# APPLICATIONS OF OPERATIONAL RESEARCH IN AIRLINE MANAGEMENT AND SCHEDULING

A DISSERTATION

SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE AWARD OF THE DEGREE
OF

MASTER OF SCIENCE
IN
**[MATHEMATICS]**

Submitted by :

**[Vedika Dixit](2K20/MSCMAT/37)**
**[Gaytri Rani](2K20/MSCMAT/10)**

Under the supervision of

[Prof.L N Das]


**DEPARTMENT OF APPLIED MATHEMATICS**
DELHI TECHNOLOGICAL UNIVERSITY

(Formerly Delhi College of Engineering)
Bawana Road,Delhi -110042

7 MAY 2022

**CANDIDATE'S DECLARATION**

We (Vedika Dixit),2K20/MSCMAT/37 and (Gaytri Rani),2K20/MSCMAT/10 of M.Sc( Applied Mthematics),hereby declare that the project Dissertation titled " APPLICATIONS OF OPERATIONAL RESEARCH IN AIRLINE MANAGEMENT AND SCHEDULING" which is submitted by us to the Department of Applied Mathematics, Delhi Technological University , Delhi in partial fulfillment of the requirement for the award of the degree of Master of Science, is original and not copied from any source without proper citation.

This work has not previously formed the basis for the award of any Degree,Diploma Associateship ,Fellowship or any other similar title or recognition.

Place : Delhi

Vedika Dixit

Gayatri Rani

## CERTIFICATE

We hereby certify that the project Dissertation titles ”APPLICATIONS OF OPERATIONAL RESEARCH IN AIRLINE MANAGEMENT AND SCHEDULING ” which is submitted by Vedika Dixit (2K20/MSCMAT/37) and Gayatri Rani (2K20/MSCMAT/10) to the department of Applied Mathematics ,this document submitted to Delhi Techno- logical University as part of the requirements for the Master of Science degree ,is the record of project work, completed by students under my supervision.This thesis has notbeen applied in part or full for any degree or diploma at the university or elsewhere, to the best of my knowledge.

Prof. L N Das
Department of Applied Mathematics
Delhi Technological University

## ACKNOWLEDGEMENT

Throughout the writing of this dissertation, We received a lot of help and encouragement.
First and foremost,we want to express our gratitude to Professor Mr. L.N Das, who was very helpful in developing the research topics and methodology. His informative remarks encouraged us to improve our thoughts and raise the quality of our work.He provided us with the tools that we needed to choose the right direction and successfully complete our dissertation.
In addition, We would like to thank God and our parents for their wise counsel and sympathetic ear.

**ABSTRACT**

This study base on AIR INDIA Airline Management and Scheduling examined certain well-known problems and their solutions, as well as current research in the field and developing areas of future significance. The Vogel's approximation method and the modified distribution method used to add new routes. The Hungarian Assignment Model and Critical Path Method used to solve airlines problem The Dijkstra Algorithm used to identify the shortest path. The paper has divided into four sections. In the first, the Vogel's approximation and the Modi approach is used to create additional routes. In the second, crew is assigned to different flights in order to optimise profit. In the third case, aircraft routing problem is solved, and in the fourth, an aircraft maintenance problem in which 24 activities opted and determine which are critical or not.It addresses airline scheduling flaws, aims to boost crew member productivity through a crew assignment model, and reduces time spent on non-critical operations, resulting in a rise in airline profit.

