Hitchcock - Koopmans problem for Vaccine Distribution

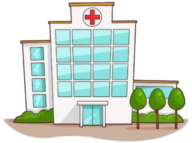
IT300 Course Project

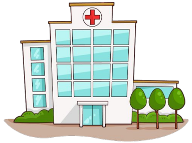
Presented by - Aniruddh Sujish - 181IT206
Krithik Vaidya - 181IT124
Josson Joe Thoppil - 181IT121
Mohith R - 181IT128

Problem Statement

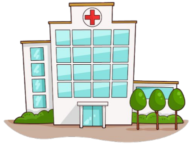
The problem setting of the Hitchcock-Koopmans transportation problem is that goods are to be transported from a set of sources to a set of destinations subject to the supply and demand of the sources and destination respectively such that the total cost of transportation is minimized.

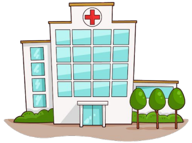
Vaccine Supply Setting

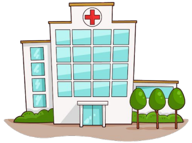




This is a system to minimise Transportation Cost. We are modelling this into a vaccination supply scenario. There will be a few vaccine suppliers who supply the vaccine and vaccine purchasers who purchases them and stores them. Since there are many potential supplier and buyer locations, we have different costs for different suppliers to different purchasers. So, the objective is to try and meet the supplies with the demands with the least cost, so that we can effectively manage the COVID-19 Pandemic.







A simple example

		DESTINATIONS			SUPPLY
		1	2	3	
SOURCE	A	2	3	1	10
	B	5	4	8	35
	C	5	6	8	25
DEMAND		20	25	25	

Optimal Allocation:

```
Allocation Matrix

0        0        10
10       25       0
10       0        15
```

The optimal least cost for transportation is 330

Steps to solve it

1. Check if supply = demand (i.e. If the problem is balanced)
   a. If Balanced, go to 2
   b. If Unbalanced, Balance it using a dummy row/control and go to 2.
2. Use any of the 3 basic methods listed below to arrive at an approximate solution
   a. North West Corner Method
   b. Least Cost Method
   c. Vogel's Approximation Method
3. Check for and fix the Degeneracy issue.
4. Optimise the above initial solution using one of the 2 methods:
   a. Stepping Stone Method
   b. MODI Method

Balancing the problem

Add a dummy row for excess demand or dummy column for excess supply

		DESTINATIONS			SUPPLY
		1	2	3	
SOURCE	A	2	3	1	20
	B	5	4	8	15
	C	5	6	8	30
DEMAND		20	15	25	60 / 65

→

		DESTINATIONS				SUPPLY
		1	2	3	DUMMY DESTINATION	
SOURCE	A	2	3	1	0	20
	B	5	4	8	0	15
	C	5	6	8	0	30
DEMAND		20	15	25	65-60= 5	65 / 65

Basic Feasible Solutions

North West Corner Method

		DESTINATION			SUPPLY	
		1	2	3		
SOURCE	A	5 (65)	7	8	70-65= 5	
	B	4	4	6	30	
	C	6	7	7	50	
DEMAND		65	42	43	42+43+65= **150**	5+30+50+65= **150**

		DESTINATION		SUPPLY	
		2	3		
SOURCE	A	7 (5)	8	5	
	B	4	6	30	
	C	7	7	50	
DEMAND		42-5= 37	43	37+43+5+65= **150**	30+50+5+65= **150**

```
for (int i = 0, j = 0; i < row && j < col;) {
    if (Supply[i] > Demand[j]) {
        Allocate[i][j] = Demand[j];
        Supply[i] -= Demand[j];
        Demand[j] = 0;
        j++;
    } else {
        Allocate[i][j] = Supply[i];
        Demand[j] -= Supply[i];
        Supply[i] = 0;
        i++;
    }
}
```

