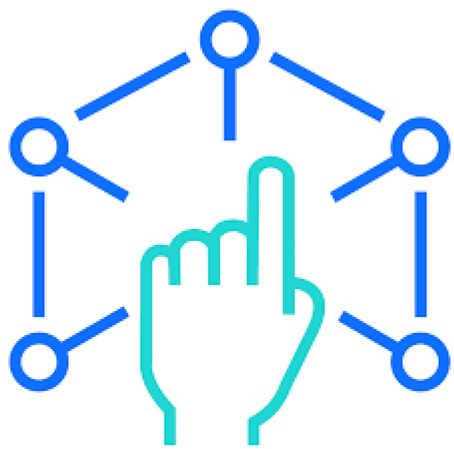
COSC364
Internet Technologies and Engineering

Second Assignment



Student Names	Student ID	Contribution
Ahmad Alsaleh	14749959	50%
Chadol Han	79364948	50%

Problem formulation & Explanations for the formulation

The Planning booklet from Learn gave us a good headstart especially **Problem 5.2.6**, and our formulation is shown below.

For equal spilt we get…

Minimize [x, d, c, r]

$$r$$

Subject to

Demand volume is calculated by the summation of x_{ikj} which is the demand volume between source node *i*, destination node j, that is routed through the transit node *k*, but in short only the source node and destination node requires a demand volume which is simply just adding i + j(h).

$$\sum_{k=1}^{Y_k} x_{ikj} = i + j = (h_{ij}) \quad for\ i\ \in \{1, \dots, X\}, \qquad j\ \in \{1, \dots, Z\}$$

Result of the utilization will always be 3 as stated in the assignment handout, **each demand volume shall be spilt over three different paths.** The second line represents each path gets an equal share of the demand volume (for equal split), which is why it is divided by the number of paths the volume is split over (in this case 3).

$$\sum_{k=1}^{y_k} u_{ikj} = 3 \quad for\ i\ \in \{1, \dots, X\}, \qquad j\ \in \{1, \dots, Z\}$$

$$x_{ijk} = \frac{(i + j)\, u_{ikj}}{3} \quad for\ i\ \in \{1, \dots, X\}, \qquad k\ \in \{1, \dots, Y\}, \qquad j\ \in \{1, \dots, Z\}$$

The source constraint becomes the summation of the demand flow in each path which will always equal the capacity (link between source node and transit node) since demand volume is equal throughout all paths.

$$\sum_{j=1}^{z} x_{ikj} \le c_{ik} \quad for\ i\ \in \{1, \dots, X\}, \qquad k\ \in \{1, \dots, Y\}$$

The destination constraint conditions are the same as the source constraint equation, but instead the capacity is the link between transit node and destination node.

$$\sum_{i=1}^{x} x_{ikj} \leq d_{kj} \quad for\ k\ \in\ \{1, \dots, Y\}, \qquad j\ \in\ \{1, \dots, Z\}$$

We use binary indicator variables for telling whether the path for the demand volume is used or not, if the path is used, u = 1, if it is not used, u = 0.

$$u_{ikj}\ \in\ \{0, 1\} \quad for\ i\ \in\ \{1, \dots, X\}, \qquad k\ \in\ \{1, \dots, Y\}, \qquad j\ \in\ \{1, \dots, Z\}$$

Bounds must be non-negative, as negative data rates make no sense. In other words, the following constraint inequalities hold.

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

$\sum_{i=1}^{x} \sum_{j=1}^{z} x_{ikj} \leq r$ All assuming bounds where...

$$x_{ikj} \geq 0, \qquad c_{ik} \geq 0, \qquad d_{kj} \geq 0, \qquad r \geq 0$$

This equation makes the formulation a load balancing problem.

x = Rate of flow of the demand volume

u= Indicator/binary variable

r = Value of objective function (auxiliary variable)

d = Capacity from transit to destination (from j to k)

c = Capacity from source to transit (from I to k)