# Contents

# 1 Chapter 1

## 1.1 Introduction

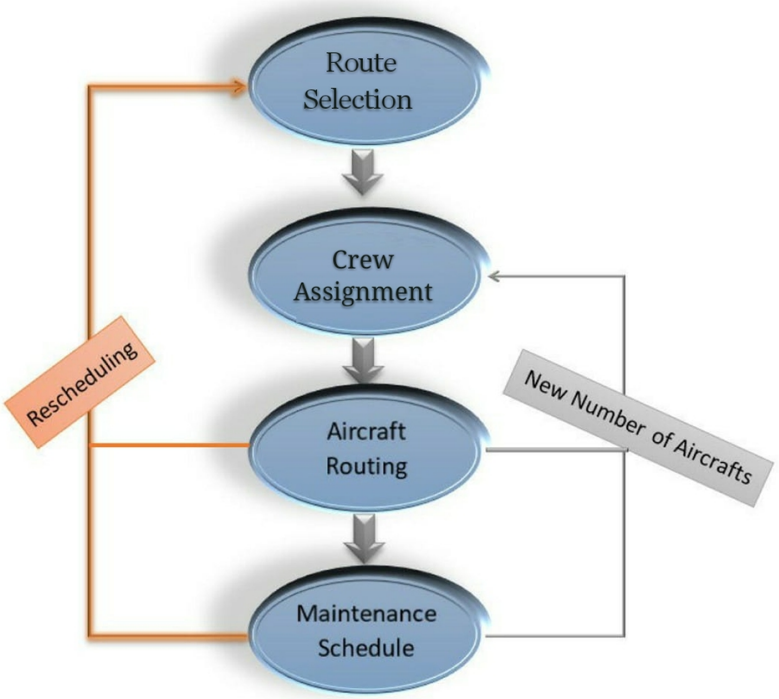
Operations research has aided the airline industry and its infrastructure in maintaining strong growth rates and moving from a novelty that catered to an elite clientele to a mass-market service business. It aids the industry in evolving in order to compete effectively in the marketplace and suit the diverse needs of the consumers. The Indian airline industry specifically AIR INDIA is going through a difficult period. Airlines' operating margins have been slashed. Many operating expenses, including as fuel, operating lease payments, repair/maintenance, and a huge number of aircraft and crew members, remain unutilized, meaning that they are prone to loss of epic proportions for AIR INDIA and are frequently beyond their control.
This paper is based on the spectacular divestiture of Air India, which ultimately failed. The lack of a management system is AIR INDIA's main concern. They fly more on routes where customer's demand is low and less on routes where demand is high. They have a significant number of aircrafts and crew on reserve, all of which are either unused or used without prior planning. The paper attempts to solve the problems that the Air India Airline faces in a number of situations by developing four separate cases. To tackle their problem, Vogel's Approximation approach and Modi method, Hungarian Method, Dijkstra's Algorithm, and Critical Path Analysis is primarily used.

## 1.2 About AIR INDIA

Air India is India's national airline, With headquarters in Delhi. Air India Ltd, a government-owned company, owns and operates a fleet of airline carriers that serve 90 domestic and international destinations across four continents. Air India is India's third-largest airline, after Indigo and Jet Airways, with an 18.6 percent domestic market share. JRD Tata Airlines was founded in 1932. After WWII, it was renamed Air India and turned into a public limited company. After a failed attempt to privatize the company, it merged with Indian Airlines and suffered losses. Due to fierce competition from private Indian airlines, Air India's domestic market share dropped from 19.2 percent to 14 percent between September 2007 and May 2011.

## 1.3 Airline and airport Management

The administration of airlines and airport is known as airline and airport management. It involves as determining airport strategy and gathering and transmitting information on airline commercial and operational plans. It provides a comprehensive review of airline management. It is also considered as field of studied that teaches airline and airport management. This gives a vast understanding of the airline sector and makes you aware of the marketing, financial, operational, and other elements that influence airline surveillance. This research based on the essential aspects of aircraft and aircraft selection and the influence of airport decision-making, as well as providing information on airline commercial and operational priorities. The procedure for airline management is as follows:



