David Hamblin
November 15, 2016
CPSC 611

Network Optimization with Gurobi

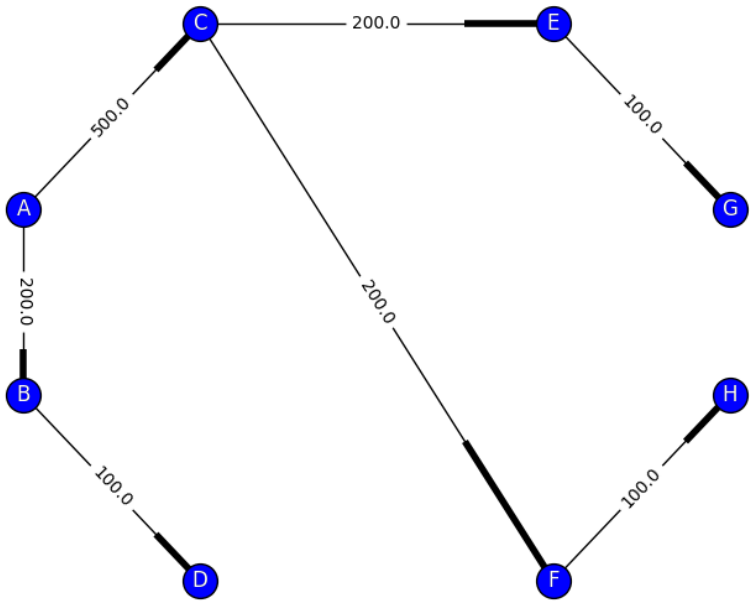
Task 1

Python Code:

