**Department of Computer Science and Software Engineering**

**University of Canterbury**

**COSC364**

**Assignment 2 – Flow Planning**

by

Kate Chamberlin (54384616)

Shan Koo (44001993)

Report submitted on 03/06/2018

# Percentage Contribution

Kate 50

Shan 50

# Problem Formulation and Explanation

The objective of the assignment was to minimise link capacities for a system that was load balanced. To do this we introduced an auxiliary variable to represent the value of our objective. The demand volume, , of all path between a source node and a destination node for all was stated as = + as shown in equation (1) and (7). There was a global requirement that each demand volume should be split equally over exactly 3 different paths, so a binary variable was used, where if the path - - is used to carry the flow then the value of else . We used this variable to determine that the sum of all flow from source to destination for all transit nodes was split into 3 different paths as in equation (2), while equation (3) ensured that the splits were done equally. It also indicated the minimum capacity needed for the link. Next, equation (4) and equation (5) defined the constraint that the sum of all flows using the path - for all destination was less than or equal to the link capacity, and the sum of all flows using the path - for all destinations was less than or equal to the link capacity, respectively. After ensuring that the load was balanced, we found the minimum capacities by finding the sum of capacities going through node for all source nodes as in equation (6). Equation (8) defines as a binary variable while Equation (9) – (12) describes that the decisions variables are of non-negative values.

|  |  |  |  |
| --- | --- | --- | --- |
| **Minimize [x, r]** |  |  |  |
| **Subject to** |  |  | ……….. (1) |
|  |  |  | ……….. (2) |
|  |  |  | ……….. (3) |
|  |  |  | ……….. (4) |
|  |  |  | ……….. (5) |
|  |  |  | ……….. (6) |
|  |  |  | ……….. (7) |
|  |  |  | ……….. (8) |
|  |  |  | ……….. (9) |
|  |  |  | ….….. (10) |
|  |  |  | ….….. (11) |
|  |  |  | .…….. (12) |

**CPLEX execution time, numbers of non-zero capacity links and the highest capacity link for varying Y**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| # of Transit Nodes | 3 | 4 | 5 | 6 | 7 |
| r | 130.666667 | 98 | 78.666667 | 65.333333 | 56 |
| Load 1  Load 2  Load 3  Load 4  Load 5  Load 6  Load 7 | 130.666667  130.666667  130.666667  -  -  -  - | 98  98  98  98  -  -  - | 78  78.666667  78.666667  78.666667  78  -  - | 65.333333  65.333333  65.333333  65.333333  65.333333  65.333333  - | 56  56  56  56  56  56  56 |
| CPLEX execution time | 0.0285 | 0.0529 | 0.0634 | 0.1202 | 0.1634 |
| # of non-zero capacity links | 42 | 56 | 69 | 82 | 94 |
| Highest capacity link | 25.666667 | 23.333333 | 21.666667 | 23 | 19 |

# Appendix I – Source Code for lp\_gen.py

#==============================================================================#

# lp\_gen.py

# Generates the lp file for CPLEX.

# Assignment 2 - Flow planning

# Shan Koo (ysk28) and Kate Chamberlin (kch114)

# Due date 25th May 2018

#==============================================================================#

**import** sys

PATHS **=** 3 #the number of paths the load must be balanced between

STD **=** "S{}T{}D{}" #Standard format for most nodes - Source #, Transit #, Dest #.

#==============================================================================#

# Format writers

#==============================================================================#

**def** writeAll**(**file**,** source**,** transit**,** dest**):**

""" Writes entire LP file"""

writeHeader**(**file**)**

writeConstraints**(**file**,** source**,** transit**,** dest**)**

writeTrailer**(**file**,** source**,** transit**,** dest**)**

**return**

**def** writeHeader**(**file**):**

""" Writes the header out to the given file"""

file**.**write**(**"Minimize\n"**)**

file**.**write**(**" r\n"**)**

file**.**write**(**"Subject to\n"**)**

**return**

#==============================================================================#

# Constraint writers

#==============================================================================#

**def** writeConstraints**(**file**,** source**,** transit**,** dest**):**

""" Writes all constraints to the given file"""

writeMinimiseObjectiveFormula**(**file**,** source**,** transit**)**

writeLoadBalancingConstraints**(**file**,** source**,** transit**,** dest**)**

writeDemandVolConstraints**(**file**,** source**,** transit**,** dest**)**

writeDemandFlowConstraints**(**file**,** source**,** transit**,** dest**)**

writeSourceCapacityConstraints**(**file**,** source**,** transit**,** dest**)**

writeDestCapacityConstraints**(**file**,** source**,** transit**,** dest**)**

writeBinaryConstraints**(**file**,** source**,** transit**,** dest**)**

**def** writeMinimiseObjectiveFormula**(**file**,** source**,** transit**):**

""" Writes the minimisation of r objective constraints to the given file.