# 2 Chapter 2

## 2.1 Model and solution

This research is focused on the airline company AIR INDIA.AIR INDIA is India's national airline, with its headquarters in New Delhi. The Indian government is selling its entire stake in AIR INDIA Airlines, which has been making losses since its integrated with Indian Airlines in 2007. After bidding, Tata's Air India company will control 4400 domestic and 1800 international flights. The main difficulty for Tata Sons is to run the AIR INDIA airline efficiently. Airindia was controlled by Tata's Sons in the 1950s and 1960s, and the airline was only profitable at that period. The key opportunity for AIR INDIA is to develop globally and actually become fairly structured and broad-based from a network perspective, which can only be achieved if routing, route planning, and aircraft maintenance are all optimized.There are many aircraft and crews that are not functioning effectively, and there is no methodology to work with when it comes to airline planning.
This paper is based on a real-life situation with four various scenarios that an airline company can face in the real world and how they can use these strategies to improve their airline planning.

# 3 Chapter 3

## 3.1 Case1 : Route Selection

Airlines must conduct extensive market research and demand studies in order to determine whether they can profitably operate the route while also recognising operational restrictions. In this case, its aims to enhance the process of deciding which destinations to include by using a Transportation Problem (TP) Model, which is a type of LPP. The TP Model aims to optimise available routes from a specific origin to a specific destination in order to obtain the lowest possible cost.
To demonstrate this, the paper uses the real-situation of an AIR INDIA airline that is aiming to expand its route network. Many airlines use a hub-and-spoke model to provide links to smaller cities via focus cities on international routes. As a result, Table 1 shows data on profit earned per seat sold on a specific sector, as well as demand for destinations across the country, while taking capacity constraints at already overburdened hubs into account.

| | HongKong | Sydney | Amsterdam | Moscow | Toronto | Bangkok | Lisbon | Supply |
|---|---|---|---|---|---|---|---|---|
| Delhi | 116 | 31 | 71 | 19 | 49 | 47 | 48 | 1200 |
| Mumbai | 35 | 12 | 108 | 70 | 49 | 89 | 90 | 900 |
| Chandigadh | 54 | 14 | 65 | 19 | 105 | 59 | 68 | 900 |
| Hydrabad | 74 | 109 | 58 | 48 | 39 | 51 | 20 | 800 |
| Ahmedabad | 100 | 102 | 90 | 30 | 20 | 58 | 68 | 700 |
| Demand | 550 | 930 | 1250 | 620 | 900 | 800 | 700 | |

Table 1: Problem Table

To make the above-mentioned information into a TP issue, a dummy row is introduced to balance supply and demand, allowing the answer to be solved. Furthermore, prof-

its are converted to 'Opportunity Loss' values, which are defined as the amount missed when compared to taking the most profitable route.
The Vogel's Approximation Method (VAM) is an iterative approach for determining a transportation problem's initial basic feasible solution (IBFS) utilising penalties. The MODI technique (Modified Distribution) is used to improve the IBFS even further. The problem was solved both manually and with thethe Python Programming Language..
From Table 1 we can say that the Total number of Supply constraints is 5 and Total number of demand constraints is 7
Here Total Need = 5750 is more significant than Total Supply =4500. So we add a dummy supply constraint with 0 unit cost and with an allotment of 1250.
Now the revised table is

| | HongKong | Sydney | Amsterdam | Moscow | Toronto | Bangkok | Lisbon | Supply |
|---|---|---|---|---|---|---|---|---|
| Delhi | 116 | 31 | 71 | 19 | 49 | 47 | 48 | 1200 |
| Mumbai | 35 | 12 | 108 | 70 | 49 | 89 | 90 | 900 |
| Chandigadh | 54 | 14 | 65 | 19 | 105 | 59 | 68 | 900 |
| Hydrabad | 74 | 109 | 58 | 48 | 39 | 51 | 20 | 800 |
| Ahmedabad | 100 | 102 | 90 | 30 | 20 | 58 | 68 | 700 |
| $S_{dummy}$ | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1250 |
| Demand | 550 | 930 | 1250 | 620 | 900 | 800 | 700 | |

