COSC 364 – Assignment 2

Isaac Foster – 64555890

Zachary Sanson – 58520526

Team Contribution

50% both ways.

Formulas

Minimize Function:

$Min_{[x,c,d,r]}(r)$

This is our minimization function which we will introduce a new variable r for the utilization of load on each transit node. In doing so we will also need to introduce a new constraint where $r \geq x_{ikj}$ so that r is not smaller than the total load on each transit node.

Demand Volume Constraint:

$$h_{ij} = \sum_{k=1}^{y} x_{ikj} = 2i + j$$

$i \in \{1, \dots, x\}, \qquad j \in \{1, \dots, z\}$

This defines the constraint where the sum of the path flows (x_{ikj}) between the source and destination nodes ($i \rightarrow j$) should add up to the given demand volume (h_{ij}). In our case for each path from $i \rightarrow j$ the demand volume for each path h_{ij} should equal $2i + j$, as defined in our brief.

Capacity Constraint:

Capacity 1:

$$\sum_{j=1}^{z} x_{ikj} \leq c_{ik}$$

$i \in \{1, \dots, x\}, \qquad k \in \{1, \dots, y\}$

Capacity 2:

$$\sum_{i=1}^{x} x_{ikj} \leq d_{kj}$$

$k \in \{1, \dots, y\}, \qquad j \in \{1, \dots, z\}$

Each path flow (x_{ikj}) between the nodes $i \rightarrow k \rightarrow j$ should not exceed the given capacity of a link between; the source nodes → transit nodes (c_{ik}) or transit nodes → destination nodes (d_{kj}). To optimise this, we must get the capacity between these links to be as close to / equal to the demand volume (h_{ij}).

Transit Load Constraint:

$$\sum_{i=1}^{x} \sum_{j=1}^{z} x_{ikj} \leq r$$

$k \in \{1, \dots, y\}$

The sum of the flow over each transit node x_{ikj} must not be greater that our minimisation function (r).

Binary Variable Constraint:

$$n_{ij} = \sum_{k=1}^{y} u_{ikj} = 2$$

$i \in \{1, \dots, x\}, \quad j \in \{1, \dots, z\}$

Our problem describes that each demand volume must be split over exactly two different paths. To do so we check whether the demand flow is used over a given path or node. This is expressed by u_{ikj} where $u_{ikj} = 1$ means that the path between $i \to j$ is already used and $u_{ikj} = 0$ meaning the path isn't used.

Path Flow:

$$x_{ikj} = \frac{u_{ikj} * h_{ij}}{n_{ij}}$$

$i \in \{1, \dots, x\}, \quad k \in \{1, \dots, y\}, \quad j \in \{1, \dots, z\}$

This is where we define an equal split of our load over our paths from the source node to destination node. For each path between the nodes $i \to j$, the demand volume h_{ij} must be equally split into exactly <u>two</u> path flows ($n_{ij} = 2$). It is important to note that h_{ij} must equal $2i + j$ like stated in our brief.

Bounds:

$x_{ikj} \geq 0$

$i \in \{1, \dots, x\}, \quad k \in \{1, \dots, y\}, j \in \{1, \dots, z\}$

We require that all variables that have been introduced to meet the non-negativity constraints. To do so the path flows from $i \to j$ must be non-negative and must satisfy $x_{ijk} \geq 0$.

Binaries:

$u_{ikj} \in \{0, 1\}$

$i \in \{1, \dots, x\}, \quad k \in \{1, \dots, y\}, \quad j \in \{1, \dots, z\}$

Binaries indicate if a path for the demand volume h_{ij} is used, and $u_{ikj} = 1$ if the path is used otherwise $u_{ikj} = 0$.

The Problem

The problem as described in the brief is about a traffic system layout, we have a set amount of; 'Source Nodes' defined as x, 'Transit Nodes' defined as y and 'Destination Nodes' defined as z.

We want to aim to balance the load (the total amount of incoming traffic flow) that runs from the source nodes to the destination nodes (through the transit nodes). Our deviation of this problem specifies that each source node must have <u>exactly</u> two paths connecting to their destination nodes. *("Each demand volume must be split over <u>two</u> different paths.")* From our outputted CPLEX file we must analyse it and determine the load the transit nodes, capacities of all links and the value of each flow.

Data

Note: To create consistent and reliable result we have run these tests over ten separate attempts to rule out the variance in execution time (due to other programs on the system running at the same time).

