COSC364

Internet Technology and Engineering

# Flow Planning

Authors
Nicholas Ranum 53147503
Erica Shipley 23108940

Supervisor
Andreas Willig

May 29 2020

University of Canterbury

## Percentage Contribution

Nicholas – 50%
Erica – 50%

## Problem Description

Given a network with X source nodes, Y transit nodes and Z source note a problem was designed to generate an LP file to be run in the CPLEX algorithm to balance the load on the transit nodes. The structure of the network is shown in Figure 1.

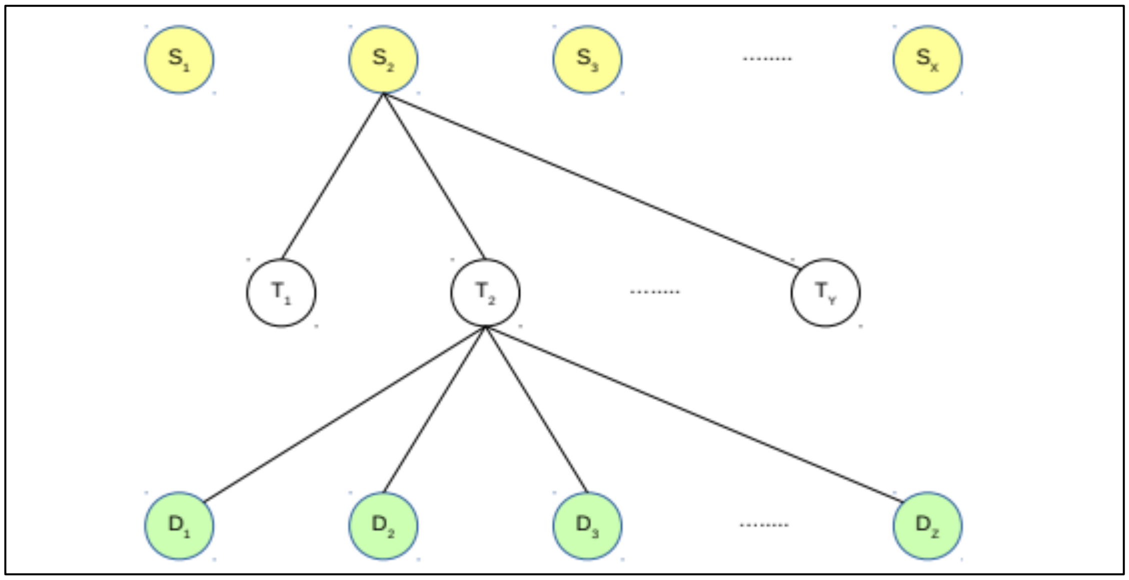

*Figure 1. Network topology*

Each source node has a demand volume for every destination node and the demand volume must be split into two equal demand flows that travel through two transit nodes.

The demand volume for source node $S_i$ to destination node $D_j$ is given by

$$h_{ij} = i + j$$

## Problem Formulation

The problem is a load balancing problem within a network constrained to restrict the demand volume to exactly two paths. It was solved by introducing binary variable constants '$u_{ij}$' and an auxiliary variable 'r' to minimize the objective function.

The objective function (Formula 1) introduces the auxiliary variable ‘r’ to linearise the objective function so that the problem can be solved as a linear problem. By minimizing ‘r’ you are effectively minimizing the flow that is run through each of the transit nodes.

$$\text{minimize}_{[x,c,d,r]}\ r \qquad \text{(Equation 1)}$$

The decision variables are the amount of demand volume between source node i and destination node j that is routed through transit node k. This leads to the first of the problem’s constraints, the demand constraint (Equation 2) which says that the sum of demand flows between $S_i$ and $D_j$ is equal to the demand volume between them, $h_{ij}$.

$$\sum_{k=1}^{Y} x_{ikj} = h_{ij} \text{ (Equation 2)}$$

The capacities of the links need to be determined, hence they are decision variables. The capacity between source node ($S_i$) and transit node ($T_k$) is denoted as $c_{ik}$. The capacity between transit node ($T_k$) and destination node ($D_j$) is denoted as $d_{kj}$. The second (Equation 3) and third (Equation 4) constraints are the capacity constraints between the source & transit nodes, and transit & destination nodes respectively.

