COSC 364

Internet Technology and Engineering Flow Assignment

Michael Zhu

ID: 57656922

May 30, 2023

Problem formulation

i = index of source node

k = index of transit node

j = index of destination node

d_{kj} = capacity link transit- destination

h_{ij} = demand flow between source and destination

x_{ikj} = auxiliary decision variable

c_{ik} = capacity link source- tranist

u_{ikj} = binary variable

$$Minimize_{[x,c,d,r]} r \tag{1}$$

Subject to

$$\sum_{j=1}^{Z} x_{ikj} \leq c_{ik} \qquad i \in \{1, \dots, X\}, k \in \{1, \dots, Y\} \tag{2}$$

$$\sum_{i=1}^{X} x_{ikj} \leq d_{ik} \qquad k \in \{1, \dots, X\}, j \in \{1, \dots, Z\} \tag{3}$$

$$\sum_{k=1}^{Y} x_{ikj} \leq h_{ij} \qquad i \in \{1, \dots, X\}, j \in \{1, \dots, Z\} \tag{4}$$

$$\sum_{i=1}^{X} \sum_{j=1}^{Z} x_{ikj} \leq r \qquad k \in \{1, \dots, Y\} \tag{5}$$

$$\sum_{k=1}^{Y} u_{ikj} = 2 \qquad i \in \{1, \dots, X\}, j \in \{1, \dots, Z\} \tag{6}$$

$$x_{ikj} = \frac{u_{ikj} h_{ij}}{2} \qquad i \in \{1, \dots, X\}, k \in \{1, \dots, Y\}, j \in \{1, \dots, Z\} \tag{7}$$

$$x_{ikj} \geq 0 \qquad i \in \{1, \dots, X\}, k \in \{1, \dots, Y\}, j \in \{1, \dots, Z\} \tag{8}$$

$$r \geq 0 \tag{9}$$

$$u_{ikj} \in \{0,1\} \qquad i \in \{1, \dots, X\}, k \in \{1, \dots, Y\}, j \in \{1, \dots, Z\} \tag{10}$$

The problem at hand is to determine the objective function for to achieve load-balancing by evenly distributing the utilization across all relevant links. To achieve linearity in the objective function, an auxiliary variable 'r', is introduced. By minimizing 'r', we simultaneously minimize the objective function and achieve even distribution load across the transit nodes. The minimization process encompasses decision variables such as 'x', 'c', 'd', and 'r' (equation 1) which have their own constraints which determine the result.

Equation 2 (capacity constraint source- transit) and equation 3 (capacity constraint transit-destination) calculate the capacities of each link from source to transit and transit to destination. The path flow x_{ikj} should be equal or smaller than the flow of the corresponding c_{ik} and d_{ik}.

Equation 4 is the demand flow constraint. The sum of the load on transit nodes should be equal to demand flow between source and destination.

Equation 5 are the constraints for the load on each transit node.

Equation 7 balances the load between the 2 paths.

Equation 8 and 9 is needed as flow does not make sense hence cannot be negative.

Equation 6 and 10 are needed to restrict demand volume to single path hence binary variables u_{ikj} is needed. If data is being carried by a path then u_{ikj} = 1 otherwise u_{ikj} = 0 and path is not used. Equation 6 tell the program that network has 2 paths.

Results

Table 1: results with Y ∈ {3,4,5,6,7} , X = 7 and Z = 7

Y	**3**	**4**	**5**	**6**	**7**
Execution time (sec)	0.0577	0.0250	0.0360	0.0573	0.0959
Num links non-zero capacities	39	45	66	73	88
Transit load on nodes	130.7	98.0	78.4	65.3	56.0
Highest capacity links	c73, d37 : 44	c17, d27 : 38	d26, d27 : 30	d57 : 59	c66 : 22

The execution time of having 4 and 5 nodes is faster than the execution time of having 3 nodes might be because 3 nodes does not have enough capacity to carry data from 7 source nodes to 7 destination nodes therefore there is a bottle neck. The time is increased for 6 and 7 transit nodes probably due to CPLEX trying to load balance between the extra nodes as the network would be more complex.

The number of links with non-zero capacities increased as nodes increased because new links would be established between the new nodes that can carry the loads. Therefore more links would be used to transfer the loads.

The transit load decreases as the number of nodes increases because there are more nodes to spread the loads to. The results from CPLEX show that the transit nodes all have the same load value which shows that all the networks are balanced.

The number of highest capacity links decrease as transit nodes increase because there are more links due to more nodes therefore the loads would be redistributed, and links would reduce in capacity. However, there is an outlier at transit node Y = 6 which could be due to performance as d57 is favoured compared to other links.

