Assignment 3
Of
Modelling & Simulation Lab (CS1052)

Masters of Technology in Computer Science And Engineering

submitted by
Arghya Bandyopadhyay
RollNo. 20CS4103

submitted to
Dr Nanda Dulal Jana
Assistant Professor
Dept. of CSE

National Institute of Technology, Durgapur

Problem 1

Problem statement 1.

Formulate the following

	D_1	D_2	D_3	D_4	supply
O_1	6	4	1	5	14
O_2	8	9	2	7	12
O_3	4	3	6	2	4
Demand	6	10	10	4	

Also determine an initial basic feasible solution of this TP using least cost method.

Problem formulation

	D_1	D_2	D_3	D_4	supply
O_1	x_{11} 6	x_{12} 4	x_{13} 1	x_{14} 5	14
O_2	x_{21} 8	x_{22} 9	x_{23} 2	x_{24} 7	12
O_3	x_{31} 4	x_{32} 3	x_{33} 6	x_{34} 2	4
Demand	6	10	10	4	

The transportation problem is formulated as an LP model as follows:-

Minimize (Total transportation cost) =

$$6x_{11} + 4x_{12} + x_{13} + 5x_{14} +$$
$$8x_{21} + 9x_{22} + 2x_{23} + 7x_{24} +$$
$$4x_{31} + 3x_{32} + 6x_{33} + 2x_{34}$$

Subject to the constraints

$$x_{11} + x_{12} + x_{13} + x_{14} = 14$$
$$x_{21} + x_{22} + x_{23} + x_{24} = 12$$
$$x_{31} + x_{32} + x_{33} + x_{34} = 4$$

and

$$x_{11} + x_{21} + x_{31} = 6$$
$$x_{12} + x_{22} + x_{32} = 10$$
$$x_{13} + x_{23} + x_{33} = 10$$
$$x_{14} + x_{24} + x_{34} = 4$$

and $x_{ij} \geq 0$ for $i = 1, 2, 3$ &

$j = 1, 2, 3, 4$

In the above LP model, there are $m \times n = 3 \times 4 = 12$ decision variables, x_{ij} & $m+n = 7$ constraints, where m are the numbers of rows and n are the numbers of columns.

<u>Existence of feasible soln</u>

Total supply = Total demand

$$\Rightarrow \sum_{i=1}^{m} a_i = \sum_{j=1}^{n} b_j$$

$$\Rightarrow 14 + 12 + 4 = 6 + 10 + 10 + 4$$

$$\Rightarrow 30 = 30$$

The total supply is equal to total demand, hence the problem is a balanced transportation problem.

<u>Solution</u>:-

(i) Select minimum of cost matrix i.e at (O_1, D_3) and then compare

	D_1	D_2	D_3	D_4	Supply
O_1	6	4 (4)	1 (10)	5	~~14~~ ~~4~~ 0 $= a_1$
O_2	8 (6)	9 (6)	2	7	~~12~~ ~~6~~ 0 $= a_2$
O_3	4	3	6	2 (4)	~~4~~ 0 $= a_3$
Demand	~~6~~ 0 $= b_1$	~~10~~ ~~6~~ 0 $= b_2$	~~10~~ 0 $= b_3$	~~4~~ 0 $= b_4$	

a_1 and b_3 i.e $10 < 14$, hence allocate $x_{13} = 10$, b_3 gets exhausted and $a_1 = 4$ delete column D_3

(2) Again select the minimum cost from the new cost matrix with D_3 removed, i.e 2 at (O_3, D_4) and then compare a_3 & b_4 where $4 = 4$, allocate $x_{34} = 4$, this exhausts a_3 & $b_4 = 0$, delete O_3 & D_4.

(3) The next minimum is 4 at (O_1, D_2) comparing a_1 & b_2 i.e $4 < 10$, allocate $x_{12} = 4$ & this exhaust $a_1 = 0$ & leaves $b_2 = 6$. delete row O_1

(4) select 8 at (O_2, Q_1). compare a_2 & b_1 i.e $6 < 12$, allocate $x_{21} = 6$ & exhaust b_1 & leave $b_2 = 6$, then select 9 i.e at (O_2, D_2) & allocate remaining value i.e $x_{22} = 6$. this exhausts all the supply & demand.