X = Source nodes

Y = Transit nodes

Z = Destination nodes

i = Source node

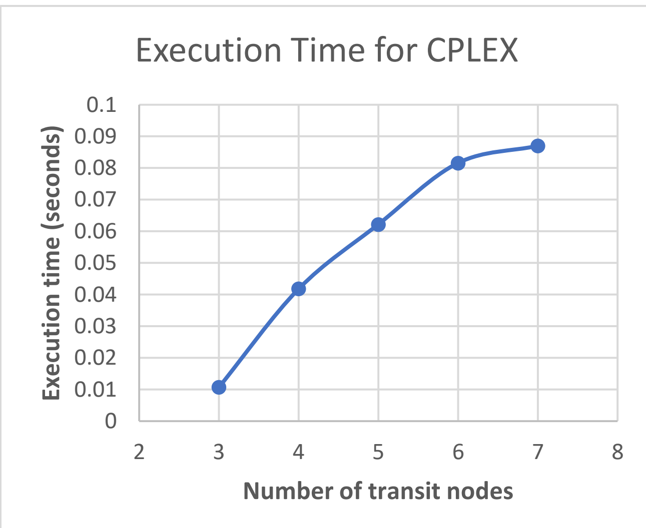
k = Transit node

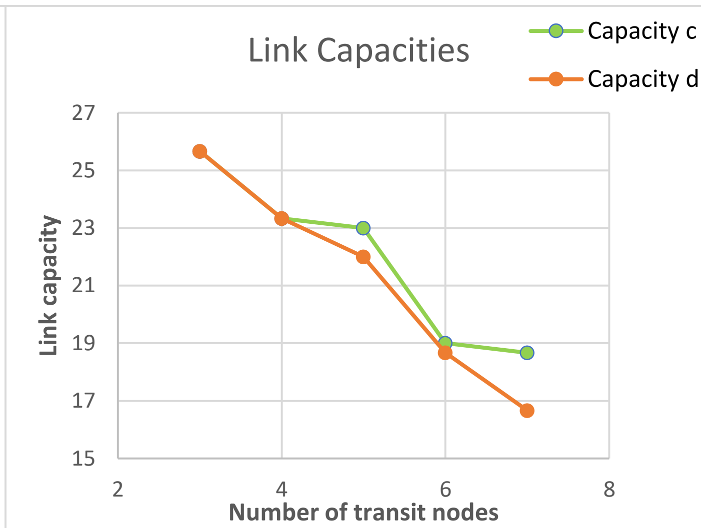
j = Destination node

CPLEX Results

The following table shows the solution and results that CPLEX generates with the LP file created by the formulation above. The source nodes are represented by X, transit nodes represented by Y, and destination nodes represented by Z.

Number of nodes (X | Y | Z)	Objective function value (r)	Execution time (seconds)	Highest capacity link c	Highest capacity link d	Links with non-zero capacities
X = 7 | Y = 3 | Z = 7	130.666667	0.0107	25.666667	25.666667	42
X = 7 | Y = 4 | Z = 7	98.00000	0.0418	23.333333	23.333333	56
X = 7 | Y = 5 | Z = 7	78.666667	0.0622	23.000000	22.000000	70
X = 7 | Y = 6 | Z = 7	65.33333	0.0816	19.000000	18.666667	83
X = 7 | Y = 7 | Z = 7	56.00000	0.0870	18.666667	16.666667	96





For links with non-zero capacities, it assumes that a link is from source to transit, or from transit to source. An example of a capacity of a link would be c11, which is a link from source node 1 to transit node 1, or d11 which is a link from transit node 1 to destination node 1.

It is evident from the table and graphs above that:

1. **The execution time increases as the number of transit node Y increase:** This Is the expected result, as Y (number of transit nodes) increases, there are more decisions to be made and more constraints are introduced. These new variables and constraints will make the problem "bigger" and in general increase the time required to numerically solve an instance of the problem. Therefore CPLEX will take more time to find a solution as the number of nodes increases.
2. **The capacities decrease as the number of transit nodes Y increase:** This result is also expected, as Y (number of transit nodes) increases, the load can be spread across more transit nodes Y. Therefore, since the load is spread across more links which are now available, each link has less capacity. We can see that the link utilization decreases according to the amount of nodes in the network.

3. **Links with non-zero capacities increase as the number of transit nodes Y increase:** This is expected due to Point 2 made above. Since there are more links available to spread the load over, more links will be needed/used. Hence more links with non-zero capacities as the number of transit nodes Y increases.