Appendix A

```python
import sys
import subprocess
import time

def gen_capacity_constraints_st(src, trans, dest):
    capacity_constraints_st = ""
    for i in range(1, src + 1):
        for k in range(1, trans + 1):
            temp = [f"x{i}{k}{j}" for j in range(1, dest + 1)]
            string = " + ".join(temp) + f" - c{i}{k} <= 0"
            capacity_constraints_st += f"{string}\n"
    return capacity_constraints_st

def gen_capacity_constraints_td(src, trans, dest):
    capacity_constraints_td = ""
    for k in range(1, trans + 1):
        for j in range(1, dest + 1):
            temp = [f"x{i}{k}{j}" for i in range(1, src + 1)]
            string = " + ".join(temp) + f" - d{k}{j} <= 0"
            capacity_constraints_td += f"{string}\n"
    return capacity_constraints_td

def gen_demand_constraints(src, trans, dest):
    demand_constraints = ""
    for i in range(1, src + 1):
        for j in range(1, dest + 1):
            temp = [f'x{i}{k}{j}' for k in range(1, trans + 1)]
            string = " + ".join(temp) + f" = {i + j}"
            demand_constraints += f"{string}\n"
    return demand_constraints

def gen_binary_constraints(src, trans, dest, paths):
    binary_constraints = ""
    for i in range(1, src + 1):
        for j in range(1, dest + 1):
            temp = [f"u{i}{k}{j}" for k in range(1, trans + 1)]
            string = " + ".join(temp) + f" = {paths}"
            binary_constraints += f"{string}\n"
```

```python
    return binary_constraints

def gen_transit_load_constraints(src, trans, dest):
    transit_load_constraints = ""
    for k in range(1, trans + 1):
        constraint = ""
        for j in range(1, src + 1):
            for i in range(1, dest + 1):
                if constraint != "":
                    constraint += " + "
                constraint += f"x{i}{k}{j}"
        constraint += f" - r <= 0"
        transit_load_constraints += f"{constraint}\n"
    return transit_load_constraints

def gen_flow_constraints(src, trans, dest, paths):
    flow_constraints = ""
    for i in range(1, src + 1):
        for j in range(1, dest + 1):
            temp = [f"{paths} x{i}{k}{j} - {i + j} u{i}{k}{j} = 0" for k in
range(1, trans + 1)]
            flow_constraints += "\n".join(temp) + "\n"
    return flow_constraints

def gen_bound_constraints(src, trans, dest):
    bound_constraints = ""
    for i in range(1, src + 1):
        for j in range(1, dest + 1):
            for k in range(1, trans + 1):
                bound_constraints += f"0 <= x{i}{k}{j}\n"
    return bound_constraints

def gen_binary_variables(src, trans, dest):
    binary_variables = ""
    for i in range(1, src + 1):
        for j in range(1, trans + 1):
            for k in range(1, dest + 1):
                binary_variables += f"u{i}{k}{j}\n"
    return binary_variables

def assemble_lp_file(demand_constraint, capacity_constraints_st,
capacity_constraints_td, bound_constraints,
                     binary_variables, binary_constraints, flow_constraints,
transit_load_constraints):
    with open('Documents/temp.lp', 'w') as f:
        f.write("Minimize\n r\nSubject to\n")
        f.write("Capacity source to transit: \n")
        f.write(capacity_constraints_st)
```

```
        f.write("Capacity transit to destination: \n")
        f.write(capacity_constraints_td)

        f.write("demand constraint:\n")
        f.write(demand_constraint)

        f.write("Transit load constraints: \n")
        f.write(transit_load_constraints)

        f.write("Demand flow: \n")
        f.write(flow_constraints)

        f.write("Binary variable constraints: \n")
        f.write(binary_constraints)

        f.write("Bounds \n")
        f.write(bound_constraints)
        f.write("0 <= r\n")

        f.write("Binary \n")
        f.write(binary_variables)

        f.write("End")

def run_cplex():
    transit_load = {}
    load = 0
    max_node = 0
    capture_load = {}

    link_count = 0
    highest_capacity = {}
    max_value = 0
    highest_cap = {}

    start_time = time.time()
    args = ["cplex", "-c", "read", "Documents/temp.lp", "optimize", "display
solution variables -"]
    process = subprocess.Popen(args, stdout=subprocess.PIPE)
    output, _ = process.communicate()
    end_time = time.time()
    run_time = end_time - start_time
    result = output.decode().strip()

    lines = result.split('\n')
    for line in lines:
        if line.startswith("x"):
```

```
            variable, value = line.split()
            transit_load[variable] = float(value)
        elif line.startswith("c") or line.startswith("d"):
            link_count += 1
            variable, value = line.split()
            highest_capacity[variable] = float(value)

    current_Y = 1
    load = 0
    for n in transit_load:
        if int(n[2]) > max_node:
            max_node = int(n[2])

    while current_Y <= max_node:
        load = 0
        for key in transit_load:
            if key[2] == str(current_Y):
                load += transit_load[key]
        capture_load[current_Y] = load
        current_Y += 1

    for key, value in highest_capacity.items():
        if value > max_value:
            max_value = value
            highest_cap.clear()
            highest_cap[key] = value
        elif value == max_value:
            highest_cap[key] = value

    print(f"Num links {link_count}")
    for key, value in highest_cap.items():
        print(f"link {key} : value {value}")

    for key, value in capture_load.items():
        print(f" node {key} : load {value}")
    return run_time, capture_load, highest_cap

def write_lp(Y):
    # if len(sys.argv) < 4:
    #     print("input three arguments: source nodes, transit nodes, 
destination nodes.")
    #     exit(-1)

    X = 3 #int(sys.argv[1])
    # Y =   int(sys.argv[2])
    Z = 3 #int(sys.argv[3])

    paths = 2
```

```
    demand_constraint = gen_demand_constraints(X, Y, Z)
    capacity_constraints_st = gen_capacity_constraints_st(X, Y, Z)
    capacity_constraints_td = gen_capacity_constraints_td(X, Y, Z)
    bound_constraints = gen_bound_constraints(X, Y, Z)
    binary_variables = gen_binary_variables(X, Y, Z)
    binary_constraints = gen_binary_constraints(X, Y, Z, paths)
    flow_constraints = gen_flow_constraints(X, Y, Z, paths)
    transit_load_constraints = gen_transit_load_constraints(X, Y, Z)
    assemble_lp_file(demand_constraint, capacity_constraints_st,
capacity_constraints_td, bound_constraints,
                     binary_variables, binary_constraints, flow_constraints,
transit_load_constraints)

def main():
    for Y in range(2, 3):
        write_lp(Y)
        run_time, transit_load, highest_capacity = run_cplex()
        print(f"Y = {Y}")
        print("CPLEX run time:", run_time)
        print("Transit load:", transit_load)
        print("Highest link capacity:", highest_capacity)
        print()

if __name__ == "__main__":
    main()
```

