DISCRETE MATHEMATICS

Kamran Muhammad Bhatti
3807942

Contents

Problem Statement

There is a transport network laid over 6 major cities ($C_1 \ldots C_6$) in a country.

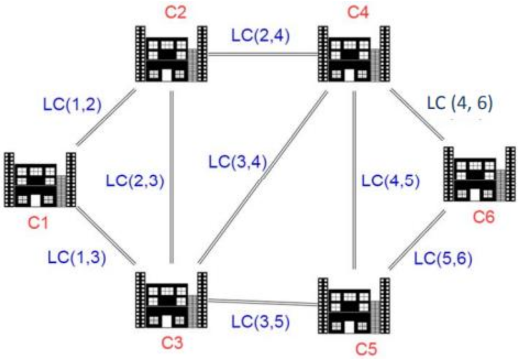


Figure 1

A logistics company transports two shipments (S_1, S_2) from C_1 to C_6. The logistic cost of them from C_i to C_j is denoted by LC (I, j) is subject to the individual transportation cost of the products:

Logistic cost between adjacent cities	Shipment 1	Shipment 2
1,2	6	7
1,3	3	2
2,3	2	3
2,4	5	8
3,4	3	8
3,5	4	3
4,5	4	5
4,6	6	3
5,6	5	9

Table 1 Logistics Cost between Adjacent Cities

Figure 2

The shipment carries two products (P_1, P_2), due to various transport conditions and customs rules each connecting road has constraints on carrying a specific number of units of each product per shipment. Also, maximum capacity of products is also capped:

Road	Product	Shipment 1	Shipment 2	Max Load
1,2	Product 1	1	1	20
	Product 2	2	3	6
1,3	Product 1	1	1	10
	Product 2	2	3	6
2,3	Product 1	1	1	40
	Product 2	3	8	12
2,4	Product 1	1	1	15
	Product 2	5	2	10
3,4	Product 1	1	1	5
	Product 2	5	10	15
3,5	Product 1	1	1	20
	Product 2	4	2	8
4,5	Product 1	1	1	5
	Product 2	2	3	6
4,6	Product 1	1	1	10
	Product 2	6	3	9
5,6	Product 1	1	1	5
	Product 2	4	2	8

Figure 3

Problem: The shipment company wants to calculate a probability density function for the subjected transportation problem by mapping the optimistic (best) and pessimistic (worst) cost of transport through all possible ways between C_1 and C_6.

What can I deduce from the problem?

Each city and each connection between cities correspond to vertices and edges of a graph respectively.

Characteristics

The graph G(V,E) has vertex set V ={(C_1, C_2, C_3, C_4, C_5, C_6} and edge set E = {LC(1,2), LC(1,3), LC(2,3), LC(2,4), LC(3,4), LC(3,5), LC(4,5), LC(4,6), LC(5,6)}. The vertices with the largest degree would be C_3 and C_4 with degree of 4. Additionally, the graph has a Hamiltonian circuit in which you can traverse every vertex exactly once before reaching the starting vertex. Furthermore, I can determine that the graph is planar as none of the edges overlap or cross.

The graph has subgraphs of C_3 or K_3 between nodes: C_1, C_2, C_3 or C_2, C_3, C_4 or C_3, C_4, C_5 or C_4, C_5, C_6. Additionally, the subgraph C_4 can be seen in nodes: C_1, C_2, C_3, C_4 or C_2, C_3, C_4, C_5 or C_3, C_4, C_5, C_6. Finally, the largest subgraph that can be seen is C_6 which contains all nodes when following the outer edges.

Chromatic Number

The subgraphs of C_4 and C_6 infer that the graph will have a minimum chromatic number of 2 but as the graph contains a subgraph of C_3, the graph will have a minimum chromatic number of 3 which is valid for the given graph. Therefore $\chi(C_n)$ when n is even makes the chromatic number 2, when n is odd the chromatic number is 3. $\chi(G)$ is 3.

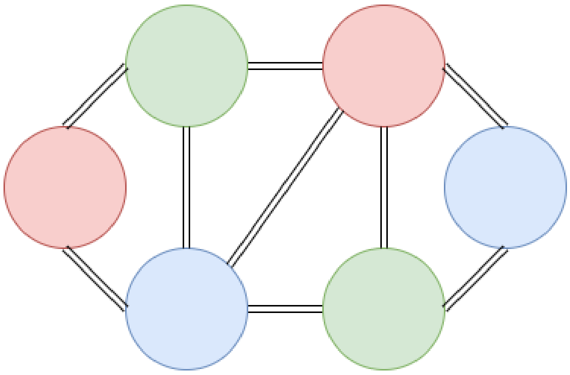


Figure 4

Rank Nullity Theorem

The rank of the graph is 6 nodes – 1 component = 5. The nullity of the graph is 5 regions – 6 nodes + 1 components = 0.

Adjacency matrix

	C_1	C_2	C_3	C_4	C_5	C_6
C_1	0	1	1	0	0	0
C_2	1	0	1	1	0	0
C_3	1	1	0	1	1	0
C_4	0	1	1	0	1	1
C_5	0	0	1	1	0	1
C_6	0	0	0	1	1	0

Incidence matrix

	E_1	E_2	E_3	E_4	E_5	E_6	E_7	E_8	E_9
C_1	1	1	0	0	0	0	0	0	0
C_2	1	0	1	1	0	0	0	0	0
C_3	0	1	1	0	1	1	0	0	0
C_4	0	0	0	1	1	0	1	1	0
C_5	0	0	0	0	0	1	1	0	1
C_6	0	0	0	0	0	0	0	1	1

How will I approach the problem?

The logistic cost between adjacent cities illustrates a problem which can be represented with the use of linear programming in the form of a minimization problem in order to keep the cost of travelling between cities low. However, the condition of the problem is to find the optimistic and pessimistic cost of transport therefore I will be finding the maximization and minimization of the cost of transport between all edges in the graph. The decision variables for both problems will be shipment 1 and shipment 2. The constraints for both problems will be the products on each of the edges between the towns.

Finally, to find the definitive answer to the maximization and minimization problems, I will have to use graph theory in order to find all paths from C_1 to C_6. The optimistic and pessimistic cost of transport will be determined by evaluating the cost of each path from C_1 to C_6. In regards to the probability density function, there will be a minimum of 2 distribution figures to depict the minimization of all the edges and the maximization of all the edges when depicting the graph for all of the paths in the graph.

Furthermore, I will use cartesian product for each of the paths to find the cost of all possible scenarios of the transportation problem, then I will make a probability density function for the undirected and directed graph as I have done just for the maximization of the paths. All of the code I will be using will be uploaded to my GitHub page with the appropriate csv files associated with the programs I have written, (Bhatti, 2019)

LPP

In this section I will carry out the linear programming problem for each of the individual edges. S_1 is shipment 1 represented as X and S_2 is shipment 2 represented as Y. The first two rows after the objective function for each of the edges will represent product 1 and product 2 being P_1 and P_2 respectively.

In this situation maximization of the linear programming problems will show the most pessimistic approach by giving the highest cost each edge for the journey, in comparison minimization of the problems will depict an optimistic approach by giving the lowest cost for each edge on the journey.

General form

$Max/Min\ Z = S1 + S2$

$subject\ to$:

$P1\ constraint$

$P2\ constraint$

$x, y \geq 0$

Program(lpp.py)

```
import numpy
from matplotlib import pyplot
from matplotlib.path import Path
from matplotlib.patches import PathPatch
# modules used to represent the graphs

fig, axis = pyplot.subplots(figsize=(4.5, 4))
x1 = numpy.linspace(0, 40)

# function definitions for each edge


def edge1():
    # bounds represents boundary of feasible region
    bounds = Path([
        (0., 0.),
        (0., 2.),
        (3., 0.),
        (0., 0.)])
    return (20 - x1), ((6 - 2 * x1) / 3), bounds, 20
    # returns constraint 1, constrain 2, boundary of feasible region and limit2 of axis


def edge2():
    # bounds represents boundary of feasible region
    bounds = Path([
        (0., 0.),
        (0., 2.),
        (3., 0.),
        (0., 0.)])
    return (10 - x1), ((6 - 2 * x1) / 3), bounds, 10
    # returns constraint 1, constrain 2, boundary of feasible region and limit2 of axis


def edge3():
    # bounds represents boundary of feasible region
    bounds = Path([
        (0., 0.),
        (0., 1.5),
        (4., 0.),
        (0., 0.)])
    return (40 - x1), ((12 - 3 * x1) / 8), bounds, 40
    # returns constraint 1, constrain 2, boundary of feasible region and limit2 of axis
```

```python
def edge4():
    # bounds represents boundary of feasible region
    bounds = Path([
        (0., 0.),
        (0., 5.),
        (2., 0.),
        (0., 0.)])
    return (15 - x1), ((10 - 5 * x1) / 2), bounds, 15
    # returns constraint 1, constrain 2, boundary of feasible region and limit2 of axis


def edge5():
    # bounds represents boundary of feasible region
    bounds = Path([
        (0., 0.),
        (0., 1.5),
        (3., 0.),
        (0., 0.)])
    return (5 - x1), ((15 - 5 * x1) / 10), bounds, 6
    # returns constraint 1, constrain 2, boundary of feasible region and limit2 of axis


def edge6():
    # bounds represents boundary of feasible region
    bounds = Path([
        (0., 0.),
        (0., 4.),
        (2., 0.),
        (0., 0.)])
    return (20 - x1), ((8 - 4 * x1) / 2), bounds, 20
    # returns constraint 1, constrain 2, boundary of feasible region and limit2 of axis


def edge7():
    # bounds represents boundary of feasible region
    bounds = Path([
        (0., 0.),
        (0., 2.),
        (3., 0.),
        (0., 0.)])
    return (5 - x1), ((6 - 2 * x1) / 3), bounds, 5
    # returns constraint 1, constrain 2, boundary of feasible region and limit2 of axis
```

```
def edge8():
    # bounds represents boundary of feasible region
    bounds = Path([
        (0., 0.),
        (0., 3.),
        (1.5, 0.),
        (0., 0.)])
    return (10 - x1), ((9 - 6 * x1) / 3), bounds, 10
    # returns constraint 1, constrain 2, boundary of feasible region and limit2 of axis


def edge9():
    # bounds represents boundary of feasible region
    bounds = Path([
        (0., 0.),
        (0., 4.),
        (2., 0.),
        (0., 0.)])
    return (5 - x1), ((8 - 4 * x1) / 2), bounds, 5
    # returns constraint 1, constrain 2, boundary of feasible region and limit2 of axis


edge = int(input('''Enter which edge you would like to see the graph of
1.  LC(1,2)
2.  LC(1,3)
3.  LC(2,3)
4.  LC(2,4)
5.  LC(3,4)
6.  LC(3,5)
7.  LC(4,5)
8.  LC(4,6)
9.  LC(5,6)
'''))
# input to represent each edge in the graph

# if an edge is selected the returned values are assigned to their respective parameters
if edge == 1:
    p1_constraint, p2_constraint, bounds, limit = edge1()
elif edge == 2:
    p1_constraint, p2_constraint, bounds, limit = edge2()
elif edge == 3:
    p1_constraint, p2_constraint, bounds, limit = edge3()
elif edge == 4:
    p1_constraint, p2_constraint, bounds, limit = edge4()
```

```
elif edge == 5:
    p1_constraint, p2_constraint, bounds, limit = edge5()
elif edge == 6:
    p1_constraint, p2_constraint, bounds, limit = edge6()
elif edge == 7:
    p1_constraint, p2_constraint, bounds, limit = edge7()
elif edge == 8:
    p1_constraint, p2_constraint, bounds, limit = edge8()
elif edge == 9:
    p1_constraint, p2_constraint, bounds, limit = edge9()

# plot line for p1 and p2 with label as p1 constraint and p2 constraint
pyplot.plot(x1, p1_constraint, linewidth=3, label='P1 constraint')
pyplot.plot(x1, p2_constraint, linewidth=3, label='P2 constraint')

# plot line for x, y >= 0 constraints
pyplot.plot(numpy.zeros_like(x1), x1, linewidth=3, label='$S_1$ Sign restriction')
pyplot.plot(x1, numpy.zeros_like(x1), linewidth=3, label='$S_2$ Sign restriction')

# shades in feasible region on graph
feasible_region = PathPatch(bounds, label='Feasible region', alpha=0.5)
axis.add_patch(feasible_region)

# label axis of graph
pyplot.xlabel('$S_1$')
pyplot.ylabel('$S_2$')

# plot the graphs between -0.1 and the limit given as some graphs dont show x, y>=0 constraint if first limit is 0
pyplot.xlim(-0.1, limit)
pyplot.ylim(-0.1, limit)

# show graph
pyplot.legend()
pyplot.show()
```

To implement the linear programming problem in python I used code from (Wang, 2019).

