```python
#Deep Space Nine - Power Grid Research Team MW2 - - - B1-2a
# Imports-------------------------------------
from gurobipy import GRB,Model
import pprint
import csv
import numpy
import matplotlib.pyplot as plt

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

# Create dictionaries from CSV files
arc_caps = {}
with open('DS9_Network_Arc_Data.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])}
        else:
            arc_caps[int(arc[0])][int(arc[1])] = int(arc[2])
d = arc_caps
# print(d)

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])),)
# print(groups)

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

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

#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))

#make fariness metric
a = ["a"]

#combine lists of arcs and nodes
vars = arcs + nodes + a

#addvars arcs and nodes in list vars
v = m.addVars(vars, vtype=GRB.CONTINUOUS, lb = -999, name = vars)
# x0x1 = m.addVar(vtype=GRB.CONTINUOUS, lb = -999, name= "x0x1")

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

#make list of arcs max caps
for i in v:
    if i[0]=='x' and i != 'x0x1':
```

```
        m.addConstr(v[i]<=d[int(i.split("x")[1])][int(i.split("x")[2])])

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

#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 fariness constraints
m.addConstr(v["a"]>=0, name="a")
for i in range (1,len(groups)+1):
    tot_sum = 0
    node_sum = 0
    for j in range(len(groups[i])):
        node_sum += v[f"y{groups[i][j][0]}"]
        tot_sum += groups[i][j][1]
    m.addConstr((node_sum/tot_sum) >= v["a"], name=f"g{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]

m.addConstr(obj >= 0.95*(103), name=f"g{i}")

m.setObjective(v["a"], GRB.MAXIMIZE)

#optimize model function
m.write("B1_2a.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))
    print('Optimal objective value:\n{}'.format(m.objVal))
```