COSC 364 Assignment2 Report

YUAN CUI ycu20 63483319
Yihong Liu yli227 49118489

1. In the project, YUAN did 60%, Yihong Liu 40%.
2. Formulate and explanation:

- Given:

(1) Demand volume **hij** between source **i** and source **j** (**hij = i + j**).

(2) Each demand volume shall be split over exactly three different paths. (**nk = 3**)

(3) For a link between source node **Si** and transit node **Tk** we denote its capacity by **cik**. For a link between transit node **Tk** and destination node **Dj** we denote its capacity by **dkj** .

- Goal: load balancing on transit nodes and get the minimum balancing load, **load** will be introduced as an auxiliary variable.

- Decision variables:

(1) **xikj** = amount of flows to go through transit node k for the demand volume between i and j.

(2) **uikj** = a binary variable, if kth-path of demand volume hij is used to carry the data, uikj = 1. Otherwise, uikj = 0.

(3) **load**

Formula:

// balance the load on all transit nodes, get the minimum value of load

Minimize[x, u, load] load

// establish demand constaints

Subject to $\sum_{k=1}^{Y} xikj = hij = i + j$ for i ∈ {1,2…X}, j ∈ {1,2…Z}

// establish capacity constraints for source-transit links

$\sum_{j=1}^{Z} xikj <=$ cik for i ∈ {1,2…X}, k ∈ {1,2…Y}

// establish capacity constraints for transit-destination links

$\sum_{i=1}^{X} xikj <=$ dkj for j ∈ {1,2…Z}, k ∈ {1,2…Y}

// establish constraints to express load balancing

$\sum_{i=1}^{X}.\sum_{j=1}^{Z} xikj <=$ load for k ∈ {1,2…Y}

// each volumn owns exactly3 paths, so nk = 3

$\sum_{k=1}^{Y} uikj <=$ nk = 3 for i ∈ {1,2…X}, j ∈ {1,2…Z}

// every splitted x value is the same, because of equally splitting

xikj = uikj * hij / nk for i ∈ {1,2…X}, j ∈ {1,2…Z}, k ∈ {1,2…Y}

// non-negative constraints

xikj >= 0 for i ∈ {1,2…X}, j ∈ {1,2…Z}, k ∈ {1,2…Y}

load >= 0 for i ∈ {1,2…X}, j ∈ {1,2…Z}, k ∈ {1,2…Y}

// binary constraints

uikj ∈ {0, 1} for i ∈ {1,2…X}, j ∈ {1,2…Z}, k ∈ {1,2…Y}

3. Result

```
[ycu20@cs17077jp ~/cosc364/assignment2]$ python3 assignment2.py 7 3 7
Cplex running time: 0.009s
The maximum load accross the transit node is 130.667
The number of links with non-zero capacities is 42
The capacity of the link with highest capacity is 26.0.
[ycu20@cs17077jp ~/cosc364/assignment2]$ python3 assignment2.py 7 4 7
Cplex running time: 0.033s
The maximum load accross the transit node is 98.000
The number of links with non-zero capacities is 56
The capacity of the link with highest capacity is 24.0.
[ycu20@cs17077jp ~/cosc364/assignment2]$ python3 assignment2.py 7 5 7
Cplex running time: 0.039s
The maximum load accross the transit node is 78.667
The number of links with non-zero capacities is 70
The capacity of the link with highest capacity is 23.0.
[ycu20@cs17077jp ~/cosc364/assignment2]$ python3 assignment2.py 7 6 7
Cplex running time: 0.058s
The maximum load accross the transit node is 65.333
The number of links with non-zero capacities is 83
The capacity of the link with highest capacity is 19.0.
[ycu20@cs17077jp ~/cosc364/assignment2]$ python3 assignment2.py 7 7 7
Cplex running time: 0.058s
The maximum load accross the transit node is 56.000
The number of links with non-zero capacities is 96
The capacity of the link with highest capacity is 19.0.
[ycu20@cs17077jp ~/cosc364/assignment2]$
```

	number of transit nodes				
	3	4	5	6	7
running time	0.009	0.033	0.039	0.058	0.058
maximum load through transit nodes	130.667	98.000	78.667	65.333	56.000
number of links with non-zero capacity	42	56	70	83	96
maximum link capacity	26	24	23	19	19

Analysis:

- With the increase of transit nodes' number, there will be more possibilities to choose the optimal route, which will cost more time.
- With the increase of transit nodes' number, every node will balance certain loads, so the maximum load will appear a decline trend.
- Because the number of transit nodes is increasing, more links will be generated. Obviously, both non-zero capacity and zero capacity links will increase.
- Because more links to afford the total load, the maximum link capacity will appear a decline trend. Meanwhile I consider that the last two same numbers 19 in the row of maximum link capacity are the bottleneck of link capacity. In other the word, the value won't change with the increasing transit node's number when transit nodes' number is larger than or equal to 6.
- *If the transit node reaches a big number, for example, in this case when it reach 9 or larger, there will not be any integer feasible solution existing as the figure shows below.*

```
[ycu20@cs18208kq ~/cosc364/assignment2]$ python3 assignment2.py 7 8 7
Cplex running time: 0.117s
The maximum load accross the transit node is 49.000
The number of links with non-zero capacities is 105
The capacity of the link with highest capacity is 19.0.
[ycu20@cs18208kq ~/cosc364/assignment2]$ python3 assignment2.py 7 9 7
No integer feasible solution exists.
[ycu20@cs18208kq ~/cosc364/assignment2]$
```

Appendix

4. Source code:

```
from time import *
import subprocess
import matplotlib.pyplot as plt
import numpy as np
import sys
import shlex

SPLIT_NO = 3 #set split number to 3

def make_lp_file(X, Y, Z):
    """create lp file"""
    file_name = "assignment2.lp"
    with open(file_name, "w") as f:
        my_string = "Minimize\n\tload\nSubject to\n"
        for i in range(1, X+1):
            for j in range(1 ,Z+1):
                for k in range(1, Y+1):
                    if k == 1:
                        my_string += "\tx{}{}{} ".format(i, k, j)
                    else:
                        my_string += "+ x{}{}{} ".format(i, k, j)
                my_string += "= " + str(i+j) + "\n"
        for i in range(1, X+1):
            for j in range(1 ,Z+1):
                for k in range(1, Y+1):
                    if k == 1:
                        my_string += "\tu{}{}{} ".format(i, k, j)
                    else:
                        my_string += "+ u{}{}{} ".format(i, k, j)
                my_string += "= {}\n".format(SPLIT_NO)
        for i in range(1, X+1):
            for j in range(1 ,Z+1):
                for k in range(1, Y+1):
                    my_string += "\t3 x{}{}{} - {} u{}{}{} = 0\n".format(i, k, j, i+j, i, k, j)
        for i in range(1, X+1):
            for k in range(1 ,Y+1):
                for j in range(1, Z+1):
                    if j == 1:
                        my_string += "\tx{}{}{} ".format(i, k, j)
                    else:
                        my_string += "+ x{}{}{} ".format(i, k, j)
```

```
            my_string += "- c{}{} <= 0\n".format(i, k)
        for k in range(1, Y+1):
            for j in range(1 ,Z+1):
                for i in range(1, X+1):
                    if i == 1:
                        my_string += "\tx{}{}{} ".format(i, k, j)
                    else:
                        my_string += "+ x{}{}{} ".format(i, k, j)
                my_string += "- d{}{} <= 0\n".format(k, j)
        for k in range(1, Y+1):
            for j in range(1, Z+1):
                for i in range(1, X+1):
                    if i == 1 and j ==1:
                        my_string += "\tx{}{}{} ".format(i, k, j)
                    else:
                        my_string += "+ x{}{}{} ".format(i, k, j)
            my_string += "- load <= 0\n"
        my_string += "Bounds\n"
        for i in range(1, X+1):
            for k in range(1 ,Y+1):
                for j in range(1, Z+1):
                    my_string += "\tx{}{}{} >= 0\n".format(i, k, j)
        my_string += "\tload >= 0\nBINARY\n"
        for i in range(1, X+1):
            for k in range(1 ,Y+1):
                for j in range(1, Z+1):
                    my_string += "\tu{}{}{}\n".format(i, k ,j)
        my_string += "END\n"
        f.write(my_string)
    return file_name

def process(file, Y):
    """read lp file, calculate optimized result, print the result on console"""
    cplex = "/home/cosc/student/ycu20/cosc364/cplex/bin/x86-64_linux/cplex"
    command = "time " + cplex +" -c read " + file + " optimize display solution variables -"
    args = shlex.split(command)
    proc = subprocess.Popen(command, stdout = subprocess.PIPE, stderr=subprocess.PIPE,
executable='/bin/bash', shell=True)
    stdout, stderr = proc.communicate()
    time_str = str(stderr)
    msg_str = str(stdout)
    real_index = time_str.index('real')
    cplex_time = time_str[real_index+8 : real_index+14]
    # if no feasible solution, print an error message
    try:
        start_index = msg_str.index('Solution Value') + 16
```

```
    except ValueError:
        return print('No integer feasible solution exists.')
    # if there is no zero variables, a value error will be caught
    # the result message is caught by a different way from result with zero variables
    try:
        end_index = msg_str.index('All other variables in the range') - 2
    except ValueError:
        end_index = -10
    msgs = msg_str[start_index:end_index].split("\\n")
    x_msg = {}
    max_c = 0.0
    capacity_count = 0
    max_load = 0.0
    for msg in msgs:
        results = msg.split("                    ")
        if results[0].startswith('x'):
            x_msg[results[0]] = results[1]
        elif (results[0].startswith('c') or results[0].startswith('d')):
            capacity_count += 1
            if float(results[1]) > max_c:
                max_c = float(results[1])
        elif (results[0].startswith('load')):
            max_load = results[0].strip().replace('load', '')
    # print result
    print("Cplex running time: {}".format(cplex_time))
    print('The maximum load accross the transit node is {:.3f}'.format(float(max_load)))
    print('The number of links with non-zero capacities is {}'.format(capacity_count))
    print('The capacity of the link with highest capacity is {}.'.format(max_c))


def main():
    # catch all invalid input
    try:
        X = int(sys.argv[1])
        Y = int(sys.argv[2])
        Z = int(sys.argv[3])
    except ValueError:
        print("Invalid argument(s)")
        return
    # automatically generate lp file
    file_name = make_lp_file(X, Y ,Z)
    # because we need to generate a lp file with 2 transit nodes.
    # so that this check is moved down to make_lp_file function.
    if Y <= SPLIT_NO -1:
        return print("Invalid arguments on number of transit nodes.\n The number of transit nodes
must be larger than split number")
```

```
    # process lp file by cplex through pipline and display result
    process(file_name, Y)

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

5. LP file with (3, 2, 4)