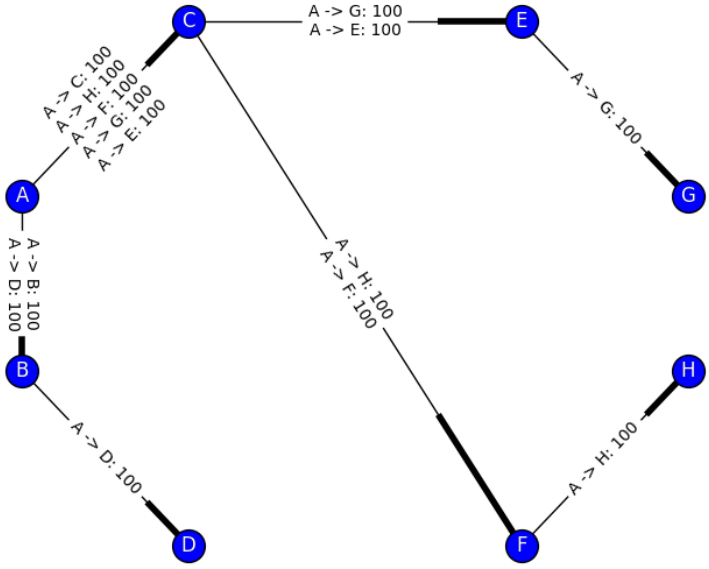
```
from gurobipy import *

m = Model('Project2Task1')

nodes = ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H']

# Capacity-dependent cost is 5 MU per 10 Mbps. Links are 1 Gbps.
cap_dep_cost = 5

# Setup cost is 100 MU * cost multiplier
setup_cost = 100

demands, dAmount = multidict({
  ('A', 'B'):   100,
  ('A', 'C'):   100,
  ('A', 'D'):   100,
  ('A', 'E'):   100,
  ('A', 'F'):   100,
  ('A', 'G'):   100,
  ('A', 'H'):   100})
demands = tuplelist(demands)

links, cost, install_cost = multidict({
    ('A', 'B'): [2*cap_dep_cost, 2*setup_cost],
    ('B', 'A'): [2*cap_dep_cost, 2*setup_cost],
    ('A', 'C'): [2*cap_dep_cost, 2*setup_cost],
    ('C', 'A'): [2*cap_dep_cost, 2*setup_cost],
    ('B', 'D'): [2*cap_dep_cost, 2*setup_cost],
    ('D', 'B'): [2*cap_dep_cost, 2*setup_cost],
    ('D', 'F'): [2*cap_dep_cost, 2*setup_cost],
    ('F', 'D'): [2*cap_dep_cost, 2*setup_cost],
    ('C', 'F'): [3*cap_dep_cost, 3*setup_cost],
    ('F', 'C'): [3*cap_dep_cost, 3*setup_cost],
    ('C', 'E'): [2*cap_dep_cost, 2*setup_cost],
    ('E', 'C'): [2*cap_dep_cost, 2*setup_cost],
    ('E', 'G'): [1*cap_dep_cost, 1*setup_cost],
    ('G', 'E'): [1*cap_dep_cost, 1*setup_cost],
    ('G', 'H'): [2*cap_dep_cost, 2*setup_cost],
    ('H', 'G'): [2*cap_dep_cost, 2*setup_cost],
    ('F', 'H'): [1*cap_dep_cost, 1*setup_cost],
    ('H', 'F'): [1*cap_dep_cost, 1*setup_cost]
})
links = tuplelist(links)

flow = {}
for i,j in links:
    for d in demands:
        flow[i,j,d] = m.addVar(name='flow_%s_%s_%s' % (i, j, d))

capacity = {}
install = {}
for i,j in links:
    capacity[i, j] = m.addVar(name='capacity_%s_%s' % (i, j))
    install[i,j] = m.addVar(vtype=GRB.INTEGER, name='install_%s_%s' % (i,j))
```

```
m.update()

# Flow balance at source, output, and interior nodes
for i in nodes:
    for d in demands:
        if i == d[0]:
            m.addConstr(
                quicksum(flow[i,j,d] for i,j in links.select(i,'*')) -
                quicksum(flow[k,i,d] for k,i in links.select('*',i))
                == dAmount[d], 'node_%s_%s' % (i, d))
        elif i == d[1]:
            m.addConstr(
                quicksum(flow[i, j, d] for i, j in links.select(i, '*')) -
                quicksum(flow[k, i, d] for k, i in links.select('*', i))
                == -dAmount[d], 'node_%s_%s' % (i, d))
        else:
            m.addConstr(
                quicksum(flow[i, j, d] for i, j in links.select(i, '*')) -
                quicksum(flow[k, i, d] for k, i in links.select('*', i))
                == 0, 'node_%s_%s' % (i, d))

# Capacity constraints
for i,j in links:
    m.addConstr(quicksum(flow[i,j,d] for d in demands) <= capacity[i,j],
                'cap_%s_%s' % (i, j))
    m.addConstr(capacity[i,j] <= 800*install[i,j],
                'install_%s_%s' % (i, j))

m.update()

totalCost = quicksum((capacity[i, j]*cost[i, j] + install[i, j]*install_cost[i, j])
for i, j in links)
m.setObjective(totalCost, GRB.MINIMIZE)
m.update()

m.optimize()
```

Resultant Graph with Link Capacity:



Resultant Graph with Demand Flow:



