Assignment 3 - Algo for Telecom

Student: Bárbara da Silva Oliveira

February 7, 2022

Network design - Green networking

Question 1

Inputs:

- $G = (V, E)$
- K: request with (s,t,b)
- α: the cost of using the arc $uv \in E$
- β: the cost for the traffic
- γ: the discount of using a pair uv and vu

Variables for the constraints:

- x^k_{uv}: fractional, the request $k \in K$ uses the link $uv \in E$
- z_{uv} binary, indicates if the arc is used at least one request
- y_{uv} binary, indicates when a pair of arcs uv and vu is used
- c integer, indicates the capacity of each link of the network.

Constraints:

- We need to indicate when an arc is used by at least one request. For doing this, we can set z_{uv} as a upper bound for every x^k_{uv}, so when x^k_{uv} is true for any request $k \in K$, z_{uv} must also be true. As z_{uv} is binary and x^k_{uv} is fractional, we can multiply z_{uv} by a large number.

$$\forall k \in K, \forall uv \in E, x^k_{uv} \leq z_{uv} * 100. \tag{1}$$

- We need calculate the value of each time a pair of arcs uv and vu is used. It is possible to do it by the following formula. z_{uv} is a binary variable, that assumes 1 if the arc is used and 0 if it is not used. If z_{uv} is 0 and z_{uv} is 0, we have that the only solution for y_{uv} is 0, because of the restriction that it must be bigger or equal to 0. If z_{uv} is 0 and z_{uv} is 1 or the opposite, the only solution is $y_{uv} = 0$, because of the restriction that it mus be less or equal to 1. And if both are true, the only solution for y_{uv} is 1.

$$\forall uv \in E 0 \leq z_{uv} + z_{vu} - 2 * y_{uv} \leq 1. \tag{2}$$

- Flow conservation constraints: The flow conservation must be attended in all path. The difference of the the flow that enters the node and goes out the node. If the node is the origin of the commodity, then the flow that goes out is the generation of throughput of that request in that node, so it is equal to b_k. If it is the destination, then the throughput will "disappear" from the node, so it is equal to $-b_k$. So, we have:

$$\forall k \in K, \sum_{uv \in E} x_{uv}^k - \sum_{vu \in E} x_{uv}^k = \begin{cases} b_k, & v = s_k \\ -b_k, & v = t_k \\ 0, & \text{otherwise.} \end{cases} \tag{3}$$

- Capacity constraints: The maximum capacity of each link is c. So, if the link $uv \in E$ is used by $k \in K$ requests, the total bandwidth must be less or equal to c.

$$\forall uv \in E, \sum_{uv \in E} x_{uv}^k \leq c \tag{4}$$

Objective:

$$\min \sum_{uv \in E} z_{u,v} * \alpha + \sum_{k \in K} \sum_{uv \in E} x_{u,v,k} * \beta - \sum_{uv} y_{u,v} * \gamma \tag{5}$$

Or:

$$\min c \tag{6}$$

Quantities of:

- Variables: KE + 2E +1 variables
- Constraints: O(K(E + V)) constraints

Question 3

For this question, I added one variable, fractional, $w_{u,k}$, that corresponds to the cost of using a node.

$$\forall k \in K, \forall u \in V, w_{u,k} = \begin{cases} 0, & u = t_k \\ \sum_{uv \in E} x_{uv}, & \text{otherwise} \end{cases} \tag{7}$$

The new objective is:

$$min \sum_{uv \in E} z_{u,v} * \alpha + \sum_{k \in K} \sum_{uv \in E} x_{u,v,k} * \beta - \sum_{uv} y_{u,v} * \gamma + \sum_{k \in K} \sum_{uinV} w_{u,k} * \delta \tag{8}$$

Or

$$\min c \tag{9}$$

Quantities of:

- Variables: 2E + K(E + V) + 1 variables
- Constraints: O(K(E + V)) constraints

Assignment3

February 7, 2022

[162]:
```python
def flow_minicost(G,K,alpa,bet,gam):
    #Minimizing the cost of fractional multicomodity flow

    from sage.numerical.mip import MixedIntegerLinearProgram, MIPSolverException

    # initialize a minimization program
    p = MixedIntegerLinearProgram(maximization=False)
    Kk = range(len(K))
    sk = np.zeros(10)
    tk = np.zeros(10)
    bk = np.zeros(10)
    for k in range(0,10):
        sk[k], tk[k], bk[k] = K[k]
    print(bk)
    # declaring variables
    x = p.new_variable(real= True, nonnegative = True, name='x')
    z = p.new_variable(binary=True, name='z') #if the arc is used at least one␣
↪request
    y = p.new_variable(binary=True, name = 'y') #a pair of arcs uv and vu is␣
↪used
    c = p.new_variable(integer = True, nonnegative = True, name = 'c') #capacity
    # add constraints:
    for k in range(0,10):
        for u, v, label in G.edges():
            p.add_constraint(x[u,v,k] <= z[u,v]*100)
    for u,v, label in G.edges():
        p.add_constraint(0<= z[u,v] + z[v,u] - 2*y[u,v])
        p.add_constraint(z[u,v] + z[v,u] - 2*y[u,v] <=1)
    for k in Kk:
        for u in G.vertices():
            if (u==sk[k]):
                p.add_constraint(sum(x[u,v,k] for v in G.neighbors(u)) -␣
↪sum(x[v,u,k] for v in G.neighbors(u)) == bk[k])
            elif (u==tk[k]):
                p.add_constraint(sum(x[u,v,k] for v in G.neighbors(u)) -␣
↪sum(x[v,u,k] for v in G.neighbors(u)) == -bk[k])
            else:
```

```
                p.add_constraint(sum(x[u,v,k] for v in G.neighbors(u)) -␣
↪sum(x[v,u,k] for v in G.neighbors(u)) == 0)
    for u,v, label in G.edges():
        p.add_constraint(sum(x[u,v,k] for k in Kk) <= c[0])

    #add objective to minimize

    #p.set_objective(sum(z[u,v] for u, v, label in G.edges())*alpa +␣
↪sum(sum(x[u,v,k] for u, v, label in G.edges()) for k in Kk)*bet - sum(y[u,v]␣
↪for u, v, label in G.edges())*gam*0.5)
    p.set_objective(c[0])
    #try:
    p.solve()
    #except MIPSolverException:
        #print("the problem has no feasible solution")
        #return []

    c_result = p.get_values(c)
    return c_result
```