$$\sum_{j=1}^{Z} x_{ikj} \leq c_{ik}\ for\ i \in \{1,\dots,X\}, k \epsilon \{1,\dots,Y\} \text{ (Equation 3)}$$

$$\sum_{i=1}^{X} x_{ikj} \leq d_{kj}\ for\ k \in \{1,\dots,Y\}, j \epsilon \{1,\dots,Z\} \text{ (Equation 4)}$$

The binary variable $u_{ijk}$ is introduced to restrict the demand volume two be split equally on two paths. $u_{ijk} = 1$ when the path $x_{ijk}$ carries any of the demand volume and $u_{ijk} = 0$ otherwise. The constraint equation to restrict the path to 2 paths is given by Equation 5 and the constraint equation to split the demand volume equally over said 2 paths is given by Equation 6.

$$\sum_{k=1}^{Y} u_{ikj} = 2\ for\ i \in \{1,\dots,X\}, j \epsilon \{1,\dots,Z\} \text{ (Equation 5)}$$

$$x_{ikj} = u_{ikj} * \left(\frac{h_{ij}}{2}\right) for\ i \in \{1,\dots,X\}, k \in \{1,\dots,Y\} j \in \{1,\dots,Z\} \text{ (Equation 6)}$$

The final constraint equation (Equation 7) describes the decision variable r as the sum of all the demand flows $x_{ijk}$ through the transit node Tk. The objective function (Equation 1) describes the problem as minimizing r, and by describing r as the sum of the demand flows through each transit node the objective function is effectively minimizing the flow, and therefore the load on each of the transit nodes.

$$\sum_{i=1}^{X}\sum_{j=1}^{Z} x_{ikj} \leq r \; for \; k \in \{1, \ldots, Y\} \text{ (Equation 7)}$$

To make this problem feasible it is necessary to create bounds on the decision variables. This leads to the non-negativity constraints described in Equation 8, Equation 9, Equation 10 and Equation 11.

$$x_{ikj} \geq 0 \text{ (Equation 8)}$$
$$c_{ik} \geq 0 \text{ (Equation 9)}$$
$$d_{kj} \geq 0 \text{ (Equation 10)}$$
$$r \geq 0 \text{ (Equation 11)}$$

It also necessary to describe the binary variable $u_{ikj}$ to be a subset of 1 and 0 ($u_{ikj} \in \{0,1\}$). This is done by creating a binary list of all the binary decision variables (Equation 12)

Bin $u_{ikj}$ for $i \in \{1,...,X\}$, $k \in \{1,...Y\}$, $j \in \{1,..,Z\}$ (Equation 12)

## CPLEX Execution

| **Y** | **Run Time (s)** | **Nonzero capacity links** | **Lowest Transit Node Utilization** | **Highest Transit Node Utilization** | **Highest Capacity Link** |
|---|---|---|---|---|---|
| **3** | 0.02 | 42 | ? | ? | $d_{17}$ (38.5) |
| **4** | 0.08 | 54 | ? | ? | $d_{37}$ (34.5) |
| **5** | 0.06 | 67 | ? | ? | $c_{64}$ (30.0) |
| **6** | 0.11 | 77 | ? | ? | $d_{37}$ (23.5) |
| **7** | 0.08 | 89 | ? | ? | $c_{65}$ (26.0) |

The run time was averaged over 5 executions. The average run time is roughly proportional to the amount of transit nodes (Y) with an exception being Y=7. This is expected, as Y increases so does the number of decision variables and hence the number of calculations made by the CPLEX algorithm.

The number of non-zero capacity links increases as Y increases. This is expected, as Y increases so does the number of links. Since the objective of the problem is to minimize the load on the links it is reasonable to expect that the new links will be taken advantage of.

We were unable to calculate the highest/lowest transit node utilization.