Total Cost of the Network: 15300

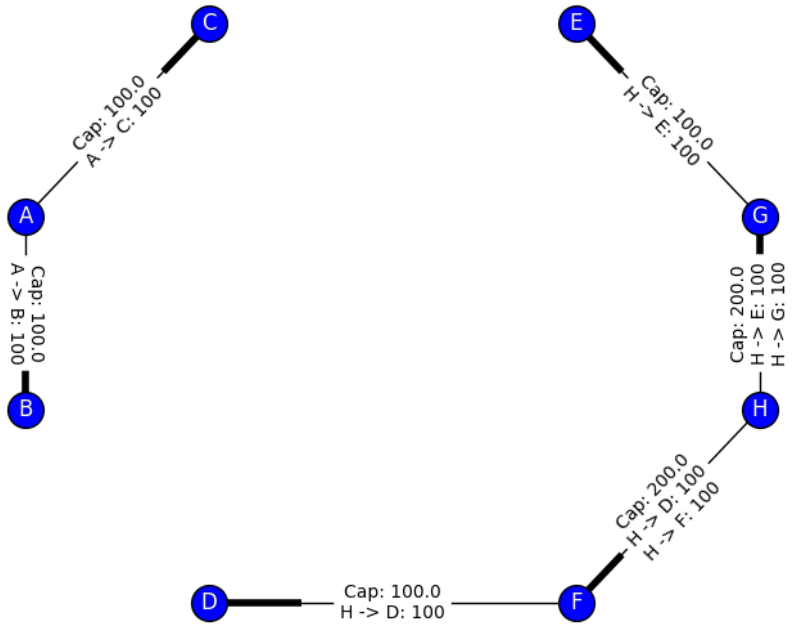
Task 2

Python Code:

```
from gurobipy import *

m = Model('Project2Task2')

nodes = ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H']

# Capacity-dependent cost is 5 MU per 10 Mbps. Links are 1 Gbps.
cap_dep_cost = 5

# Setup cost is 100 MU * cost multiplier
setup_cost = 100

demands, dAmount = multidict({
  ('A', 'B'):   100,
  ('A', 'C'):   100,
  ('H', 'D'):   100,
  ('H', 'E'):   100,
  ('H', 'F'):   100,
  ('H', 'G'):   100})
demands = tuplelist(demands)

links, cost, install_cost = multidict({
    ('A', 'B'): [2*cap_dep_cost, 2*setup_cost],
    ('B', 'A'): [2*cap_dep_cost, 2*setup_cost],
    ('A', 'C'): [2*cap_dep_cost, 2*setup_cost],
    ('C', 'A'): [2*cap_dep_cost, 2*setup_cost],
    ('B', 'D'): [2*cap_dep_cost, 2*setup_cost],
    ('D', 'B'): [2*cap_dep_cost, 2*setup_cost],
    ('D', 'F'): [2*cap_dep_cost, 2*setup_cost],
    ('F', 'D'): [2*cap_dep_cost, 2*setup_cost],
    ('C', 'F'): [3*cap_dep_cost, 3*setup_cost],
    ('F', 'C'): [3*cap_dep_cost, 3*setup_cost],
    ('C', 'E'): [2*cap_dep_cost, 2*setup_cost],
    ('E', 'C'): [2*cap_dep_cost, 2*setup_cost],
    ('E', 'G'): [1*cap_dep_cost, 1*setup_cost],
    ('G', 'E'): [1*cap_dep_cost, 1*setup_cost],
    ('G', 'H'): [2*cap_dep_cost, 2*setup_cost],
    ('H', 'G'): [2*cap_dep_cost, 2*setup_cost],
    ('F', 'H'): [1*cap_dep_cost, 1*setup_cost],
    ('H', 'F'): [1*cap_dep_cost, 1*setup_cost]
})
links = tuplelist(links)

flow = {}
for i,j in links:
    for d in demands:
        flow[i,j,d] = m.addVar(name='flow_%s_%s_%s' % (i, j, d))

capacity = {}
install = {}
for i,j in links:
    capacity[i, j] = m.addVar(name='capacity_%s_%s' % (i, j))
    install[i,j] = m.addVar(vtype=GRB.INTEGER, name='install_%s_%s' % (i,j))

m.update()

# Flow balance at source, output, and interior nodes
for i in nodes:
    for d in demands:
```

```
        if i == d[0]:
            m.addConstr(
                quicksum(flow[i,j,d] for i,j in links.select(i,'*')) -
                quicksum(flow[k,i,d] for k,i in links.select('*',i))
                == dAmount[d], 'node_%s_%s' % (i, d))
        elif i == d[1]:
            m.addConstr(
                quicksum(flow[i, j, d] for i, j in links.select(i, '*')) -
                quicksum(flow[k, i, d] for k, i in links.select('*', i))
                == -dAmount[d], 'node_%s_%s' % (i, d))
        else:
            m.addConstr(
                quicksum(flow[i, j, d] for i, j in links.select(i, '*')) -
                quicksum(flow[k, i, d] for k, i in links.select('*', i))
                == 0, 'node_%s_%s' % (i, d))

# Capacity constraints
for i,j in links:
    m.addConstr(quicksum(flow[i,j,d] for d in demands) <= capacity[i,j],
                'cap_%s_%s' % (i, j))
    m.addConstr(capacity[i,j] <= 2000*install[i,j],
                'install_%s_%s' % (i, j))

m.update()

totalCost = quicksum((capacity[i, j]*cost[i, j] + install[i, j]*install_cost[i, j])
for i, j in links)
m.setObjective(totalCost, GRB.MINIMIZE)
m.update()

m.optimize()
```