No. transit nodes	**3**	**4**	**5**	**6**	**7**	**8**
Average Execution Time (Seconds) (Below is every attempt from 1-10)	0.065 0.04 0.04 0.03 0.04 0.04 0.09 0.13 0.10 0.06 0.08	0.086 0.05 0.05 0.11 0.12 0.11 0.13 0.12 0.05 0.07 0.05	0.124 0.08 0.09 0.08 0.08 0.12 0.14 0.17 0.15 0.17 0.16	0.182 0.11 0.21 0.20 0.21 0.20 0.20 0.18 0.16 0.17 0.18	0.204 0.12 0.17 0.25 0.23 0.23 0.23 0.22 0.21 0.19 0.19	0.230 0.19 0.22 0.22 0.22 0.22 0.18 0.23 0.24 0.33 0.25
No. links with a non-zero capacity (Below the links with a non-zero capacity)	52 c11 c13 c21 c23 c31 c32 c33 c41 c42 c43 c51 c52 c53 c61 c62 c63 c71 c72 c73 c81 c82 c83 c91 c92 c93 d11 d12 d13 d14 d15 d16 d17 d18 d19 d21 d22 d23 d24 d25 d26 d27 d28 d29 d31 d32 d33 d34 d35 d36 d37 d38 d39	67 c11 c14 c21 c22 c24 c31 c32 c34 c41 c42 c43 c44 c51 c52 c53 c54 c61 c62 c63 c64 c71 c72 c73 c74 c81 c82 c83 c84 c91 c92 c93 c94 d11 d12 d13 d14 d15 d16 d17 d18 d19 d21 d22 d23 d24 d25 d26 d27 d28 d29 d32 d33 d34 d35 d36 d37 d38 d39 d41 d42 d43 d44 d45 d46 d47 d48 d49	80 c11 c12 c13 c15 c21 c22 c25 c31 c32 c35 c41 c42 c44 c45 c51 c52 c53 c54 c55 c61 c62 c63 c64 c65 c71 c72 c73 c74 c75 c81 c82 c83 c84 c85 c92 c93 c94 d11 d12 d13 d14 d15 d16 d17 d18 d19 d21 d22 d23 d24 d25 d26 d27 d28 d29 d32 d33 d34 d35 d36 d37 d38 d39 d41 d43 d44 d45 d46 d47 d48 d49 d51 d52 d53 d54 d55 d56 d57 d58 d59	106 c11 c13 c14 c15 c16 c21 c22 c23 c24 c25 c31 c32 c33 c34 c35 c36 c41 c42 c43 c44 c45 c46 c51 c52 c53 c54 c55 c56 c61 c62 c63 c64 c65 c66 c71 c72 c73 c74 c75 c76 c81 c82 c83 c84 c85 c86 c91 c92 c93 c94 c95 c96 d11 d12 d13 d14 d15 d16 d17 d18 d19 d21 d22 d23 d24 d25 d26 d27 d28 d29 d31 d32 d33 d34 d35 d36 d37 d38 d39 d41 d42 d43 d44 d45 d46 d47 d48 d49 d51 d52 d53 d54 d55 d56 d57 d58 d59 d61 d62 d63 d64 d65 d66 d67 d68 d69	118 c11 c12 c13 c15 c16 c17 c21 c22 c23 c24 c25 c26 c27 c31 c32 c33 c34 c35 c36 c37 c41 c42 c43 c44 c45 c46 c47 c51 c52 c53 c54 c55 c56 c57 c61 c62 c63 c64 c65 c66 c67 c72 c73 c74 c75 c76 c77 c81 c83 c84 c85 c86 c87 c92 c93 c94 c95 c96 c97 d11 d12 d13 d14 d15 d16 d17 d18 d19 d21 d22 d23 d24 d25 d26 d27 d28 d29 d31 d33 d35 d36 d37 d38 d39 d42 d43 d44 d45 d46 d47 d48 d49 d52 d53 d54 d55 d56 d57 d58 d59 d61 d62 d63 d64 d65 d66 d67 d68 d69 d71 d72 d73 d74 d75 d76 d77 d78 d79	131 c11 c12 c13 c14 c16 c17 c18 c21 c22 c23 c24 c25 c27 c31 c32 c33 c34 c36 c37 c38 c41 c42 c43 c44 c45 c46 c47 c48 c51 c52 c54 c55 c56 c57 c58 c61 c62 c63 c64 c66 c67 c68 c71 c73 c74 c75 c76 c77 c78 c82 c83 c84 c85 c87 c88 c91 c93 c94 c95 c96 c97 c98 d11 d12 d13 d14 d15 d16 d17 d18 d19 d21 d22 d23 d24 d25 d26 d27 d28 d29 d31 d32 d33 d34 d36 d37 d38 d39 d41 d42 d43 d44 d45 d46 d47 d48 d49 d51 d52 d53 d54 d55 d56 d58 d59 d61 d62 d63 d64 d65 d66 d67 d68 d69 d71 d72 d73 d74 d75 d76 d77 d78 d79 d81 d82 d83 d84 d85 d86 d87 d89
No. links with highest capacity	1	1	2	1	1	1
Links with highest capacity (Below is the capacity value)	c92 (104.0)	c93 (94.0)	c93 c94 (94.0)	c96 (60.0)	c83 (65.0)	c95 (68.0)