LC (1,2)

$Max/Min\ Z = 6x + 7y$ $subject\ to:$ $x + y \leq 20$ $2x + 3y \leq 6$ $x, y \geq 0$	**Argument matrix** $\begin{pmatrix} 1 & 1 \\ 2 & 3 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 20 \\ 6 \end{pmatrix}$	**Intercepting Form** $\frac{x}{20} + \frac{y}{20} \leq 1$ $\frac{x}{3} + \frac{y}{2} \leq 1$

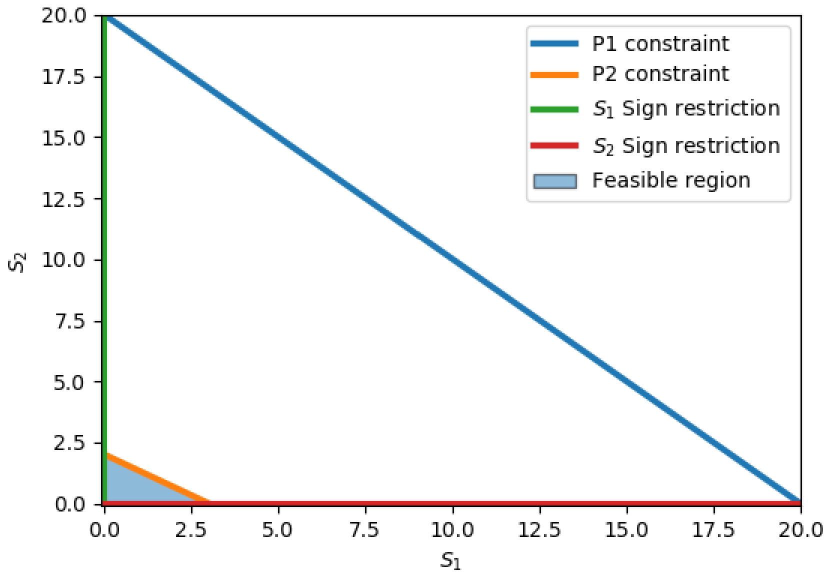


Figure 5

Maximization

In [15]:
```
prob1 = LpProblem("LC(1,2) Maximize", LpMaximize)
s1 = LpVariable("s1", lowBound = 0)
s2 = LpVariable("s2", lowBound = 0)
prob1 += 6*s1+7*s2
prob1 += 1*s1+1*s2<=20
prob1 += 2*s1+3*s2<=6
prob1
```

Out[15]:
```
LC(1,2)_Maximize:
MAXIMIZE
6*s1 + 7*s2 + 0
SUBJECT TO
_C1: s1 + s2 <= 20

_C2: 2 s1 + 3 s2 <= 6

VARIABLES
s1 Continuous
s2 Continuous
```

In [16]:
```
status = prob1.solve()
LpStatus[status]
```

Out[16]: 'Optimal'

In [17]:
```
value(prob1.objective)
```

Out[17]: 18.0

Figure 6

Minimization

In [21]:
```
prob1 = LpProblem("LC(1,2) Minimize", LpMinimize)
s1 = LpVariable("s1", lowBound = 0)
s2 = LpVariable("s2", lowBound = 0)
prob1 += 6*s1+7*s2
prob1 += 1*s1+1*s2<=20
prob1 += 2*s1+3*s2<=6
prob1
```

Out[21]:
```
LC(1,2)_Minimize:
MINIMIZE
6*s1 + 7*s2 + 0
SUBJECT TO
_C1: s1 + s2 <= 20

_C2: 2 s1 + 3 s2 <= 6

VARIABLES
s1 Continuous
s2 Continuous
```

In [22]:
```
status = prob1.solve()
LpStatus[status]
```

Out[22]: 'Optimal'

In [23]:
```
value(prob1.objective)
```

Out[23]: 0.0

Figure 7

S_1	S_2	Z
0	0	0
0	2	14
3	0	18

The maximum value is when S_1 is 3 and S_2 is 0 making Z = 18.

The minimum value is when S_1 and S_2 are both 0 making Z = 0.

LC (1,3)

$Max/Min\ Z = 3x + 2y$	**Argument matrix**	**Intercepting Form**
$subject\ to:$ $x + y \leq 10$ $2x + 3y \leq 6$ $x, y \geq 0$	$\begin{pmatrix} 1 & 1 \\ 2 & 3 \end{pmatrix}\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 10 \\ 6 \end{pmatrix}$	$\frac{x}{10} + \frac{y}{10} \leq 1$ $\frac{x}{3} + \frac{y}{2} \leq 1$

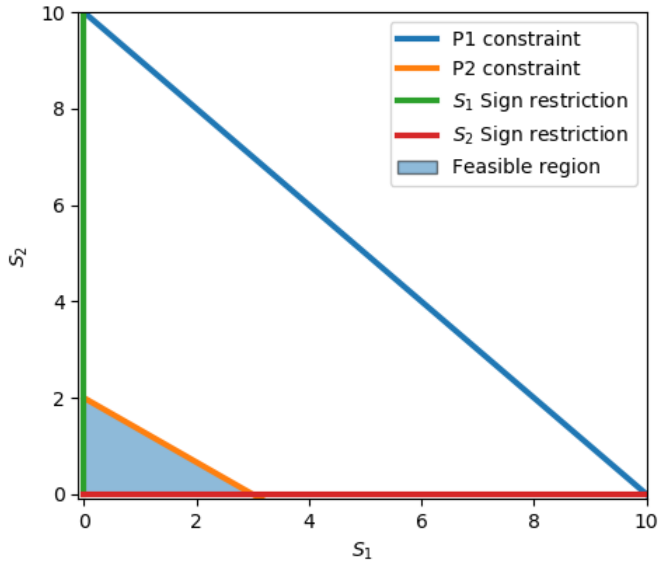


Figure 8

Maximization

```
In [6]: prob1 = LpProblem("LC(1,3)_Maximize", LpMaximize)
        s1 = LpVariable("s1", lowBound = 0)
        s2 = LpVariable("s2", lowBound = 0)
        prob1 += 3*s1+2*s2
        prob1 += 1*s1+1*s2<=10
        prob1 += 2*s1+3*s2<=6
        prob1

Out[6]: LC(1,3)_Maximize:
        MAXIMIZE
        3*s1 + 2*s2 + 0
        SUBJECT TO
        _C1: s1 + s2 <= 10

        _C2: 2 s1 + 3 s2 <= 6

        VARIABLES
        s1 Continuous
        s2 Continuous

In [7]: status = prob1.solve()
        LpStatus[status]

Out[7]: 'Optimal'

In [8]: value(prob1.objective)

Out[8]: 9.0
```

Figure 9

Minimization

```
In [9]: prob1 = LpProblem("LC(1,3)_Minimize", LpMinimize)
        s1 = LpVariable("s1", lowBound = 0)
        s2 = LpVariable("s2", lowBound = 0)
        prob1 += 3*s1+2*s2
        prob1 += 1*s1+1*s2<=10
        prob1 += 2*s1+3*s2<=6
        prob1

Out[9]: LC(1,3)_Minimize:
        MINIMIZE
        3*s1 + 2*s2 + 0
        SUBJECT TO
        _C1: s1 + s2 <= 10

        _C2: 2 s1 + 3 s2 <= 6

        VARIABLES
        s1 Continuous
        s2 Continuous

In [10]: status = prob1.solve()
         LpStatus[status]

Out[10]: 'Optimal'

In [11]: value(prob1.objective)

Out[11]: 0.0
```

Figure 10

S_1	S_2	Z
0	0	0
0	2	4
3	0	9

The maximum value is when S_1 is 3 and S_2 is 0 making Z = 9.

The minimum value is when S_1 and S_2 are both 0 making Z = 0.

LC (2,3)

$Max/Min\ Z = 2x + 3y$ $subject\ to$: $x + y \leq 40$ $3x + 8y \leq 12$ $x, y \geq 0$	**Argument matrix** $\begin{pmatrix} 1 & 1 \\ 3 & 8 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 40 \\ 12 \end{pmatrix}$	**Intercepting Form** $\frac{x}{40} + \frac{y}{40} \leq 1$ $\frac{x}{4} + \frac{y}{1.5} \leq 1$

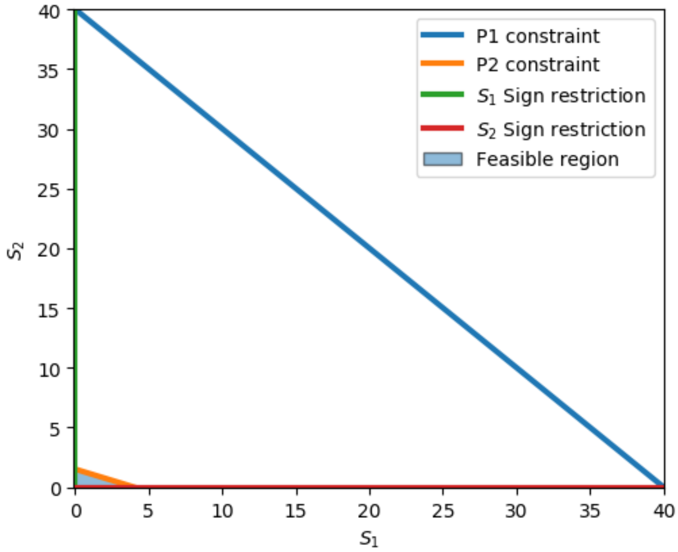


Figure 11

Maximization

```
In [10]: prob1 = LpProblem("LC(2,3)_Maximize", LpMaximize)
         s1 = LpVariable("s1", lowBound = 0)
         s2 = LpVariable("s2", lowBound = 0)
         prob1 += 2*s1+3*s2
         prob1 += 1*s1+1*s2<=40
         prob1 += 3*s1+8*s2<=12
         prob1

Out[10]: LC(2,3)_Maximize:
         MAXIMIZE
         2*s1 + 3*s2 + 0
         SUBJECT TO
         _C1: s1 + s2 <= 40

         _C2: 3 s1 + 8 s2 <= 12

         VARIABLES
         s1 Continuous
         s2 Continuous

In [4]: status = prob1.solve()
        LpStatus[status]

Out[4]: 'Optimal'

In [5]: value(prob1.objective)

Out[5]: 8.0
```

Figure 12

Minimization

```
In [11]: prob1 = LpProblem("LC(2,3)_Minimize", LpMinimize)
         s1 = LpVariable("s1", lowBound = 0)
         s2 = LpVariable("s2", lowBound = 0)
         prob1 += 2*s1+3*s2
         prob1 += 1*s1+1*s2<=40
         prob1 += 3*s1+8*s2<=12
         prob1

Out[11]: LC(2,3)_Minimize:
         MINIMIZE
         2*s1 + 3*s2 + 0
         SUBJECT TO
         _C1: s1 + s2 <= 40

         _C2: 3 s1 + 8 s2 <= 12

         VARIABLES
         s1 Continuous
         s2 Continuous

In [7]: status = prob1.solve()
        LpStatus[status]

Out[7]: 'Optimal'

In [8]: value(prob1.objective)

Out[8]: 0.0
```

Figure 13

S_1	S_2	Z
0	0	0
0	1.5	4.5
4	0	8

The maximum value is when S_1 is 4 and S_2 is 0 making Z = 8.

The minimum value is when S_1 and S_2 are both 0 making Z = 0.

LC (2,4)

$Max/Min\ Z = 5x + 8y$ $subject\ to:$ $x + y \leq 15$ $5x + 2y \leq 10$ $x, y \geq 0$	**Argument matrix** $\begin{pmatrix} 1 & 1 \\ 5 & 2 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 15 \\ 10 \end{pmatrix}$	**Intercepting Form** $\frac{x}{15} + \frac{y}{15} \leq 1$ $\frac{x}{2} + \frac{y}{5} \leq 1$

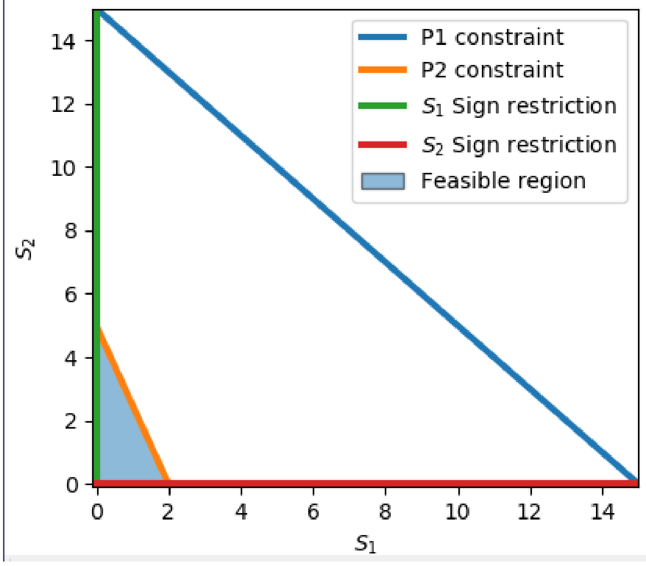


Figure 14

Maximization

```
In [12]: prob1 = LpProblem("LC(2,4)_Maximize", LpMaximize)
         s1 = LpVariable("s1", lowBound = 0)
         s2 = LpVariable("s2", lowBound = 0)
         prob1 += 5*s1+8*s2
         prob1 += 1*s1+1*s2<=15
         prob1 += 5*s1+2*s2<=10
         prob1

Out[12]: LC(2,4)_Maximize:
         MAXIMIZE
         5*s1 + 8*s2 + 0
         SUBJECT TO
         _C1: s1 + s2 <= 15

         _C2: 5 s1 + 2 s2 <= 10

         VARIABLES
         s1 Continuous
         s2 Continuous

In [13]: status = prob1.solve()
         LpStatus[status]

Out[13]: 'Optimal'

In [14]: value(prob1.objective)

Out[14]: 40.0
```

Figure 15

Minimization

```
In [15]: prob1 = LpProblem("LC(2,4)_Minimize", LpMinimize)
         s1 = LpVariable("s1", lowBound = 0)
         s2 = LpVariable("s2", lowBound = 0)
         prob1 += 5*s1+8*s2
         prob1 += 1*s1+1*s2<=15
         prob1 += 5*s1+2*s2<=10
         prob1

Out[15]: LC(2,4)_Minimize:
         MINIMIZE
         5*s1 + 8*s2 + 0
         SUBJECT TO
         _C1: s1 + s2 <= 15

         _C2: 5 s1 + 2 s2 <= 10

         VARIABLES
         s1 Continuous
         s2 Continuous

In [16]: status = prob1.solve()
         LpStatus[status]

Out[16]: 'Optimal'

In [17]: value(prob1.objective)

Out[17]: 0.0
```

Figure 16

S_1	S_2	Z
0	0	0
0	5	40
2	0	10

The maximum value is when S_1 is 0 and S_2 is 5 making Z = 40.