Allocation matrix -

	D_1	D_2	D_3	D_4	Supply
O_1	0	4	10	0	14
O_2	6	6	0	0	12
O_3	0	0	0	4	4
Demand	6	10	10	4	30

Total cost = $4 \times 4 + 1 \times 10 + 8 \times 6 + 9 \times 6 + 2 \times 4$

$= 16 + 10 + 48 + 54 + 8$

$= 136$

The number of allocations = 5, & $m + n - 1 = 6 \Rightarrow 5 \neq 6$ hence the solution is degenerate

To solve degeneracy problem we have to select a position in the matrix which is unallocated & also have maximum cost & donot create a loop.

So if we put the value epsilon at (O_1, D_1) or (O_2, D_3), it will create loop whether rest of the position is independent ~~of~~ and we can put epsilon on the position with minimum cost i.e 3 at (O_3, D_2)

	D_1	D_2	D_3	D_4	Supply
O_1	0	4	10	0	14
O_2	6	6	0	0	12
O_3	0	ϵ	0	4	4
Demand	6	10	10	4	0

Total cost $= 4\times4 + 10\times1 + 6\times8 + 6\times9$
$+ 3\times\epsilon + 4\times2$

$= 16 + 10 + 48 + 54 + 8 + 3\epsilon$

$= 136 + 3\epsilon$

$\simeq 136$ where $\epsilon = 0$

Python Code:

```
import numpy as np

def check_loop(p, row, column):
        p[row, column] = -1
        flag = 1
        while flag != 0:
                flag = 0
                if p.size != 0:
                        row = np.count_nonzero(p, axis=1)
                        f = 0
                        for index in range(len(row)):
                                if row[index] < 2:
                                        flag = 1
                                        p = np.delete(p, (index - f), axis=0)
                                        f += 1
                if p.size != 0:
                        e = 0
                        col = np.count_nonzero(p, axis=0)
                        for index in range(len(col)):
                                if col[index] < 2:
                                        flag = 1
                                        p = np.delete(p, (index - e), axis=1)
                                        e += 1
        if p.size != 0:
                return 0
        else:
                return 1

if __name__ == '__main__':
        cm = np.array([[6.0, 4.0, 1.0, 5.0],
                [8.0, 9.0, 2.0, 7.0],
                [4.0, 3.0, 6.0, 2.0]])
        s = np.array([14.0, 12.0, 4.0])
        d = np.array([6.0, 10.0, 10.0, 4.0])
        c = cm.copy()
        print("The Cost Matrix is: ")
        print(c)
        print("The Supply is: ", s)
        print("The Demand is: ", d)
        m, n = c.shape
        print("No of Rows & No of Columns: (", m, ", ", n, ")")
        total_cost = 0
        no_alloc = 0
        total_demand = np.sum(d)
        total_supply = np.sum(s)
        alloc = []
        if total_demand == total_supply:
                print("It is a Balanced Transportation Problem")
```

```
else:
        print("It is an UnBalanced Transportation Problem")
        if total_demand > total_supply:
                new = np.array(np.zeros(n))
                c = np.row_stack((c, new))
                s = np.append(s, total_demand - total_supply)
                m = m + 1
        else:
                new = np.array(np.zeros(m))
                c = np.column_stack((c, new))
                d = np.append(d, total_supply - total_demand)
                n = n + 1
        print("The New Balanced Cost Matrix is: ")
        print(c)
        print("The Supply is: ", s)
        print("The Demand is: ", d)
a = np.zeros(c.shape)
min_cost = np.amin(c)
while min_cost != np.inf:
        indexes = np.where(c == min_cost)
        i = indexes[0][0]
        j = indexes[1][0]
        x = min(s[i], d[j])
        s[i] -= x
        d[j] -= x
        total_cost += (x * c[i, j])
        no_alloc += 1
        a[i, j] = x
        alloc.append((i, j))
        if s[i] < d[j]:
                x = 0
                while x < n:
                        c[i, x] = np.inf
                        x += 1
        elif s[i] > d[j]:
                y = 0
                while y < m:
                        c[y, j] = np.inf
                        y += 1
        else:
                x = 0
                while x < n:
                        c[i, x] = np.inf
                        x += 1
                y = 0
                while y < m:
                        c[y, j] = np.inf
                        y += 1
        min_cost = np.amin(c)
print("Total Cost: ", total_cost)
unalloc = []
```

```
for i in range(m):
        for j in range(n):
                if not (i, j) in alloc:
                        unalloc.append((i, j))
print("List of Allocated Positions: ", alloc)
print("List of Unallocated Positions: ", unalloc)
print("Allocation Matrix: ")
print(a)
no_loop = []
if no_alloc == m + n - 1:
        print("Non Degeneracy")
else:
        print("Degeneracy")
        for i in unalloc:
                g = check_loop(a.copy(), i[0], i[1])
                if g == 1:
                        no_loop.append(i)
                min_epi_list = []
        for i in no_loop:
                min_epi_list.append(cm[i[0], i[1]])
        min_epi = min(min_epi_list)
        ind = min_epi_list.index(min_epi)
        loc = no_loop[ind]
        a[loc[0], loc[1]] = -1
        print("Allocation Matrix After Converting
                Degeneracy to Non-Degeneracy is : ")
        print(a)
```

