COSC 364

Internet Technologies and Engineering

Flow Assignment

Blake Manson – 54426511

bma206@uclive.ac.nz

28/05/2024

Problem Formulation

The objective of this assignment is to balance the load on all transit nodes.

Variable definitions:

i = Index of source node, X = number of source nodes, S_i = source node i

k = Index of transit node, Y = number of transit nodes, T_k = transit node k

j = Index of destination node, Z = number of destination nodes, D_j = destination node j

x_{ikj} = Flow path from S_i to D_j through T_k

h_{ij} = Demand volume from S_i to D_j

c_{ik} = Link capacity from S_i to T_k, d_{kj} = Link capacity from T_k to D_j

u_{ikj} = Binary Variable for path x_{ikj}

Constraints:

1. Each demand volume must be split over exactly 2 different paths.
$$Y \geq 2$$
2. Demand constraint: The sum of all path flows from source node i through transit node k to destination node j must give the total demand volume h.
$$\sum_{k=1}^{Y} x_{ikj} = h_{ij} \;\; for\; i \in \{1, \dots, X\}, j \in \{1, \dots, Z\}$$
3. Non-negativity constraints: Each path flow and link capacity must be non-negative, as negative data rates do not make sense.
$$x_{ikj} \geq 0, \qquad c_{ik} \geq 0, \qquad d_{kj} \geq 0$$
4. Capacity constraints:
    a. The link capacity between a source node S_i and a transit node T_k is represented by c_{ik}. The sum of all flows through each link must be less than or equal to the link capacity c_{ik}.
    $$\sum_{j=1}^{Z} x_{ikj} \leq c_{ik} \;\; for\; i \in \{1, \dots, X\}, k \in \{1, \dots, Y\}$$
    b. The link capacity between a transit node T_k and a destination node D_k is represented by d_{kj}. The sum of all flows through each link must be less than or equal to the link capacity d_{kj}.
    $$\sum_{i=1}^{X} x_{ikj} \leq d_{kj} \;\; for\; k \in \{1, \dots, Y\}, j \in \{1, \dots, Z\}$$

5. Binary constraint: For this problem, each demand volume needs to be split over exactly 2 paths. To minimise the load on each transit node, these flows must be equal.

$$\sum_{k=1}^{Y} u_{ikj} = 2 \ \ for\ i \in \{1, \dots, X\}, j \in \{1, \dots, Z\}, u \in \{0,1\}$$

$$x_{ikj} = u_{ikj} \times \frac{h_{ij}}{2}$$

6. Minimising the load is done by introducing an auxiliary variable *r* used to represent the utilisation of load at each transit node. By minimizing r, the demand volume of each transit node becomes as small as possible, balancing loads over all transit nodes.

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

These constraints form the minimisation problem below:

Minimize $_{[x,\ c,\ d\ r]}$ r

Subject to

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

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

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

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

$$\sum_{i=1}^{X} x_{ikj} \leq d_{kj} \qquad for\ k \in \{1, \dots, Y\}, j \in \{1, \dots, Z\}$$

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

$$h_{ij} = i + j \qquad for\ i \in \{1, \dots, X\}, j \in \{1, \dots, Z\}$$

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

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

$$c_{ik} \geq 0 \qquad for\ i \in \{1, \dots, X\}, k \in \{1, \dots, Y\}$$

$$d_{kj} \geq 0 \qquad for\ k \in \{1, \dots, Y\}, j \in \{1, \dots, Z\}$$

$$r \geq 0$$

Results

# of T nodes	3	4	5	6	7
Execution time (s)	0.494	0.285	0.317	0.306	1.849
N of non-zero capacities	42	53	64	79	83
Lowest transit load	1, 2 = 130.5	1, 2, 3, 4 = 98	2 = 78	1, 4 = 65	1, 2, 3, 4, 5, 6, 7 = 56
Highest transit load	3 = 131	1, 2, 3, 4 = 98	1, 3, 4, 5 = 78.5	2, 3, 5, 6=65.5	1, 2, 3, 4, 5, 6, 7 = 56
Highest capacity links	D_{17} = 35	D_{37} = 35	C_{73} = 30	C_{55} = 25	C_{56} = 25

Table 1: CPLEX results for varying Y, with fixed X = Y = 7.

Explanation

The execution time is not what was expected, as the times all seemed very close together until a massive jump at 7 transit nodes, whereas it was expected that the time would steadily increase as more nodes means a more complex calculation. The number of non-zero capacities however did steadily increase with the number of transit nodes, which is expected as with more nodes, the demand is spread out across more links to reduce the load per link. For every number of transit nodes, the transit loads were all within 0.5 of each other, meaning that the load balancing was successful. The overall load average decreases against the number of transit nodes, as more transit nodes means the load is evenly distributed over a larger number. The highest capacity links get values smaller as the number of transit nodes increases because as the number of non-zero capacity links increases, the load again is split over more links giving a lower load on each link.