The minimum value is when S_1 and S_2 are both 0 making Z = 0.

LC (3,4)

$Max/Min\ Z = 3x + 8y$ subject to: $x + y \leq 5$, $5x + 10y \leq 15$, $x, y \geq 0$	**Argument matrix** $\begin{pmatrix} 1 & 1 \\ 5 & 10 \end{pmatrix}\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 5 \\ 15 \end{pmatrix}$	**Intercepting Form** $\frac{x}{5} + \frac{y}{5} \leq 1$, $\frac{x}{3} + \frac{y}{1.5} \leq 1$

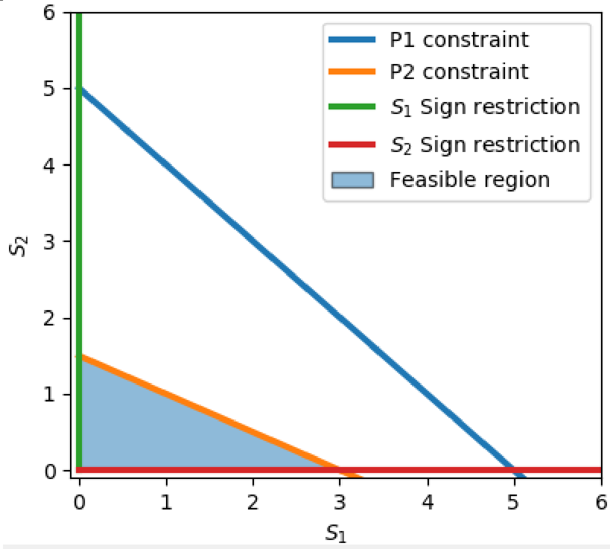


Figure 17

Maximization

```
In [19]: prob1 = LpProblem("LC(3,4)_Maximize", LpMaximize)
         s1 = LpVariable("s1", lowBound = 0)
         s2 = LpVariable("s2", lowBound = 0)
         prob1 += 3*s1+8*s2
         prob1 += 1*s1+1*s2<=5
         prob1 += 5*s1+10*s2<=15
         prob1

Out[19]: LC(3,4)_Maximize:
         MAXIMIZE
         3*s1 + 8*s2 + 0
         SUBJECT TO
         _C1: s1 + s2 <= 5

         _C2: 5 s1 + 10 s2 <= 15

         VARIABLES
         s1 Continuous
         s2 Continuous

In [20]: status = prob1.solve()
         LpStatus[status]

Out[20]: 'Optimal'

In [21]: value(prob1.objective)

Out[21]: 12.0
```

Figure 18

Minimization

```
In [22]: prob1 = LpProblem("LC(3,4)_Minimize", LpMinimize)
         s1 = LpVariable("s1", lowBound = 0)
         s2 = LpVariable("s2", lowBound = 0)
         prob1 += 3*s1+8*s2
         prob1 += 1*s1+1*s2<=5
         prob1 += 5*s1+10*s2<=15
         prob1

Out[22]: LC(3,4)_Minimize:
         MINIMIZE
         3*s1 + 8*s2 + 0
         SUBJECT TO
         _C1: s1 + s2 <= 5

         _C2: 5 s1 + 10 s2 <= 15

         VARIABLES
         s1 Continuous
         s2 Continuous

In [23]: status = prob1.solve()
         LpStatus[status]

Out[23]: 'Optimal'

In [24]: value(prob1.objective)

Out[24]: 0.0
```

Figure 19

S_1	S_2	Z
0	0	0
0	1.5	12
3	0	9

The maximum value is when S_1 is 0 and S_2 is 1.5 making Z = 12.

The minimum value is when S_1 and S_2 are both 0 making Z = 0.

LC (3,5)

$Max/Min\ Z = 4x + 3y$	**Argument matrix**	**Intercepting Form**
$subject\ to:$ $x + y \le 20$ $4x + 2y \le 8$ $x, y \ge 0$	$\begin{pmatrix} 1 & 1 \\ 4 & 2 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 20 \\ 8 \end{pmatrix}$	$\frac{x}{20} + \frac{y}{20} \le 1$ $\frac{x}{2} + \frac{y}{4} \le 1$

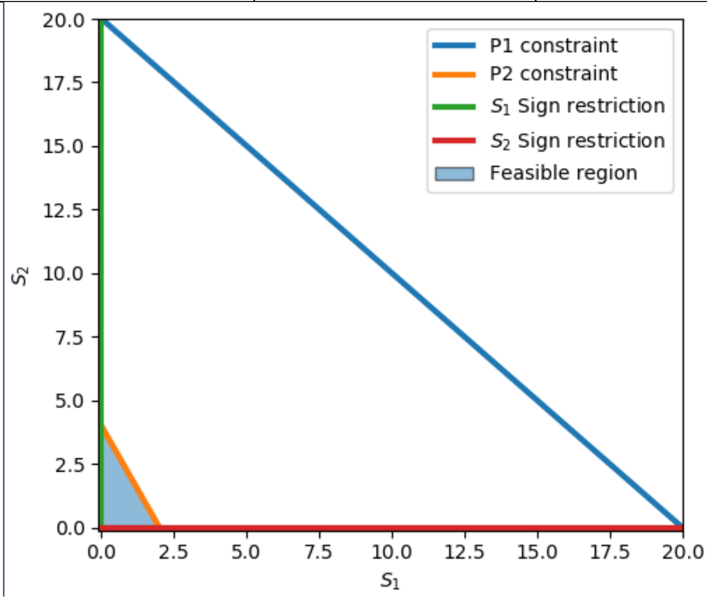


Figure 20

Maximization	**Minimization**

In [26]: `prob1 = LpProblem("LC(3,5)_Maximize", LpMaximize)`
`s1 = LpVariable("s1", lowBound = 0)`
`s2 = LpVariable("s2", lowBound = 0)`
`prob1 += 4*s1+3*s2`
`prob1 += 1*s1+1*s2<=20`
`prob1 += 4*s1+2*s2<=8`
`prob1`
	In [29]: `prob1 = LpProblem("LC(3,5)_Minimize", LpMinimize)`
	`s1 = LpVariable("s1", lowBound = 0)`
	`s2 = LpVariable("s2", lowBound = 0)`
	`prob1 += 4*s1+3*s2`
	`prob1 += 1*s1+1*s2<=20`
	`prob1 += 4*s1+2*s2<=8`
	`prob1`

Out[26]: LC(3,5)_Maximize: MAXIMIZE 4*s1 + 3*s2 + 0 SUBJECT TO _C1: s1 + s2 <= 20 _C2: 4 s1 + 2 s2 <= 8 VARIABLES s1 Continuous s2 Continuous	Out[29]: LC(3,5)_Minimize: MINIMIZE 4*s1 + 3*s2 + 0 SUBJECT TO _C1: s1 + s2 <= 20 _C2: 4 s1 + 2 s2 <= 8 VARIABLES s1 Continuous s2 Continuous
In [27]: `status = prob1.solve()` `LpStatus[status]`	In [30]: `status = prob1.solve()` `LpStatus[status]`
Out[27]: 'Optimal'	Out[30]: 'Optimal'
In [28]: `value(prob1.objective)`	In [31]: `value(prob1.objective)`
Out[28]: 12.0	Out[31]: 0.0
Figure 21	*Figure 22*

S_1	S_2	Z
0	0	0
0	4	12
2	0	8

The maximum value is when S_1 is 0 and S_2 is 4 making Z = 12.

The minimum value is when S_1 and S_2 are both 0 making Z = 0.

LC (4,5)

$Max/Min\ \ Z = 4x + 5y$ $subject\ to:$ $x + y \leq 5$ $2x + 3y \leq 6$ $x, y \geq 0$	**Argument matrix** $\begin{pmatrix} 1 & 1 \\ 2 & 3 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 5 \\ 6 \end{pmatrix}$	**Intercepting Form** $\frac{x}{5} + \frac{y}{5} \leq 1$ $\frac{x}{3} + \frac{y}{2} \leq 1$

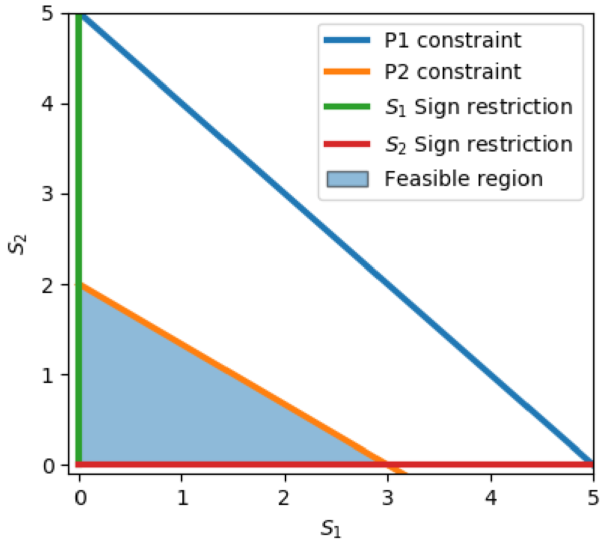


Figure 23

Maximization	Minimization
see below	see below

Maximization

```
In [33]: prob1 = LpProblem("LC(4,5)_Maximize", LpMaximize)
         s1 = LpVariable("s1", lowBound = 0)
         s2 = LpVariable("s2", lowBound = 0)
         prob1 += 4*s1+5*s2
         prob1 += 1*s1+1*s2<=5
         prob1 += 2*s1+3*s2<=6
         prob1

Out[33]: LC(4,5)_Maximize:
         MAXIMIZE
         4*s1 + 5*s2 + 0
         SUBJECT TO
         _C1: s1 + s2 <= 5

         _C2: 2 s1 + 3 s2 <= 6

         VARIABLES
         s1 Continuous
         s2 Continuous

In [34]: status = prob1.solve()
         LpStatus[status]

Out[34]: 'Optimal'

In [35]: value(prob1.objective)

Out[35]: 12.0
```

Figure 24

Minimization

```
In [36]: prob1 = LpProblem("LC(4,5)_Minimize", LpMinimize)
         s1 = LpVariable("s1", lowBound = 0)
         s2 = LpVariable("s2", lowBound = 0)
         prob1 += 4*s1+5*s2
         prob1 += 1*s1+1*s2<=5
         prob1 += 2*s1+3*s2<=6
         prob1

Out[36]: LC(4,5)_Minimize:
         MINIMIZE
         4*s1 + 5*s2 + 0
         SUBJECT TO
         _C1: s1 + s2 <= 5

         _C2: 2 s1 + 3 s2 <= 6

         VARIABLES
         s1 Continuous
         s2 Continuous

In [37]: status = prob1.solve()
         LpStatus[status]

Out[37]: 'Optimal'

In [38]: value(prob1.objective)

Out[38]: 0.0
```

Figure 25

S_1	S_2	Z
0	0	0
0	2	10
3	0	12

The maximum value is when S_1 is 3 and S_2 is 0 making Z = 12.

The minimum value is when S_1 and S_2 are both 0 making Z = 0.

LC (4,6)

$Max/Min\ Z = 6x + 3y$	**Argument matrix**	**Intercepting Form**
$subject\ to:$ $x + y \leq 10$ $6x + 3y \leq 9$ $x, y \geq 0$	$\begin{pmatrix} 1 & 1 \\ 6 & 3 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 10 \\ 9 \end{pmatrix}$	$\frac{x}{10} + \frac{y}{10} \leq 1$ $\frac{x}{1.5} + \frac{y}{3} \leq 1$

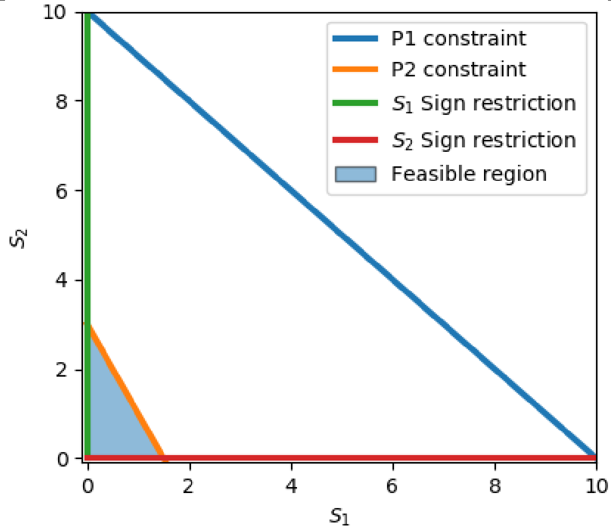


Figure 26

Maximization

```
In [40]: prob1 = LpProblem("LC(4,6)_Maximize", LpMaximize)
         s1 = LpVariable("s1", lowBound = 0)
         s2 = LpVariable("s2", lowBound = 0)
         prob1 += 6*s1+3*s2
         prob1 += 1*s1+1*s2<=10
         prob1 += 6*s1+3*s2<=9
         prob1

Out[40]: LC(4,6)_Maximize:
         MAXIMIZE
         6*s1 + 3*s2 + 0
         SUBJECT TO
         _C1: s1 + s2 <= 10

         _C2: 6 s1 + 3 s2 <= 9

         VARIABLES
         s1 Continuous
         s2 Continuous

In [41]: status = prob1.solve()
         LpStatus[status]

Out[41]: 'Optimal'

In [42]: value(prob1.objective)

Out[42]: 9.0
```

Figure 27

Minimization

```
In [43]: prob1 = LpProblem("LC(4,6)_Minimize", LpMinimize)
         s1 = LpVariable("s1", lowBound = 0)
         s2 = LpVariable("s2", lowBound = 0)
         prob1 += 6*s1+3*s2
         prob1 += 1*s1+1*s2<=10
         prob1 += 6*s1+3*s2<=9
         prob1

Out[43]: LC(4,6)_Minimize:
         MINIMIZE
         6*s1 + 3*s2 + 0
         SUBJECT TO
         _C1: s1 + s2 <= 10

         _C2: 6 s1 + 3 s2 <= 9

         VARIABLES
         s1 Continuous
         s2 Continuous

In [44]: status = prob1.solve()
         LpStatus[status]

Out[44]: 'Optimal'

In [45]: value(prob1.objective)

Out[45]: 0.0
```

Figure 28

S_1	S_2	Z
0	0	0
0	3	9
1.5	0	9

The maximum value is when S_1 is 0 and S_2 is 3 / S_1 is 1.5 and S_2 is 0 making Z = 9.