Source code, highlighted using http://www.planetb.ca/syntax-highlight-word

```
################################################################################
#### COSC364 Assignment 2
#### The following is a program which creates an LP file to be read on cplex
#### Ahmad Alsaleh (ID: 14749959) | Chadol Han (ID: 79364948)
################################################################################
import codecs
import subprocess
import time
#GLOBALS
global X
global Y
global Z
#Creates LP file
f = open("LPfile.lp","w+")

#Used to check for input positive integers error
class Error(UserWarning):
    pass

#Indenting string recipe, taken from http://code.activestate.com/recipes/576867-indent-a-string/
def indent(txt, stops=1):
    return '\n'.join(" " * 4 * stops + line for line in  txt.splitlines())


def write_initial():
    """"Function that writes required text to LP file"""
    f.write("Minimize\n")
    f.write(indent('r'))
    f.write("\nSubject to\n")

def inputs():
    """"Function that asks the user for the amount of nodes required and
    checks whether they are positive integers"""
    while 1:
        try:
            global X
            global Y
            global Z
            source_nodes = input("Enter amount of source nodes: ")
            transit_nodes = input("Enter amount of transit nodes: ")
            destination_nodes = input("Enter amount of destination nodes: ")
            X = int(source_nodes)
            Y = int(transit_nodes)
            Z = int(destination_nodes)
            if X <= 0:
                raise Error
            elif Y <= 0:
                raise Error
            elif Z <= 0:
                raise Error
        except Error:
            print("Nodes only accept positive integers, please try again!")
            continue
```

```python
        except ValueError:
            print("The inputs must be a number, try again!")
            continue
        else:
            break

def write_demand_volume(source, transit, destination):
    """writes the demand volume from i to j to the LP file"""
    for i in range(1,source+1):
        string = ""
        for k in range(1,destination+1):
            string = ""
            for j in range(1,transit+1):
                string += "x" + str(i) + str(j) + str(k) + " + "
                h = i + k
            f.write(indent(string[:-2] + "= {}".format(h))+"\n")


def write_utilsation_u(source, transit, destination):
    """writes utilisation constraints for the transit nodes to the LP file"""
    for i in range(1,source+1):
        string = ""
        for k in range(1,destination+1):
            string = ""
            for j in range(1,transit+1):
                string += "u" + str(i) + str(j) + str(k) + " + "
                n_k = 3
            f.write(indent(string[:-2] + "= {}".format(n_k)) + "\n")


def write_demand_flow(source, transit, destination):
    """writes demand flow from i to j to the LP file"""
    for i in range(1,source+1):
            string = ""
            for j in range(1,transit+1):
                string = ""
                for k in range(1,destination+1):
                    h = i + k
                    string += "3 x" + str(i) + str(j) + str(k) + " - " + str(h) + " u" + str(i) + str(j) + str(k) + " = 0"
                    f.write(indent(string) + "\n")
                    string = ""


def write_source_constraint(source, transit, destination):
    """writes capacity constraints from source to transit to the LP file"""
    for i in range(1,source+1):
        string = ""
        for j in range(1,transit+1):
            string = ""
            for k in range(1,destination+1):
                string += "x" + str(i) + str(j) + str(k) + " + "
            f.write(indent(string[:-2] + "- c" + str(i) + str(j) + " = 0") + "\n")


def write_destination_constraint(source, transit, destination):
    """writes capacity constraints from transit to destination to the LP file"""
    for k in range(1,destination+1):
        string = ""
        for j in range(1,transit+1):
            string = ""
```

```
            for i in range(1,source+1):
                string += "x" + str(i) + str(j) + str(k) + " + "
            f.write(indent(string[:-2] + "- d" + str(j) + str(k) + " = 0") + "\n")


def write_transit_constraints(source, transit, destination):
    """writes transit constraints to the LP file"""
    for j in range(1,transit+1):
        string = ""
        for i in range(1,source+1):
            for k in range(1,destination+1):
                string += "x" + str(i) + str(j) + str(k) + " + "
        f.write(indent(string[:-2] + "- r <= 0") + "\n")

def write_path_flow_bounds(source, transit, destination):
    """writes bounds for the path flow (non-negativity)"""
    f.write("\nBounds\n")
    f.write(indent("r >=0") + "\n")
    for i in range(1,source+1):
        string = ""
        for j in range(1,transit+1):
            string = ""
            for k in range(1,destination+1):
                string += "x" + str(i) + str(j) + str(k) + " + "
                f.write(indent(string[:-2] + ">= 0") + "\n")
                string = ""

def write_capacity_S_to_T(source, transit):
    """writes bounds for the capacity c from source to transit(non-negativity)"""
    for i in range(1,source+1):
        string = ""
        for j in range(1,transit+1):
            string += "c" + str(i) + str(j)
            f.write(indent(string + " >= 0") + "\n")
            string = ""

def write_capacity_T_to_D(transit, destination):
    """writes bounds for the capacity d from transit to dest(non-negativity)"""
    for j in range(1,transit+1):
        string = ""
        for k in range(1,destination+1):
            string += "d" + str(j) + str(k)
            f.write(indent(string + " >= 0") + "\n")
            string = ""

def write_to_binary(source, transit, destination):
    """writes the binary values and adds an end so the LP file ends"""
    f.write("\n\nBinary\n")
    for i in range(1,source+1):
        string = ""
        for j in range(1,transit+1):
            string = ""
            for k in range(1,destination+1):
                string += "u" + str(i) + str(j) + str(k)
                f.write(indent(string) + "\n")
                string = ""
    f.write("End")

def run_in_cplex(file):
    """The following function runs the LP file created by this program
    into CPLEX, CPLEX then reads it and optimizes it"""
```

```python
    #Add "display solution variables -" after optimize to show solutions
    #It was removed so the execution time could be properly calculated
    proc = subprocess.Popen(["./cplex"] + ["-c", "read", file, "optimize"], stdout=subprocess.PIPE)
    out,err = proc.communicate()
    result = out.decode("utf-8")
    return result

def main():
    write_initial()
    inputs()
    write_demand_volume(X,Y,Z)
    f.write("\n")
    write_utilsation_u(X,Y,Z)
    f.write("\n")
    write_demand_flow(X, Y, Z)
    f.write("\n")
    write_source_constraint(X,Y,Z)
    f.write("\n")
    write_destination_constraint(X, Y, Z)
    f.write("\n")
    write_transit_constraints(X, Y, Z)
    write_path_flow_bounds(X, Y, Z)
    write_capacity_S_to_T(X, Y)
    write_capacity_T_to_D(Y, Z)
    write_to_binary(X, Y, Z)

    #Close file
    f.close()

    #To time the execution time
    before = time.time()
    print(run_in_cplex("LPfile.lp"))
    elapsed_time = time.time() - before
    print(elapsed_time)

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

LP File generated for X = 3 | Y = 2 | Z = 4

