COSC364 – FLOW PLANING ASSIGNMENT 2

Yuxi (Harry) Chen : 31597634

Table of Contents

Introduction

The objective is to minimise the maximum load across all transit nodes. The traffic between source and destination pair is constraint to exactly 2 paths. Additionally, the load must be distributed evenly between the transit nodes. The subsequent sections detail the formulation of the problem as well as the LP file results. The source code is found in Appendix A, B and the 3-2-3 LP file is found in Appendix C.

Symbol notation:

- X = number of source nodes
- Y = number of transit nodes
- Z = number of destination nodes
- i = i^{th} source node
- k = k^{th} transit node
- j = j^{th} destination node
- h_{ij} = i + j (Demand flow between S_i and D_j)
- c_{ik} = link capacity between S_i and T_k
- d_{kj} = link capacity between T_k and D_j
- x_{ikj} = auxiliary decision variable based on the path from source to destination
- u_{ikj} = binary decision variable

Problem Formulation

The introduction of the auxiliary variable r enables linearization of the objective function, which allows the following to be solvable using CPLEX (once they're formatted into an LP file). The r represents the maximum load on any transit node.

Note: $\boldsymbol{i} \in \{\mathbf{1},\ldots,\boldsymbol{X}\}$, $\boldsymbol{k} \in \{\mathbf{1},\ldots,\boldsymbol{Y}\}$, $\boldsymbol{j} \in \{\mathbf{1},\ldots,\boldsymbol{Z}\}$ otherwise specified in the equation.

Objective function:

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

Subject to:

$$\sum_{j=1}^{Z} x_{ikj} \leq c_{ik} \tag{2}$$

$$\sum_{i=1}^{X} x_{ikj} \leq d_{kj} \tag{3}$$

$$\sum_{k=1}^{Y} x_{ikj} = h_{ij} \tag{4}$$

$$\sum_{i=1}^{X} \sum_{j=1}^{Z} x_{ikj} \leq r \tag{5}$$

$$\sum_{k=1}^{Y} u_{ikj} = 2 \tag{6}$$

$$x_{ikj} = \frac{u_{ikj}\, h_{ij}}{2} \tag{7}$$

$$u_{ikj} \in \{0,1\} \quad (8)$$

$$r \geq 0 \quad (9)$$

$$x_{ikj} \geq 0 \quad (10)$$

$$d_{kj} \geq 0 \quad (11)$$

$$c_{ik} \geq 0 \quad (12)$$

Equation Rational

The following lists the explanations for each equation provided from the section prior.

- Equation (1): This is the objective function it's used to minimise the maximum load (r) across all transit nodes. This means trying to find the smallest possible values for r such that the maximum load on any transit node doesn't exceed r.

- Equation (2) and (3): These are the constraints that ensures the total flow from a node does not exceed the capacity. More specifically, the Equation (1) and (2) ensures the flow from a source node i to a transit node k don't exceed the capacity, C_{ik}. Similarly, the flow from node k to a destination node j don't exceed the capacity, d_{kj}. These prevents any single link from being overloaded.

- Equation (4): This constraint ensures that the demand flow is meet. The constraint ensures that the total load on each transit node for a given node i and node j equals the demand h_{ij}.

- Equation (5): This is the constraint that ensures the total load on any transit node don't exceed the maximum load r. This helps in balancing the load across all transit nodes.

- Equation (6): This constraint ensures that exactly 2 paths should be used for each pair of source and destination node. The binary variable given indicates whether a path is used (1 if used, 0 otherwise).

- Equation (7): This constraint ensures the load is evenly split between the 2 paths used for each pair of source and destination node. This is done by dividing the total demand by 2 and allocated to each path.

- Equation (8): The constraint enables the usage of binary variable. It ensures that it can only take 2 values, 1 or 0.

- Equation (9), (10), (11), and (12): These constraints ensure that all variables are non-negative. This is required for a linear programming to ensure that the solution is physically feasible (i.e. flows and capacities can't be less than 0).