The minimum value is when S_1 and S_2 are both 0 making Z = 0.

LC (5,6)

$Max/Min\ \ Z = 5x + 9y$	**Argument matrix**	**Intercepting Form**
$subject\ to:$ $x + y \leq 5$ $4x + 2y \leq 8$ $x, y \geq 0$	$\begin{pmatrix} 1 & 1 \\ 4 & 2 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 5 \\ 8 \end{pmatrix}$	$\frac{x}{5} + \frac{y}{5} \leq 1$ $\frac{x}{2} + \frac{y}{4} \leq 1$

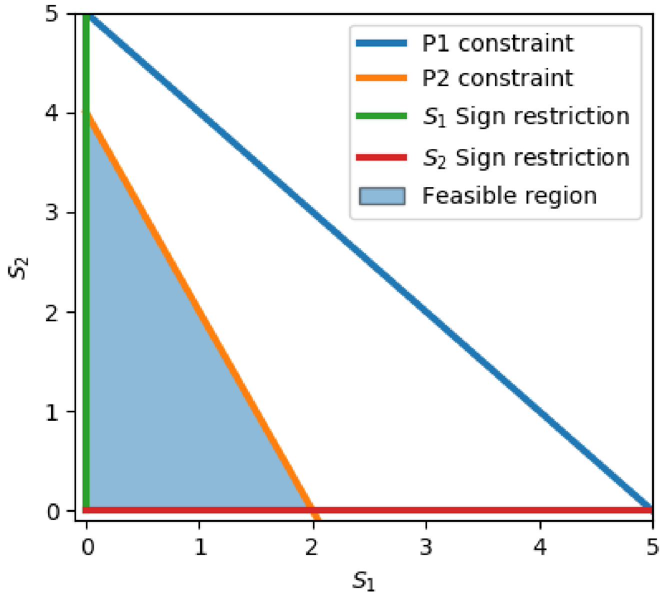


Figure 29

Maximization

```
In [49]: prob1 = LpProblem("LC(5,6)_Maximize", LpMaximize)
         s1 = LpVariable("s1", lowBound = 0)
         s2 = LpVariable("s2", lowBound = 0)
         prob1 += 5*s1+9*s2
         prob1 += 1*s1+1*s2<=5
         prob1 += 4*s1+2*s2<=8
         prob1

Out[49]: LC(5,6)_Maximize:
         MAXIMIZE
         5*s1 + 9*s2 + 0
         SUBJECT TO
         _C1: s1 + s2 <= 5

         _C2: 4 s1 + 2 s2 <= 8

         VARIABLES
         s1 Continuous
         s2 Continuous

In [50]: status = prob1.solve()
         LpStatus[status]

Out[50]: 'Optimal'

In [51]: value(prob1.objective)

Out[51]: 36.0
```

Figure 30

Minimization

```
In [52]: prob1 = LpProblem("LC(5,6)_Minimize", LpMinimize)
         s1 = LpVariable("s1", lowBound = 0)
         s2 = LpVariable("s2", lowBound = 0)
         prob1 += 5*s1+9*s2
         prob1 += 1*s1+1*s2<=5
         prob1 += 4*s1+2*s2<=8
         prob1

Out[52]: LC(5,6)_Minimize:
         MINIMIZE
         5*s1 + 9*s2 + 0
         SUBJECT TO
         _C1: s1 + s2 <= 5

         _C2: 4 s1 + 2 s2 <= 8

         VARIABLES
         s1 Continuous
         s2 Continuous

In [53]: status = prob1.solve()
         LpStatus[status]

Out[53]: 'Optimal'

In [54]: value(prob1.objective)

Out[54]: 0.0
```

Figure 31

S_1	S_2	Z
0	0	0
0	4	36
2	0	10

The maximum value is when S_1 is 0 and S_2 is 4 making Z = 36.

The minimum value is when S_1 and S_2 are both 0 making Z = 0.

Conclusion

From drawing the graphs for each edge, I have evaluated that none of the graphs intersect when the constraint $x, y \geq 0$ is used. Additionally, all of the feasible solutions are members of the feasible regions for each individual edges' graph thus, a set of each edges' feasible solutions would be a proper subset of each edges' feasible region. If F_s represents feasible solutions and F_r represents the feasible region, $Fs \in Fr$ and $Fs \subseteq Fr$.

Furthermore, as all of the graphs for each edge have a minimum value of 0, depicting the transportation problem with 0 values on each edge wouldn't help in solving the problem and calculating a probability density function. Therefore, I will only be using the maximum values of each edge on the transportation graph.

Graph

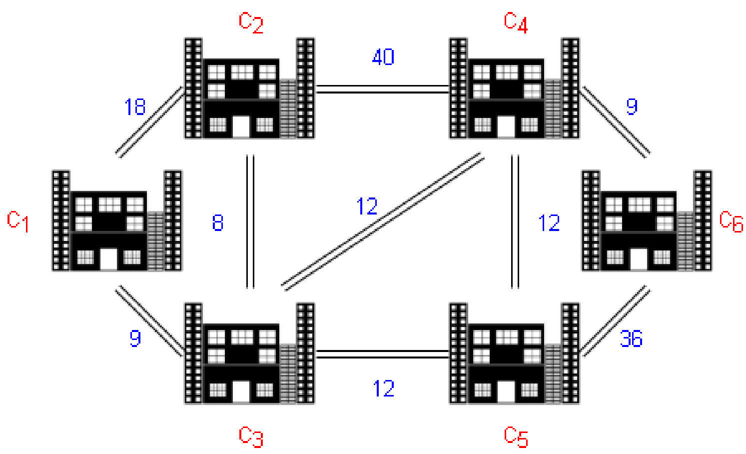


Figure 32

Graph Theory

In this section I will be determining all the paths that can be found in the transportation graph and the total cost it takes to travel along each path.

Dijkstra's algo.py

```
INFINITY = 9999999
# maximum weight
graph = {'a': {'b': 18, 'c': 9}, 'b': {'a': 18, 'c': 8, 'd': 40}, 'c': {'a': 9, 'b': 8, 'd': 12, 'e': 12},
         'd': {'b': 40, 'c': 12, 'e': 12, 'f': 9}, 'e': {'c': 12, 'd': 12, 'f': 36}, 'f': {'d': 9, 'e': 36}}
# all connections on the graph
totalCost = {}
parentNodes = {}
route = []
for node in graph:
    totalCost[node] = INFINITY
totalCost['a'] = 0
# initialises total cost of each node
while graph:
    smallest = None
    for node in graph:
        if smallest is None:
            smallest = node
        elif totalCost[node] < totalCost[smallest]:
            smallest = node
        # changes smallest node to the current node in the list
    for childNode, cost in graph[smallest].items():
        if cost + totalCost[smallest] < totalCost[childNode]:
            totalCost[childNode] = cost + totalCost[smallest]
            parentNodes[childNode] = smallest
        # append nodes to parentnodes dictionary, update values in total cost dictionary
    graph.pop(smallest)
    # remove smallest node from graph in each loop
currentNode = 'f'
while currentNode != 'a':
    try:
        route.insert(0, currentNode)
        currentNode = parentNodes[currentNode]
        # try to make a route from the parentNodes dictionary to the currentNode
    except KeyError:
        print('Path not reachable')
        break
        # print message when route isnt possible
route.insert(0, 'a')
if totalCost['f'] != INFINITY:
    print(
        "The shortest route from C1-C6 represented with a-f respectively is {} has total cost {}".format(route, totalCost['f']))
    # print the shortest route and how to get there
```

I used source code from (Sullivan, 2017) to implement Dijkstra's algorithm in python.

```
In [1]: %run "dijkstra's algo"
        The shortest route from C1-C6 represented with a-f respectively is ['a', 'c', 'd', 'f'] has total cost 30
```

Running this algorithm shows that the shortest path would be from A, C, D, F giving a cost of 30 which would be from C_1, C_3, C_4 and finally C_6.

Paths

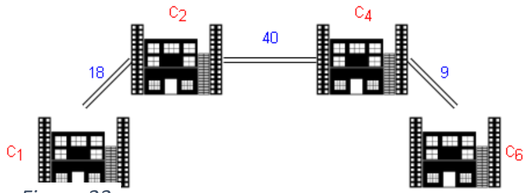
<table>
<tr><th colspan="2">Number</th><th>Cost</th></tr>
<tr><td>1</td><td>


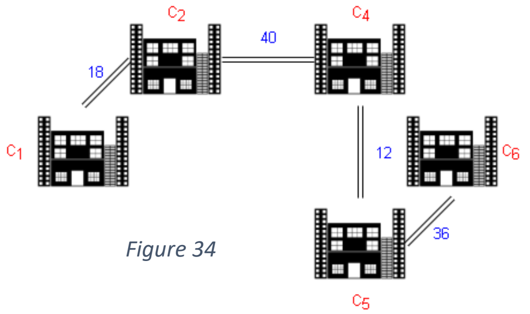
Figure 33</td><td>67</td></tr>
<tr><td>2</td><td>


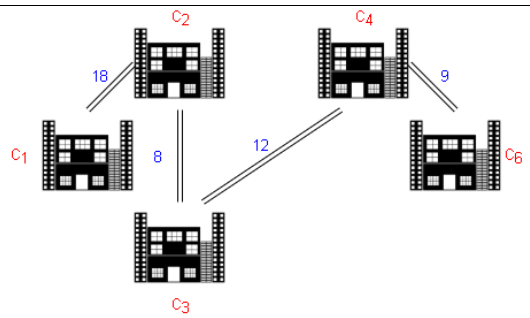
Figure 34</td><td>106</td></tr>
<tr><td>3</td><td>


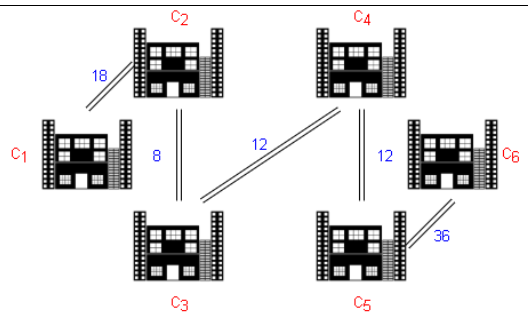
Figure 35</td><td>47</td></tr>
<tr><td>4</td><td>


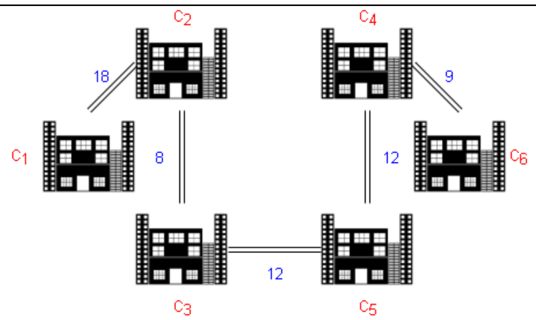
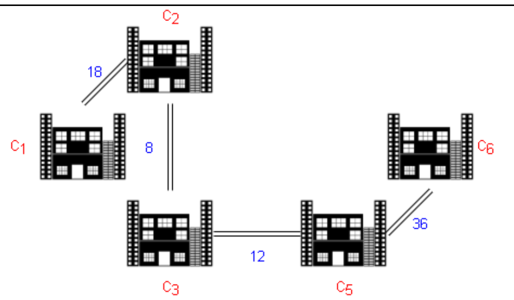
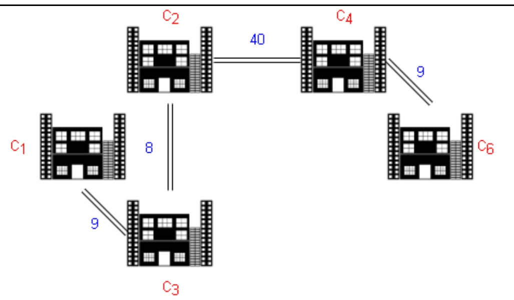
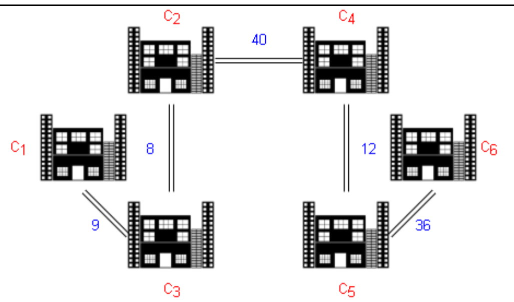
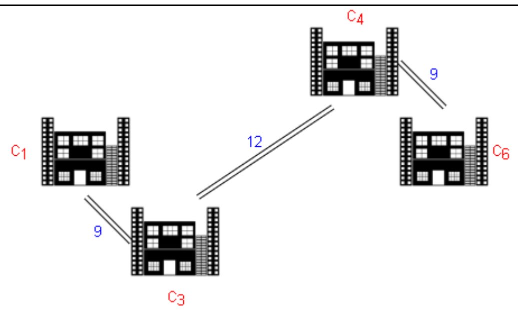
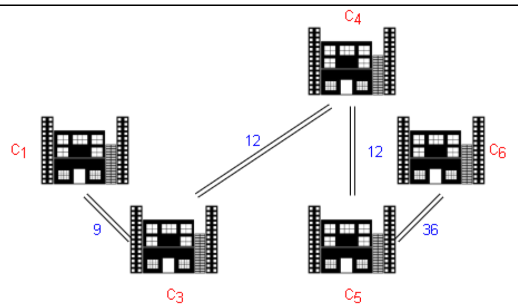
Figure 36</td><td>86</td></tr>
<tr><td>5</td><td>