[163]:
```
import numpy as np

G = graphs.CycleGraph(10)
G.add_edges([(1, 3), (4, 9), (5, 8)])
G = DiGraph(G)
K = [(0, 1, 1), (0, 5, 2), (0, 6, 1), (1, 2, 1), (2, 9, 3), (1, 6, 2), (2, 7,␣
↪1), (2, 8, 1), (6, 5, 1), (8, 1, 2)]
alpa = 100
bet = 5
gam = 20
c_result = flow_minicost(G,K,alpa,bet,gam)
print(c_result)
```

```
[1. 2. 1. 1. 3. 2. 1. 1. 1. 2.]
{0: 5.0}
```

[1]:
```
def flow_minicost2(G,K,alpa,bet,gam,delt):
    #Minimizing the cost of fractional multicomodity flow

    from sage.numerical.mip import MixedIntegerLinearProgram, MIPSolverException

    # initialize a minimization program
    p = MixedIntegerLinearProgram(maximization=False)
    Kk = range(len(K))
    sk = np.zeros(10)
    tk = np.zeros(10)
    bk = np.zeros(10)
```

```
    for k in range(0,10):
        sk[k], tk[k], bk[k] = K[k]
    print(bk)
    # declaring variables
    x = p.new_variable(real= True, nonnegative = True, name='x')
    z = p.new_variable(binary=True, name='z') #if the arc is used at least one request
    y = p.new_variable(binary=True, name = 'y') #a pair of arcs uv and vu is used
    c = p.new_variable(integer = True, nonnegative = True, name = 'c') #capacity
    w = p.new_variable(real = True, nonnegative = True, name = 'w')

    # add constraints:
    for k in range(0,10):
        for u, v, label in G.edges():
            p.add_constraint(x[u,v,k] <= z[u,v]*100)
    for u,v, label in G.edges():
        p.add_constraint(0<= z[u,v] + z[v,u] - 2*y[u,v])
        p.add_constraint(z[u,v] + z[v,u] - 2*y[u,v] <=1)
    #for u, v, label in G.edges():
        #p.add_constraint(sum(sum(x[u,v,k] for u, v, label in G.edges()) for k in K == t[u,v])
    for k in Kk:
        for u in G.vertices():
            if (u==sk[k]):
                p.add_constraint(sum(x[u,v,k] for v in G.neighbors(u)) - sum(x[v,u,k] for v in G.neighbors(u)) == bk[k])
            elif (u==tk[k]):
                p.add_constraint(sum(x[u,v,k] for v in G.neighbors(u)) - sum(x[v,u,k] for v in G.neighbors(u)) == -bk[k])
            else:
                p.add_constraint(sum(x[u,v,k] for v in G.neighbors(u)) - sum(x[v,u,k] for v in G.neighbors(u)) == 0)
    for u,v, label in G.edges():
        p.add_constraint(sum(x[u,v,k] for k in Kk) <= c[0])

    for u in G.vertices():
        for k in Kk:
            if(u == tk[k]):
                p.add_constraint(w[u,k] == 0)
            else:
                p.add_constraint(w[u,k] == sum(x[u,v,k] for v in G.neighbors(u)))
    #add objective to minimize
```

```python
    p.set_objective(sum(z[u,v] for u, v, label in G.edges())*alpa + sum(sum(x[u,v,k] for u, v, label in G.edges()) for k in Kk)*bet - sum(y[u,v] for u, v, label in G.edges())*gam*0.5 + sum(sum(w[u,k] for u in G.vertices()) for k in Kk)*delt)
    #p.set_objective(c[0])
    #try:
    p.solve()
    #except MIPSolverException:
        #print("the problem has no feasible solution")
        #return []
    c_result = p.get_values(c)
    return c_result
```

[2]:
```python
import numpy as np

G = graphs.CycleGraph(10)
G.add_edges([(1, 3), (4, 9), (5, 8)])
G = DiGraph(G)
K = [(0, 1, 1), (0, 5, 2), (0, 6, 1), (1, 2, 1), (2, 9, 3), (1, 6, 2), (2, 7, 1), (2, 8, 1), (6, 5, 1), (8, 1, 2)]
alpa = 100
bet = 5
gam = 20
delt = 10
c_result = flow_minicost2(G,K,alpa,bet,gam,delt)
print(c_result)
```

```
[1. 2. 1. 1. 3. 2. 1. 1. 1. 2.]
{0: 10.0}
```

The effect of the adding is that we now need to also minimize the quantities of nodes used in the routing traffic.

[0]:
```python

```