Output :

```
The Cost Matrix is:
[[6. 4. 1. 5.]
 [8. 9. 2. 7.]
 [4. 3. 6. 2.]]
The Supply is:  [14. 12.  4.]
The Demand is:  [ 6. 10. 10.  4.]
No of Rows & No of Columns: ( 3 ,  4 )
It is a Balanced Transportation Problem
Total Cost:  136.0
List of Allocated Positions:  [(0, 2), (2, 3), (0, 1), (1, 0), (1, 1)]
List of Unallocated Positions:  [(0, 0), (0, 3), (1, 2), (1, 3), (2, 0), (2, 1), (2, 2)]
Allocation Matrix:
[[ 0.  4. 10.  0.]
 [ 6.  6.  0.  0.]
 [ 0.  0.  0.  4.]]
Degeneracy
Allocation Matrix After Converting Degeneracy to Non-Degeneracy is :
[[ 0.  4. 10.  0.]
 [ 6.  6.  0.  0.]
 [ 0. -1.  0.  4.]]
```

Problem statement

Formulate the following TP & find IBFS by LCM.

	1	2	3	4	5	Available
A	4	3	1	2	6	80
B	5	2	3	4	5	40
C	3	5	6	3	2	40
D	2	4	4	5	3	20
Required	60	60	30	40	10	200/180

Problem formulation

The transportat^{n} problem is formulated as an LP as follows:-

	1	2	3	4	5	Available
A	x_{11} 4	x_{12} 3	x_{13} 1	x_{14} 2	x_{15} 6	80
B	x_{21} 5	x_{22} 2	x_{23} 3	x_{24} 4	x_{25} 5	40
C	x_{31} 3	x_{32} 5	x_{33} 6	x_{34} 3	x_{35} 2	40
D	x_{41} 2	x_{42} 4	x_{43} 4	x_{44} 5	x_{45} 3	20
Required	60	60	30	40	10	200/180

Existence of feasible solution

$\Rightarrow \sum_{i=1}^{m} a_i = \sum_{j=1}^{n} b_j$

$\Rightarrow 80 + 40 + 40 + 20 = 60 + 60 + 30 + 40 + 10$

$180 \neq 200$

Hence the ~~solution~~ problem is unbalanced transportation problem, so to solve this question, we need to insert a dummy row E, with cost (0) to all the cells and availability = |Total demand − Total supply| = 20.

Hence the new cost matrix =

	1	2	3	4	5	Avail.
A	x_{11} 4	x_{12} 3	x_{13} 1	x_{14} 2	x_{15} 6	80
B	x_{21} 5	x_{22} 2	x_{23} 3	x_{24} 4	x_{25} 5	40
C	x_{31} 3	x_{32} 5	x_{33} 6	x_{34} 3	x_{35} 2	40
D	x_{41} 2	x_{42} 4	x_{43} 4	x_{44} 5	x_{45} 3	20
E	x_{51} 0	x_{52} 0	x_{53} 0	x_{54} 0	x_{55} 0	20
Required	60	60	30	40	10	200

Total cost $= 4x_{11} + 3x_{12} + 1x_{13} + 2x_{14} + 6x_{15} +$
$5x_{21} + 2x_{22} + 3x_{23} + 4x_{24} + 5x_{25} +$
$3x_{31} + 5x_{32} + 6x_{33} + 3x_{34} + 2x_{35} +$
$2x_{41} + 4x_{42} + 4x_{43} + 5x_{44} + 3x_{45} +$
$0x_{51} + 0x_{52} + 0x_{53} + 0x_{54} + 0x_{55}$

Subject to constraint

$x_{11} + x_{12} + x_{13} + x_{14} + x_{15} = 80$

$x_{21} + x_{22} + x_{23} + x_{24} + x_{25} = 40$

$x_{31} + x_{32} + x_{33} + x_{34} + x_{35} = 40$

$x_{41} + x_{42} + x_{43} + x_{44} + x_{45} = 20$

$x_{51} + x_{52} + x_{53} + x_{54} + x_{55} = 20$

$x_{11} + x_{21} + x_{31} + x_{41} + x_{51} = 60$

$x_{12} + x_{22} + x_{32} + x_{42} + x_{52} = 60$

$x_{13} + x_{23} + x_{33} + x_{43} + x_{53} = 30$

$x_{14} + x_{24} + x_{34} + x_{44} + x_{54} = 40$

$x_{15} + x_{25} + x_{35} + x_{45} + x_{55} = 10$.

and $x_{ij} \geq 0$ for $i = 1, 2, 3, 4, 5$ &
$j = 1, 2, 3, 4, 5$.