Sum of all capacities through a transit node - r <= 0

This auxilliary variable r is what we are minimising. """

line **=** ""

**for** k **in** range**(**1**,** transit **+** 1**):**

line **+=** " r{}: "**.**format**(**k**)**

**for** i **in** range**(**1**,** source **+** 1**):**

line **+=** **(**"yS{}T{}"**.**format**(**i**,** k**))**

**if** **(**i **==** source**):** line **+=** **(**" - r <= 0\n"**)**

**else:** line **+=** **(**" + "**)**

file**.**write**(**line**)**

**return**

**def** writeLoadBalancingConstraints**(**file**,** source**,** transit**,** dest**):**

""" Writes the load balancing constraints to the given file.

Sum of all path flows through a transit node - load = 0. """

line **=** ""

**for** k **in** range**(**1**,** transit **+** 1**):**

line **+=** " load{}: "**.**format**(**k**)**

**for** i **in** range**(**1**,** source **+** 1**):**

**for** j **in** range**(**1**,** dest **+** 1**):**

line **+=** **(**"x" **+** STD**.**format**(**i**,** k**,** j**))**

**if** **(**i **==** source **and** j **==** dest**):** line **+=** **(**" - lT{} = 0\n"**.**format**(**k**))**

**else:** line **+=** **(**" + "**)**

file**.**write**(**line**)**

**return**

**def** writeDemandVolConstraints**(**file**,** source**,** transit**,** dest**):**

""" Writes the demand volume constraints out to the given file

Here the demand volume is equal to i + j. """

line **=** ""

**for** i **in** range**(**1**,** source **+** 1**):**

**for** j **in** range**(**1**,** dest **+** 1**):**

line **+=** **(**" hS{}D{}: "**.**format**(**i**,** j**))**

**for** k **in** range**(**1**,** transit **+** 1**):**

line **+=** **(**"x" **+** STD**.**format**(**i**,** k**,** j**))**

**if** **(**k **!=** transit**):** line **+=** **(**" + "**)**

**else:** line **+=** **(**" = {}\n"**.**format**(**i **+** j**))**

file**.**write**(**line**)**

**return**

**def** writeDemandFlowConstraints**(**file**,** source**,** transit**,** dest**):**

""" Writes the demand flow constraints out to the given file.

This is for load balancing equation x=(u\*h)/n"""

line **=** ""

**for** i **in** range**(**1**,** source **+** 1**):**

**for** j **in** range**(**1**,** dest **+** 1**):**

**for** k **in** range**(**1**,** transit **+** 1**):**

line **+=** **(**" df{}: {} x{} - {} u{} = 0\n"

**.**format**(**STD**.**format**(**i**,** k**,** j**),**

PATHS**,**

STD**.**format**(**i**,** k**,** j**),**

i **+** j**,**

STD**.**format**(**i**,** k**,** j**)))**

file**.**write**(**line**)**

**return**

**def** writeSourceCapacityConstraints**(**file**,** source**,** transit**,** dest**):**

""" Writes the capacity constraints for the source to transit link.

Writes to the given file"""

line **=** ""

**for** i **in** range**(**1**,** source **+** 1**):**

**for** k **in** range**(**1**,** transit **+** 1**):**

line **+=** " cS{}T{}: "**.**format**(**i**,**k**)**

**for** j **in** range**(**1**,** dest **+** 1**):**

line **+=** **(**"x" **+** STD**.**format**(**i**,** k**,** j**))**

**if** **(**j **!=** dest**):** line **+=** **(**" + "**)**

**else:** line **+=** **(**" - yS{}T{} = 0\n"**.**format**(**i**,** k**))**

file**.**write**(**line**)**

**return**

**def** writeDestCapacityConstraints**(**file**,** source**,** transit**,** dest**):**

""" Writes the capacity constraints for the transit to destination link.