## Appendix

### Source Code

```
''' COSC 364 Flow Assignment
Authors:
Nicholas Ranum (53147503)
Erica Shipley (23108940)
'''

import subprocess
import sys

def demand(sources,transit,dests):
    ''' Create demand constraints for the demand between each source and dest node '''
    demandLines = []
    for i in range(sources):
        i += 1
        for j in range(dests):
            j += 1
            line = ""
            for k in range(transit):
                k += 1
                line += "x{}{}{} + ".format(i,k,j)
            line = line[0:-3]
            line += " = {}".format(i+j)
            demandLines.append(line)
    return demandLines

def capacity(sources,transit,dests):
    ''' Creade capacity constraints for the capacity of each source-transit and
    each transit-dest node '''
    capacityLines = []
    y = 1
    # Linj capacities for source -> transit links
    for i in range(sources):
        i += 1
        for k in range(transit):
            k +=1
            line = ""
            for j in range(dests):
                j += 1
                line += "x{}{}{} + ".format(i,k,j)
            line = line[0:-3]
            line += " - c{}{} = 0".format(i,k)
            capacityLines.append(line)
            y += 1
    # Link capacities for transit -> destination links
    for k in range(transit):
        k += 1
        for j in range(dests):
            j +=1
            line = ""
            for i in range(sources):
                i += 1
                line += "x{}{}{} + ".format(i,k,j)
            line = line[0:-3]
            line += " - d{}{} = 0".format(k,j)
            capacityLines.append(line)
            y += 1
    return capacityLines

def bounds(sources,transit,dests):
    ''' Generates bounds for all demand volumes and auxiliary variable r '''
    boundLines = []
    boundLines.append("\nBounds")
    for i in range(sources):
        i += 1
        for k in range(transit):
            k += 1
            boundLines.append("c{}{} >= 0".format(i,k))
    for k in range(transit):
        k += 1
        for j in range(dests):
            j += 1
            boundLines.append("d{}{} >= 0".format(k,j))

    for i in range(sources):
        i += 1
        for k in range(transit):
            k += 1
            for j in range(dests):
                j += 1
                boundLines.append("0 <= x{}{}{}".format(i,k,j))
    boundLines.append("0 <= r")
    return boundLines

def splitLimit(sources,transit,dests,splits):
    ''' constrains the binary variable to be the number of demand volume splits '''
    splitLines = []
    splitLines.append("")
    for i in range(sources):
        i += 1
        for j in range(dests):
            j += 1
            line = ""
            for k in range(transit):
                k += 1
                line += "u{}{}{} + ".format(i,k,j)
            line = line[0:-3]
            line += " = {}".format(splits)
            splitLines.append(line)
    return splitLines

def binary(sources,transit,dests):
    ''' Bounds each of the integer based decision variables to a 1 or 0 '''
    binaryLines = []
    binaryLines.append("\nBIN")
    for i in range(sources):
        i += 1
        for k in range(transit):
            k += 1
        for j in range(dests):
            j += 1
            binaryLines.append("u{}{}{}".format(i,k,j))
    return binaryLines

def halfFlow(sources,transit,dests):
    ''' Constrains a path to take half of the demand volume if it takes any at all '''
    halfLines = []
    halfLines.append("")
    for i in range(sources):
        i += 1
        for k in range(transit):
            k += 1
            for j in range(dests):
                j += 1
                halfLines.append("2 x{}{}{} - {} u{}{}{}  = 0".format(i,k,j,i+j,i,k,j))
    return halfLines

def auxiliary(sources,transit,dests):
    ''' Introduces auxiliary variable r representing the value of the objective
    function. '''
    auxLines = []
    for k in range(transit):
        k += 1
        line = ""
        for i in range(sources):
            i += 1
            for j in range(dests):
                j += 1
                line += "x{}{}{} + ".format(i,k,j)
        line = line[0:-3]
        line += " -r <= 0"
        auxLines.append(line)
    return auxLines

def main():
    # Number of sources, destinations and transit nodes
    sources = int(sys.argv[1])
    transit = int(sys.argv[2])
    dests = int(sys.argv[3])
```

```
splits = 2 # amount of paths the demand volume can take

lines = []
file_ = open("flow.lp","w")

# Adds objective function to the lp file
lines.append("Minimize\nr")

lines.append("Subject to")

# Generates and adds the demand constraints to the lp file list
lines.append("\nDemand:")
demandLines = demand(sources,transit,dests)
for line in demandLines:
    lines.append(line)

# Generates and adds the capacity constraints to the lp file list
lines.append("\nCapacity:")
capacityLines = capacity(sources,transit,dests) # Link capacity constraints
for line in capacityLines:
    lines.append(line)

# Generates and adds the binary value constraints to the lp file list
splitLines = splitLimit(sources,transit,dests,splits)
for line in splitLines:
    lines.append(line)

# Generates and adds constraint for a demand flow to take half of the
# demand volume to the lp file list
halfLines = halfFlow(sources,transit,dests)
for line in halfLines:
    lines.append(line)

# Generates and adds the auxiliary file constraints to the lp file list
auxLines = auxiliary(sources,transit,dests)
for line in auxLines:
    lines.append(line)

# Generates and adds the bounds to the lp file list
```

```
    boundsLines = bounds(sources,transit,dests)
    for line in boundsLines:
        lines.append(line)

    # Generates and adds the binary bounds to the lp file list
    binaryLines = binary(sources,transit,dests)
    for line in binaryLines:
        lines.append(line)

    lines.append("End")

    # Write each of the lp file list to the actual lp file
    for line in lines:
        file_.write(line + "\n")
    sol_file = open("flow_sol.txt","w")

    # Runs the cplex command lines in the terminal and writes the stdout to
    # flow_sol.txt.
    out = subprocess.Popen("./cplex -c \" read flow.lp\" \"optimize\" \"display solution variables -
\"",shell=True, stdout = sol_file)
    sol_file.close()
    file_.close()
main()
```