Load on Transit Nodes (Transit Node No. **:** Load on Node)	1: 405.0 2: 405.0 3: 405.0	1: 304.0 2: 303.5 3: 304.0 4: 303.5	1: 243.0 2: 243.0 3: 243.0 4: 243.0 5: 243.0	1: 202.5 2: 202.5 3: 202.5 4: 202.5 5: 202.5 6: 202.5	1: 173.5 2: 173.0 3: 174.0 4: 174.0 5: 174.0 6: 174.0 7: 172.5	1: 152.0 2: 152.0 3: 152.0 4: 152.0 5: 152.0 6: 152.0 7: 151.0 8: 152.0

Results

Our analysed data above contains information about execution times, nodes with a non-zero capacity, nodes with the highest capacity and the load on the transit nodes. We run our optimisation problem with a varying amount of transit nodes from $y = 3 \dots 8$ and run our tests over ten attempts to minimise outliers. To understand these results we will run over each section.

Firstly we have the execution time which is the amount of time that it took CPLEX to optimise our '.LP' file (which contains a problem to optimise). From our results we can see that with a minimal amount of transit nodes ($y = 3$) that the execution time is relatively low, running in 0.065 seconds. Although as our problem includes more transit nodes, the more complex our problem becomes and therefore taking longer for CPLEX to optimise our problem. We can see this when we set the number of transit nodes to a higher number. E.g. $y = 7$, we can see that with seven transit nodes it takes CPLEX 0.204 seconds which is drastically more than having three nodes.

Secondly we have the non-zero capacities, this relates to the amount of links that are currently being used (if a link has a capacity of zero, it is not being used). In our data we can see that with a low amount of transit nodes we also have a low amount of non-zero capacity links. Although as we increase the number of transit nodes in our system we effectively balance out our traffic over more nodes, therefore making the amount of non-zero capacities increase. *(We have also included the name of the capacities underneath.)*

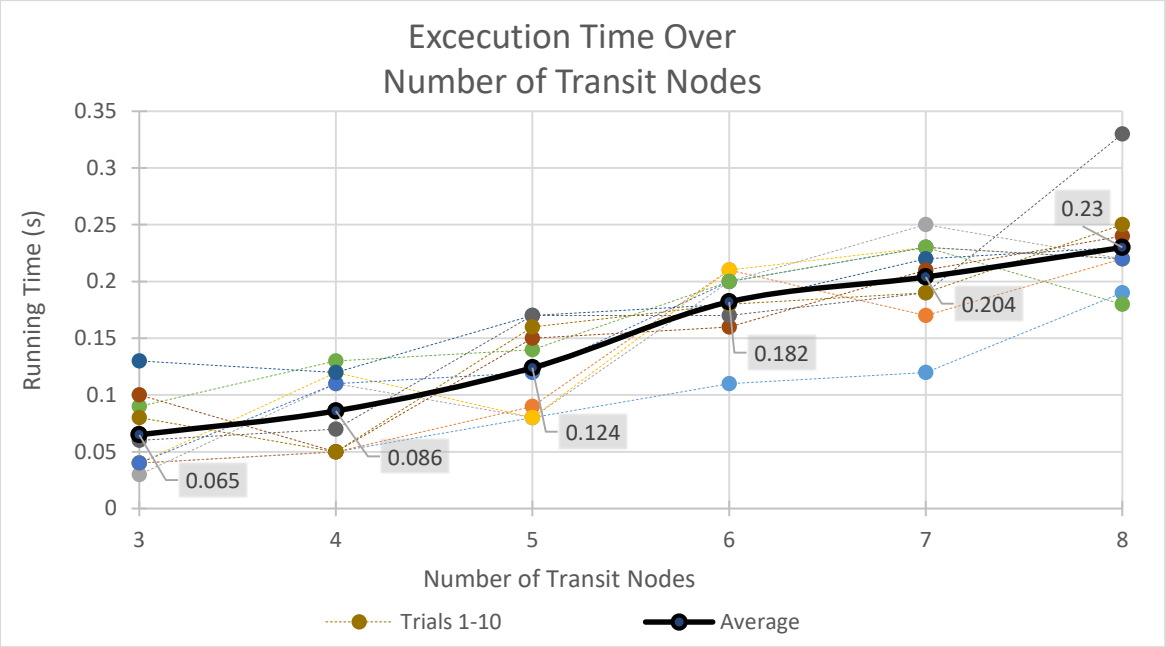
Thirdly we have the links which are the highest capacity in the system. We can see here that the capacity of the links is relatively high when there is a low amount of transit nodes, much like what happened with the non-zero capacity links. When we increase the amount of transit nodes, you will find that the highest capacity of all the links has actually decreased numerically. Since the load of our system is now balanced over more transit nodes, meaning that the links capacities aren't as high anymore.