Resultant Graph with Link Capacity/Demand Flow:



Total Cost of the Network: 7500

Task 3

Task 3.1

Python Code:

```
from gurobipy import *

m = Model('Project2Task3-1')

nodes = ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H']

# Capacity-dependent cost is 5 MU per 10 Mbps. Links are 1 Gbps.
cap_dep_cost = 5

# Setup cost is 100 MU * cost multiplier
setup_cost = 100

DemandNode, DemandSink, DemandSource = multidict({
    ('A'): [0, 700],
    ('B'): [100, 0],
    ('C'): [100, 0],
    ('D'): [100, 0],
    ('E'): [100, 0],
    ('F'): [100, 0],
    ('G'): [100, 0],
    ('H'): [100, 0]
})

links, cost, install_cost = multidict({
    ('A', 'B'): [2*cap_dep_cost, 2*setup_cost],
    ('B', 'A'): [2*cap_dep_cost, 2*setup_cost],
    ('A', 'C'): [2*cap_dep_cost, 2*setup_cost],
    ('C', 'A'): [2*cap_dep_cost, 2*setup_cost],
    ('B', 'D'): [2*cap_dep_cost, 2*setup_cost],
    ('D', 'B'): [2*cap_dep_cost, 2*setup_cost],
    ('D', 'F'): [2*cap_dep_cost, 2*setup_cost],
    ('F', 'D'): [2*cap_dep_cost, 2*setup_cost],
    ('C', 'F'): [3*cap_dep_cost, 3*setup_cost],
    ('F', 'C'): [3*cap_dep_cost, 3*setup_cost],
    ('C', 'E'): [2*cap_dep_cost, 2*setup_cost],
    ('E', 'C'): [2*cap_dep_cost, 2*setup_cost],
    ('E', 'G'): [1*cap_dep_cost, 1*setup_cost],
    ('G', 'E'): [1*cap_dep_cost, 1*setup_cost],
    ('G', 'H'): [2*cap_dep_cost, 2*setup_cost],
    ('H', 'G'): [2*cap_dep_cost, 2*setup_cost],
    ('F', 'H'): [1*cap_dep_cost, 1*setup_cost],
    ('H', 'F'): [1*cap_dep_cost, 1*setup_cost]
})
links = tuplelist(links)

# G = nx.read_gml('task3.gml')

flow = {}
for i,j in links:
# for i,j in nx.edges(G):
    flow[i,j] = m.addVar(name='flow_%s_%s' % (i, j))

capacity = {}
install = {}
for i,j in links:
    capacity[i, j] = m.addVar(name='capacity_%s_%s' % (i, j))
```

```
    install[i, j] = m.addVar(vtype=GRB.INTEGER, name='install_%s_%s' % (i, j))

m.update()

# Flow balance at source, output, and interior nodes
for i in nodes:
    m.addConstr(
        quicksum(flow[i,j] for i,j in links.select(i,'*')) -
        quicksum(flow[k,i] for k,i in links.select('*',i))
        == DemandSource[i] - DemandSink[i], 'node_%s' % (i))

# Capacity constraints
for i,j in links:
    m.addConstr(flow[i,j] <= capacity[i,j],
                'cap_%s_%s' % (i, j))
    m.addConstr(capacity[i,j] <= 2000*install[i,j],
                'install_%s_%s' % (i, j))

m.update()

totalCost = quicksum((capacity[i, j]*cost[i, j] + install[i, j]*install_cost[i, j])
for i, j in links)
m.setObjective(totalCost, GRB.MINIMIZE)
m.update()

m.optimize()
```