Allocation

		DESTINATION			SUPPLY
		1	2	3	
SOURCE	A	65 5	5 7	8	70
	B	4	30 4	6	30
	C	6	7 7	43 7	50
DEMAND		65	42	43	**150** / **150**

Least Cost Method

		DESTINATION			SUPPLY	
		1	2	3		
SOURCE	A	5	7	8	70	
	~~B~~	~~4~~ 30	~~4~~	~~6~~	~~30~~	
	C	6	7	7	50	
DEMAND		65-30= 35	42	43	35+42+43+30= **150**	70+50+30= **150**

		DESTINATION			SUPPLY	
		~~1~~	2	3		
SOURCE	A	~~5~~ 35	7	8	70-35= 35	
	C	~~6~~	7	7	50	
DEMAND		~~35~~	42	43	42+43+30+35= **150**	35+50+30+35= **150**

```c
while (1) {
    minx = miny = -1;
    for (int i = 0, j = 0; j < col || (j = 0, ++i) < row; j++) {
        if (Supply[i] && Demand[j] && (minx < 0 || Cost[i][j] < Cost[minx][miny])) {
            minx = i, miny = j;
        }
    }
    if (minx < 0) {
        break;
    }
    if (Supply[minx] > Demand[miny]) {
        Allocate[minx][miny] = Demand[miny];
        Supply[minx] -= Demand[miny];
        Demand[miny] = 0;
    } else {
        Allocate[minx][miny] = Supply[minx];
        Demand[miny] -= Supply[minx];
        Supply[minx] = 0;
    }
}
```

Allocation

		DESTINATION 1	DESTINATION 2	DESTINATION 3		SUPPLY
SOURCE	A	5 [35]	7 [35]	8		70
SOURCE	B	4 [30]	4	6		30
SOURCE	C	6	7 [7]	7 [43]		50
DEMAND		65	42	43	150	150

Vogel’s Approximation Method

		DESTINATION			SUPPLY	ROW DIFFERENCE
		1	2	3		
SOURCE	A	5	7	8	70	7-5= 2
	~~B~~	~~4~~	30 ~~4~~	~~6~~	~~30~~	4-4= 0
	C	6	7	7	50	7-6= 1
DEMAND		65	42-30= 12	43	65+12+43+30= **150**	70+50+30= **150**
COLUMN DIFFERENCE		5-4= 1	7-4= 3	7-6= 1		

		DESTINATION			SUPPLY	ROW DIFFERENCE
		~~1~~	2	3		
SOURCE	A	65 ~~5~~	7	8	70-65= 5	7-5= 2
	C	~~6~~	7	7	50	7-6= 1
DEMAND		~~65~~	12	43	12+43+30+65= **150**	5+50+30+65= **150**
COLUMN DIFFERENCE		6-5= 1	7-7= 0	8-7= 1		

```
for (int i = 0; i < row; i++)
{
    RowDiff[i] = -1;
    min1 = min2 = -1;
    for (int j = 0; Supply[i] && j < col; j++)
    {
        if (Demand[j])
        {
            if (min1 == -1 || Cost[i][j] < min1)
            {
                min2 = min1;
                min1 = Cost[i][j];
            }
            else if (min2 == -1 || Cost[i][j] < min2)
            {
                min2 = Cost[i][j];
            }
        }
    }
    if (Supply[i] && min1 >= 0)
    {
        RowDiff[i] = min2 > 0 ? min2 - min1 : min1;
        if (maxx == -1 || RowDiff[i] > RowDiff[maxx])
        {
            maxx = i;
        }
    }
}
```

```
if (RowDiff[maxx] > ColDiff[maxy])
{
    maxy = -1;
    for (int i = 0; i < col; i++)
    {
        if (Demand[i] && (maxy == -1 || Cost[maxx][i] < Cost[maxx][maxy]))
        {
            maxy = i;
        }
    }
}
else
{
    maxx = -1;
    for (int i = 0; i < row; i++)
    {
        if (Supply[i] && (maxx == -1 || Cost[i][maxy] < Cost[maxx][maxy]))
        {
            maxx = i;
        }
    }
}

if (maxx < 0 || maxy < 0)
    break;

if (Supply[maxx] > Demand[maxy])
{
    Allocate[maxx][maxy] = Demand[maxy];
    Supply[maxx] -= Demand[maxy];
    Demand[maxy] = 0;
}
else
{
    Allocate[maxx][maxy] = Supply[maxx];
    Demand[maxy] -= Supply[maxx];
    Supply[maxx] = 0;
}
```