The final section we have is the load on the transit nodes and their corresponding values. The average load on our transit nodes should be decreasing as we add more transit nodes to our problem and we can actually see our data mimicking this behaviour. As we have a small amount of transit nodes the load on all transit nodes is quite high, up at 405 units. Although as we increase the number of transit nodes our load is spread out over more units making the average load and the singular load on each transit node lower.

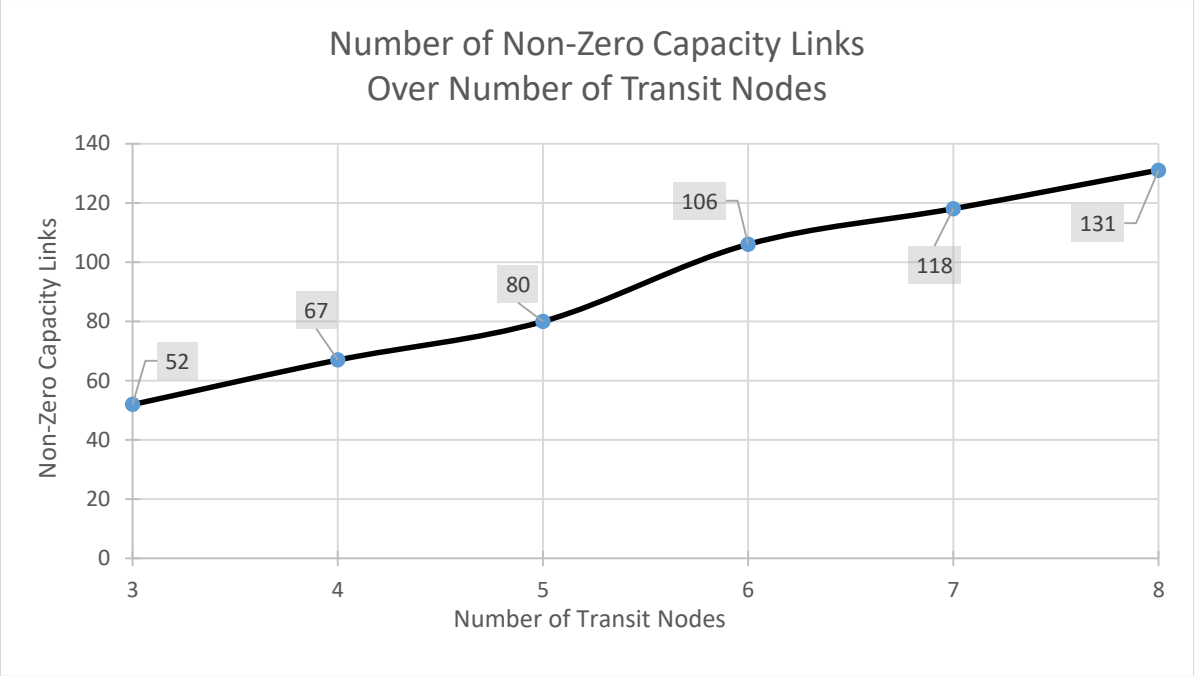
Note: In the brief we are told to show & explain the difference/spread between the transit node loads. Although since the loads are balanced evenly over every node (for all amounts of transit nodes, $y = 3 \dots 8$) so there is not much spread. *(Our transit load data doesn't contain 0 value loads.)* Here is a brief summary of the differences between the nodes if still required:

$y = 3$	$\min\{T1 \dots T3\} - \max\{T1 \dots T3\}$	$405 - 405 = 0$
$y = 4$	$\min\{T1 \dots T4\} - \max\{T1 \dots T4\}$	$304 - 303.5 = 0.5$
$y = 5$	$\min\{T1 \dots T5\} - \max\{T1 \dots T5\}$	$243 - 243 = 0$
$y = 6$	$\min\{T1 \dots T6\} - \max\{T1 \dots T6\}$	$202.5 - 202.5 = 0$
$y = 7$	$\min\{T1 \dots T7\} - \max\{T1 \dots T7\}$	$174 - 172.5 = 1.5$
$y = 8$	$\min\{T1 \dots T8\} - \max\{T1 \dots T8\}$	$152 - 152 = 0$

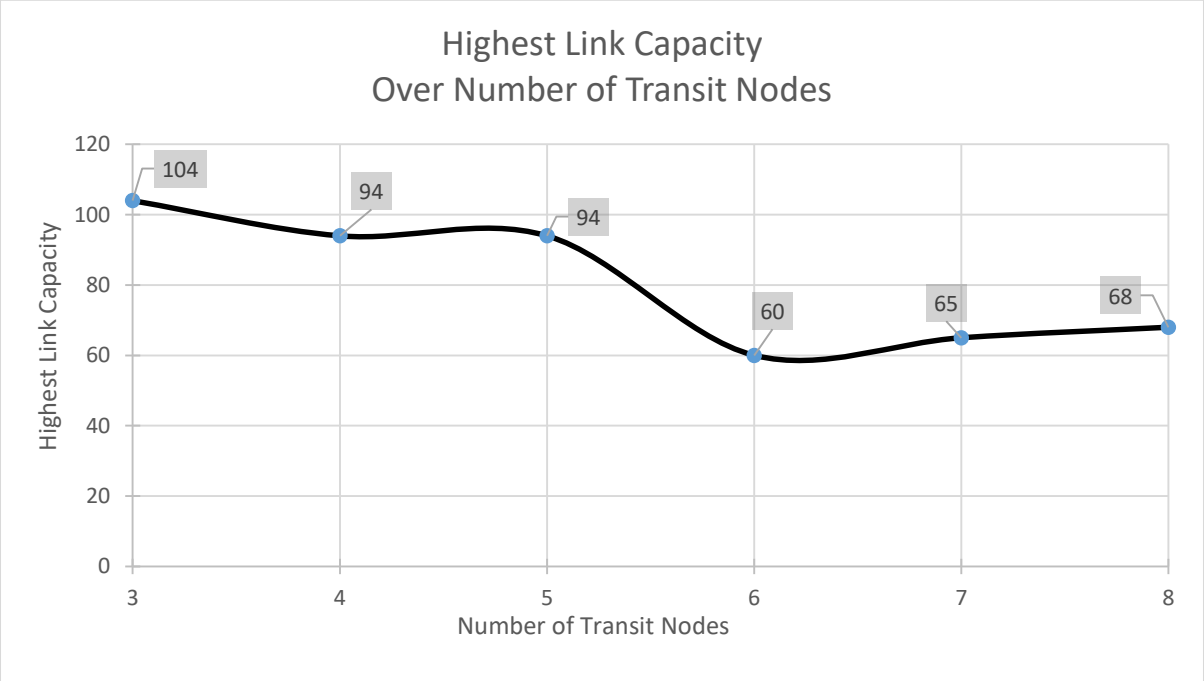
Graphs



This graph shows how the running time of optimising a solution increases as you increase the amount of transit nodes. As you add in more transit nodes, the CPLEX analyser must calculate more paths between the source and destination. By doing so we are making the problem that CPLEX has to solve more complex and therefore increasing the execution time. We can see this reflect in the graph above where we start with $x = 9,\ y = 3,\ z = 9$ (where x is the number of source nodes, y is the number of transit nodes and z is the number of destination nodes) with a low execution time of 0.065 seconds. Although as we increase the amount of transit nodes ($y \to 3 \dots 8$) the execution time will increase also. E.g. with $y = 7$ we have an execution time of 0.204.

