COSC 364:
Internet Technologies and Engineering

Assignment 2 : Flow planning

Name	Student ID
Ariel Evangelista	aev35 (62193622)
Aiman Hazashah	mah198 (76504305)

Marking distribution Marks are distributed equally between both partners of 50% each.

0.1 Problem formulation and solution

According to the booklet in subsection 6.2.2 Problem, the formulation given is used as a references for our formulation. The assignment requirements were to formulate an optimization problem for values X, Y and Z with (with Y $>$ 3). Furthermore, amount of demand volume between source node S_i (1<i<X) and destination node j(i<j<Z) are written as $i + j = (h_{ij})$

and for equal split:

$$\text{minimize[x,c,d,r]} \quad \text{r}$$

$$\text{Subject to} \quad \sum_{k=1}^{y_k} x_{ikj} = i + j = (h_{ij}) \quad \text{for i} \in \{1, ..., X\}, \quad \text{j} \in \{1, ..., Z\}$$

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

$$x_{ikj} = \frac{h_{ij}{*}u_{ikj}}{3} \quad \text{for i} \in \{1, ..., X\}, \quad \text{k} \in \{1, ..., Y\}, \quad \text{j} \in \{1, ..., Z\}$$

$$\sum_{i=1}^{x_i} \sum_{j=1}^{z_k} x_{ikj}\delta_{ikj} \leq \text{r}$$

$$\sum_{i=1}^{x_i} \sum_{j=1}^{z_k} x_{ikj}$$

$$\sum_{j=1}^{z_j} x_{ikj} \leq c \quad \text{for i} \in \{1, ..., X\}, \quad \text{k} \in \{1, ..., Y\}$$

$$\sum_{i=1}^{x_i} x_{ikj} \leq d \quad \text{for k} \in \{1, ..., Y\}, \quad \text{j} \in \{1, ..., Z\}$$

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

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

$$c_{ik} \geq 0 \quad \text{for i} \in \{1, ..., X\}, \quad \text{k} \in \{1, ..., Y\}$$

$$d_{kj} \geq 0 \quad \text{k} \in \{1, ..., Y\}, \quad \text{j} \in \{1, ..., Z\}$$

$$r \geq 0$$

The utilization would be split in 3 different paths where each path received a balanced demand volume that is split equally. Below are the source and destination constraints equation in which the capacity of the links are between i to j:

$\sum_{j=1}^{z_j} x_{ikj} \leq c \quad \text{for i} \in \{1, ..., X\}, \quad \text{k} \in \{1, ..., Y\}$

$\sum_{i=1}^{x_i} x_{ikj} \leq d \quad \text{for k} \in \{1, ..., Y\}, \quad \text{j} \in \{1, ..., Z\}$

Next the indicator variables determines either the path of the demand flow is utilized where '1' means utilized and '0' not being utilized:

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

0.2 CPLEX Results

The program run for each y ∈{3,4,5,6,7}, which solve the resulting LP file with CPLEX that record the following outputs:-

[x = 7,y=3,z=7]

Objective function (r)	Links of non-zero capacities (total)	CPLEX run time (seconds)	Highest capacity of c	Highest capacity of d
R = 130.666667	42	0.011988	26.00	26.00

[x = 7,y=4,z=7]

Objective function (r)	Links of non-zero capacities (total)	CPLEX run time (seconds)	Highest capacity of c	Highest capacity of d
R = 98.00	56	0.053926	23.00	23.00

[x = 7,y=5,z=7]

Objective function (r)	Links of non-zero capacities (total)	CPLEX run time (seconds)	Highest capacity of c	Highest capacity of d
r = 78.666667	70	0.061613	23.00	22.00

[x = 7,y=6,z=7]

Objective function (r)	Links of non-zero capacities (total)	CPLEX run time (seconds)	Highest capacity of c	Highest capacity of d
r = 65.33	84	0.065729	21.00	20.00

[x = 7,y=7,z=7]

Objective function (r)	Links of non-zero capacities (total)	CPLEX run time (seconds)	Highest capacity of c	Highest capacity of d
r = 56.00	96	0.088036	19.00	17.00