Writes to the given file"""

line **=** ""

**for** j **in** range**(**1**,** dest **+** 1**):**

**for** k **in** range**(**1**,** transit **+** 1**):**

line **+=** " dT{}D{}: "**.**format**(**k**,** j**)**

**for** i **in** range**(**1**,** source **+** 1**):**

line **+=** **(**"x" **+** STD**.**format**(**i**,** k**,** j**))**

**if** **(**i **!=** source**):** line **+=** **(**" + "**)**

**else:** line **+=** **(**" - yT{}D{} = 0\n"**.**format**(**k**,** j**))**

file**.**write**(**line**)**

**return**

**def** writeBinaryConstraints**(**file**,** source**,** transit**,** dest**):**

""" Writes the binary constraints to the given file.

Sum of binaries should equal the PATHS constant."""

line **=** ""

**for** i **in** range**(**1**,** source **+** 1**):**

**for** j **in** range**(**1**,** dest **+** 1**):**

line **+=** " uS{}D{}: "**.**format**(**i**,** j**)**

**for** k **in** range**(**1**,** transit **+** 1**):**

line **+=** **(**"u" **+** STD**.**format**(**i**,** k**,** j**))**

**if** **(**k **!=** transit**):** line **+=** **(**" + "**)**

**else:** line **+=** **(**" = {}\n"**.**format**(**PATHS**))**

file**.**write**(**line**)**

**return**

#==============================================================================#

# Other writers

#==============================================================================#

**def** writeTrailer**(**file**,** source**,** transit**,** dest**):**

""" Writes all other LP information to file"""

file**.**write**(**"Bounds\n"**)**

writeFlowBounds**(**file**,** source**,** transit**,** dest**)**

writeSourceBounds**(**file**,** source**,** transit**)**

writeDestBounds**(**file**,** transit**,** dest**)**

file**.**write**(**" r >= 0\n"**)**

file**.**write**(**"Binaries\n"**)**

writeBinaries**(**file**,** source**,** transit**,** dest**)**

file**.**write**(**"End"**)**

**return**

**def** writeFlowBounds**(**file**,** source**,** transit**,** dest**):**

""" Writes the flow bounds to given file"""

line **=** ""

**for** i **in** range**(**1**,** source **+** 1**):**

**for** j **in** range**(**1**,** dest **+** 1**):**

**for** k **in** range**(**1**,** transit **+** 1**):**

line **+=** **(**" x" **+** STD**.**format**(**i**,** k**,** j**)** **+** " >= 0\n"**)**

file**.**write**(**line**)**

**return**

**def** writeSourceBounds**(**file**,** source**,** transit**):**

""" Writes the source -> transit capacity bounds to given file"""

line **=** ""

**for** i **in** range**(**1**,** source **+** 1**):**

**for** k **in** range**(**1**,** transit **+** 1**):**

line **+=** **(**" yS{}T{} >= 0\n"**.**format**(**i**,** k**))**

file**.**write**(**line**)**

**return**

**def** writeDestBounds**(**file**,** transit**,** dest**):**

""" Writes the transit -> dest capacity bounds to given file"""

line **=** ""

**for** j **in** range**(**1**,** dest **+** 1**):**

**for** k **in** range**(**1**,** transit **+** 1**):**

line **+=** **(**" yT{}D{} >= 0\n"**.**format**(**k**,** j**))**

file**.**write**(**line**)**

**return**

**def** writeBinaries**(**file**,** source**,** transit**,** dest**):**

""" Writes all the binary variables to the given file"""

line **=** ""

**for** i **in** range**(**1**,** source **+** 1**):**

**for** j **in** range**(**1**,** dest **+** 1**):**

**for** k **in** range**(**1**,** transit **+** 1**):**

line **+=** " u" **+** STD**.**format**(**i**,** k**,** j**)** **+** "\n"

file**.**write**(**line**)**

**return**

#==============================================================================#

# Test

#==============================================================================#

**if** **(**\_\_name\_\_ **==** "\_\_main\_\_"**):**

#generic lp generation

**for** y **in** range**(**3**,** 8**):**

f **=** open**(**"out%s.lp" **%**y**,** 'w'**)**

source**,** transit**,** dest **=** 7**,** y**,** 7

writeAll**(**f**,** source**,** transit**,** dest**)**

f**.**close**()**

# Appendix II – Source Code for lp\_cplex.py

**import** subprocess

**import** time

#==============================================================================#

# Global Variables

#==============================================================================#

# cplex path (change if your cplex is in a different path)

CPLEX **=** "/home/cosc/student/ysk28/cplex/cplex/bin/x86-64\_linux/cplex"