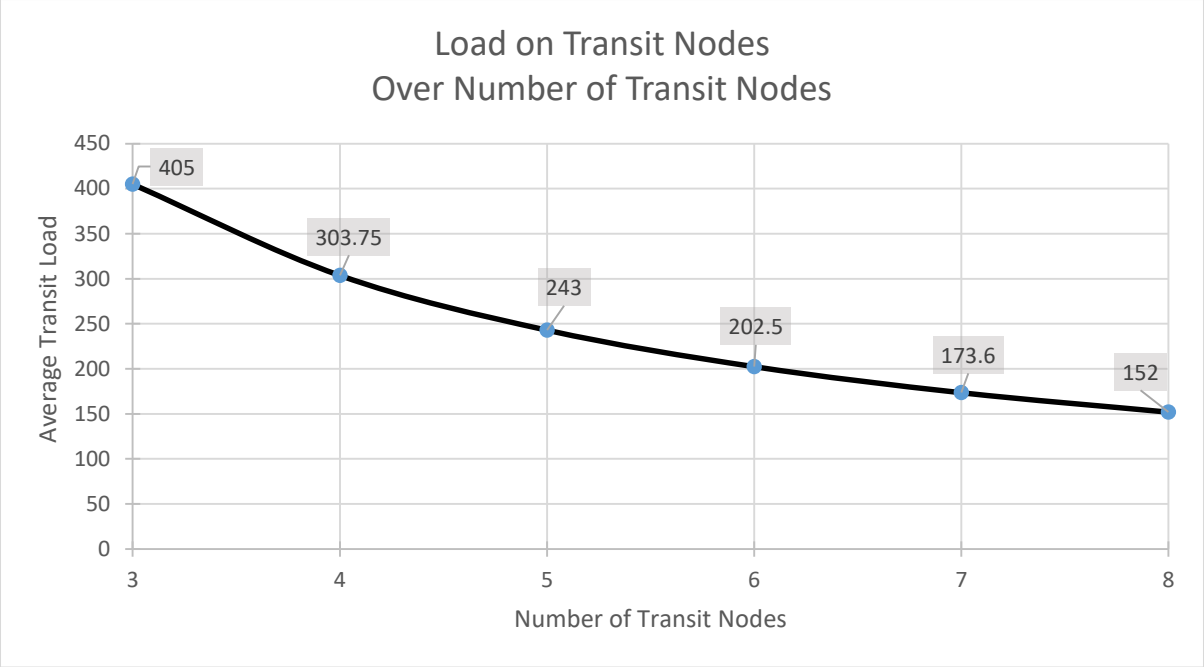


Non-zero capacity is definitely following the trend that as we increase the number of transit nodes, the number of non-zero capacities increases. This is due to the increase in links as you increase the number of transit node, since more links are being created and utilized.



We can see on the graph that the capacity of the links is relatively high with a value of 104 units when there is a low amount of transit nodes. This will be due to the load not being spread over a large amount of nodes and therefore the capacity of each node requires a higher value. When we increase the amount of transit nodes we therefore spread the load over more paths, thus the required capacity per link has decreased. E.g. when $y = 7$, we get a capacity of 65 units.

This data originally trends towards a lower capacity per node increase (which we expected), although the trend changes direction past six transit nodes (heads towards a 65-68 trend), this could be due to an outlier or just general trend change.



This graphs shows what the average load spread over the number of transit node. In the beginning we only start with three transit nodes ($y = 3$) and nine source & destination nodes. This means that for those nine source nodes, they have to communicate through three transit nodes. This of course increases the stress and load on those transit nodes. As we increase the number of transit nodes in our system ($y \rightarrow 3 \dots 8$) the average load decreases. This is due to the fact that more transit nodes connecting the source and destination together in terms means that they can share the load more efficiently and with lower load costs. Since there are more possible paths to allocate to the source nodes (which are unique). Looking at the graph we can definitely see this happening; when we only have three transit nodes the average load of our system is at 405 units, whereas when we increase the number to eight transit nodes the load is spread more evenly across those nodes, giving us a value of 152 units.