In the above model, there are $5 \times 5 = 25$ decision variables & $m+n = 5+5 = 10$ constraints, where m = rows & n columns.

Solution

	1	2	3	4	5	Avail.
A	4	3 (10)	1 (30)	2 (40)	6	~~80~~ ~~50~~ ~~10~~ 0 a_1
B	5	2 (40)	3	4	5	~~40~~ 0 a_2
C	3 (20)	5 (10)	6	3	2 (10)	~~40~~ ~~30~~ ~~10~~ 0 a_3
D	2 (20)	4	4	5	3	~~20~~ 0 a_4
E	0 (20)	0	0	0	0	~~20~~ 0 a_5
Required	~~60~~ ~~40~~ ~~20~~ 0 $= b_1$	~~60~~ ~~30~~ ~~10~~ 0 $= b_2$	~~30~~ 0 $= b_3$	~~40~~ 0 $= b_4$	~~10~~ 0 $= b_5$	

(1) The minimum of the cost matrix i.e 0 at (E, 1) and then compare a_5 & b_1 $\Rightarrow$ $b_1 > a_5$ hence allocate $x_{51} = 20$ which exhausts a_5 hence delete row E.

② Similarly select 1 at $(A, 3)$ allocate $x_{13} = 30$ $\because$ $b_3 < a_1$. this exhausts b_3 hence delete column 3.

③ select 2 at $(A, 4)$, allocate $x_{14} = 40$. $\because$ $b_4 < a_1$, this exhausts b_4 hence delete column 4.

④ select 2 at $(B, 2)$ allocate $x_{22} = 40$ $\because$ $b_2 > a_2$, this exhausts a_2 hence delete row B.

⑤ select 2 at $(C, 5)$, $x_{35} = 10$, delete column 5^{th} since $b_5 < a_3$ and b_5 gets exhausted

⑥ select 2 at $(D, 1)$, $x_{41} = 20$, delete row D $\because$ $b_1 > a_4$ and a_4 gets exhausted.

⑦ ~~sto~~ select 3 at $(A, 2)$ $x_{12} = 10$, delete row A $\because$ $b_2 > a_1$ and a_1 gets exhausted.

(5) At last select min = 3 at (C, 1) $x_{31} = 20$ delete column 1 and left with min = 5 at (C, 2) $x_{32} = 10$, & delete 2nd column & All the available & required gets exhausted.

Allocation matrix

	1	2	3	4	5	available
A	0	10	30	40	0	80
B	0	40	0	0	0	40
C	20	10	0	0	10	40
D	20	0	0	0	0	20
E	20	0	0	0	0	20
Required	60	60	30	40	10	200

No of allocation = 9 & $m+n-1 = 5+5-1 = 9$ hence the soln is non degenerate.

Total cost = $(10*3) + (30*1) + (40*2) + (40*2) + (20*3) + (10*5) + (10*2) + (20*2) + (20*0)$