Figure 37</td><td>59</td></tr>
</table>

6	 *Figure 38*	74
7	 *Figure 39*	66
8	 *Figure 40*	105
9	 *Figure 41*	30
10	 *Figure 42*	69

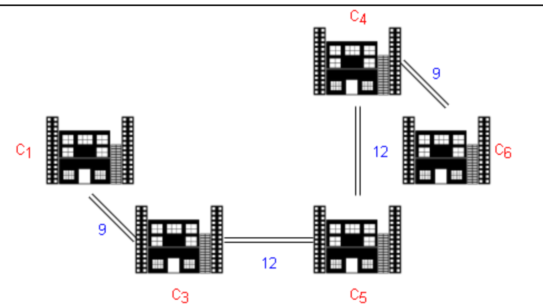
<table>
<tr><td>11</td><td>

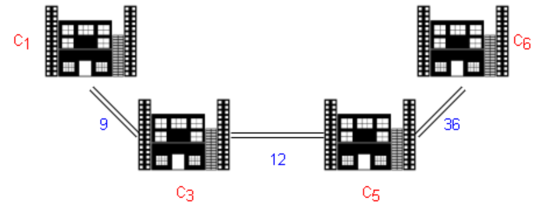
Figure 43</td><td>42</td></tr>
<tr><td>12</td><td>

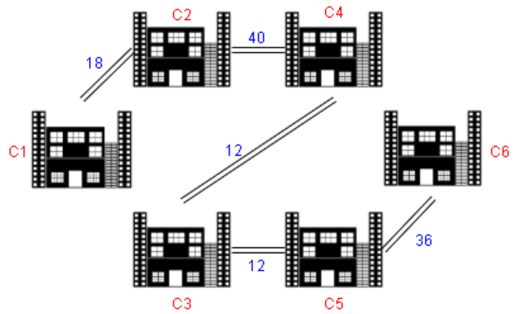
Figure 44</td><td>57</td></tr>
<tr><td>13</td><td>

Figure 45</td><td>118</td></tr>
</table>

Conclusion

All of the paths shown have a vertex set which is a proper subset of the vertex set of the transportation graph and this can be similarly linked to the edge set of the paths with the edge set of the transportation graph thus making all of the vertices and edges members of the total vertex and edge sets of the transportation graph. This can be depicted as $Vp \in Vg$and $Ep \in Eg$ and $set\ Vp \subseteq set\ Vg$ and $set\ Ep \subseteq set\ Eg$ where V represents vertices and E represents edges. Paths 4, 5, 8 and 13 are minimal spanning trees as they traverse the tree whilst visiting every node thus these paths are Hamiltonian paths. Each of the paths are subgraphs of the transportation graph such that they can be represented as components of the transportation graph. From the graphs I can deduce that the optimum path is C_1, C_3, C_4, C_6 with a cost of 30 which is validated by the Dijkstra's algorithm I implemented in python.

If the transportation problem is depicted as a digraph then paths 5,7,8,11 and 13 wouldn't be included as they oppose the direction of the edges (assuming that the edges are directed such that LC (1,2) is a direction from node 1 to node 2 and that the other edges follow this rule). Due to this discrepancy, when calculating the probability distribution function and analysing the data for the paths, I will calculate the data for an undirected graph then have an additional section for if the graph is a digraph.

Statistics

Basic Information.py

```
from statistics import stdev, variance
# import modules for standard deviation and variance
from matplotlib import pyplot
import numpy
from scipy import stats
import csv
import pandas
# import modules for boxplot, gaussian distribution and probability density function


def sort(list):
    for i in range(len(list)):
        for j in range(i, len(list)):
            if list[i] > list[j]:
                # if the ith value is greater than the jth value
                list[j], list[i] = list[i], list[j]
                # swap there places
    mean = 0
    for i in list:
        mean += i
    mean /= len(list)
    return mean, list


# to store cost of each path
choice = int(input(
    "Do you want the solution for the undirected or directed graph\n1.\tUndirected\n2.\tDirected\n> "))
list1 = []
list2 = []
with open('file.csv', encoding='utf-8-sig') as file:
    count = 0
    for row in csv.reader(file, delimiter=','):
        for value in row:
            if count == 0:
                if value != '':
                    if value == '118':
                        count += 1
                    list1.append(int(value))
            else:
                if value != '':
                    list2.append(int(value))
if choice == 1:
    mean, list = sort(list1)
    print("For the unidrected graph\n")
```

```python
else:
    mean, list = sort(list2)
    print("For the directed graph\n")

print("\nThe sorted list is {}\n".format(list))
# print the list when its sorted

if len(list) % 2 == 0:
    m1 = len(list) / 2
    m1 = int(m1)
    middle = (list[m1 - 1] + list[m1]) / 2
    # get middle of list
    first1 = int(len(list) / 4)
    first_quartile = (list[first1] + list[first1 - 1]) / 2
    # get first quartile value
    third1 = int(first1 * 3)
    third_quartile = (list[third1] + list[third1 - 1]) / 2
    # get third quartile value
    iqr = third_quartile - first_quartile
    # get iqr
else:
    mid = len(list) / 2
    middle = list[int(mid) - 1]
    # get middle of list
    first_quartile = int(len(list) / 4)
    third_quartile = int(first_quartile * 3)
    first_quartile = list[first_quartile]
    # get first quartile value
    third_quartile = list[third_quartile]
    # get third quartile value
    iqr = third_quartile - first_quartile
    # get iqr

print("The mean is {}\nThe median is {}\nThe standard deviation is {}\nThe variance is {}\n".format(
    mean, middle, stdev(list), variance(list)))
print("The 1st quartile is {}\nThe 3rd quartile is {}\nThe IQR is {}\n".format(
    first_quartile, third_quartile, iqr))
# print information about the data sets

df = pandas.DataFrame([list], index=['Cost'])
df.T.boxplot(vert=False)
pyplot.subplots_adjust(left=0.25)
pyplot.show()
```

```python
sigma = stdev(list)
# get standard deviation
x = numpy.linspace(mean - 3 * sigma, mean + 3 * sigma, 100)
pyplot.plot(x, stats.norm.pdf(x, mean, sigma))
pyplot.show()
# plot gaussian distribution
```

To fix the encoding on the csv file to be used in a python list I followed (senshin, 2015) which led me to implementing the boxplot with the aid of (unutbu, Horizontal box plots in matplotlib/Pandas, 2013) and the Gaussian Distribution with inspiration from (unutbu, python pylab plot normal distribution, 2012). This code reads in the file called “file.csv”.

PDF.py

```
import pandas
from matplotlib import pyplot
import seaborn
# import pandas to read in csv and import matplotlib and seaborn for pdf and histogram
file = 'pdf.csv'
choice = int(input("Would you like to see the PDF of the:\n1.\tUndirected graph\n2.\tDirected graph\n> "))
if choice == 1:
    title = "Undirected Graph PDF"
    column = 0
else:
    title = "Directed Graph PDF"
    column = 1
# user chooses the title and column in the csv file
ds = pandas.read_csv(file, usecols=[column])
# choose the data set from the column in the csv file
seaborn.distplot(ds, bins=7, hist=True, kde=True,
                 hist_kws={'edgecolor': 'black'}, kde_kws={'linewidth': 4})
# plot the histogram and pdf with seaborn
pyplot.title(title)
pyplot.ylabel('Density')
pyplot.xlabel('Cost')
# give the figure a title and labels on the axis
pyplot.show()
# show the graph
```

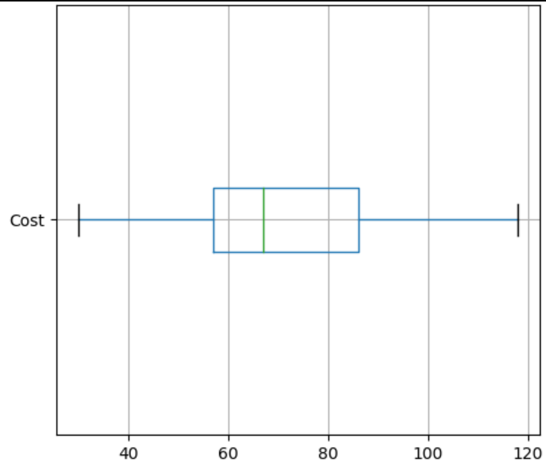
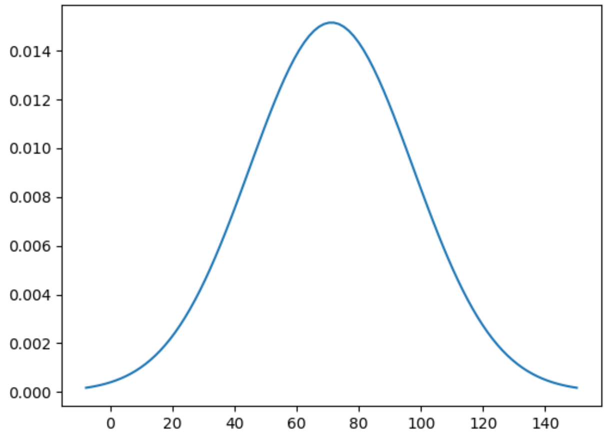
To plot the probability density function, I used (Koehrsen, 2018). This code reads in the file called "pdf.csv".

Path approach

Undirected Graph

```
In [4]: %run "basic information"

Do you want the solution for the undirected or directed graph
1.	Undirected
2.	Directed
> 1
For the unidrected graph


The sorted list is [30, 42, 47, 57, 59, 66, 67, 69, 74, 86, 105, 106, 118]

The mean is 71.23076923076923
The median is 66
The standard deviation is 26.309547842794785
The variance is 692.1923076923077

The 1st quartile is 57
The 3rd quartile is 86
The IQR is 29
```

Figure 46

Box plot	**Gaussian Distribution**
 *Figure 47*	 *Figure 48*

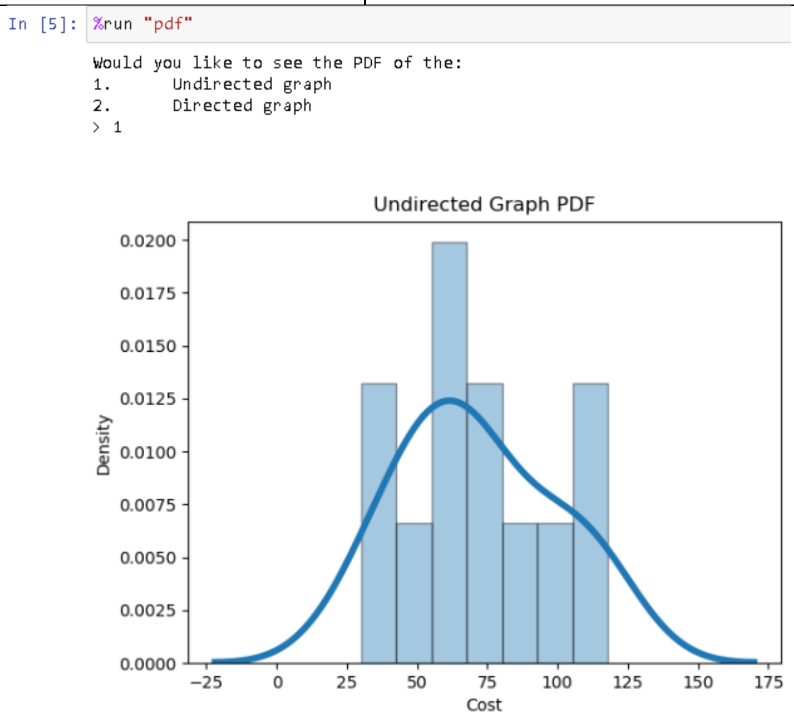


Figure 49

The mean is greater than the median thus the distribution is positively skewed which is depicted clearly in the probability density function. When calculating for outliers (which is approximately 1.5 times the standard deviation from the mean) I found that the costs of 118 and 30 would be considered outliers as the acceptable range of values would lie between 32.1529315176 and 110.847068482. This means that incorporating these values in my calculations and figures could be the cause of this distribution being positively skewed. However, as the figure is now, from the graph I can determine that there is a high concentration of costs within the bins for 50 to 75. This means the majority of paths lie between a cost of 50and 75.

