COSC364
Internet Technologies and Engineering
Assignment2 Report

May 25, 2018

Xiao Meng 88211337(50%)
Xiaoshi Xie 22378644(50%)

Formulations

F1 $\text{Minimize}_{[x, c, d, r]} \quad r$

Subject to

F2 demand volume: $\sum_{k=1}^{Y} x_{ikj} = h_{ij}$, for $i \in \{1, \dots, X\}$, $j \in \{1, \dots, Z\}$

F3 capacity 1 (Source → transit): $\sum_{j=1}^{Z} x_{ikj} \leq C_{ik}$, for $i \in \{1, \dots, X\}$, $k \in \{1, \dots, Y\}$

F4 capacity 2 (transit → destination): $\sum_{i=1}^{X} x_{ikj} \leq d_{kj}$, for $k \in \{1, \dots, Y\}$, $j \in \{1, \dots, Z\}$

F5 Transit load: $\sum_{i=1}^{X} \sum_{j=1}^{Z} x_{ikj} \leq r$, for $k \in \{1, \dots, Y\}$

F6 Binary variable: $\sum_{k=1}^{Y} u_{ikj} = n_{ij}$, for $i \in \{1, \dots, X\}$, $j \in \{1, \dots, Z\}$

F7 path flow: $x_{ikj} = \dfrac{u_{ikj} \cdot h_{ij}}{n_{ij}}$, for $i \in \{1, \dots, X\}$, $k \in \{1, \dots, Y\}$, $j \in \{1, \dots, Z\}$,

F8 Bounds: $x_{ikj} \geq 0$, for $i \in \{1, \dots, X\}$, $k \in \{1, \dots, Y\}$, $j \in \{1, \dots, Z\}$

F9 Binaries: $u_{ikj} \in \{0, 1\}$ for $i \in \{1, \dots, X\}$, $k \in \{1, \dots, Y\}$, $j \in \{1, \dots, Z\}$

Formulation and explanation

variables we used:

i: index of source node

k: index of transit node

j: index of destination node

xikj: flow on path from source i using transit k to destination j
hij: demand volume from source i to destination j
cik: capacity on link from source i to transit k
dkj: capacity on link from transit k to destination j
uikj: binary variable on path from source i using transit k to destination j
nij: number of split paths

F1: Minimize objective function
We introduce an auxiliary r for the utilization of the load on each transit node, so we can make the load on transit nodes to balance by adjusting the value of r. Then introduce the new constraint for r which r should not be smaller than the total value of load on each transit node xikj. In this way, when r gets the minimized value, it means demand volume which allocates on each transit node is minimized, the utilization of transit nodes is optimized.

F2: Demand volume constraint
Each path flow xikj should add up to hij which is the given demand volume between node i and j.

F3 – F4: Capacity constraints
Each path flow xikj which involves the link ik or kj should not be greater than the capacity of this link. In this problem for the optimized situation, the capacity should be equal to the demand volume to make the cost of capacity of each link lowest.

F5: Load on each transit node constraint
The sum of flow on each transit node xikj should not be greater than the introduced variable r which means r is a upper bound for load on transit node. The eligible flow on each transit node is smaller, meaning the utilization of transit nodes is more balanced.

As the load on transit node influences path flow, we think it is more reasonable to put the constraints F3 – F5 before F6 and F7 which is a little different from the fomulations on booklet.

F6: Binary variable constraint
The binaray variable uikj is equal to 1 meaning the demand volume split into the path i – k – j is used and 0 means not into this path. As in this problem, each demand volume split over exactly three different paths, the amount of paths which is used is 3, the sum of binary variable uikj on each path between i and j is 3.

F7: Path flow equation
As this is an equal split load balancing problem, for each used path i – k - j, demand volume hij between node i and j should be equally splited into nij = 3 path flows xikj.

F8: Bounds
All the path flow xikj should be non-negative.

F9: Binaries
All the binary variables which are equal to one of used paths.

Execution result

Given X = Y= 7:

number of transit nodes(Y)

	3	4	5	6	7
Execution time(in seconds)	0.01338	0.08076	0.07471	0.07672	0.10540
NO. of links with non-zero capacities	42	56	70	83	95
Highest capacity	26	24	23	19	17
Links with highest capacity	c71, c72, c73, d17, d27, d37	d26	c75	c74, d67	c67, c75
Load on the transit nodes(excluding unused nodes)	1:130.667 2:130.667 3:130.667	1:98.000 2:97.999 3:98.000 4:98.0	1:78.333 2:78.667 3:78.667 4:77.667 5:78.667	1:65.333 2:65.333 3:65.333 4:65.333 5:65.333 6:65.333	1:55.999 2:56.000 3:55.999 4:56.000 5:56.000 6:56.0 7:56.000

Explaination of execution results:

The execution result is same as we expected. As the number of transit nodes increasing, the execution time becomes longer. This is because the number of variables has increased, which makes the problem more complex to solve and hence takes more time. The number of links with non-zero capacities has also increased because demand is distributed to more transit nodes in order to reduce the capacity on each link. When the demand volume is fixed, less capacity on a link means less cost. We can also clearly find that the load on the used transit nodes are almost equal thanks to load balancing. The load on each transit node is decreasing with the addition of transit nodes since load is distributed to more transit nodes. Thus each node is responsible for less load on average.