# lp file path (change if your files are in a different file)

FILE **=** "/home/cosc/student/ysk28/Desktop/Flow-Planning-master/"

#==============================================================================#

# Main function - loops through all the lp file generated by lp\_gen.py and pass

# it through to cplex and outputs it into a text file

#==============================================================================#

**for** i **in** range **(**3**,** 8**):**

filename **=** "out%s.lp" **%**i

# cplex command line

cmd **=** CPLEX**,** "-c"**,** 'read ' **+** FILE **+** filename**,** "optimize"**,** "display solution variables -"

# calls a subprocess to run cplex

process **=** subprocess**.**Popen**(**cmd**,** stdout **=** open**(**"out%s.txt" **%**i**,** "wb"**));**

start\_time **=** time**.**time**()**

# wait for process to finish

process**.**wait**()**

process\_time **=** time**.**time**()** **-** start\_time

**print(**"Process time for %s transit node is: " **%**i**,** process\_time**)**

# Appendix III – LP file generated for (X = 3, Y = 2, Z = 4)

Minimize

r

Subject to

r1: yS1T1 + yS2T1 + yS3T1 - r <= 0

r2: yS1T2 + yS2T2 + yS3T2 - r <= 0

load1: xS1T1D1 + xS1T1D2 + xS1T1D3 + xS1T1D4 + xS2T1D1 + xS2T1D2 + xS2T1D3 + xS2T1D4 + xS3T1D1 + xS3T1D2 + xS3T1D3 + xS3T1D4 - lT1 = 0

load2: xS1T2D1 + xS1T2D2 + xS1T2D3 + xS1T2D4 + xS2T2D1 + xS2T2D2 + xS2T2D3 + xS2T2D4 + xS3T2D1 + xS3T2D2 + xS3T2D3 + xS3T2D4 - lT2 = 0