Directed Graph

```
In [1]: %matplotlib notebook
        %run "basic information"
```

```
Do you want the solution for the undirected or directed graph
1.      Undirected
2.      Directed
> 2
For the directed graph


The sorted list is [30, 47, 57, 67, 69, 74, 86, 106]

The mean is 67.0
The median is 68.0
The standard deviation is 23.311554461866574
The variance is 543.4285714285714

The 1st quartile is 52.0
The 3rd quartile is 80.0
The IQR is 28.0
```

Figure 50

```
In [2]: %run "pdf"
```

```
Would you like to see the PDF of the:
1.      Undirected graph
2.      Directed graph
> 2
```

```
C:\Users\kamra\Anaconda3\lib\site-packages\numpy\lib\histograms.py:824: R
er_equal
  keep = (tmp_a >= first_edge)
C:\Users\kamra\Anaconda3\lib\site-packages\numpy\lib\histograms.py:825: R
equal
  keep &= (tmp_a <= last_edge)
C:\Users\kamra\Anaconda3\lib\site-packages\statsmodels\nonparametric\kde.
in greater
  X = X[np.logical_and(X > clip[0], X < clip[1])] # won't work for two cc
C:\Users\kamra\Anaconda3\lib\site-packages\statsmodels\nonparametric\kde.
in less
  X = X[np.logical_and(X > clip[0], X < clip[1])] # won't work for two cc
```

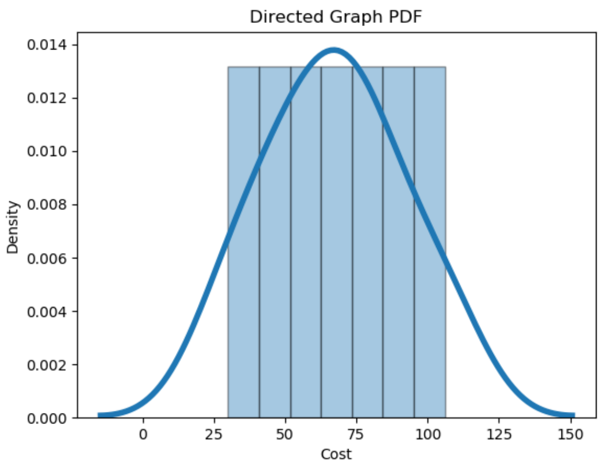


Figure 51

Boxplot	Gaussian Distribution

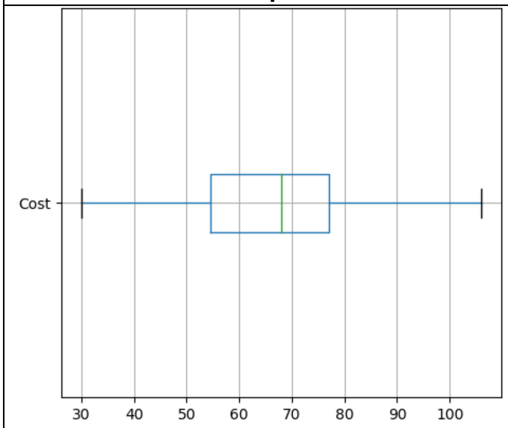


Figure 52

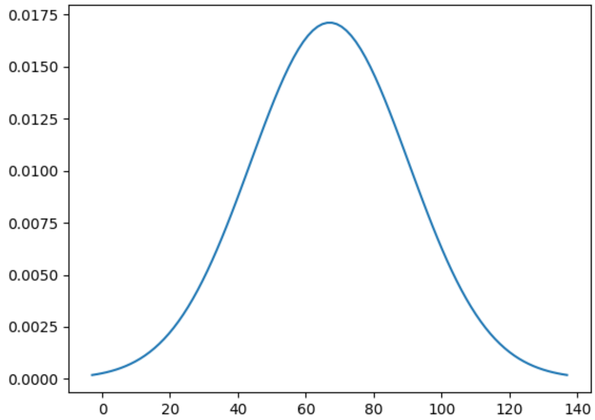


Figure 53

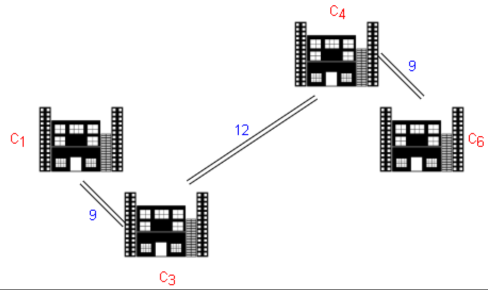
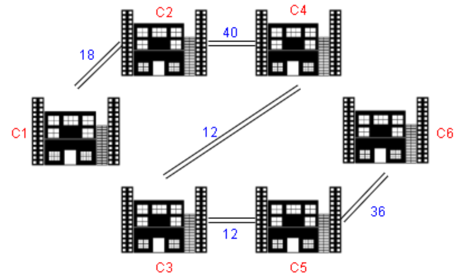
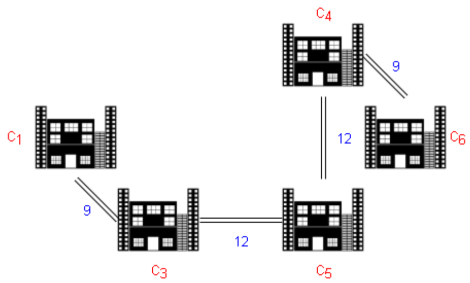
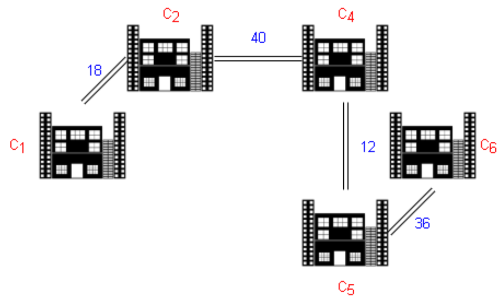
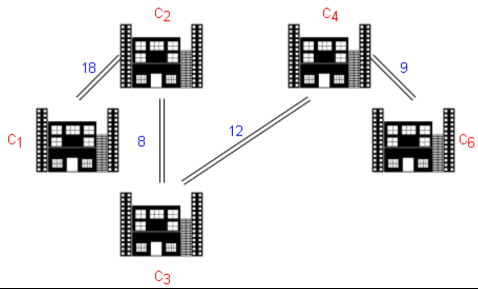
The mean is less than the median thus the distribution is negatively skewed. When calculating for outliers (which is approximately 1.5 times the standard deviation from the mean) I found that the costs of 106 and 30 would be considered outliers as the acceptable values would approximately be in the range of 32.0326680729 and 101.967331927. This means that incorporating these values within my calculations could be the cause of the distribution being negatively skewed in the probability density function. Even though there appears to be a negative skewness in the data, the probability density function plotted appears to depict that the skewness isn't perceptible as the graph appears similar to a Gaussian distribution due to its symmetrical nature which could be depicted by $X \sim N(67, 543.43)$.

What I can conclude?

On both graphs the minimum and maximum points on the boxplots are the same as both paths are applicable to both scenarios. The mean in the undirected graph is fractionally larger than the mean in the directed graph but the means are very similar. Although the directed graph has less data points in the distribution than the undirected graph; the first quartile, last quartile and inter quartile range are the same in both situations thus insinuating that most of the data is heavily weighted within the region enclosed by the first quartile and last quartile.

The standard deviation is the same for both scenarios thus showing that the data of both graphs are similarly distributed however, the variance in the directed graph is higher than the variance in the undirected graph thus the data in the directed graph is further distributed from the mean than the data in the undirected graph. This is further emphasised by the probability density function of both graphs as for the directed graph, the data is distributed evenly but in the undirected graph, the data is more concentrated between an approximate cost of 40 and 70.

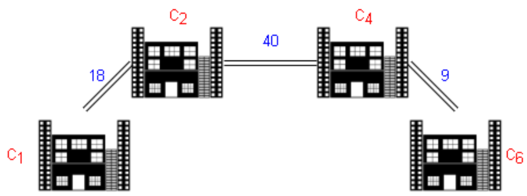
If the outliers are included the optimistic cost is 30 and pessimistic cost is 106 however, if not included the pessimistic cost in both scenarios is 86 but for the undirected graph, the optimistic cost would be 42 whilst for the directed graph the optimistic cost would be 47 while operating under the assumption that optimistic means least costly and pessimistic means most costly means of transport.

<table>
<tr><th>Situation</th><th>Optimistic</th><th>Pessimistic</th></tr>
<tr><td>With outliers</td><td>Cost = 30
</td><td>Cost = 118
</td></tr>
<tr><td>Undirected without outliers</td><td>Cost = 42
</td><td rowspan="2">Cost = 106
</td></tr>
<tr><td>Directed without outliers</td><td>Cost = 47
</td></tr>
</table>

Cartesian Product Approach

I will be using set theory to determine the cartesian product of all of the given paths. Each member of the set made by the cartesian product will be a tuple.

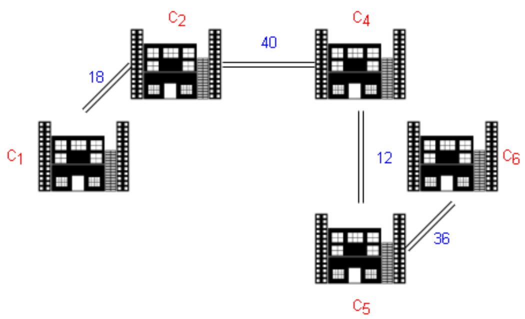
Path 1



{0,18} x {0,40} x {0,9} = {(0,0,0), (0,0,9), (0,40,0), (0,40,9), (18,0,0), (18,0,9), (18,40,0), (18,40,9)}

Costs = 0, 9, 40, 49, 18, 27, 58, 67

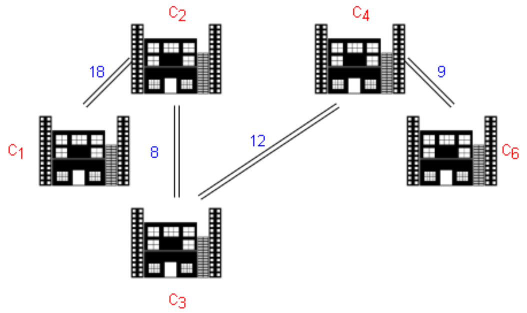
Path 2



{0,18} x {0,40} x {0,12} x {0,36} = {(0,0,0,0), (0,0,0,36), (0,0,12,0), (0,0,12,36), (0,40,0,0), (0,40,0,36), (0,40,12,0), (0,40,12,36), (18,0,0,0), (18,0,0,36), (18,0,12,0), (18,0,12,36), (18,40,0,0), (18,40,0,36), (18,40,12,0), (18,40,12,36)}

Costs = 0, 36, 12, 48, 40, 76, 52, 88, 18, 54, 30, 66, 58, 94, 70, 106

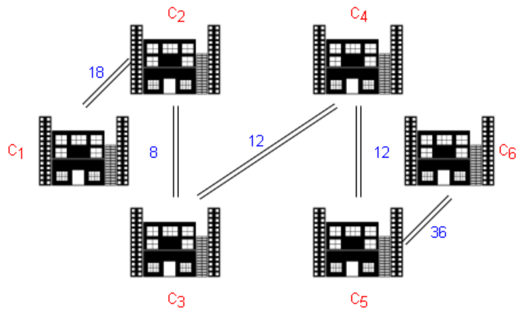
Path 3



{0,18} x {0,8} x {0,12} x {0,9} = {(0,0,0,0), (0,0,0,9), (0,0,12,0), (0,0,12,9), (0,8,0,0), (0,8,0,9), (0,8,12,0), (0,8,12,9), (18,0,0,0), (18,0,0,9), (18,0,12,0), (18,0,12,9), (18,8,0,0), (18,8,0,9), (18,8,12,0), (18,8,12,9)}

Costs = 0, 9, 12, 21, 8, 17, 20, 29, 18, 27, 30, 39, 26, 35, 38, 47

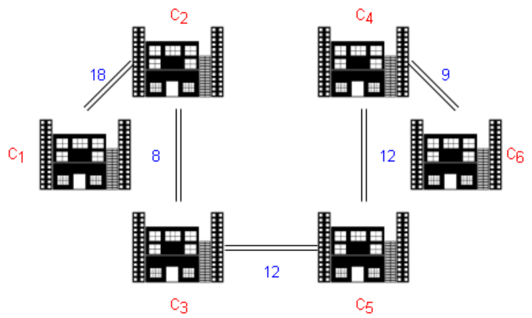
Path 4



{0,18} x {0,8} x {0,12} x {0,12} x {0,36} = {(0,0,0,0,0), (0,0,0,0,36), (0,0,0,12,0), (0,0,0,12,36), (0,0,12,0,0), (0,0,12,0,36), (0,0,12,12,0), (0,0,12,12,36), (0,8,0,0,0), (0,8,0,0,36), (0,8,0,12,0), (0,8,0,12,36), (0,8,12,0,0), (0,8,12,0,36), (0,8,12,12,0), (0,8,12,12,36), (18,0,0,0,0), (18,0,0,0,36), (18,0,0,12,0), (18,0,0,12,36), (18,0,12,0,0), (18,0,12,0,36), (18,0,12,12,0), (18,0,12,12,36), (18,8,0,0,0), (18,8,0,0,36), (18,8,0,12,0), (18,8,0,12,36), (18,8,12,0,0), (18,8,12,0,36), (18,8,12,12,0), (18,8,12,12,36)}

Costs = 0, 36, 12, 48, 12, 48, 24, 60, 8, 44, 20, 56, 20, 56, 32, 68, 18, 54, 30, 66, 30, 66, 42, 78, 26, 62, 38, 74, 38, 74, 50, 86