0.3 Conclusion from the CPLEX records:

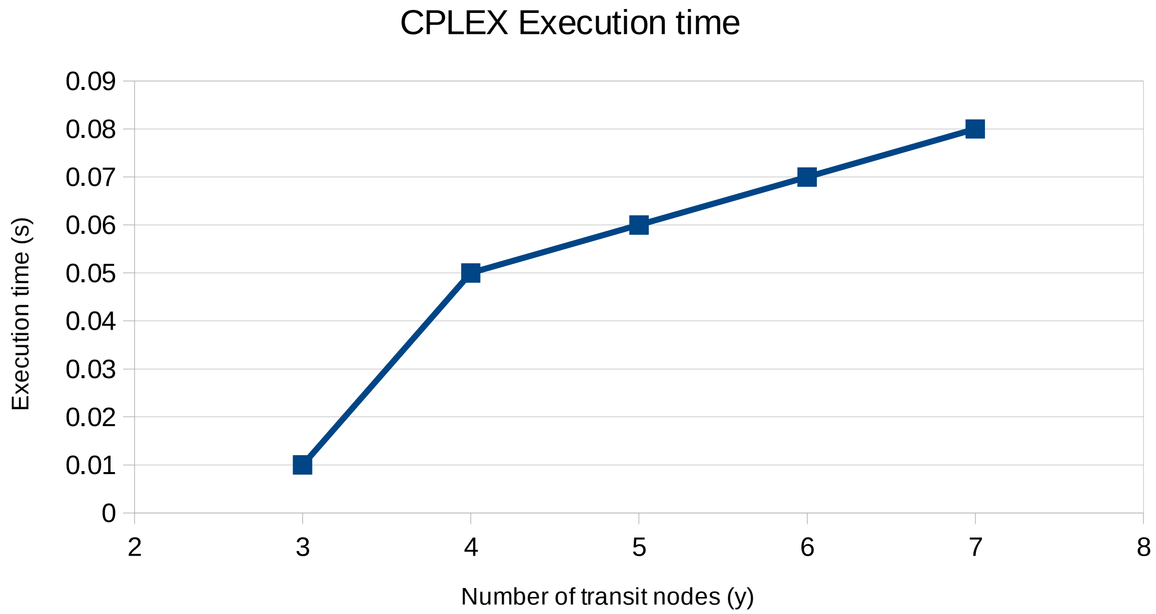

Figure 1: shows the Increment of execution time as there are increase in transit nodes.

1. **The higher the transit nodes, the decrease the constraint capacity.**
As the transit node (y) increases, the capacity burden would reduced as there are more links towards destination nodes. The load balancing process would ensure each transit nodes are optimized.

2. **The higher the transit nodes, the increase of execution time.**
As the transit nodes (y) increases, the computational requirement to calculate the load would be greater. The spread of links would demand more time to process which in turn increment the execution time.

3. **The higher the Links of non-zero capacities, the increase in execution time.**
As the Links of non-zero capacities increases, the process to spread the load over all other transit would be increase. This require more process as there are more transit nodes to handle.

4. **The higher the transit node, the decrease in objective functions**
As the transit node (y) increases, the objective functions (r) would decrease. This would reduction of total cost occurs when more load are balanced among the transit nodes (y).

0.4 Generated LP file (for X= 3,Y= 2 and Z= 4)

The solutions is not feasible because Y could not spread enough route to catter the Destination. It would reach its maximum capacity for the transport node. The Transport node minimum integer should be more than 3 to sustain any proper component function. When we run the LP file on CPLEX, we got

$ CPLEX> No integer feasible solution exists.