Appendix 1

Source code:

createLPfile.py (used to generate lp file)

```
#generate the nodes of source, transit, destination
X = int(input('The amount of source nodes:'))
Y = int(input('The amount of transit nodes:'))
Z = int(input('The amount of destination nodes:'))
N = 3

print('Source nodes : {}\ntransit nodes : {}\nDestination node : {}'.format(X, Y, Z))
print('---------------------')


def DV_contraint():
    '''return the demand volume constraint:
    Xikj = hij
    which Xikj means the sum of load between source i to destination j'''
    DV = []
    demand_equation = []
    for i in range(1, X + 1):
        for j in range(1, Z + 1):
            dv = []
            for k in range(1, Y + 1):
                dv.append("x{}{}{}".format(i, k, j))
            DV = ' + '.join(dv) + ' = {}'.format(i + j)
            demand_equation.append(DV)
    demand_constraint = '\n'.join(demand_equation)
    return demand_constraint


def ST_capp_constraint():
    '''return the cappacity constraint from source to transit node'''
    ST = []
    capp1_equation = []
    for i in range (1, X + 1):
        for k in range (1, Y +1):
            st = []
            for j in range(1, Z+1):
                st.append('x{}{}{}'.format(i, k, j))
            ST = ' + '.join(st) + ' - c{}{} <= 0'.format(i, k)
            capp1_equation.append(ST)
    capp1_constraint = '\n'.join(capp1_equation)
    return capp1_constraint


def TD_capp_constraint():
    '''return the cappacity constraint from transit node to dest node'''
```

```
    TD = []
    capp2_equation = []
    for k in range(1, Y+1):
        for j in range(1, Z+1):
            td = []
            for i in range(1, X+1):
                td.append('x{}{}{}'.format(i, k, j))
            TD = ' + '.join(td) + ' - d{}{} <= 0'.format(k, j)
            capp2_equation.append(TD)
    capp2_constraint = '\n'.join(capp2_equation)
    return capp2_constraint


def TN_constraint():
    '''return the constraint of load of transit nodes which should be minimize'''
    TN = []
    tn_equation = []
    for k in range(1, Y + 1):
        tn = []
        for j in range(1, Z + 1):
            for i in range(1, X + 1):
                tn.append('x{}{}{}'.format(i, k, j))
        TN = ' + '.join(tn) + ' - r <= 0'.format(k, j)
        tn_equation.append(TN)
    tn_constraint = '\n'.join(tn_equation)
    return tn_constraint


def BV_constriant():
    '''return the binary variables constraint:
    Uikj = Nk
    which Uikj means the sum of used paths and Nk = N in this problem'''
    BV = []
    binary_equation = []
    for i in range(1, X + 1):
        for j in range(1, Z + 1):
            bv = []
            for k in range(1, Y + 1):
                bv.append("u{}{}{}".format(i, k, j))
            BV = ' + '.join(bv) + ' = {}'.format(N)
            binary_equation.append(BV)
    binary_constraint = '\n'.join(binary_equation)
    return binary_constraint


def DF_constraint():
    '''return the demand flow constraint:
    Nk * Xikj = hij * Uikj'''
    DF = []
    flow_equation = []
```

```
    for i in range(1, X + 1):
        for j in range(1, Z + 1):
            for k in range(1, Y + 1):
                DF = '{} x{}{}{} - {} u{}{}{} = 0'.format(N, i, k, j, i + j, i, k, j)
                flow_equation.append(DF)
    demand_flow_constraint = '\n'.join(flow_equation)
    return demand_flow_constraint


def Bounds_variable():
    '''return the bounds of demand variable : Xikj of this problem'''
    bound_x = []
    bound_unequation_x = []
    for i in range(1, X + 1):
        for j in range(1, Z + 1):
            for k in range(1, Y + 1):
                bound_x = '0 <= x{}{}{}'.format(i, k, j)
                bound_unequation_x.append(bound_x)
    Bounds_x = '\n'.join(bound_unequation_x)
    return Bounds_x


def binary_constraint():
    '''return binary constraints'''
    bc = ''
    for i in range(1, X+1):
        for k in range(1, Y+1):
            for j in range(1, Z+1):
                bc += 'u{}{}{}\n'.format(i,k,j)
    return bc


def createLP():
    '''create a LP file for this problem'''
    f = open(filename, 'w')
    content = \
    '''Minimize
    r
Subject to
demand volume: \n{}
srouce to tranfer node capp1: \n{}
transit to destination node capp2: \n{}
transit nodes: \n{}
binary variables: \n{}
demand flow: \n{}
Bounds
{}
0 <= r
Binaries
{}
```

```
End'''.format(demand_volume, ST_capacity,
            TD_capacity, transit_nodes, binary_variables, demand_flow, bounds_x, binaries)
    f.write(content)
    f.close


def set_filename():
    '''return the filename : XYZ.lp which Y belongs to {3, 4, 5, 6, 7}'''
    filename = '{}{}{}.lp'.format(X, Y, Z)
    return filename


demand_volume = DV_contraint()
ST_capacity = ST_capp_constraint()
TD_capacity = TD_capp_constraint()
transit_nodes = TN_constraint()
binary_variables = BV_constriant()
demand_flow = DF_constraint()
bounds_x = Bounds_variable()
binaries = binary_constraint()
filename = set_filename()

createLP()
```