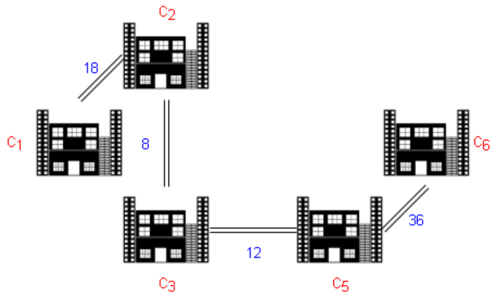
Path 5



{0,18} x {0,8}, {0,12} x {0,12} x {0,9} = {(0,0,0,0,0), (0,0,0,0,9), (0,0,0,12,0), (0,0,0,12,9), (0,0,12,0,0), (0,0,12,0,9), (0,0,12,12,0), (0,0,12,12,9), (0,8,0,0,0), (0,8,0,0,9), (0,8,0,12,0), (0,8,0,12,9), (0,8,12,0,0), (0,8,12,0,9), (0,8,12,12,0), (0,8,12,12,9), (18,0,0,0,0), (18,0,0,0,9), (18,0,0,12,0), (18,0,0,12,9), (18,0,12,0,0), (18,0,12,0,9), (18,0,12,12,0), (18,0,12,12,9), (18,8,0,0,0), (18,8,0,0,9), (18,8,0,12,0), (18,8,0,12,9), (18,8,12,0,0), (18,8,12,0,9), (18,8,12,12,0), (18,8,12,12,9)}

Costs = 0, 9, 12, 21, 12, 21, 24, 33, 8, 17, 20, 29, 20, 29, 32, 41, 18, 27, 30, 39, 30, 39, 42, 51, 26, 35, 38, 47, 38, 47, 50, 59

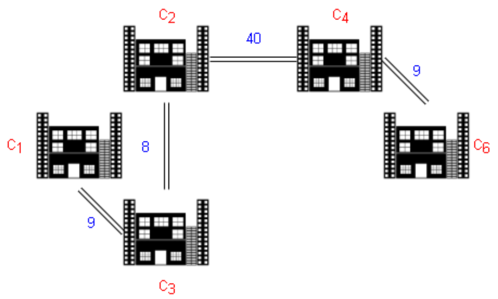
Path 6



{0,18} x {0,8} x {0,12} x {0,36} = {(0,0,0,0), (0,0,0,36), (0,0,12,0), (0,0,12,36), (0,8,0,0), (0,8,0,36), (0,8,12,0), (0,8,12,36), (18,0,0,0), (18,0,0,36), (18,0,12,0), (18,0,12,36), (18,8,0,0), (18,8,0,36), (18,8,12,0), (18,8,12,36)}

Costs = 0, 36, 12, 48, 8, 44, 20, 56, 18, 54, 30, 66, 26, 62, 38, 74

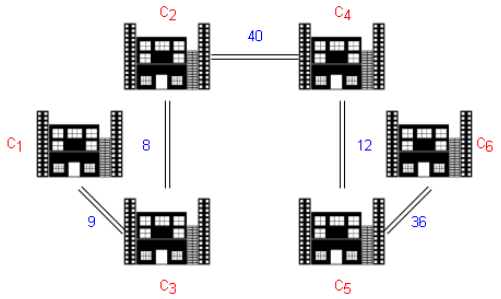
Path 7



{0,9} x {0,8} x {0,9} x {0,40} = {(0,0,0,0), (0,0,0,40), (0,0,9,0), (0,0,9,40), (0,8,0,0), (0,8,0,40), (0,8,9,0), (0,8,9,40), (9,0,0,0), (9,0,0,40), (9,0,9,0), (9,0,9,40), (9,8,0,0), (9,8,0,40), (9,8,9,0), (9,8,9,40)}

Costs = 0, 40, 9, 49, 8, 48, 17, 57, 9, 49, 18, 58, 17, 57, 26, 66

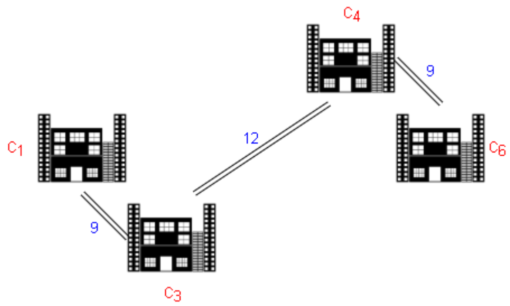
Path 8



{0,40} x {0,8}, {0,12} x {0,36} x {0,9} = {(0,0,0,0,0), (0,0,0,0,9), (0,0,0,36,0), (0,0,0,36,9), (0,0,12,0,0), (0,0,12,0,9), (0,0,12,36,0), (0,0,12,36,9), (0,8,0,0,0), (0,8,0,0,9), (0,8,0,36,0), (0,8,0,36,9), (0,8,12,0,0), (0,8,12,0,9), (0,8,12,36,0), (0,8,12,36,9), (40,0,0,0,0), (40,0,0,0,9), (40,0,0,12,0), (40,0,0,36,9), (40,0,12,0,0), (40,0,12,0,9), (40,0,12,36,0), (40,0,12,36,9), (40,8,0,0,0), (40,8,0,0,9), (40,8,0,36,0), (40,8,0,36,9), (40,8,12,0,0), (40,8,12,0,9), (40,8,12,36,0), (40,8,12,36,9)}

Costs = 0,9, 36, 45, 12, 21, 48, 57, 8, 17, 44, 53, 20, 29, 56, 65, 40, 49, 52, 85, 52, 61, 88, 97, 48, 57, 84, 93, 60, 69, 96, 105

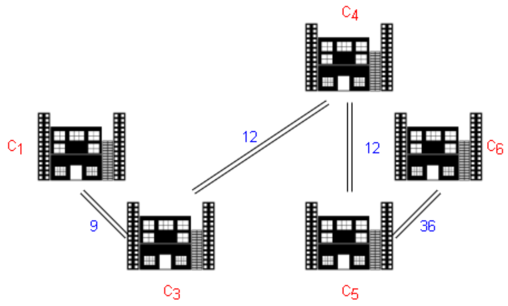
Path 9



{0,9} x {0,12} x {0,9} = {(0,0,0), (0,0,9), (0,12,0), (0,12,9), (9,0,0), (9,0,9), (9,12,0), (9,12,9)}

Costs = 0, 9, 12, 21, 9, 18, 21, 30

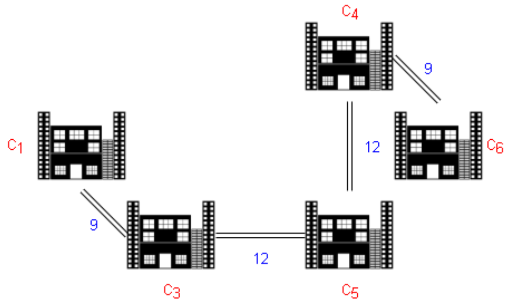
Path 10



{0,9} x {0,12} x {0,12} x {0,36} = {(0,0,0,0), (0,0,0,36), (0,0,12,0), (0,0,12,36), (0,12,0,0), (0,12,0,36), (0,12,12,0), (0,12,12,36), (9,0,0,0), (9,0,0,36), (9,0,12,0), (9,0,12,36), (9,12,0,0), (9,12,0,36), (9,12,12,0), (9,12,12,36)}

Costs = 0, 36, 12, 48, 12, 48, 24, 60, 9, 45, 21, 57, 21, 57, 33, 69

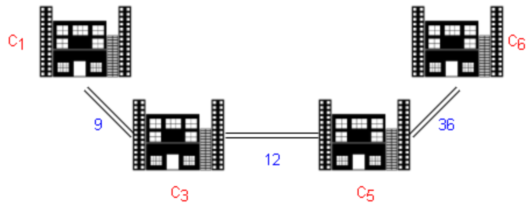
Path 11



{0,9} x {0,12} x {0,12} x {0,9} = {(0,0,0,0), (0,0,0,9), (0,0,12,0), (0,0,12,9), (0,12,0,0), (0,12,0,9), (0,12,12,0), (0,12,12,9), (9,0,0,0), (9,0,0,9), (9,0,12,0), (9,0,12,9), (9,12,0,0), (9,12,0,9), (9,12,12,0), (9,12,12,9)}

Costs = 0, 9, 12, 21, 12, 21, 24, 33, 9, 18, 21, 30, 21, 30, 33, 42

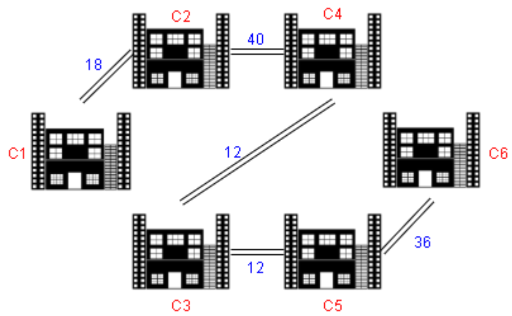
Path 12



{0,9} x {0,12} x {0,36} = {(0,0,0}, (0,0,36), (0,12,0), (0,12,36), (9,0,0), (9,0,36), (9,12,0), (9,12,36)}

Costs = 0, 36, 12, 48, 9, 45, 21, 57

Path 13



{0,18} x {0,40} x {0,12}x {0,12} x {0,36} = {(0,0,0,0,0), (0,0,0,0,36), (0,0,0,12,0), (0,0,0,12,36), (0,0,12,0,0), (0,0,12,0,36), (0,0,12,12,0), (0,0,12,12,36), (0,40,0,0,0), (0,40,0,0,36), (0,40,0,12,0), (0,40,0,12,36), (0,40,12,0,0), (0,40,12,0,36), (0,40,12,12,0), (0,40,12,12,36), (18,0,0,0,0), (18,0,0,0,36), (18,0,0,12,0), (18,0,0,12,36), (18,0,12,0,0), (18,0,12,0,36), (18,0,12,12,0), (18,0,12,12,36), (18,40,0,0,0), (18,40,0,0,36), (18,40,0,12,0), (18,40,0,12,36), (18,40,12,0,0), (18,40,12,0,36), (18,40,12,12,0), (18,40,12,12,36)}

Costs = 0, 36, 12, 48, 12, 48, 24, 60, 40, 76, 52, 88, 52, 88, 64, 100, 18, 54, 30, 66, 30, 66, 42, 78, 58, 94, 70, 106, 70, 106, 82, 118

Basic Information v2.py

```
from statistics import stdev, variance
# import modules for standard deviation and variance
from matplotlib import pyplot
import numpy
from scipy import stats
import csv
import pandas
# import modules for boxplot, gaussian distribution and probability density function


def sort(list):
    for i in range(len(list)):
        for j in range(i, len(list)):
            if list[i] > list[j]:
                # if the ith value is greater than the jth value
                list[j], list[i] = list[i], list[j]
                # swap there places
    mean = 0
    for i in list:
        mean += i
    mean /= len(list)
    return mean, list


# to store cost of each path
choice = int(input(
    "Do you want the solution for the undirected or directed graph\n1.\tUndirected\n2.\tDirected\n> "))
list1 = []
list2 = []
with open('cartesian file.csv', encoding='utf-8-sig') as file:
    count = 0
    for row in csv.reader(file, delimiter=','):
        for value in row:
            if count == 0:
                if value != '':
                    if value == '118':
                        count += 1
                    list1.append(int(value))
            else:
                if value != '':
                    list2.append(int(value))
if choice == 1:
    mean, list = sort(list1)
    print("For the unidrected graph\n")
```

```
else:
    mean, list = sort(list2)
    print("For the directed graph\n")

print("\nThe sorted list is {}\n".format(list))
# print the list when its sorted

if len(list) % 2 == 0:
    m1 = len(list) / 2
    m1 = int(m1)
    middle = (list[m1 - 1] + list[m1]) / 2
    # get middle of list
    first1 = int(len(list) / 4)
    first_quartile = (list[first1] + list[first1 - 1]) / 2
    # get first quartile value
    third1 = int(first1 * 3)
    third_quartile = (list[third1] + list[third1 - 1]) / 2
    # get third quartile value
    iqr = third_quartile - first_quartile
    # get iqr
else:
    mid = len(list) / 2
    middle = list[int(mid) - 1]
    # get middle of list
    first_quartile = int(len(list) / 4)
    third_quartile = int(first_quartile * 3)
    first_quartile = list[first_quartile]
    # get first quartile value
    third_quartile = list[third_quartile]
    # get third quartile value
    iqr = third_quartile - first_quartile
    # get iqr

print("The mean is {}\nThe median is {}\nThe standard deviation is {}\nThe variance is {}\n".format(
    mean, middle, stdev(list), variance(list)))
print("The 1st quartile is {}\nThe 3rd quartile is {}\nThe IQR is {}\n".format(
    first_quartile, third_quartile, iqr))
# print information about the data sets

df = pandas.DataFrame([list], index=['Cost'])
df.T.boxplot(vert=False)
pyplot.subplots_adjust(left=0.25)
pyplot.show()
# plots the box plot for the data sets
```

```
sigma = stdev(list)
# get standard deviation
x = numpy.linspace(mean - 3 * sigma, mean + 3 * sigma, 100)
pyplot.plot(x, stats.norm.pdf(x, mean, sigma))
pyplot.show()
# plot gaussian distribution
```

This is essentially the same as the original 'Basic Information.py' file inspired from however, instead of reading from 'file.csv', I am reading from 'cartesian.csv'.