$= 30 + 30 + 80 + 80 + 60 + 50 + 20 + 40 + 0$

$= 390$

Python Code:

```
import numpy as np

def check_loop(p, row, column):
        p[row, column] = -1
        flag = 1
        while flag != 0:
                flag = 0
                if p.size != 0:
                        row = np.count_nonzero(p, axis=1)
                        f = 0
                        for index in range(len(row)):
                                if row[index] < 2:
                                        flag = 1
                                        p = np.delete(p, (index - f), axis=0)
                                        f += 1
                if p.size != 0:
                        e = 0
                        col = np.count_nonzero(p, axis=0)
                        for index in range(len(col)):
                                if col[index] < 2:
                                        flag = 1
                                        p = np.delete(p, (index - e), axis=1)
                                        e += 1
        if p.size != 0:
                return 0
        else:
                return 1

if __name__ == '__main__':
        cm = np.array([
                [4.0, 3.0, 1.0, 2.0, 6.0],
                [5.0, 2.0, 3.0, 4.0, 5.0],
                [3.0, 5.0, 6.0, 3.0, 2.0],
                [2.0, 4.0, 4.0, 5.0, 3.0]])
        s = np.array([80.0, 40.0, 40.0, 20.0])
        d = np.array([60.0, 60.0, 30.0, 40.0, 10.0])
        c = cm.copy()
        print("The Cost Matrix is: ")
        print(c)
        print("The Supply is: ", s)
        print("The Demand is: ", d)
        m, n = c.shape
        print("No of Rows & No of Columns: (", m, ", ", n, ")")
        total_cost = 0
        no_alloc = 0
        total_demand = np.sum(d)
        total_supply = np.sum(s)
        alloc = []
```

```
if total_demand == total_supply:
        print("It is a Balanced Transportation Problem")
else:
        print("It is an UnBalanced Transportation Problem")
        if total_demand > total_supply:
                new = np.array(np.zeros(n))
                c = np.row_stack((c, new))
                s = np.append(s, total_demand - total_supply)
                m = m + 1
        else:
                new = np.array(np.zeros(m))
                c = np.column_stack((c, new))
                d = np.append(d, total_supply - total_demand)
                n = n + 1
        print("The New Balanced Cost Matrix is: ")
        print(c)
        print("The Supply is: ", s)
        print("The Demand is: ", d)
a = np.zeros(c.shape)
min_cost = np.amin(c)
while min_cost != np.inf:
        indexes = np.where(c == min_cost)
        i = indexes[0][0]
        j = indexes[1][0]
        x = min(s[i], d[j])
        s[i] -= x
        d[j] -= x
        total_cost += (x * c[i, j])
        no_alloc += 1
        a[i, j] = x
        alloc.append((i, j))
        if s[i] < d[j]:
                x = 0
                while x < n:
                        c[i, x] = np.inf
                        x += 1
        elif s[i] > d[j]:
                y = 0
                while y < m:
                        c[y, j] = np.inf
                        y += 1
        else:
                x = 0
                while x < n:
                        c[i, x] = np.inf
                        x += 1
                y = 0
                while y < m:
                        c[y, j] = np.inf
                        y += 1
        min_cost = np.amin(c)
print("Total Cost: ", total_cost)
unalloc = []
```

```
for i in range(m):
        for j in range(n):
                if not (i, j) in alloc:
                        unalloc.append((i, j))
print("List of Allocated Positions: ", alloc)
print("List of Unallocated Positions: ", unalloc)
print("Allocation Matrix: ")
print(a)
no_loop = []
if no_alloc == m + n - 1:
        print("Non Degeneracy")
else:
        print("Degeneracy")
        for i in unalloc:
                g = check_loop(a.copy(), i[0], i[1])
                if g == 1:
                        no_loop.append(i)
        min_epi_list = []
        for i in no_loop:
                min_epi_list.append(cm[i[0], i[1]])
        min_epi = min(min_epi_list)
        ind = min_epi_list.index(min_epi)
        loc = no_loop[ind]
        a[loc[0], loc[1]] = -1
        print("Allocation Matrix After Converting
                Degeneracy to Non-Degeneracy is : ")
        print(a)
```

Output :

```
The Cost Matrix is:
[[4. 3. 1. 2. 6.]
 [5. 2. 3. 4. 5.]
 [3. 5. 6. 3. 2.]
 [2. 4. 4. 5. 3.]]
The Supply is:  [80. 40. 40. 20.]
The Demand is:  [60. 60. 30. 40. 10.]
No of Rows & No of Columns: ( 4 ,  5 )
It is an UnBalanced Transportation Problem
The New Balanced Cost Matrix is:
[[4. 3. 1. 2. 6.]
 [5. 2. 3. 4. 5.]
 [3. 5. 6. 3. 2.]
 [2. 4. 4. 5. 3.]
 [0. 0. 0. 0. 0.]]
```

```
The Supply is:  [80. 40. 40. 20. 20.]

The Demand is:  [60. 60. 30. 40. 10.]

Total Cost:  390.0

List of Allocated Positions:  [(4, 0), (0, 2), (0, 3), (1, 1), (2, 4), (3, 0), (0, 1), (2, 0), (2, 1)]

List of Unallocated Positions:  [(0, 0), (0, 4), (1, 0), (1, 2), (1, 3), (1, 4), (2, 2), (2, 3), (3, 1), (3, 2), (3, 3), (3, 4), (4, 1), (4, 2), (4, 3), (4, 4)
```

```
Allocation Matrix:
[[ 0. 10. 30. 40.  0.]
 [ 0. 40.  0.  0.  0.]
 [20. 10.  0.  0. 10.]
 [20.  0.  0.  0.  0.]
 [20.  0.  0.  0.  0.]]
Non Degeneracy
```