```
Minimize
   r
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 x112 - 3 u112 = 0
   3 x113 - 4 u113 = 0
   3 x114 - 5 u114 = 0
   3 x121 - 2 u121 = 0
   3 x122 - 3 u122 = 0
   3 x123 - 4 u123 = 0
   3 x124 - 5 u124 = 0
   3 x211 - 3 u211 = 0
   3 x212 - 4 u212 = 0
   3 x213 - 5 u213 = 0
   3 x214 - 6 u214 = 0
   3 x221 - 3 u221 = 0
   3 x222 - 4 u222 = 0
   3 x223 - 5 u223 = 0
   3 x224 - 6 u224 = 0
   3 x311 - 4 u311 = 0
```

3 x312 - 5 u312 = 0
3 x313 - 6 u313 = 0
3 x314 - 7 u314 = 0
3 x321 - 4 u321 = 0
3 x322 - 5 u322 = 0
3 x323 - 6 u323 = 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
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
x114 + x214 + x314 - d14 = 0
x124 + x224 + x324 - d24 = 0

x111 + x112 + x113 + x114 + x211 + x212 + x213 + x214 + x311 + x312 + x313 + x314 - r <= 0
x121 + x122 + x123 + x124 + x221 + x222 + x223 + x224 + x321 + x322 + x323 + x324 - r <= 0

Bounds
r >=0
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
   c11 >= 0
   c12 >= 0
   c21 >= 0
   c22 >= 0
   c31 >= 0
   c32 >= 0
   d11 >= 0
   d12 >= 0
   d13 >= 0
   d14 >= 0
   d21 >= 0
   d22 >= 0
   d23 >= 0
   d24 >= 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
```