Allocation

SOURCE	DESTINATION 1	DESTINATION 2	DESTINATION 3		SUPPLY
A	5 65	7 5	8		70
B	4	4 30	6		30
C	6	7 7	7 43		50
DEMAND	65	42	43	150	150

Optimisation

Loop

		DESTINATION		
		1	2	3
SOURCE	A	5 (65)	7 (5)	8
	B	4	4 (30)	6
	C	6	7 (7)	7 (43)

Loop beginning at (1,0)

Degeneracy In Basic Solution

If number of allocated cells < #supply + #demand - 1, solution is degenerate

To fix degeneracy:

- Step 1: Pick any unallocated cell (i,j) such that there is no loop formed with (i,j) as the starting point.
- Step 2: Allocate this cell with an arbitrary value Epsilon
- Step 3: Proceed with Optimization as if there is no degeneracy
- Step 4: Once optimality condition is satisfied, calculate final solution taking Epsilon = 0

Stepping Stone Method

Stepping Stone Method Steps (Rule)	
Step-1:	Find an initial basic feasible solution using any one of the three methods NWCM, LCM or VAM.
Step-2:	1. Draw a closed path (or loop) from an unoccupied cell. The right angle turn in this path is allowed only at occupied cells and at the original unoccupied cell. Mark (+) and (-) sign alternatively at each corner, starting from the original unoccupied cell. 2. Add the transportation costs of each cell traced in the closed path. This is called net cost change. 3. Repeat this for all other unoccupied cells.
Step-3:	1. If all the net cost change are ≥ 0, an optimal solution has been reached. Now stop this procedure. 2. If not then select the unoccupied cell having the highest negative net cost change and draw a closed path.
Step-4:	1. Select minimum allocated value among all negative position (-) on closed path 2. Assign this value to selected unoccupied cell (So unoccupied cell becomes occupied cell). 3. Add this value to the other occupied cells marked with (+) sign. 4. Subtract this value to the other occupied cells marked with (-) sign.
Step-5:	Repeat Step-2 to step-4 until optimal solution is obtained. This procedure stops when all net cost change ≥ 0 for unoccupied cells.

Calculating Net Cost Change for Each Unallocated (i,j):

Net Cost Change at (1,0) = 4 - 5 +7 - 4 = +2

		DESTINATION		
		1	2	3
SOURCE	A	5 [65] -1	7 [5] +1	8
	B	4 +1	4 [30] -1	6
	C	6	7 [7]	7 [43]

Net Cost Change calculated for all Unallocated Cells

- Net Cost Change for 0,2 = 1.000000
- Net Cost Change for 1,0 = 2.000000
- Net Cost Change for 1,2 = 2.000000
- Net Cost Change for 2,0 = 1.000000

If all values are ≥ 0 then solution is optimal

MODI Method (UV)

MODI Method Steps (Rule)	
Step-1:	Find an initial basic feasible solution using any one of the three methods NWCM, LCM or VAM.
Step-2:	Find u_i and v_j for rows and columns. To start a. assign 0 to u_i or v_j where maximum number of allocation in a row or column respectively. b. Calculate other u_i's and v_j's using $c_{ij}=u_i+v_j$, for all occupied cells.
Step-3:	For all unoccupied cells, calculate $d_{ij}=\left(u_i+v_j\right)-c_{ij}$, .
Step-4:	Check the sign of d_{ij} a. If $d_{ij}<0$, then current basic feasible solution is optimal and stop this procedure. b. If $d_{ij}=0$ then alternative solution exists, with different set allocation and same transportation cost. Now stop this procedure. b. If $d_{ij}>0$, then the given solution is not an optimal solution and further improvement in the solution is possible.