CPLEX Results

The LP file generated using the code in Appendix B is ran through the CPLEX program. X=7, Z=7, and Y ∈ $\{3, 4, 5, 6, 7\}$. The results collected are shown in Table 1 below. The execute time is calculated by averaging over 5 values.

Table 1:CPLEX results from running the LP files with these node values X=7, Z=7, and Y ∈ $\{3, 4, 5, 6, 7\}$

Y	Run time (s)	Nonzero capacity links	Lowest Load on transit nodes	Highest load on transit nodes	Highest capacity link
3	0.039	42	130.5	131.0	D_{17} = 34.5
4	0.045	54	98.0	98.0	D_{37} = 34.5
5	0.051	67	78.0	78.5	C_{73} = 29.0
6	0.064	80	65.0	65.5	C_{53}, C_{54} = 23.5
7	0.12	92	56.0	56.0	D_{57} = 23.0

Conclusion

Using the results from Table 1 above it can be identified that as the number of transit nodes (Y) increases, the complexity of the system also increases. This causes longer computation times as the CPLEX program must solve more paths and constraints.

Additionally, nonzero capacity links increases with more transit nodes. This is simply due to the increase in possible links as stated before. The CPLEX solver attempts to evenly distribute the load across the transit as much as possible. However, due to the topology of the node network there will still be minor discrepancies between the load balanced on the transit nodes (seen from the difference between lowest and highest load on transit nodes).

The highest capacity links are proportionally decreasing with increasing Y. The dependency on a single link is decreased as more transit nodes are introduced. As such the load gets more distributed across multiple links, thus reducing the capacity for any single link.

Finally, the constraints and objective function drive the solution towards minimizing the maximum load on any transit node. This is reflected in the decreasing trend on the lowest load on transit nodes.