Below is the generated LP file:

```
Source Nodes:     3
Transit Nodes:    2
Destination Nodes: 4
Minimize
r
Subject to
 xS1T1D1 + xS1T2D1 = 2
 xS1T1D2 + xS1T2D2 = 3
 xS1T1D3 + xS1T2D3 = 4
 xS1T1D4 + xS1T2D4 = 5
 xS2T1D1 + xS2T2D1 = 3
 xS2T1D2 + xS2T2D2 = 4
 xS2T1D3 + xS2T2D3 = 5
 xS2T1D4 + xS2T2D4 = 6
 xS3T1D1 + xS3T2D1 = 4
 xS3T1D2 + xS3T2D2 = 5
 xS3T1D3 + xS3T2D3 = 6
 xS3T1D4 + xS3T2D4 = 7
 xS1T1D1 + xS1T1D2 + xS1T1D3 + xS1T1D4 - yS1T1 = 0
 xS1T2D1 + xS1T2D2 + xS1T2D3 + xS1T2D4 - yS1T2 = 0
 xS2T1D1 + xS2T1D2 + xS2T1D3 + xS2T1D4 - yS2T1 = 0
 xS2T2D1 + xS2T2D2 + xS2T2D3 + xS2T2D4 - yS2T2 = 0
 xS3T1D1 + xS3T1D2 + xS3T1D3 + xS3T1D4 - yS3T1 = 0
 xS3T2D1 + xS3T2D2 + xS3T2D3 + xS3T2D4 - yS3T2 = 0
 xS1T1D1 + xS2T1D1 + xS3T1D1 - yT1D1 = 0
 xS1T1D2 + xS2T1D2 + xS3T1D2 - yT1D2 = 0
 xS1T1D3 + xS2T1D3 + xS3T1D3 - yT1D3 = 0
 xS1T1D4 + xS2T1D4 + xS3T1D4 - yT1D4 = 0
 xS1T2D1 + xS2T2D1 + xS3T2D1 - yT2D1 = 0
 xS1T2D2 + xS2T2D2 + xS3T2D2 - yT2D2 = 0
 xS1T2D3 + xS2T2D3 + xS3T2D3 - yT2D3 = 0
 xS1T2D4 + xS2T2D4 + xS3T2D4 - yT2D4 = 0
 yS1T1 - cS1T1 <= 0
 yS1T2 - cS1T2 <= 0
 yS2T1 - cS2T1 <= 0
 yS2T2 - cS2T2 <= 0
 yS3T1 - cS3T1 <= 0
 yS3T2 - cS3T2 <= 0
 yT1D1 - dT1D1 <= 0
 yT1D2 - dT1D2 <= 0
 yT1D3 - dT1D3 <= 0
 yT1D4 - dT1D4 <= 0
 yT2D1 - dT2D1 <= 0
 yT2D2 - dT2D2 <= 0
 yT2D3 - dT2D3 <= 0
 yT2D4 - dT2D4 <= 0
 yS1T1 + yS2T1 + yS3T1 - r <= 0
 yS1T2 + yS2T2 + yS3T2 - r <= 0
 uS1T1D1 + uS1T2D1 = 3
 uS1T1D2 + uS1T2D2 = 3
 uS1T1D3 + uS1T2D3 = 3
 uS1T1D4 + uS1T2D4 = 3
 uS2T1D1 + uS2T2D1 = 3
 uS2T1D2 + uS2T2D2 = 3
 uS2T1D3 + uS2T2D3 = 3
```

```
 uS2T1D4 + uS2T2D4 = 3
 uS3T1D1 + uS3T2D1 = 3
 uS3T1D2 + uS3T2D2 = 3
 uS3T1D3 + uS3T2D3 = 3
 uS3T1D4 + uS3T2D4 = 3
 3 xS1T1D1 - 2 uS1T1D1 = 0
 3 xS1T1D2 - 3 uS1T1D2 = 0
 3 xS1T1D3 - 4 uS1T1D3 = 0
 3 xS1T1D4 - 5 uS1T1D4 = 0
 3 xS1T2D1 - 2 uS1T2D1 = 0
 3 xS1T2D2 - 3 uS1T2D2 = 0
 3 xS1T2D3 - 4 uS1T2D3 = 0
 3 xS1T2D4 - 5 uS1T2D4 = 0
 3 xS2T1D1 - 3 uS2T1D1 = 0
 3 xS2T1D2 - 4 uS2T1D2 = 0
 3 xS2T1D3 - 5 uS2T1D3 = 0
 3 xS2T1D4 - 6 uS2T1D4 = 0
 3 xS2T2D1 - 3 uS2T2D1 = 0
 3 xS2T2D2 - 4 uS2T2D2 = 0
 3 xS2T2D3 - 5 uS2T2D3 = 0
 3 xS2T2D4 - 6 uS2T2D4 = 0
 3 xS3T1D1 - 4 uS3T1D1 = 0
 3 xS3T1D2 - 5 uS3T1D2 = 0
 3 xS3T1D3 - 6 uS3T1D3 = 0
 3 xS3T1D4 - 7 uS3T1D4 = 0
 3 xS3T2D1 - 4 uS3T2D1 = 0
 3 xS3T2D2 - 5 uS3T2D2 = 0
 3 xS3T2D3 - 6 uS3T2D3 = 0
 3 xS3T2D4 - 7 uS3T2D4 = 0
 S1T1D1 + xS1T1D2 + xS1T1D3 + xS1T1D4 + xS2T1D1 + xS2T1D2 + xS2T1D3 + xS2T1D4 + xS3T1D1 + xS3T1D2 + xS3T1D3 +
xS3T1D4 - lT1 = 0
 S1T2D1 + xS1T2D2 + xS1T2D3 + xS1T2D4 + xS2T2D1 + xS2T2D2 + xS2T2D3 + xS2T2D4 + xS3T2D1 + xS3T2D2 + xS3T2D3 +
xS3T2D4 - lT2 = 0
Bounds
 yS1T1 >= 0
 yS1T2 >= 0
 yS2T1 >= 0
 yS2T2 >= 0
 yS3T1 >= 0
 yS3T2 >= 0
 yT1D1 >= 0
 yT1D2 >= 0
 yT1D3 >= 0
 yT1D4 >= 0
 yT2D1 >= 0
 yT2D2 >= 0
 yT2D3 >= 0
 yT2D4 >= 0
 xS1T1D1 >= 0
 xS1T1D2 >= 0
 xS1T1D3 >= 0
 xS1T1D4 >= 0
 xS1T2D1 >= 0
 xS1T2D2 >= 0
 xS1T2D3 >= 0
 xS1T2D4 >= 0
 xS2T1D1 >= 0
 xS2T1D2 >= 0
 xS2T1D3 >= 0
 xS2T1D4 >= 0
 xS2T2D1 >= 0
 xS2T2D2 >= 0
 xS2T2D3 >= 0
 xS2T2D4 >= 0
 xS3T1D1 >= 0
 xS3T1D2 >= 0
 xS3T1D3 >= 0
 xS3T1D4 >= 0
 xS3T2D1 >= 0
 xS3T2D2 >= 0
 xS3T2D3 >= 0
 xS3T2D4 >= 0
 r >= 0
```

```
Binary
 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
```