Appendix B

Minimize

r

Subject to

Capacity source to transit:

x111 + x112 + x113 - c11 <= 0

x121 + x122 + x123 - c12 <= 0

x211 + x212 + x213 - c21 <= 0

x221 + x222 + x223 - c22 <= 0

x311 + x312 + x313 - c31 <= 0

x321 + x322 + x323 - c32 <= 0

Capacity transit to destination:

x111 + x211 + x311 - d11 <= 0

x112 + x212 + x312 - d12 <= 0

$x113 + x213 + x313 - d13 <= 0$

$x121 + x221 + x321 - d21 <= 0$

$x122 + x222 + x322 - d22 <= 0$

$x123 + x223 + x323 - d23 <= 0$

demand constraint:

$x111 + x121 = 2$

$x112 + x122 = 3$

$x113 + x123 = 4$

$x211 + x221 = 3$

$x212 + x222 = 4$

$x213 + x223 = 5$

$x311 + x321 = 4$

$x312 + x322 = 5$

$x313 + x323 = 6$

Transit load constraints:

$x111 + x211 + x311 + x112 + x212 + x312 + x113 + x213 + x313 - r <= 0$

$x121 + x221 + x321 + x122 + x222 + x322 + x123 + x223 + x323 - r <= 0$

Demand flow:

$2 x111 - 2 u111 = 0$

$2 x121 - 2 u121 = 0$

$2 x112 - 3 u112 = 0$

$2 x122 - 3 u122 = 0$

$2 x113 - 4 u113 = 0$

$2 x123 - 4 u123 = 0$

$2 x211 - 3 u211 = 0$

$2 x221 - 3 u221 = 0$

$2 x212 - 4 u212 = 0$

$2 x222 - 4 u222 = 0$

$2 x213 - 5 u213 = 0$

$2 x223 - 5 u223 = 0$

$2 x311 - 4 u311 = 0$

2 x321 - 4 u321 = 0

2 x312 - 5 u312 = 0

2 x322 - 5 u322 = 0

2 x313 - 6 u313 = 0

2 x323 - 6 u323 = 0

Binary variable constraints:

u111 + u121 = 2

u112 + u122 = 2

u113 + u123 = 2

u211 + u221 = 2

u212 + u222 = 2

u213 + u223 = 2

u311 + u321 = 2

u312 + u322 = 2

u313 + u323 = 2

Bounds

0 <= x111

0 <= x121

0 <= x112

0 <= x122

0 <= x113

0 <= x123

0 <= x211

0 <= x221

0 <= x212

0 <= x222

0 <= x213

0 <= x223

0 <= x311

0 <= x321

0 <= x312

0 <= x322

0 <= x313

0 <= x323

0 <= r

Binary

u111

u121

u131

u112

u122

u132

u211

u221

u231

u212

u222

u232

u311

u321

u331

u312

u322

u332

End