Appendix A

```
"""
COSC364 flow assignment
Created by Yuxi (Harry) Chen

"""
import sys

MAX_PATH_NUM = 2

def createDemandConstraint(s,t,d):
    """creates the demand volmue ( hij = i+j) for a path from
source->transit->destination (Xijk)
        returns a string
    """
    string = "\n\CONSTRAINTS \n"
    for i in range(1, s + 1):
        for j in range(1, d + 1):
            for k in range(1,t + 1):
                if k == t:
                    string += f"x{i}{k}{j} = {i + j} \n"
                else:
                    string += f"x{i}{k}{j} + "
    return string

def createStoTCapacity(s,t,d):
    """Calculate capacity contraints for each link from source to transit
nodes
        returns a string
    """
    string = "\n\CAPACITY FOR SOURCE TO TRANSIT \n"
    for i in range(1, s+1):
        for k in range(1, t+1):
            for j in range(1,d+1):
                if j == d:
                    string += f"x{i}{k}{j} - c{i}{k} = 0 \n"
                else:
                    string += f"x{i}{k}{j} + "
    return string

def createTtoDCapacity(s,t,d):
    """Calculate capacity contraints for each link from transit to dest
nodes"""
    string = "\n\CAPCITY FOR TRANSIT TO DEST \n"
    for k in range(1,t + 1):
        for j in range(1, d + 1):
            for i in range(1, s + 1):
```

```
                if i == s:
                    string += f"x{i}{k}{j} - d{k}{j} = 0 \n"
                else:
                    string += f"x{i}{k}{j} + "
    return string

def createNonNegativityConstraints(s,t,d):
    """Create non-negative constraints onto variables"""
    string = "\nBOUNDS \nr >= 0 \n"
    #Ensure non-negativity for each path
    for i in range(1, s + 1):
        for k in range(1,t + 1):
            for j in range(1, d + 1):
                string+=f"x{i}{k}{j} >= 0 \n"

    #Ensure non-negative capacity from Source to transit
    for i in range(1, s + 1):
        for k in range(1, t + 1):
            string+=f"c{i}{k} >= 0\n"

    #Ensure non-negative capacity from transit to destination
    for k in range(1,t+1):
        for j in range(1,d+1):
            string += f"d{k}{j} >= 0 \n"
    return string

def createDemandFlow(s,t,d):
    """Calculates and returns a string for the equal path flow requirement"""
    string = "\n\DEMAND FLOW \n"
    for i in range(1, s + 1):
        for k in range(1, t+ 1):
            for j in range(1, d + 1):
                string += (f"2x{i}{k}{j} - {i + j}u{i}{k}{j} = 0 \n")
    return string

def createMaxPathConstraint(s,t,d):
    """Ensure 2 active transit nodes( max path is 2 ) and return it as a
string"""
    #Ensure 2 path max per source to destination (from specs)
    string = "\n\BINARY VARIABLES \n"
    for i in range(1,s + 1):
        for j in range(1, d + 1):
            for k in range(1, t + 1):
                if k == t:
                    string += (f"u{i}{k}{j} = {MAX_PATH_NUM} \n")
                else:
                    string += (f"u{i}{k}{j} + ")
    return string
```

```
def createTransitLoad(s,t,d):
    """Calculate load constraint for each path and return it as a String"""
    string = "\n\TRANSIT LOAD CONSTRAINTS\n"
    #Calculate the sum of each transit load
    for k in range (1, t + 1):
        for i in range(1, s + 1):
            for j in range(1,d + 1):
                if (j == d) and (i == s):
                    string += (f"x{i}{k}{j} - r <= 0 \n")
                else:
                    string += (f"x{i}{k}{j} + ")
    return string

def createBinaryVariables(s,t,d):
    """create binary variables and return it as a String"""
    string = "\nBINARY \n"
    for i in range(1,s+1):
        for j in range(1, d + 1):
            for k in range(1, t + 1):
                string += (f"u{i}{k}{j} \n")
    return string

def generateLPFile(s,t,d):
    """Create LP file based on required specifications
        Note:
        total Source nodes = s
        total transit nodes = t
        total destination nodes = d

        individual Source node = i,
        individual Transit node = k,
        individual Destination node = j
    """
    with open(f"{s}{t}{d}.lp", "w") as file:
        file.write("MINIMIZE \nr \n\nSUBJECT TO \n")
        file.write(createDemandConstraint(s,t,d))
        file.write(createStoTCapacity(s,t,d))
        file.write(createTtoDCapacity(s,t,d))
        file.write(createTransitLoad(s,t,d))
        file.write(createDemandFlow(s,t,d))
        file.write(createMaxPathConstraint(s,t,d))
        file.write(createNonNegativityConstraints(s,t,d))
        file.write(createBinaryVariables(s,t,d))
        file.write("END \n")

def checkInputParameters():
    """Check correct inputs from user"""
```

```
    try:
        #Assume the user knows that nodes can't be negative
        source, transit, dest = abs(int(sys.argv[1])), abs(int(sys.argv[2])),
abs(int(sys.argv[3]))
        if (transit >= MAX_PATH_NUM):
            return [source, transit, dest]
        else:
            print("Incorrect arguments are given: Transit nodes must be >= 2
")
    except Exception as e:
        print(f"Error: {e}.")

def main():
    userInputs = checkInputParameters()
    if userInputs:
        generateLPFile(userInputs[0], userInputs[1], userInputs[2])

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

Appendix B

```
"""
COSC364 flow assignment
Created by Yuxi (Harry) Chen
10/05/24
"""
import subprocess
import time

def print_results(transit_nodes, vars_c, vars_d, execute_time):
    print("Execute_time:", execute_time)

    #print the list of C capacity
    highest_capacity_c = vars_c[0]
    print("CAPACITY")
    for item in vars_c:
        if item[1] > highest_capacity_c[1]:
            highest_capacity_c = item
            print(highest_capacity_c)