<u>The source code of your program as an appendix.</u>

```
"""
COSC364 Flow Planning
WRITTEN BY:
Aiman Hazashah
Ariel Evangelista - aev35
"""

# ==============================
# IMPORTS
import os
import time
# GLOBAL VARIABLE(s)
filename = 'flow.lp'    # file to be created
cplex_path = "cplex" # cplex path
# ==============================

def get_input():
    """Gets the number of Nodes"""
    s = int(input("Source Nodes:      "))
    t = int(input("Transit Nodes:     "))
    d = int(input("Destination Nodes: "))
    return s,t,d


def create_nodes(s, t, d):
    """Creates nodes on a list"""
    start = []
    trans = []
    dest = []

    for i in range (1, s + 1):
        start.append(str("S" + str(i)))

    for i in range (1, t + 1):
        trans.append(str("T" + str(i)))

    for i in range (1, d + 1):
        dest.append(str("D" + str(i)))

    return start, trans, dest


def get_dem_vol(start, trans, dest):
    """Calculates the demand volume using Hij = i + j"""
    demands = []
    x_var = []
    for s in start:
        for d in dest:
            dem = start.index(s) + dest.index(d) + 2 # 2 for incrementing index
            form = ''
            for t in trans:
                dem_var = "x{}{}{}".format(s, t, d)
                form += dem_var
                x_var.append(dem_var)
                if t != trans[-1]:
                    form += " + "
                else:
                    form += " = {}".format(dem)
            demands.append(form)
    return sorted(demands), sorted(x_var)


def get_source_trans(start, trans, dest):
    """Calculates the link capacity from Source to Transit Nodes with
    some variable ySiTj to make the equation linear"""
    cap = []
    for s in start:
        for t in trans:
            form = ''
            for d in dest:
                form += "x{}{}{}".format(s, t, d)
                if d != dest[-1]:
```

```python
                form += " + "
            else:
                form += " - y{}{} = 0".format(s, t)
        cap.append(form)
    return sorted(cap)


def get_trans_dest(start, trans, dest):
    """Calculates the link capacity from Transit to Destination Nodes with
    some variable yTiDj to make the equation linear"""
    cap = []
    for t in trans:
        for d in dest:
            form = ''
            for s in start:
                form += "x{}{}{}".format(s, t, d)
                if s != start[-1]:
                    form += " + "
                else:
                    form += " - y{}{} = 0".format(t, d)
            cap.append(form)
    return sorted(cap)


def get_source_const(source_trans):
    """Generate constraint for source nodes"""
    constraints = []
    minimum = []

    for i in source_trans:
        value = i.split(' ')
        form = "{} - c{} <= 0".format(value[-3], value[-3][1:])
        mini = "{} >= 0".format(value[-3])
        constraints.append(form)
        minimum.append(mini)

    return constraints, minimum


def get_trans_const(trans_dest):
    """Generate constraint for destination nodes"""
    constraints = []
    minimum = []

    for i in trans_dest:
        value = i.split(' ')
        form = "{} - d{} <= 0".format(value[-3], value[-3][1:])
        mini = "{} >= 0".format(value[-3])
        constraints.append(form)
        minimum.append(mini)

    return constraints, minimum


def get_constraints(source_trans, trans_dest, x_var, start, trans):
    """Generates constraints"""
    constraints = []
    minimum = []

    source_const = get_source_const(source_trans)
    constraints += source_const[0]
    minimum += source_const[1]

    trans_const = get_trans_const(trans_dest)
    constraints += trans_const[0]
    minimum += trans_const[1]

    # Generates constraint for all Xijk
    for i in x_var:
        form = "{} >= 0".format(i)
        minimum.append(form)

    # Generate r values
    for t in trans:
        form = ''
        for s in start:
            form += "y{}{}".format(s, t)
            if s != start[-1]:
                form += " + "
            else:
```

```python
            form += " - r <= 0".format(s, t)
        constraints.append(form)

    return [constraints, minimum]


def get_binary_path(start, trans, dest):
    """Generates binary path (Default: 3 | Start -> Transit -> Destination)"""
    paths = []
    binaries = []

    # all binary paths when summed, is equal to 3 (S -> T -> D)
    for s in start:
        for d in dest:
            form = ''
            for t in trans:
                var = 'u{}{}{}'.format(s, t, d)
                binaries.append(var)
                if t != trans[-1]:
                    form += '{} + '.format(var)
                else:
                    form += '{} = 3'.format(var)
            paths.append(form)

    # paths for demand volumes
    for s in start:
        for t in trans:
            for d in dest:
                dem = start.index(s) + dest.index(d) + 2 # for incrementing index
                dem_var = "{}{}{}".format(s, t, d)
                form = '3 x{} - {} u{} = 0'.format(dem_var, dem, dem_var)
                paths.append(form)

    return paths, binaries


def get_trans_load(trans, x_var):
    """Calculates the demand for each transit nodes"""
    trans_load = []

    for t in trans:
        form = 'x'
        for var in x_var:
            if t in var:
                form += var
                form += ' + '
        form = form[1:-3]
        form += ' - l{} = 0'.format(t)
        trans_load.append(form)

    return trans_load


def create_lp(demand_volume, source_trans, trans_dest, constraints, minimum,
              binary_path, binaries, trans_load):
    """Generates an LP file based on the generated optimization problem"""
    form = "Minimize\nr\nSubject to\n"

    for i in range(0, len(demand_volume)):
        form += '  {}\n'.format(demand_volume[i])

    for i in range(0, len(source_trans)):
        form += '  {}\n'.format(source_trans[i])

    for i in range(0, len(trans_dest)):
        form += '  {}\n'.format(trans_dest[i])

    for i in range(0, len(constraints)):
        form += '  {}\n'.format(constraints[i])

    for i in range(0, len(binary_path)):
        form += '  {}\n'.format(binary_path[i])

    for i in range(0, len(trans_load)):
        form += '  {}\n'.format(trans_load[i])

    form += 'Bounds\n'

    for i in range(0, len(minimum)):
        form += '  {}\n'.format(minimum[i])
```

```python
    form += '  r >= 0\n'
    form += 'Binary\n'

    for i in range(0, len(binaries)):
        form += '  {}\n'.format(binaries[i])

    form += 'End'

    f = open(filename, 'w')
    f.write(form)
    f.close()


def run_cplex():
    """Executes cplex via python"""

    # CPLEX FULL PATH OF CURRENT MACHINE
    # THIS IS DECLARED ON THE GLOBAL VARIABLES
    #cplex_path = "/home/chaosbib/cplex/cplex/bin/x86-64_linux/cplex"
    #cplex_path = 'cplex'

    cplex = cplex_path + " -c 'read {}'".format(filename)
    cplex += " 'optimize' 'display solution variables -'"

    os.system(cplex)


def main():
    s, t, d = get_input()
    start, trans, dest = create_nodes(s, t, d)
    demand_volume, x_var = get_dem_vol(start, trans, dest)
    source_trans = get_source_trans(start, trans, dest)
    trans_dest = get_trans_dest(start, trans, dest)
    constraints, minimum = get_constraints(source_trans, trans_dest,
                                           x_var, start, trans)
    binary_path, binaries = get_binary_path(start, trans, dest)
    trans_load = get_trans_load(trans, x_var)
    create_lp(demand_volume, source_trans, trans_dest, constraints, minimum,
              binary_path, binaries, trans_load)

    start_time = time.time()
    run_cplex()
    print("\nCPLEX Execution Time: {}".format(time.time() - start_time))


main()
```