---

analyser.py (used to run CPLEX with given lp files and extract the useful information)

```
import subprocess
import time
import json

def CPLEX(filename):
    '''run CPLEX to solve the problem of given lp file'''
    #args1 = ['/Users/mac/Desktop/364/a2/cplex', '-c', 'read /Users/mac/Desktop/364/a2/' +
filename, 'optimize',
          #'display solution variable -']
    args1 = ['/Users/zelta/Desktop/cplex', '-c', 'read /Users/zelta/Desktop/' + filename,
'optimize',
          'display solution variable -']

    time1 = time.time()
    process1 = subprocess.Popen(args1, stdout = subprocess.PIPE)

    output, error = process1.communicate()
    time2 = time.time()
    execution_time = time2 - time1

   # print('Execution_time: '+str("%.5f" % (execution_time))+'seconds')
    result = output.decode("utf-8").split()
```

```
    start = result.index("Incumbent")
    load = {}
    links = []
    link_count = 0
    cappacities = {}
    cappacities = {'max':[0,[]]}
    max_cappacity = 0.0
    for i in range(1,8):
        load[i] = float(0)
    for n in result[start:]:
        if n.startswith('x'):
            transit_node = int(n[2])
            x_index = result.index(n)
            load[transit_node] += float(result[x_index+1])
        if (n.startswith('c') or n.startswith('d')) and len(n) == 3:
            capp_index = result.index(n)
            cappacity = float(result[capp_index+1])
            if cappacity > 0:
                link_count += 1
                links.append(n)
            if cappacity == max_cappacity:
                cappacities['max'][1].append(n)
            if cappacity > max_cappacity:
                max_cappacity = cappacity
                cappacities['max'][0] = max_cappacity
                cappacities['max'][1]= [n]

    result = 'Execution_time: '+str("%.5f" % (execution_time))+'seconds\n'+'Load on transit
nodes: '+json.dumps(load)+'\n'+'Maximum cappacity: '+json.dumps(cappacities)+'\nNon-zero
capacity link count: '+str(link_count)+'\nNon-zero capacity links: '+' '.join(links)
    return result



def main():

    for n in range(3, 8):
        filename = '7{}7.lp'.format(n)
        print(filename)
        result = CPLEX(filename)
        f = open(filename + '_analyser.txt','w')
        f.write(result)
        f.close()

main()
```

Appendix 2

324.lp (lp file for situation where X = 3, Y = 2, Z= 4)

Minimize
r
Subject to
demand volume:
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
source to transit node capp1:
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
transit to destination node capp2:
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
transit nodes:
x111 + x211 + x311 + x112 + x212 + x312 + x113 + x213 + x313 + x114 + x214 + x314 - r <= 0
x121 + x221 + x321 + x122 + x222 + x322 + x123 + x223 + x323 + x124 + x224 + x324 - r <= 0
binary variables:
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
demand flow:
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
Bounds
0 <= x111
0 <= x121
0 <= x112
0 <= x122
0 <= x113
0 <= x123
0 <= x114
0 <= x124
0 <= x211
0 <= x221
0 <= x212
0 <= x222
0 <= x213
0 <= x223
0 <= x214
0 <= x224
0 <= x311
0 <= x321
0 <= x312
0 <= x322

0 <= x313
0 <= x323
0 <= x314
0 <= x324
0 <= r
Binaries
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