hS1D1: xS1T1D1 + xS1T2D1 = 2

hS1D2: xS1T1D2 + xS1T2D2 = 3

hS1D3: xS1T1D3 + xS1T2D3 = 4

hS1D4: xS1T1D4 + xS1T2D4 = 5

hS2D1: xS2T1D1 + xS2T2D1 = 3

hS2D2: xS2T1D2 + xS2T2D2 = 4

hS2D3: xS2T1D3 + xS2T2D3 = 5

hS2D4: xS2T1D4 + xS2T2D4 = 6

hS3D1: xS3T1D1 + xS3T2D1 = 4

hS3D2: xS3T1D2 + xS3T2D2 = 5

hS3D3: xS3T1D3 + xS3T2D3 = 6

hS3D4: xS3T1D4 + xS3T2D4 = 7

dfS1T1D1: 3 xS1T1D1 - 2 uS1T1D1 = 0

dfS1T2D1: 3 xS1T2D1 - 2 uS1T2D1 = 0

dfS1T1D2: 3 xS1T1D2 - 3 uS1T1D2 = 0

dfS1T2D2: 3 xS1T2D2 - 3 uS1T2D2 = 0

dfS1T1D3: 3 xS1T1D3 - 4 uS1T1D3 = 0

dfS1T2D3: 3 xS1T2D3 - 4 uS1T2D3 = 0

dfS1T1D4: 3 xS1T1D4 - 5 uS1T1D4 = 0

dfS1T2D4: 3 xS1T2D4 - 5 uS1T2D4 = 0

dfS2T1D1: 3 xS2T1D1 - 3 uS2T1D1 = 0

dfS2T2D1: 3 xS2T2D1 - 3 uS2T2D1 = 0

dfS2T1D2: 3 xS2T1D2 - 4 uS2T1D2 = 0

dfS2T2D2: 3 xS2T2D2 - 4 uS2T2D2 = 0

dfS2T1D3: 3 xS2T1D3 - 5 uS2T1D3 = 0

dfS2T2D3: 3 xS2T2D3 - 5 uS2T2D3 = 0

dfS2T1D4: 3 xS2T1D4 - 6 uS2T1D4 = 0

dfS2T2D4: 3 xS2T2D4 - 6 uS2T2D4 = 0

dfS3T1D1: 3 xS3T1D1 - 4 uS3T1D1 = 0

dfS3T2D1: 3 xS3T2D1 - 4 uS3T2D1 = 0

dfS3T1D2: 3 xS3T1D2 - 5 uS3T1D2 = 0

dfS3T2D2: 3 xS3T2D2 - 5 uS3T2D2 = 0

dfS3T1D3: 3 xS3T1D3 - 6 uS3T1D3 = 0

dfS3T2D3: 3 xS3T2D3 - 6 uS3T2D3 = 0

dfS3T1D4: 3 xS3T1D4 - 7 uS3T1D4 = 0

dfS3T2D4: 3 xS3T2D4 - 7 uS3T2D4 = 0

cS1T1: xS1T1D1 + xS1T1D2 + xS1T1D3 + xS1T1D4 - yS1T1 = 0

cS1T2: xS1T2D1 + xS1T2D2 + xS1T2D3 + xS1T2D4 - yS1T2 = 0

cS2T1: xS2T1D1 + xS2T1D2 + xS2T1D3 + xS2T1D4 - yS2T1 = 0

cS2T2: xS2T2D1 + xS2T2D2 + xS2T2D3 + xS2T2D4 - yS2T2 = 0

cS3T1: xS3T1D1 + xS3T1D2 + xS3T1D3 + xS3T1D4 - yS3T1 = 0

cS3T2: xS3T2D1 + xS3T2D2 + xS3T2D3 + xS3T2D4 - yS3T2 = 0

dT1D1: xS1T1D1 + xS2T1D1 + xS3T1D1 - yT1D1 = 0

dT2D1: xS1T2D1 + xS2T2D1 + xS3T2D1 - yT2D1 = 0

dT1D2: xS1T1D2 + xS2T1D2 + xS3T1D2 - yT1D2 = 0

dT2D2: xS1T2D2 + xS2T2D2 + xS3T2D2 - yT2D2 = 0

dT1D3: xS1T1D3 + xS2T1D3 + xS3T1D3 - yT1D3 = 0

dT2D3: xS1T2D3 + xS2T2D3 + xS3T2D3 - yT2D3 = 0

dT1D4: xS1T1D4 + xS2T1D4 + xS3T1D4 - yT1D4 = 0

dT2D4: xS1T2D4 + xS2T2D4 + xS3T2D4 - yT2D4 = 0

uS1D1: uS1T1D1 + uS1T2D1 = 3

uS1D2: uS1T1D2 + uS1T2D2 = 3

uS1D3: uS1T1D3 + uS1T2D3 = 3

uS1D4: uS1T1D4 + uS1T2D4 = 3

uS2D1: uS2T1D1 + uS2T2D1 = 3

uS2D2: uS2T1D2 + uS2T2D2 = 3

uS2D3: uS2T1D3 + uS2T2D3 = 3

uS2D4: uS2T1D4 + uS2T2D4 = 3

uS3D1: uS3T1D1 + uS3T2D1 = 3

uS3D2: uS3T1D2 + uS3T2D2 = 3

uS3D3: uS3T1D3 + uS3T2D3 = 3

uS3D4: uS3T1D4 + uS3T2D4 = 3

Bounds

xS1T1D1 >= 0

xS1T2D1 >= 0

xS1T1D2 >= 0

xS1T2D2 >= 0

xS1T1D3 >= 0

xS1T2D3 >= 0

xS1T1D4 >= 0

xS1T2D4 >= 0

xS2T1D1 >= 0

xS2T2D1 >= 0

xS2T1D2 >= 0

xS2T2D2 >= 0

xS2T1D3 >= 0

xS2T2D3 >= 0

xS2T1D4 >= 0

xS2T2D4 >= 0

xS3T1D1 >= 0

xS3T2D1 >= 0

xS3T1D2 >= 0

xS3T2D2 >= 0

xS3T1D3 >= 0

xS3T2D3 >= 0

xS3T1D4 >= 0

xS3T2D4 >= 0

yS1T1 >= 0

yS1T2 >= 0

yS2T1 >= 0

yS2T2 >= 0

yS3T1 >= 0

yS3T2 >= 0

yT1D1 >= 0

yT2D1 >= 0

yT1D2 >= 0

yT2D2 >= 0

yT1D3 >= 0

yT2D3 >= 0

yT1D4 >= 0

yT2D4 >= 0

r >= 0

Binaries

uS1T1D1

uS1T2D1

uS1T1D2

uS1T2D2

uS1T1D3

uS1T2D3

uS1T1D4

uS1T2D4

uS2T1D1

uS2T2D1

uS2T1D2

uS2T2D2

uS2T1D3

uS2T2D3

uS2T1D4

uS2T2D4

uS3T1D1

uS3T2D1

uS3T1D2

uS3T2D2

uS3T1D3

uS3T2D3

uS3T1D4

uS3T2D4

End