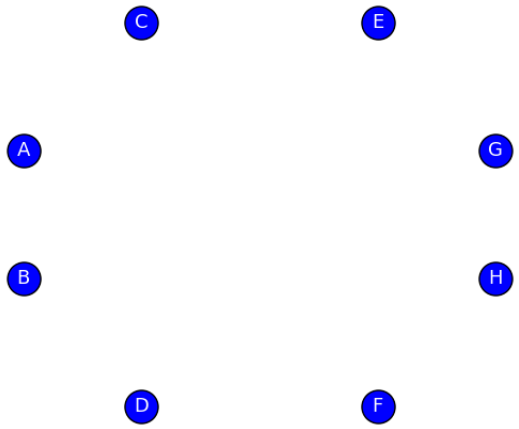
Task 3.2

Python Code:

```
from gurobipy import *

m = Model('Project2Task3-2')

nodes = ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H']

# Capacity-dependent cost is 5 MU per 10 Mbps. Links are 1 Gbps.
cap_dep_cost = 5

# Setup cost is 100 MU * cost multiplier
setup_cost = 100

DemandNode, DemandSink = multidict({
    ('A'): 0,
    ('B'): 100,
    ('C'): 100,
    ('D'): 100,
    ('E'): 100,
    ('F'): 100,
    ('G'): 100,
    ('H'): 100
})

links, cost, install_cost = multidict({
    ('A', 'B'): [2*cap_dep_cost, 2*setup_cost],
    ('B', 'A'): [2*cap_dep_cost, 2*setup_cost],
    ('A', 'C'): [2*cap_dep_cost, 2*setup_cost],
    ('C', 'A'): [2*cap_dep_cost, 2*setup_cost],
    ('B', 'D'): [2*cap_dep_cost, 2*setup_cost],
    ('D', 'B'): [2*cap_dep_cost, 2*setup_cost],
```

```
    ('D', 'F'): [2*cap_dep_cost, 2*setup_cost],
    ('F', 'D'): [2*cap_dep_cost, 2*setup_cost],
    ('C', 'F'): [3*cap_dep_cost, 3*setup_cost],
    ('F', 'C'): [3*cap_dep_cost, 3*setup_cost],
    ('C', 'E'): [2*cap_dep_cost, 2*setup_cost],
    ('E', 'C'): [2*cap_dep_cost, 2*setup_cost],
    ('E', 'G'): [1*cap_dep_cost, 1*setup_cost],
    ('G', 'E'): [1*cap_dep_cost, 1*setup_cost],
    ('G', 'H'): [2*cap_dep_cost, 2*setup_cost],
    ('H', 'G'): [2*cap_dep_cost, 2*setup_cost],
    ('F', 'H'): [1*cap_dep_cost, 1*setup_cost],
    ('H', 'F'): [1*cap_dep_cost, 1*setup_cost]
})
links = tuplelist(links)

flow = {}
for i,j in links:
    flow[i,j] = m.addVar(name='flow_%s_%s' % (i, j))

capacity = {}
install = {}
for i,j in links:
    capacity[i, j] = m.addVar(name='capacity_%s_%s' % (i, j))
    install[i, j] = m.addVar(vtype=GRB.INTEGER, name='install_%s_%s' % (i, j))

DemandSource = {}
for i in nodes:
    DemandSource[i] = m.addVar(name='demand-source_%s' % i)

m.update()

# Flow balance at source, output, and interior nodes
for i in nodes:
    m.addConstr(
        quicksum(flow[i,j] for i,j in links.select(i,'*')) -
        quicksum(flow[k,i] for k,i in links.select('*',i))
        == DemandSource[i] - DemandSink[i], 'node_%s' % (i))

# Capacity constraints
for i,j in links:
    m.addConstr(flow[i,j] <= capacity[i,j],
                'cap_%s_%s' % (i, j))
    m.addConstr(capacity[i,j] <= 2000*install[i,j],
                'install_%s_%s' % (i, j))

m.update()

totalCost = quicksum((capacity[i, j]*cost[i, j] + install[i, j]*install_cost[i, j])
for i, j in links)
m.setObjective(totalCost, GRB.MINIMIZE)
m.update()

m.optimize()
```

Why is the total cost zero? In this example, the optimizer will place a server in each of the nodes, which means that there is no demand to send out over links as requests are processed in the same node. In a graph with no links, there is no cost.