LP File for X=3, Y=2, Z=3

```
Minimize
r
Subject to

Demand:
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:

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

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 x112 - 3 u112 = 0

2 x113 - 4 u113 = 0

2 x121 - 2 u121 = 0

2 x122 - 3 u122 = 0

2 x123 - 4 u123 = 0

2 x211 - 3 u211 = 0

2 x212 - 4 u212 = 0

2 x213 - 5 u213 = 0

2 x221 - 3 u221 = 0

2 x222 - 4 u222 = 0

2 x223 - 5 u223 = 0

2 x311 - 4 u311 = 0

2 x312 - 5 u312 = 0

2 x313 - 6 u313 = 0

```
2 x321 - 4 u321  = 0
2 x322 - 5 u322  = 0
2 x323 - 6 u323  = 0
x111 + x112 + x113 + x211 + x212 + x213 + x311 + x312 + x313 -r <= 0
x121 + x122 + x123 + x221 + x222 + x223 + x321 + x322 + x323 -r <= 0

Bounds
c11 >= 0
c12 >= 0
c21 >= 0
c22 >= 0
c31 >= 0
c32 >= 0
d11 >= 0
d12 >= 0
d13 >= 0
d21 >= 0
d22 >= 0
d23 >= 0
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 <= r

BIN
```

u111
u112
u113
u121
u122
u123
u211
u212
u213
u221
u222
u223
u311
u312
u313
u321
u322
u323
End

# Plagiarism Declaration

This form needs to accompany your COSC 364 assignment submission.

I understand that plagiarism means taking someone else's work (text, program code, ideas, concepts) and presenting them as my own, without proper attribution. Taking someone else's work can include verbatim copying of text, figures/images, or program code, or it can refer to the extensive use of someone else's original ideas, algorithms or concepts.

I hereby declare that:

- My assignment is my own original work. I have not reproduced or modified code, figures/images, or writings of others without proper attribution. I have not used original ideas and concepts of others and presented them as my own.
- I have not allowed others to copy or modify my own code, figures/images, or writings. I have not allowed others to use original ideas and concepts of mine and present them as their own.
- I accept that plagiarism can lead to consequences, which can include partial or total loss of marks, no grade being awarded and other serious consequences, including notification of the University Proctor.

| | |
|---|---|
| Name: | Nicholas Ranum |
| Student ID: | 53147503 |
| Signature: | [illegible] |
| Date: | 28/05/2020 |

# Plagiarism Declaration

This form needs to accompany your COSC 364 assignment submission.

I understand that plagiarism means taking someone else's work (text, program code, ideas, concepts) and presenting them as my own, without proper attribution. Taking someone else's work can include verbatim copying of text, figures/images, or program code, or it can refer to the extensive use of someone else's original ideas, algorithms or concepts.

I hereby declare that:

- My assignment is my own original work. I have not reproduced or modified code, figures/images, or writings of others without proper attribution. I have not used original ideas and concepts of others and presented them as my own.
- I have not allowed others to copy or modify my own code, figures/images, or writings. I have not allowed others to use original ideas and concepts of mine and present them as their own.
- I accept that plagiarism can lead to consequences, which can include partial or total loss of marks, no grade being awarded and other serious consequences, including notification of the University Proctor.

| | |
|---|---|
| Name: | Erica Shipley |
| Student ID: | 23108940 |
| Signature: | [illegible] |
| Date: | 28/05/20 |