    #print the list of D capacity
    highest_capacity_d = vars_d[0]
    for item in vars_d:
        if item[1] > highest_capacity_d[1]:
            highest_capacity_d = item
            print(highest_capacity_d)

    #Calculate number of non-zero capacity links
    non_zero_links_d = len([value for (name, value) in vars_d if value != 0])
    non_zero_links_c = len([value for (name, value) in vars_c if value != 0])
    non_zero_links = non_zero_links_d+non_zero_links_c
    print("Number of non-zero links:", non_zero_links)

    #Print the load for each transit node
    for (transit_node, load) in transit_nodes:
        print(f"Transit node {transit_node}: {load}")

def run_cplex():
    start_time = time.time()
    command =  ["cplex", "-c", "read", r"#PATH_TO_FILE", "optimize", "display
solution variables -"]
    proccess = subprocess.Popen(command, stdout=subprocess.PIPE)
    results = proccess.stdout.read().splitlines()
    end_time = time.time()
    run_time = end_time - start_time
    data = [d.decode('utf-8') for d in results]
```

```
    #lists for capacity and nodes
    vars_x = []
    vars_c = []
    vars_d = []
    transit_nodes = {}

    #Extract the needed data and process it as tuples
    for line in data:
        parts = line.split()
        if len(parts) == 2:
            name, value = parts
            try:
                value = float(value)
                if name.startswith('x'):
                    vars_x.append((name, value))
                elif name.startswith('c'):
                    vars_c.append((name, value))
                elif name.startswith('d'):
                    vars_d.append((name, value))
            except ValueError:
                #Skips the non-readable lines/useless lines from data
                continue

    #Calculate transit node load and append to dict
    for node, load in vars_x:
        # Extract the transit node K from str(xikj)
        transit_node = node[2]
        if transit_node in transit_nodes:
            transit_nodes[transit_node] += load
        else:
            transit_nodes[transit_node] = load

    return transit_nodes, vars_c, vars_d, run_time

def main():
    transit_nodes, vars_c,vars_d, run_time=run_cplex()
    print_results(transit_nodes, vars_c,vars_d, run_time)

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

Appendix C

MINIMIZE

r

SUBJECT TO

\CONSTRAINTS

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

\CAPACITY FOR 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

\CAPCITY FOR TRANSIT TO DEST

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

\TRANSIT LOAD CONSTRAINTS

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

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

\DEMAND FLOW

2x111 - 2u111 = 0

2x112 - 3u112 = 0

2x113 - 4u113 = 0

2x121 - 2u121 = 0

2x122 - 3u122 = 0

2x123 - 4u123 = 0

2x211 - 3u211 = 0

2x212 - 4u212 = 0

2x213 - 5u213 = 0

2x221 - 3u221 = 0

2x222 - 4u222 = 0

2x223 - 5u223 = 0

2x311 - 4u311 = 0

2x312 - 5u312 = 0

2x313 - 6u313 = 0

2x321 - 4u321 = 0

2x322 - 5u322 = 0

2x323 - 6u323 = 0

\BINARY VARIABLES

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

r >= 0

x111 >= 0

x112 >= 0

x113 >= 0

x121 >= 0

x122 >= 0

x123 >= 0

x211 >= 0

x212 >= 0

x213 >= 0

x221 >= 0

x222 >= 0

x223 >= 0

x311 >= 0

x312 >= 0

x313 >= 0

x321 >= 0

x322 >= 0

x323 >= 0

c11 >= 0

c12 >= 0

c21 >= 0

c22 >= 0

c31 >= 0

c32 >= 0

d11 >= 0

d12 >= 0

d13 >= 0

d21 >= 0

d22 >= 0

d23 >= 0

BINARY

u111

u121

u112

u122

u113

u123

u211

u221

u212

u222

u213

u223

u311

u321

u312

u322

u313

u323

END