Task 3.3

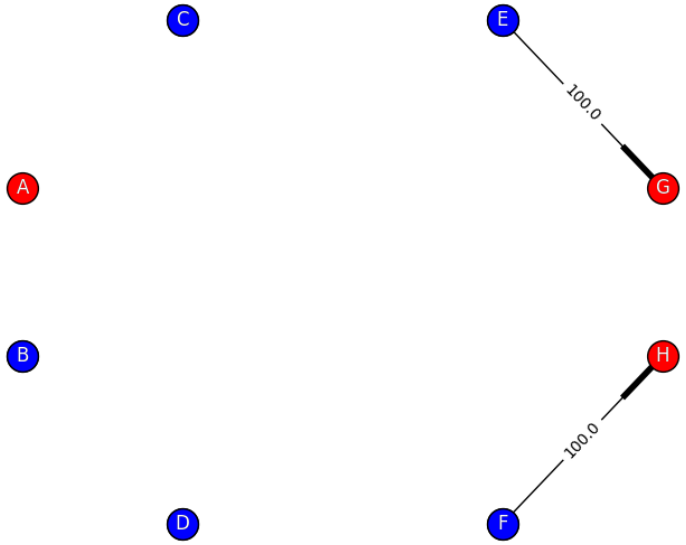
Python Code:

```
from gurobipy import *

m = Model('Project2Task3-3')

nodes = ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H']

# Capacity-dependent cost is 5 MU per 10 Mbps. Links are 1 Gbps.
cap_dep_cost = 5

# Setup cost is 100 MU * cost multiplier
setup_cost = 100

DemandNode, DemandSink = multidict({
    ('A'): 0,
    ('B'): 100,
    ('C'): 100,
    ('D'): 100,
    ('E'): 100,
    ('F'): 100,
    ('G'): 100,
    ('H'): 100
})

links, cost, install_cost = multidict({
    ('A', 'B'): [2*cap_dep_cost, 2*setup_cost],
    ('B', 'A'): [2*cap_dep_cost, 2*setup_cost],
    ('A', 'C'): [2*cap_dep_cost, 2*setup_cost],
    ('C', 'A'): [2*cap_dep_cost, 2*setup_cost],
    ('B', 'D'): [2*cap_dep_cost, 2*setup_cost],
    ('D', 'B'): [2*cap_dep_cost, 2*setup_cost],
    ('D', 'F'): [2*cap_dep_cost, 2*setup_cost],
    ('F', 'D'): [2*cap_dep_cost, 2*setup_cost],
    ('C', 'F'): [3*cap_dep_cost, 3*setup_cost],
    ('F', 'C'): [3*cap_dep_cost, 3*setup_cost],
    ('C', 'E'): [2*cap_dep_cost, 2*setup_cost],
    ('E', 'C'): [2*cap_dep_cost, 2*setup_cost],
    ('E', 'G'): [1*cap_dep_cost, 1*setup_cost],
    ('G', 'E'): [1*cap_dep_cost, 1*setup_cost],
    ('G', 'H'): [2*cap_dep_cost, 2*setup_cost],
    ('H', 'G'): [2*cap_dep_cost, 2*setup_cost],
    ('F', 'H'): [1*cap_dep_cost, 1*setup_cost],
    ('H', 'F'): [1*cap_dep_cost, 1*setup_cost]
```

```
})
links = tuplelist(links)

flow = {}
for i,j in links:
    flow[i,j] = m.addVar(name='flow_%s_%s' % (i, j))

capacity = {}
install = {}
for i,j in links:
    capacity[i, j] = m.addVar(name='capacity_%s_%s' % (i, j))
    install[i, j] = m.addVar(vtype=GRB.INTEGER, name='install_%s_%s' % (i, j))

DemandSource = {}
for i in nodes:
    DemandSource[i] = m.addVar(name='demand-source_%s' % i)

server_install = {}
for i in nodes:
    server_install[i] = m.addVar(vtype=GRB.INTEGER, name='server-install_%s' % i)

m.update()

# Flow balance at source, output, and interior nodes
for i in nodes:
    m.addConstr(
        quicksum(flow[i,j] for i,j in links.select(i,'*')) -
        quicksum(flow[k,i] for k,i in links.select('*',i))
        == DemandSource[i] - DemandSink[i], 'node_%s' % (i))

# Capacity constraints
for i,j in links:
    m.addConstr(flow[i,j] <= capacity[i,j],
                'cap_%s_%s' % (i, j))
    m.addConstr(capacity[i,j] <= 2000*install[i,j],
                'install_%s_%s' % (i, j))

for i in nodes:
    m.addConstr(DemandSource[i] <= 2000 * server_install[i],
                'server-install-max_%s' % (i))

m.update()

server_cost = 1000

totalCost = quicksum((capacity[i, j]*cost[i, j] + install[i, j]*install_cost[i, j])
for i, j in links) + quicksum(server_install[i]*server_cost for i in nodes)
m.setObjective(totalCost, GRB.MINIMIZE)
m.update()

m.optimize()
```