Step-5:	Select the unoccupied cell with the largest value of d_{ij}, and included in the next solution.
Step-6:	Draw a closed path (or loop) from the unoccupied cell (selected in the previous step). The right angle turn in this path is allowed only at occupied cells and at the original unoccupied cell. Mark (+) and (-) sign alternatively at each corner, starting from the original unoccupied cell.
Step-7:	1. Select the minimum value from cells marked with (-) sign of the closed path. 2. Assign this value to selected unoccupied cell (So unoccupied cell becomes occupied cell). 3. Add this value to the other occupied cells marked with (+) sign. 4. Subtract this value to the other occupied cells marked with (-) sign.
Step-8:	Repeat Step-2 to step-7 until optimal solution is obtained. This procedure stops when all $d_{ij} \leq 0$ for unoccupied cells.

Calculating u_i and v_j :

1. Start with $u_1 = 0$
2. $u_i + v_j = Cost_{ij}$

SOURCE		DESTINATION			SUPPLY	
		$v_1 = 5$	$v_2 = 7$	$v_3 = 7$		
		1	*2*	*3*		
$u_1 = 0$	A	5 (65)	7 (5)	8	70	
$u_2 = -3$	B	4	4 (30)	6	30	
$u_3 = 0$	C	6	7 (7)	7 (43)	50	
DEMAND		65	42	43	**150**	**150**

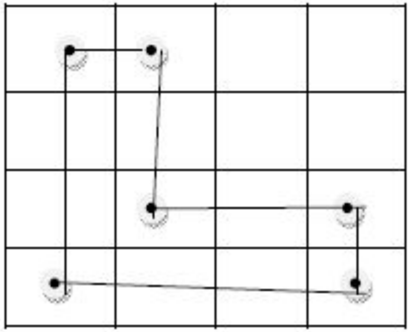
Calculating Penalties:

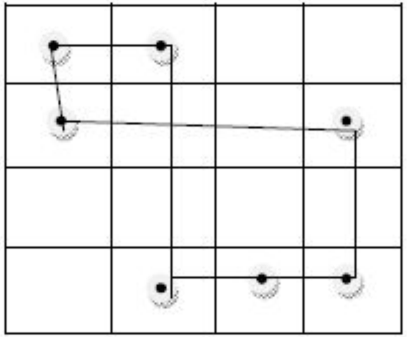
For each unallocated (i,j): $Penalty_{i,j} = u_i + v_j - Cost_{i,j}$

- Penalty of 0,2 is -1
- Penalty of 1,0 is -2
- Penalty of 1,2 is -2
- Penalty of 2,0 is -1

If all penalties are ≤ 0 then solution is optimal

Finding Loops Efficiently





References

- NPTEL Lectures on Transportation Optimization [1] [2] [3]
- https://en.wikipedia.org/wiki/Transportation_theory_(mathematics)
- https://www.researchgate.net/profile/Nneka_Ukanwoke/publication/287799854_On_the_Optimization_of_Transportation_Problem/links/5d3861ba4585153e591deea9/On-the-Optimization-of-Transportation-Problem.pdf
- GeeksForGeeks articles on Transportation Problem [1][2][3][4][5][6][7]

Contact Details

- Aniruddh Sujish (181IT206)- 7411199859
- Josson Thoppil (181IT121) - 7829729842
- Krithik Vaidya (181IT124) - 7030424864
- Mohith R (181IT128) - 9741309932

THANK YOU :)