Alec Whitticase

58124306

Flow Assignment

Optimization problem.

Decision variables are r, flow,capacity of source-transit and capacity of transit-destination.

Minimize [r,c,d,x]   r

Subject to

1 Capacity of source transit links:

$\Sigma Z, j = 1, \;\; xikj \;\; <= cik, \; for \; i\{1, X], k\{1, Y\}$

Sum of xikj's for (source,transit pair) <= c(source,transit pair).

The sum of every demand volume along a source transit link must be less than or equal to the capacity of that link. (cik)

2 Capacity of transit destination links:

$\Sigma X, i = 1, \; xikj \;\; <= \;\; dkj, \; for \; k\{1, Y], j\{1, Z\}$

Sum of Xikj for (transit,destination pair) <= d(transit,destination).

The total of every demand volume along a transit destination link must be less than or equal to the capacity of that link. (dkj)

3 Demand volume:

$\Sigma Y, k = 1, \;\; xikj \;\; = hij, \; for \; i\{1, X], j\{1, Z\}$

Sum of xikj for (source,destination pair) = i + j (flow along link).

Sum of demand volume from source to destination along each transit should equal i+j, which is the total demand.

4: Number of paths used should be exactly 2.

$\Sigma Y, k = 1, \;\; uikj \;\; = 2, \; for \; i\{1, X], j\{1, Z\}$

Sum of u values for every transit path from source to destination equals 2, meaning of all the possible paths only two are being used, meaning the flow is split correctly.

5: Demand volume equally.

2 * xikj - i+j * uikj = 0, for i{1,X},k{1,Y},j{1,Z}

For each path in X,Y,Z, the cost of it's path doubled should equal the total load on that link times the binary value for if it is used. So if the path is used, the load should be half the total load, if not, it should equal zero.

6:
$\Sigma X, i = 1, \Sigma z, j = 1, \quad xikj \ <= \ r, \ for\, k\{1, Y]$
All source, destination pair (all transits) <= r (minimizing load)
The total volume on all transit nodes is less than or equal to r. This creates the value to be minimized.

7.
Bounds
r >= 0
Xikj >= 0
Cik >= 0
Dkj >= 0

Path flows, r, and link capacities for both source-transit and transit-destination cannot be negative.

8.
Binarys
Uikj

Set all binaries.

Results.

Number of Transit Nodes	Cplex Time	Non-zero capacity links	Lowest and Highest Transit node load	Highest capacity links
3	0.077	42	All nodes = 130.7776	D17 = 35
4	0.077	53	All nodes = 98.0000	D37 = 35
5	0.078	64	All nodes = 78.4000	C73 = 30
6	0.094	78	All nodes = 65.333	C64 = 24
7	0.141	90	All nodes = 56.0000	C56,D17 = 21

Cplex time:

Through 3 to 5, there is very little change, and then the run time increases more strongly from 5 to 6 to 7 transit nodes. It makes sense that runtime would increase as the number of nodes increases, as the complexity of the problem is increasing.

Non-zero capacity links.

The more transit nodes there are, the higher the number of links that will be utilized. This column shows a linear increase in the number of links utilized, which makes sense.

Lowest and highest transit node load.

In each instance, the load on every transit node is the same. When minimizing r, you are minimizing the load on every transit node, and in order to minimize this all loads need to be the same. The fact that all the loads are the same indicates the problem formation is correct.

Highest capacity links.

This shows that with more transit nodes, and therefore more links, the strain on each individual link is decreasing. It's also notable that in each case a link with a higher i+j was the most used, even if it wasn't strictly the maximum of 7-7 each time.

Appendices on the following pages.