Appendix 1 – Source Code

```
#!/usr/bin/env python3
"""
main.py
CPLEX LP file creator
Isaac Foster, Zachary Sanson
"""

import sys

FILENAME = "Path"
PATH_SPLIT = 2


def demand_eq(i, j): return 2 * i + j


def create_LP(x, y, z):
    """
    :param x: Number of total source nodes
    :param y: Number of total transit nodes
    :param z: Number of total destination nodes
    Creates a CPLEX LP file by running through all the sub-functions to create
    the required calculations, constraints, capacities, etc.
    """
    with(open((FILENAME + str(y) + ".lp"), "w+")) as file:
        file.write("Minimize\n"
                   "\tr\n"
                   "Subject to\n"
                   "\tDemand Volume:{0}\n"
                   "\tCapacity ST:{1}\n"
                   "\tCapacity TD:{2}\n"
                   "\tTransit nodes:{3}\n"
                   "\tBinary Variables:{4}\n"
                   "\tDemand Flow:{5}\n"
                   "Bounds{6}\n"
                   "Binaries{7}\n"
                   "End".format(demand_volume(x, y, z),
                                source_transit_capacity(x, y, z),
                                transit_destination_capacity(x, y, z),
                                transit_nodes(x, y, z),
                                binary_variables(x, y, z),
                                demand_flow(x, y, z),
                                bounds(x, y, z),
                                binary(x, y, z)))


def demand_volume(x, y, z):
    """
    :param x: Number of total source nodes
    :param y: Number of total transit nodes
    :param z: Number of total destination nodes
    :return: String of demand volume constraints
    Function to return the demand volume constraint.
    That is the sum of the load between source i and Destination j:
    expected output:  x=1,y=3,z=3
                      (xikj = hij)
                      x111 + x121 + x131 = 2
                      x112 + x122 + x132 = 3
                      x113 + x123 + x133 = 4
```

```
                    ....
    """
    string = ""
    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{0}{1}{2}".format(i, k, j))
            string += "\n\t\t" + " + ".join(dv) + " = {0}".format(demand_eq(i, j))
    return string


def transit_nodes(x, y, z):
    """
    :param x: Number of total source nodes
    :param y: Number of total transit nodes
    :param z: Number of total destination nodes
    :return: String of transit node constraints
    """
    string = ""
    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{0}{1}{2}".format(i, k, j))
        string += "\n\t\t" + " + ".join(tn) + " - r <= 0"
    return string


def source_transit_capacity(x, y, z):
    """
    :param x: Number of total source nodes
    :param y: Number of total transit nodes
    :param z: Number of total destination nodes
    :return: String of source to transit capacities
    """
    string = ""
    for i in range(1, x + 1):
        for k in range(1, y + 1):
            st_cap = []
            for j in range(1, z + 1):
                st_cap.append("x{0}{1}{2}".format(i, k, j))
            string += "\n\t\t" + " + ".join(st_cap) + \
                      " - c{0}{1} <= 0".format(i, k)
    return string


def transit_destination_capacity(x, y, z):
    """
    :param x: Number of total source nodes
    :param y: Number of total transit nodes
    :param z: Number of total destination nodes
    :return: String of transit to destination capacities
    """
    string = ""
    for k in range(1, y + 1):
        for j in range(1, z + 1):
            td_cap = []
            # string += "\n\t\t"
            for i in range(1, x + 1):
                td_cap.append("x{0}{1}{2}".format(i, k, j))
```

```
            string += "\n\t\t" + " + ".join(td_cap) + \
                      " - d{0}{1} <= 0".format(k, j)
    return string


def binary_variables(x, y, z):
    """
    :param x: Number of total source nodes
    :param y: Number of total transit nodes
    :param z: Number of total destination nodes
    :return: String of binary variables
    Check whether the demand flow is used over a given path or not.
    Path will equal 0 if not used, or 1 if the path is loaded.
    """
    string = ""
    for i in range(1, x + 1):
        for j in range(1, z + 1):
            bv_cap = []
            for k in range(1, y + 1):
                bv_cap.append("u{0}{1}{2}".format(i, k, j))
            string += "\n\t\t" + " + ".join(bv_cap) + " = {0}".format(PATH_SPLIT)
    return string


def demand_flow(x, y, z):
    """
    :param x: Number of total source nodes
    :param y: Number of total transit nodes
    :param z: Number of total destination nodes
    :return: String of demand flow constraints
    """
    string = ""
    for i in range(1, x + 1):
        for j in range(1, z + 1):
            for k in range(1, y + 1):
                string += "\n\t\t{0} x{1}{2}{3} - {4} u{5}{6}{7} = 0".format(
                    PATH_SPLIT, i, k, j, demand_eq(i, j), i, k, j)
    return string


def bounds(x, y, z):
    """
    :param x: Number of total source nodes
    :param y: Number of total transit nodes
    :param z: Number of total destination nodes
    :return: String of bounds
    Constraint to ensure all variables that have been introduced
    meet the non-negativity constraints.
    """
    string = ""
    for i in range(1, x + 1):
        for j in range(1, x + 1):
            for k in range(1, y + 1):
                string += "\n\t0 <= x{0}{1}{2}".format(i, k, j)
    return string + "\n\t0 <= r"


def binary(x, y, z):
    """
    :param x: Number of total source nodes
    :param y: Number of total transit nodes
```

```
    :param z: Number of total destination nodes
    :return: String of node combinations
    Creates a string of all possible paths.
    """
    string = ""
    for i in range(1, x + 1):
        for k in range(1, y + 1):
            for j in range(1, z + 1):
                string += "\n\tu{0}{1}{2}".format(i, k, j)
    return string


# Main function for which the program initially starts from
if __name__ == '__main__':
    # x, y, z = map(int, sys.argv[1:])    #Use if starting with terminal parameters
    for y in range(3, 9):
        create_LP(9, y, 9)
```

