```python
#Deep Space Nine - Power Grid Research Team MW2 - - - B2-4
# Imports-----------------------------------
from gurobipy import GRB,Model
import pprint
import csv
import matplotlib.pyplot as plt

#SET flag if the arc data should be bidericitional
biderictional = True

# Create the model-----------------------------------
m = Model('problem A')

# Create Dictionaries from CSV files
arc_caps = {}
with open('DS9_Network_Arc_Data_B2.csv', 'r') as csvfile:
    reader = csv.reader(csvfile, delimiter=' ', quotechar='|')
    next(reader)
    for row in reader:
        arc = row[0].split(",")
        if int(arc[0]) not in arc_caps.keys():
            arc_caps[int(arc[0])] = {int(arc[1]):(int(arc[2]), int(arc[3]))}
        else:
            arc_caps[int(arc[0])][int(arc[1])] = (int(arc[2]), int(arc[3]))
d = arc_caps

node_demand = {}
with open('DS9_Network_Node_Data.csv', 'r') as csvfile:
    reader = csv.reader(csvfile, delimiter=' ', quotechar='|')
    next(reader)
    for row in reader:
        arc = row[0].split(",")
        node_demand[int(arc[0])] = int(arc[1])
demand = node_demand
# print(demand)

groups = {}
with open('DS9_Network_Node_Data.csv', 'r') as csvfile:
    reader = csv.reader(csvfile, delimiter=' ', quotechar='|')
    next(reader)
    for row in reader:
        arc = row[0].split(",")
        if int(arc[2]) not in groups:
            groups[int(arc[2])] = [(int(arc[0]), int(arc[1]))]
        else:
            groups[int(arc[2])].append((int(arc[0]),int(arc[1])),)

# Set parameters
m.setParam('OutputFlag',True)

# Add variables-----------------------------------

# define time variable
time = ["time"]

#make list of arcs
arcs = []
arcs.append("x0x1")
for f_key in d:
    for t_key in d[f_key]:
        arc = "x"+str(f_key)+"x"+str(t_key)
        arcs.append(arc)

#make list of nodes
nodes = []
for i in range(1,31):
    nodes.append("y"+str(i))

#combine lists of arcs and nodes, including time
vars = arcs + nodes + time

#addvars arcs and nodes in list vars
v = m.addVars(vars, vtype=GRB.CONTINUOUS, lb = -999, name = vars)
bolts = m.addVars(vars, vtype=GRB.INTEGER, name="bolts")
fixed = m.addVars(vars, vtype=GRB.BINARY, name="fixed")

# Add constraints-----------------------------------

#make list of arcs max caps
for i in arcs:
```

```
    if i[0]=='x' and i != 'x0x1':
        m.addConstr(v[i]<=d[int(i.split("x")[1])][int(i.split("x")[2])][0] + bolts[i], "maxcaps")
#     print(i)

#make list of arcs min caps
for i in arcs:
    if i[0]=='x' and i != 'x0x1':
       m.addConstr(v[i]>=-d[int(i.split("x")[1])][int(i.split("x")[2])][0] + bolts[i], "mincaps")

#make list of arcs caps with bolts fixed and maximp
for i in arcs:
    if i[0]=='x' and i != 'x0x1':
       m.addConstr(bolts[i]<=d[int(i.split("x")[1])][int(i.split("x")[2])][1]*fixed[i], "boltcaps")

# define requirement for 3hrs of work if bolts added to a given arc
time = 0
for i  in arcs:
    time += 3* fixed[i] + bolts[i]
m.addConstr(time >= 0,"time")

#make list of relations
values = []
for node in nodes:
    pos = []
    neg = []
    exp = 0
    for a in arcs:
        land = int(a.split("x")[2])
        send = int(a.split("x")[1])
        send_n = int(node[1:])
        if land == send_n:
            exp+=v[a]
        if send == send_n:
            exp-=v[a]
    m.addConstr(v[node]==exp, name="a"+node)

#set max and min constraints for nodes
for i in v:
    if i[0] == "y":
        m.addConstr(v[i]>=0, name="l"+i)
        m.addConstr(v[i]<=demand[int(i[1])], name="u"+i)

#set total demand satisfied to be greater than or equal to a percent of 99
obj = 0
for i in v:
    if i[0] == 'y':
        obj+=v[i]
        print(i)

# m.setObjective(v["a"], GRB.MAXIMIZE)
m.setObjective(obj-time*(.1), GRB.MAXIMIZE)

#optimize model function
m.write("B4.lp")
m.optimize()

#print results
status_code = {1:'LOADED', 2:'OPTIMAL', 3:'INFEASIBLE', 4:'INF_OR_UNBD', 5:'UNBOUNDED'}
status = m.status
print('The optimization status is {}'.format(status_code[status]))
if status == 2:
    # Retrieve variables value
    print('Optimal solution:')
    for v in m.getVars():
        print('%s = %g' % (v.varName, v.x))
        if v.varname == "x0x1":
            source = v.x

time = 0
for i  in arcs:
    time += 3* fixed[i].X + bolts[i].X
print("Demand Satisfied:")
print(source) #returns 154 units of demand satisfied
print("Hours Spent:")
print(time) #returns 77 minimum man hours spent to hit max demand satisfied
```