323.lp

```
Minimize
r
Subject to
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
x121 + x221 + x321 - d21 <= 0
x112 + x212 + x312 - d12 <= 0
x122 + x222 + x322 - d22 <= 0
x113 + x213 + x313 - d13 <= 0
x123 + x223 + x323 - d23 <= 0
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 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
r >= 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
c11 >= 0
c21 >= 0
c31 >= 0
c12 >= 0
c22 >= 0
c32 >= 0
```

```
d11 >= 0
d12 >= 0
d13 >= 0
d21 >= 0
d22 >= 0
d23 >= 0
Binary
u111
u112
u113
u121
u122
u123
u211
u212
u213
u221
u222
u223
u311
u312
u313
u321
u322
u323
End
```

```
import sys

def source_to_transit_capacity(sources,transit,destination):
    capacity_string = ""
    #generates the arguments that cik for each source transit pair <= link capacity
    for i in range(1,sources +1):
        for k in range(1, transit + 1):
            entry = ""
            for j in range(1,destination+1):
                entry += "x{}{}{} + ".format(i,k,j)
            entry = entry[0:-2]
            entry += "- c{}{} <= 0 \n".format(i,k)
            capacity_string += entry
    return capacity_string

def transit_to_desination_capacity(sources,transit,destination):
    capacity_string = ""
    for j in range(1,destination+1):
        for k in range(1, transit + 1):
            entry = ""
            for i in range(1, sources + 1):
                entry += "x{}{}{} + ".format(i,k,j)
            entry = entry[0:-2]
            entry += "- d{}{} <= 0 \n".format(k,j)
            capacity_string += entry
    return capacity_string

def source_to_dest_demand_volume(sources,transit,destination):
    #generates the load count hij = i + j for each path x ikj
    demand_string = ""
    # in sources, in destination (create one for each in next), in transit
    for i in range(1,sources + 1):
        for j in range(1,destination + 1):
            #source * destination entires, create here
            entry = ""
            for k in range(1,transit+1):
                entry += ("x{}{}{} + ".format(i,k,j))
            entry = entry[0:-2]
            entry += "= {} \n".format(i+j)
            demand_string += entry
    return demand_string

def split_along_two_paths(sources,transit,destination):
    binary_string = ""
    #generates and adds each binary and supporting constraight to ensure each
    #demand load goes over exactly two different paths
    #first, add that all possible source:destination pairs only use one souce node
    for i in range(1,sources + 1):
        for j in range(1,destination + 1):
            entry = ""
            #now we're in source/destination pairs, find binarys
            for k in range(1,transit+1):
                entry += "u{}{}{} + ".format(i,k,j)
            entry = entry[0:-2]
            entry += "= 2 \n"
            binary_string += entry
    #That first section covers u111 + u121 + u131 type stuff
    #second part is 2 (path) - load (path binary) = 0
    #checks that either both are 0 or both are equal to load dependent
    # on if the binary is true
    #if it's being used, flow = half max
    #otherwise, should be zero
    for i in range(1,sources+1):
        for k in range(1,transit+1):
            for j in range(1,destination+1):
                binary_string += "2 x{}{}{} - {} u{}{}{} = 0 \n".format(i,k,j,i+j,i,k,j)
    return binary_string

def load_balance_r(sources,transit,destination):
    #make the r equations to minimise
    r_string = ""
    for k in range(1,transit+1):
        big_entry = ""
        for i in range(1,sources+1):
            sub_entry = ""
```

```
            for j in range(1,destination+1):
                sub_entry += "x{}{}{} + ".format(i,k,j)
            big_entry += sub_entry
        big_entry = big_entry[0:-2]
        big_entry += "- r <= 0\n"
        r_string += big_entry
    return r_string

def bounds(sources,transit,destination):
    #bounds section (r,path flow,transit to destination and source to transit)
    bound_string = ""

    bound_string += "r >= 0 \n"

    for i in range(1,sources + 1):
        for k in range(1, transit + 1):
            for j in range(1, destination + 1):
                bound_string += "x{}{}{} >= 0 \n".format(i,k,j)

    for k in range(1,transit + 1):
        for i in range(1, sources + 1):
            bound_string += "c{}{} >= 0 \n".format(i,k)

    for k in range(1,transit + 1):
        for j in range(1,destination + 1):
            bound_string += "d{}{} >= 0 \n".format(k,j)

    return bound_string

def binarys(sources,transit,destination):
    binary_string = ""
    for i in range(1,sources + 1):
        for k in range(1,transit + 1):
            for j in range(1, destination + 1):
                binary_string += "u{}{}{} \n".format(i,k,j)

    return binary_string




def main():
    bar = "----------------------------------------------"
    #sources = 3
    #transit = 2
    #destination = 3
    sources = int(sys.argv[1])
    transit = int(sys.argv[2])
    destination = int(sys.argv[3])

    # order mentioned in problem description
    # source to transit capacity,transit to destination capacity,
    # #source to destination demand load, split over 2 paths

    if transit < 2:
        print("Invalid. Transit nodes must number at least 2.")
        sys.exit()

    #start generating the lp file
    lp_file = ""
    lp_file += "Minimize\n"
    lp_file += "r\n"
    lp_file += "Subject to\n"

    #source to transit capacity, cik
    source_cap = source_to_transit_capacity(sources,transit,destination)
    lp_file += source_cap

    #transit to destination capacity, dkj
    transit_cap = transit_to_desination_capacity(sources,transit,destination)
    lp_file += transit_cap

    #source to destination demand load
    demand_load = source_to_dest_demand_volume(sources,transit,destination)
```

```python
    lp_file += demand_load

    #split over 2 paths/binary func
    binary_values = split_along_two_paths(sources,transit,destination)
    lp_file += binary_values

    # everything for a given transit -r <= 0
    min_r_line = load_balance_r(sources,transit,destination)
    lp_file += min_r_line

    #bounds section (r,path flow,transit to destination and source to transit)
    lp_file += "Bounds \n"
    bound = bounds(sources,transit,destination)
    lp_file += bound

    #binarys
    lp_file += "Binary \n"
    binary = binarys(sources,transit,destination)
    lp_file += binary

    lp_file += "End"

    with open('323.lp', 'w') as f:
        f.write(lp_file)


main()
```