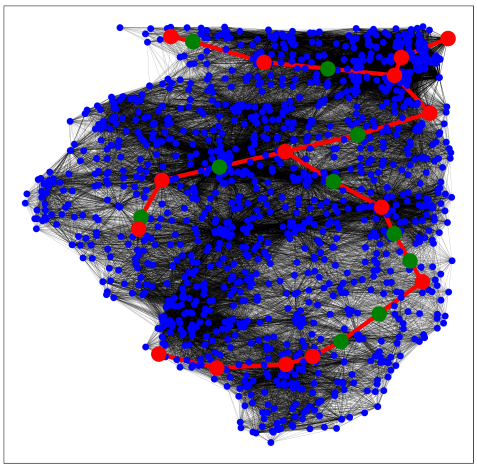
For $k = 5$:



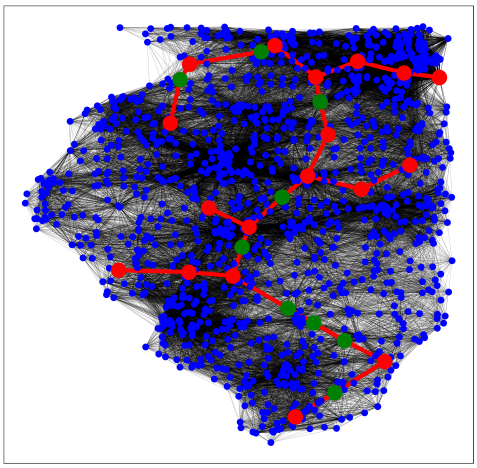
For $k = 10$:



For $k = 15$:



For $k = 20$:



Question 4: Connectedness

(a) A network is said to be k-edge-connected if it remains connected after removing $k-1$ edges. Here, we ask you to focus on a specific pair of nodes s, t. Then, the network is said to be k-edge-connected as far as $s-t$ are concerned if s can send flow to t upon removal of $k-1$ edges. Formulate this problem as a maximum flow problem.

(b) A network is said to be k-node-connected if it remains connected after removing $k-1$ nodes. Here, we ask you to focus on a specific pair of nodes s, t. Then, the network is said to be k-node-connected as far as $s-t$ are concerned if s can send flow to t upon removal of $k-1$ nodes, other than the source and the terminal themselves. Formulate this problem as a maximum flow problem.

Answer (a) Let $G(V, E)$ be the graph we are interested in checking whether it is k-edge-connected. Now, create $\hat{G}(V, E)$ with the same set of nodes and edges; however, associated with each edge (i, j) with have a capacity $u_{ij} = 1$. Solving the maximum flow problem between a source s and a terminal t is guaranteed to show us whether it is k-edge-connected:

- Assume that the maximum flow is $\geq k$; then the graph is k-edge-connected as far as the $s-t$ pair is concerned.
- Assume that the maximum flow is $< k$; then the graph is not k-edge-connected as far as the $s-t$ pair is concerned.

(b) In a similar manner, consider $\hat{G}\left(\hat{V}, \hat{E}\right)$. For the new set of nodes, duplicate each node i into i_{in}, i_{out}; for the new set of edges, replace every edge (i, j) with (i_{out}, j_{in}) and add new edges (i_{in}, i_{out}) for every node $i \in V$. Finally, set all edge capacities $u_{ij} = 1$. Solving the maximum flow problem between a source s and a terminal t is guaranteed to show us whether it is k-node-connected:

- Assume that the maximum flow is $\geq k$; then the original graph $G(V, E)$ is k-node-connected as far as the $s-t$ pair is concerned.
- Assume that the maximum flow is $< k$; then the original graph $G(V, E)$ is not k-node-connected as far as the $s-t$ pair is concerned.