Path PDF.py

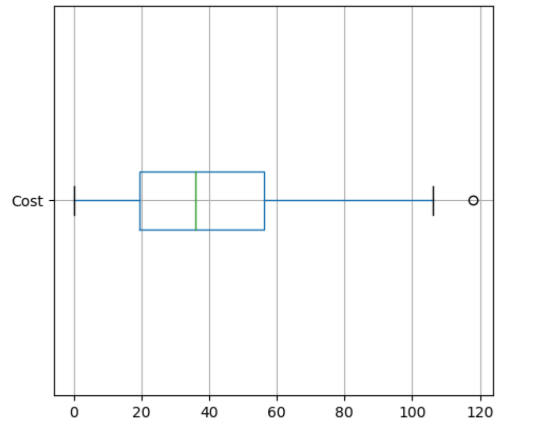
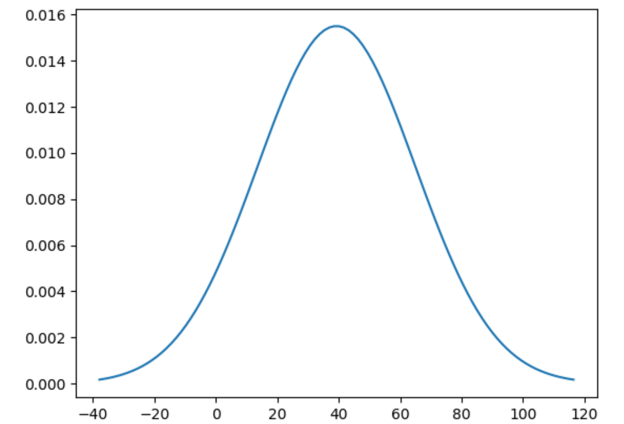
```
import pandas
from matplotlib import pyplot
import seaborn
# import pandas to read in csv and import matplotlib and seaborn for pdf and histogram
file = 'path pdf.csv'
choice = int(input("Would you like to see the PDF of the:\n1.\tUndirected graph\n2.\tDirected graph\n> "))
if choice == 1:
    title = "Undirected Graph PDF"
    column = 0
    bins = 22
else:
    title = "Directed Graph PDF"
    column = 1
    bins = 17
# user chooses the title and column in the csv file
ds = pandas.read_csv(file, usecols=[column])
# choose the data set from the column in the csv file
seaborn.distplot(ds, bins=bins, hist=True, kde=True,
                 hist_kws={'edgecolor': 'black'}, kde_kws={'linewidth': 4})
# plot the histogram and pdf with seaborn
pyplot.title(title)
pyplot.ylabel('Density')
pyplot.xlabel('Cost')
# give the figure a title and labels on the axis
pyplot.show()
# show the graph
```

This is essentially the same as the original PDF.py but instead of reading ‘pdf.csv’, the program is reading in ‘path pdf.csv’.

Undirected graph

```
In [6]: %run "basic information v2.py"

Do you want the solution for the undirected or directed graph
1.      Undirected
2.      Directed
> 1
For the unidrected graph


The sorted list is [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 8, 8, 8, 8, 8, 8, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 12, 12, 1
2, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 17, 17, 17, 17, 17, 18, 18, 18, 18, 18, 18, 18, 18, 18, 18, 20, 20, 20,
20, 20, 20, 20, 21, 21, 21, 21, 21, 21, 21, 21, 21, 21, 21, 21, 21, 24, 24, 24, 24, 24, 26, 26, 26, 26, 26, 27, 27, 27, 2
9, 29, 29, 29, 30, 30, 30, 30, 30, 30, 30, 30, 30, 30, 30, 32, 32, 33, 33, 33, 33, 35, 35, 36, 36, 36, 36, 36, 36, 36, 38,
38, 38, 38, 38, 38, 39, 39, 40, 40, 40, 40, 40, 41, 42, 42, 42, 42, 44, 44, 44, 45, 45, 45, 47, 47, 47, 48, 48, 48, 48, 4
8, 48, 48, 48, 48, 48, 48, 49, 49, 49, 49, 50, 50, 51, 52, 52, 52, 52, 52, 53, 54, 54, 54, 54, 56, 56, 56, 56, 57, 57, 57,
57, 57, 57, 57, 58, 58, 58, 58, 59, 60, 60, 60, 60, 61, 62, 62, 64, 65, 66, 66, 66, 66, 66, 66, 66, 67, 68, 69, 69, 70, 7
0, 70, 74, 74, 74, 76, 76, 78, 78, 82, 84, 85, 86, 88, 88, 88, 88, 93, 94, 94, 96, 97, 100, 105, 106, 106, 106, 118]

The mean is 39.27459016393443
The median is 36.0
The standard deviation is 25.724610117064255
The variance is 661.7555656749646

The 1st quartile is 19.0
The 3rd quartile is 56.5
The IQR is 37.5
```

Figure 54

Box plot	Gaussian Distribution
 *Figure 55*	 *Figure 56*

```
In [7]: %run "path pdf.py"

Would you like to see the PDF of the:
1.      Undirected graph
2.      Directed graph
> 1
```

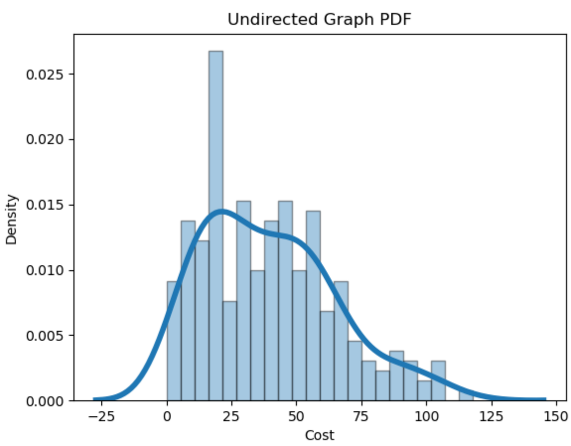


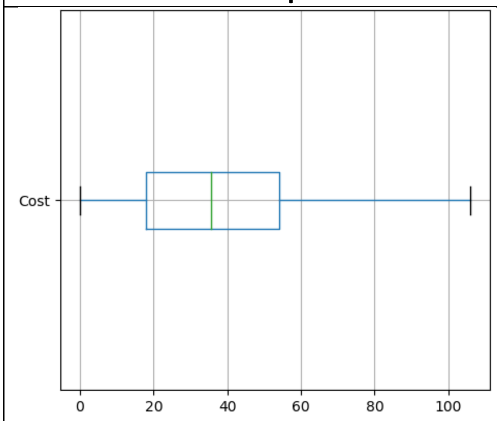
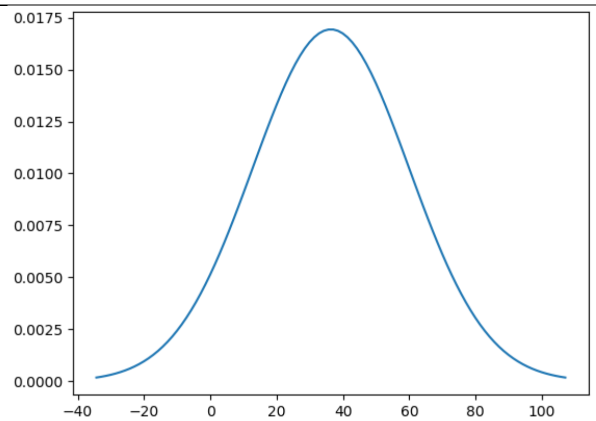
Figure 57

The mean is greater than the median therefore the distribution should appear to be positively skewed which is clearly displayed in the probability density function. The circles in the boxplot represent outliers which don't coincide with the distribution of data. When calculating for outliers (which is approximately 1.5 times the standard deviation from the mean) I found that any costs below approximately 1.5 and above 76.5 could cause the distribution to be skewed in a given direction. As most of the concentration of data is within the region closer to 0, I can assume that outliers below 1.5 contributed to giving the distribution a positive skewness. However, just from the graph I can infer that the majority of paths have a cost which lies between the 0 and 25 bin as the density of that area is much greater than any other point on the graph.

Directed graph

```
In [6]: %run "basic information v2.py"

Do you want the solution for the undirected or directed graph
1.      Undirected
2.      Directed
> 2
For the directed graph


The sorted list is [0, 0, 0, 0, 0, 0, 0, 0, 8, 8, 8, 9, 9, 9, 9, 9, 9, 12, 12, 12, 12, 12, 12, 12, 12, 12, 17, 18, 18, 18,
18, 18, 18, 20, 20, 20, 20, 21, 21, 21, 21, 21, 21, 24, 24, 26, 26, 26, 27, 27, 29, 30, 30, 30, 30, 30, 30, 32, 33, 35, 3
6, 36, 36, 36, 36, 38, 38, 38, 38, 39, 40, 40, 42, 44, 44, 45, 45, 47, 48, 48, 48, 48, 48, 48, 48, 49, 50, 52, 54, 54, 54,
56, 56, 56, 57, 57, 57, 58, 58, 60, 60, 62, 62, 66, 66, 66, 66, 67, 68, 69, 70, 74, 74, 74, 76, 78, 86, 88, 94, 106]

The mean is 36.333333333333336
The median is 35.5
The standard deviation is 23.56444171130023
The variance is 555.2829131652661

The 1st quartile is 18.0
The 3rd quartile is 54.0
The IQR is 36.0
```

Figure 58

Box plot	Gaussian Distribution
 *Figure 59*	 *Figure 60*

```
In [7]: %run "path pdf.py"

Would you like to see the PDF of the:
1.      Undirected graph
2.      Directed graph
> 2

C:\Users\kamra\Anaconda3\lib\site-packages\numpy\lib\histograms.py:824: R
er_equal
  keep = (tmp_a >= first_edge)
C:\Users\kamra\Anaconda3\lib\site-packages\numpy\lib\histograms.py:825: R
equal
  keep &= (tmp_a <= last_edge)
C:\Users\kamra\Anaconda3\lib\site-packages\statsmodels\nonparametric\kde.
in greater
  X = X[np.logical_and(X > clip[0], X < clip[1])] # won't work for two co
C:\Users\kamra\Anaconda3\lib\site-packages\statsmodels\nonparametric\kde.
in less
  X = X[np.logical_and(X > clip[0], X < clip[1])] # won't work for two co
```

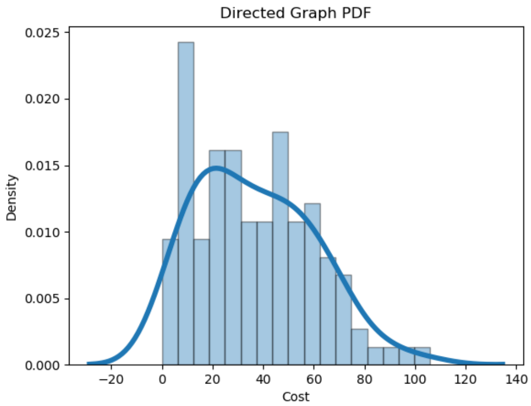


Figure 61

The mean is greater than the median therefore the distribution should appear to be positively skewed which is clearly displayed in the probability density function. When calculating for outliers (which is approximately 1.5 times the standard deviation from the mean) I found that any costs below approximately 0.99 and above 71.68 could cause the distribution to be skewed in a given direction. As most of the concentration of data is within the region closer to 0, I can assume that outliers below 0.99 contributed to giving the distribution a positive skewness. However, just from looking at the graph I can infer that the highest concentration of costs is within bins 10 to 40 as the density for that region is much higher than any other region of the probability density function. This means that most paths have a cost lying within this region.

Conclusion

The mean in the undirected graph is fractionally smaller than the mean in the directed graph but the means are very similar however, the median for the undirected graph is greater than the median for the directed graph.

Although the directed graph has less data points in the distribution than the undirected graph, the first quartile is the same in both graphs but the third quartile and interquartile range are much larger in the directed graph. From this I can infer that the data in the directed graph is more distributed than the undirected graph as more data points lie within the region enclosed by the directed graphs inter-quartile range than the undirected graphs inter-quartile range. This is further emphasised by the probability density functions as the directed graph has less kutosis than the undirected graph to which, in the directed graph the concentration appears to gradually build nearer the mean but in the undirected graph the concentration appears to build nearer the mean faster thus making the distribution appear sharper.

Given that outliers are not included I have concluded that the optimistic cost of transport would be a path giving a cost of 8 and the pessimistic cost would be a path giving a cost of 106. However, if the outliers are included then the optimistic cost of transport would be a path giving a cost of 0 and the pessimistic cost would be a path giving a cost of 118. This is applicable for both scenarios as the values I have chosen for optimistic and pessimistic costs are in both graphs.

Bibliography

Bhatti, K. M. (2019, December 8). *Discrete Maths*. Retrieved from GitHub: https://github.com/k5924/DiscreteMaths

Koehrsen, W. (2018, March 23). *Histograms and Density Plots in Python*. Retrieved from Towards Data Science: https://towardsdatascience.com/histograms-and-density-plots-in-python-f6bda88f5ac0

senshin. (2015, December 21). *Why does my Python code print the extra characters "ï»¿" when reading from a text file?* Retrieved from stackoverflow: https://stackoverflow.com/questions/34399172/why-does-my-python-code-print-the-extra-characters-%C3%AF-when-reading-from-a-tex/34399309

Sullivan, I. (2017, June 14). *Implementation of dijkstra in python*. Retrieved from Youtube: https://www.youtube.com/watch?v=IG1QioWSXRI&t=578s

unutbu. (2012, April 13). *python pylab plot normal distribution*. Retrieved from stackoverflow: https://stackoverflow.com/questions/10138085/python-pylab-plot-normal-distribution

unutbu. (2013, August 28). *Horizontal box plots in matplotlib/Pandas*. Retrieved from stackoverflow: https://stackoverflow.com/questions/18500011/horizontal-box-plots-in-matplotlib-pandas

Wang, Y. (2019, August 20). *18B_Linear_Programming_using_PuLP.* Retrieved from GitHub: https://github.com/yongtwang/engineering-python/tree/master/18B_Linear_Programming_using_PuLP