```
Minimize
        load
Subject to
        x111 + x121 = 2
        x112 + x122 = 3
        x113 + x123 = 4
        x114 + x124 = 5
        x211 + x221 = 3
        x212 + x222 = 4
        x213 + x223 = 5
        x214 + x224 = 6
        x311 + x321 = 4
        x312 + x322 = 5
        x313 + x323 = 6
        x314 + x324 = 7
        u111 + u121 = 3
        u112 + u122 = 3
        u113 + u123 = 3
        u114 + u124 = 3
        u211 + u221 = 3
        u212 + u222 = 3
        u213 + u223 = 3
        u214 + u224 = 3
        u311 + u321 = 3
        u312 + u322 = 3
        u313 + u323 = 3
        u314 + u324 = 3
        3 x111 - 2 u111 = 0
        3 x121 - 2 u121 = 0
        3 x112 - 3 u112 = 0
        3 x122 - 3 u122 = 0
        3 x113 - 4 u113 = 0
        3 x123 - 4 u123 = 0
        3 x114 - 5 u114 = 0
        3 x124 - 5 u124 = 0
        3 x211 - 3 u211 = 0
        3 x221 - 3 u221 = 0
```

3 x212 - 4 u212 = 0
3 x222 - 4 u222 = 0
3 x213 - 5 u213 = 0
3 x223 - 5 u223 = 0
3 x214 - 6 u214 = 0
3 x224 - 6 u224 = 0
3 x311 - 4 u311 = 0
3 x321 - 4 u321 = 0
3 x312 - 5 u312 = 0
3 x322 - 5 u322 = 0
3 x313 - 6 u313 = 0
3 x323 - 6 u323 = 0
3 x314 - 7 u314 = 0
3 x324 - 7 u324 = 0
x111 + x112 + x113 + x114 - c11 <= 0
x121 + x122 + x123 + x124 - c12 <= 0
x211 + x212 + x213 + x214 - c21 <= 0
x221 + x222 + x223 + x224 - c22 <= 0
x311 + x312 + x313 + x314 - c31 <= 0
x321 + x322 + x323 + x324 - c32 <= 0
x111 + x211 + x311 - d11 <= 0
x112 + x212 + x312 - d12 <= 0
x113 + x213 + x313 - d13 <= 0
x114 + x214 + x314 - d14 <= 0
x121 + x221 + x321 - d21 <= 0
x122 + x222 + x322 - d22 <= 0
x123 + x223 + x323 - d23 <= 0
x124 + x224 + x324 - d24 <= 0
x111 + x211 + x311 + x112 + x212 + x312 + x113 + x213 + x313 + x114 + x214 + x314 - load <= 0
x121 + x221 + x321 + x122 + x222 + x322 + x123 + x223 + x323 + x124 + x224 + x324 - load <= 0
Bounds
x111 >= 0
x112 >= 0
x113 >= 0
x114 >= 0
x121 >= 0
x122 >= 0
x123 >= 0
x124 >= 0
x211 >= 0
x212 >= 0
x213 >= 0
x214 >= 0
x221 >= 0

x222 >= 0
x223 >= 0
x224 >= 0
x311 >= 0
x312 >= 0
x313 >= 0
x314 >= 0
x321 >= 0
x322 >= 0
x323 >= 0
x324 >= 0
load >= 0
BINARY
u111
u112
u113
u114
u121
u122
u123
u124
u211
u212
u213
u214
u221
u222
u223
u224
u311
u312
u313
u314
u321
u322
u323
u324
END