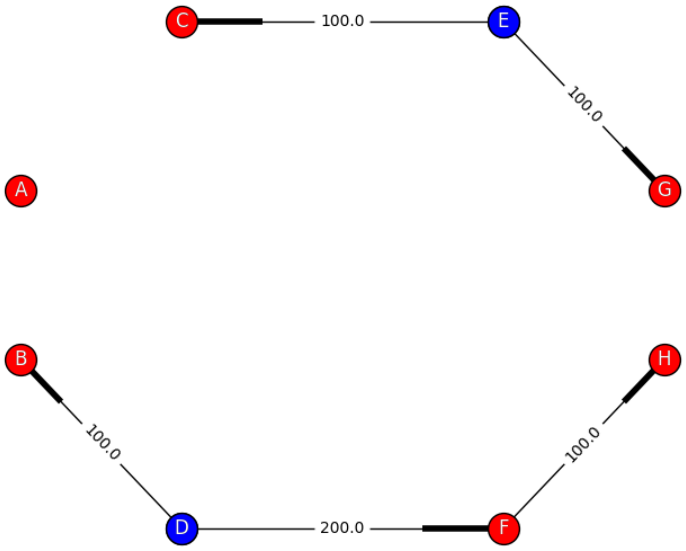
For the following, blue represent nodes with servers installed, whereas red nodes do not have servers.

$C_s = 1000$ Graph:



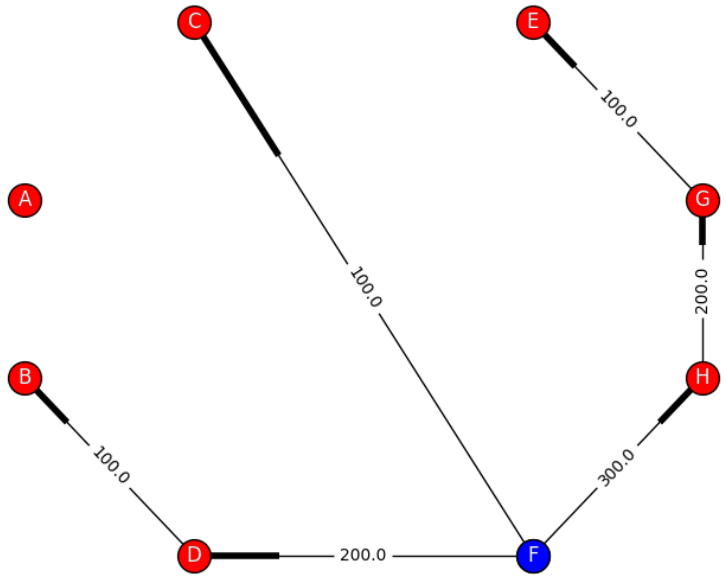
Total Cost of the Network: 6200

$C_s = 3000$ Graph:



Total Cost of the Network: 11800

$C_s = 5000$ Graph:



Total Cost of the Network: 14600

Brief Discussion of Server Cost:

While servers are inexpensive ($C_s = 1000$) the network operates cheaper with more servers than links, except for in the case where the cost multiplier of the link is 1 – between F and H, E and G. While this cost rises, the number of servers decreases and links are restored, as it is cheaper to maintain the links compared to installing new servers.