Appendix 2 – Sample LP file

($x = 3,\ y = 2,\ z = 4$ as mentioned in brief)

```
Minimize
   r
Subject to
   Demand Volume:
      x111 + x121 = 3
      x112 + x122 = 4
      x113 + x123 = 5
      x114 + x124 = 6
      x211 + x221 = 5
      x212 + x222 = 6
      x213 + x223 = 7
      x214 + x224 = 8
      x311 + x321 = 7
      x312 + x322 = 8
      x313 + x323 = 9
      x314 + x324 = 10
   Capacity ST:
      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
   Capacity TD:
      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 = 2
      u112 + u122 = 2
      u113 + u123 = 2
      u114 + u124 = 2
      u211 + u221 = 2
      u212 + u222 = 2
      u213 + u223 = 2
      u214 + u224 = 2
      u311 + u321 = 2
      u312 + u322 = 2
      u313 + u323 = 2
      u314 + u324 = 2
   Demand Flow:
      2 x111 - 3 u111 = 0
      2 x121 - 3 u121 = 0
      2 x112 - 4 u112 = 0
      2 x122 - 4 u122 = 0
      2 x113 - 5 u113 = 0
      2 x123 - 5 u123 = 0
```

```
      2 x114 - 6 u114 = 0
      2 x124 - 6 u124 = 0
      2 x211 - 5 u211 = 0
      2 x221 - 5 u221 = 0
      2 x212 - 6 u212 = 0
      2 x222 - 6 u222 = 0
      2 x213 - 7 u213 = 0
      2 x223 - 7 u223 = 0
      2 x214 - 8 u214 = 0
      2 x224 - 8 u224 = 0
      2 x311 - 7 u311 = 0
      2 x321 - 7 u321 = 0
      2 x312 - 8 u312 = 0
      2 x322 - 8 u322 = 0
      2 x313 - 9 u313 = 0
      2 x323 - 9 u323 = 0
      2 x314 - 10 u314 = 0
      2 x324 - 10 u324 = 0
Bounds
   0 <= x111
   0 <= x121
   0 <= x112
   0 <= x122
   0 <= x113
   0 <= x123
   0 <= x211
   0 <= x221
   0 <= x212
   0 <= x222
   0 <= x213
   0 <= x223
   0 <= x311
   0 <= x321
   0 <= x312
   0 <= x322
   0 <= x313
   0 <= x323
   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

Plagiarism Declaration

This form needs to accompany your COSC 364 assignment submission.

I understand that plagiarism means taking someone else's work (text, program code, ideas, concepts) and presenting them as my own, without proper attribution. Taking someone else's work can include verbatim copying of text, figures/images, or program code, or it can refer to the extensive use of someone else's original ideas, algorithms or concepts.

I hereby declare that:

- My assignment is my own original work. I have not reproduced or modified code, figures/images, or writings of others without proper attribution. I have not used original ideas and concepts of others and presented them as my own.
- I have not allowed others to copy or modify my own code, figures/images, or writings. I have not allowed others to use original ideas and concepts of mine and present them as their own.
- I accept that plagiarism can lead to consequences, which can include partial or total loss of marks, no grade being awarded and other serious consequences, including notification of the University Proctor.

Name: Zachary Sanson

Student ID: 85520526

Signature: ZachSanson

Date: 31/05/19

Plagiarism Declaration

This form needs to accompany your COSC 364 assignment submission.

I understand that plagiarism means taking someone else's work (text, program code, ideas, concepts) and presenting them as my own, without proper attribution. Taking someone else's work can include verbatim copying of text, figures/images, or program code, or it can refer to the extensive use of someone else's original ideas, algorithms or concepts.

I hereby declare that:

- My assignment is my own original work. I have not reproduced or modified code, figures/images, or writings of others without proper attribution. I have not used original ideas and concepts of others and presented them as my own.
- I have not allowed others to copy or modify my own code, figures/images, or writings. I have not allowed others to use original ideas and concepts of mine and present them as their own.
- I accept that plagiarism can lead to consequences, which can include partial or total loss of marks, no grade being awarded and other serious consequences, including notification of the University Proctor.

Name: Isaac Foster

Student ID: 64555890

Signature: 

Date: 31/05/19