Appendix

Source Code:

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

"""
flow.py
Program to create an LP file of constraints to balance loads equally across
links
    - Each load must be split over exactly 2 paths

Note: This program only runs on Linux systems with CPLEX - check directories
in CPLEX function
"""
# Author: Blake Manson
# Date Created: 02/05/2024
# Date Modified: 29/05/2024

import time
import subprocess


def createLPFileName() -> str:
    """
    Creates the name for an lp file with inputs for X, Y and Z
    Returns:
        Inputted values for X, Y and Z
        The name for the file of form XYZ.lp
    """
    X = int(input("Number of source nodes (X): "))
    Y = int(input("Number of transit nodes (Y): "))
    Z = int(input("Number of destination nodes (Z): "))
    filename = f"{X}{Y}{Z}.lp"
    return X, Y, Z, filename

def writeContraints(X: int, Y: int, Z: int, filename: str):
    """
    Writes all constraints and bounds of minimization formula to XYZ.lp
    """
    f = open(filename, 'w')
    f.write(f"Minimize\n    r\nSubject to\n")
    f.write(demandVolume(X, Y, Z))
    f.write(binaryVariable(X, Y, Z))
    f.write(demandFlow(X, Y, Z))
    f.write(sourceTransitCap(X, Y, Z))
    f.write(transitDestinationCap(X, Y, Z))
```

```
    f.write(loadBalancing(X, Y, Z))
    f.write("Bounds\n")
    f.write(bounds(X, Y, Z))
    f.write("    r >= 0\n")
    f.write("Binaries\n")
    f.write(binaries(X, Y, Z))
    f.write("End")
    f.close()
    return

def CPLEX(X: int, Y: int, Z: int, filename: str):
    """
    Runs CPLEX optimisation on the LP file
    Saves STDOUT as a file in of form XYZOut.txt
    """
    # Create output filename for CPLEX
    outputFile = f"{X}{Y}{Z}CPLEX.txt"

    # Paths for CPLEX and the LP file - Change if relevant
    CPLEXPath = "/csse/users/bma206/cplex/cplex/bin/x86-64_linux/cplex"
    filePath = "/media/bma206/8C0D-21E5/COSC364/PlanningAssignment/flow-
planning/"

    # Generate command from file paths and filename
    command = [CPLEXPath, "-c", "read " + filePath + filename, "optimize",
"display solution variables -"]

    # Starts a timer to measure execution time
    startTime = time.time()

    # Run CPLEX as a subprocess and wait for finish, then stop timer
    process = subprocess.Popen(command, stdout=open(outputFile, "wb"))
    process.wait()
    endTime = time.time()

    # Calculate and print execution time to 3dp
    timeTaken = endTime - startTime
    print(f"CPLEX Execution Time: {timeTaken:.3f}s")

    return

def demandVolume(X: int, Y: int, Z: int) -> str:
    """Returns demand volume constraints"""
    output = ""
    for i in range(1, X+1):
        for j in range(1, Z+1):
            output += "    "
            for k in range(1, Y+1):
```

```
                if k != (Y):
                    output += f"x{i}{k}{j} + "
                else:
                    output += f"x{i}{k}{j} = {i+j}\n"
    return output

def binaryVariable(X: int, Y: int, Z: int) -> str:
    """Returns binary variable constraints"""
    output = ""
    for i in range(1, X+1):
        for j in range(1, Z+1):
            output += "    "
            for k in range(1, Y+1):
                if k != (Y):
                    output += f"u{i}{k}{j} + "
                else:
                    output += f"u{i}{k}{j} = 2\n"
    return output

def demandFlow(X: int, Y: int, Z: int) -> str:
    """Returns demand flow constraints"""
    output = ""
    for i in range(1, X+1):
        for j in range(1, Z+1):
            for k in range(1, Y+1):
                output += f"    2 x{i}{k}{j} - {i+j} u{i}{k}{j} = 0\n"
    return output

def sourceTransitCap(X: int, Y: int, Z: int) -> str:
    """Returns source-transit capacity constraints"""
    output = ""
    for i in range (1, X+1):
        for k in range (1, Y+1):
            output += "    "
            for j in range(1, Z+1):
                if j != (Z):
                    output += f"x{i}{k}{j} + "
                else:
                    output += f"x{i}{k}{j} - c{i}{k} <= 0\n"
    return output

def transitDestinationCap(X: int, Y: int, Z: int) -> str:
    """Returns transit-destination capacity constraints"""
    output = ""
    for k in range(1, Y+1):
        for j in range(1, Z+1):
            output += "    "
            for i in range(1, X+1):
```

```
                if i != (X):
                    output += f"x{i}{k}{j} + "
                else:
                    output += f"x{i}{k}{j} - d{k}{j} <= 0\n"
    return output

def loadBalancing(X: int, Y: int, Z: int) -> str:
    """Returns load balancing constraints"""
    output = ""
    for k in range(1, Y+1):
        output += "    "
        for j in range(1, Z+1):
            for i in range(1, X+1):
                if i != (X) or j != (Z):
                    output += f"x{i}{k}{j} + "
                else:
                    output += f"x{i}{k}{j} - r <= 0\n"
    return output

def bounds(X: int, Y: int, Z: int) -> str:
    """Returns the bounds of x, c and d"""
    output = ""
    for i in range(1, X+1):
        for k in range(1, Y+1):
            for j in range(1, Z+1):
                output += f"    x{i}{k}{j} >= 0\n"
    for i in range(1, X+1):
        for k in range(1, Y+1):
            output += f"    c{i}{k} >= 0\n"
    for k in range(1, Y+1):
        for j in range(1, Z+1):
            output += f"    d{k}{j} >= 0\n"
    return output

def binaries(X: int, Y: int, Z: int) -> str:
    """Returns binary constraints"""
    output = ""
    for i in range(1, X+1):
        for k in range(1, Y+1):
            for j in range(1, Z+1):
                output += f"    u{i}{k}{j}\n"
    return output

def main():
    """
    Main function: Creates file with inputted X, Y, and Z
    Writes constraints formula to XYZ.lp
    Runs CPLEX optimization on file and saves to XYZCPLEX.txt
```

```
    """
    X, Y, Z, filename = createLPFileName()
    writeContraints(X, Y, Z, filename)
    CPLEX(X, Y, Z, filename)

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

LP file (XYZ) = (323):

Minimize

r

Subject to

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

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

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

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

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

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

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

Bounds

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

r >= 0

Binaries

u111

u112

u113

u121

u122

u123

u211

u212

u213

u221

u222

u223

u311

u312

u313

u321

u322

u323

End