Table 2: Revised Table

Now we will calculate the row and column penalties

| | HongKong | Sydney | Amsterdam | Moscow | Toronto | Bangkok | Lisbon | Supply | Row penalty |
|---|---|---|---|---|---|---|---|---|---|
| Delhi | 116 | 31 | 71 | 19 | 49 | 47 | 48 | 1200 | 12=31-19 |
| Mumbai | 35 | 12 | 108 | 70 | 49 | 89 | 90 | 900 | 23=35-12 |
| Chandigadh | 54 | 14 | 65 | 19 | 105 | 59 | 68 | 900 | 5=19-4 |
| Hyderabad | 74 | 109 | 58 | 48 | 39 | 51 | 20 | 800 | 19=39-20 |
| Ahmedabad | 100 | 102 | 90 | 30 | 20 | 58 | 68 | 700 | 10=30-20 |
| $S_{dummy}$ | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0-0=0 |
| Demand | 550 | 930 | 1250 | 620 | 900 | 800 | 700 | | |
| Columnpenalty | 35=35-0 | 12=12-0 | 58=58-0 | 19=19-0 | 20=20-0 | 47=47-0 | 20=20-0 | | |

Table 3: Revised Table

The max Penalty is 58 which occurs at Amsterdam.
The min $C_{ij}$ in this queue is 0
The max allotment in this cell is min(1250,1250)=1250
It satisfies supply of $S_{dummy}$ and the need of Amsterdam.
The max penalty is 28 which occurs at Lisbon.
The min $C_{ij}$ in this queue is 20
The max allotment in this cell is $min(800,700) = 700$
It satisfies the need of Lisbon and adjust the supply of Hyderabad from 800 to 100(800-700=100)

| | Hongkong | Sydney | Amsterdam | Moscow | Toronto | Bangkok | Lisbon | Supply | Row penalty |
|---|---|---|---|---|---|---|---|---|---|
| Delhi | 116 | 31 | 71 | 19 | 49 | 47 | 48 | 1200 | 12=31-19 |
| Mumbai | 35 | 12 | 108 | 70 | 49 | 89 | 90 | 900 | 23=35-12 |
| Chandigadh | 54 | 14 | 65 | 19 | 105 | 59 | 68 | 900 | 5=19-14 |
| Hydrabad | 74 | 109 | 58 | 48 | 39 | 51 | 20 | 800 | 19=39-20 |
| Ahemdabad | 100 | 102 | 90 | 30 | 20 | 58 | 68 | 700 | 10=30-20 |
| ~~$S_{dummy}$~~ | ~~0~~ | ~~0~~ | ~~0(1250)~~ | ~~0~~ | ~~0~~ | ~~0~~ | ~~0~~ | 0 | - |
| Demand | 550 | 930 | 0 | 620 | 900 | 800 | 700 | | |
| Columnpenalty | 19=54-35 | 2=14-12 | - | 0=19-19 | 19=39-20 | 4=51-47 | 28=48-20 | | |

Figure 1: Reduced Table

| | Hongkong | Sydney | Amsterdam | Moscow | Toronto | Bangkok | Lisbon | Supply | Row penalty |
|---|---|---|---|---|---|---|---|---|---|
| Delhi | 116 | 31 | 71 | 19 | 49 | 47 | 48 | 1200 | 12=31-19 |
| Mumbai | 35 | 12 | 108 | 70 | 49 | 89 | 90 | 900 | 23=35-12 |
| Chandigadh | 54 | 14 | 65 | 19 | 105 | 59 | 68 | 900 | 5=19-14 |
| Hydrabad | 74 | 109 | 58 | 48 | 39 | 51 | 20(700) | 100 | 9=48-39 |
| Ahemdabad | 100 | 102 | 90 | 30 | 20 | 58 | 68 | 700 | 10=30-20 |
| ~~$S_{dummy}$~~ | ~~0~~ | ~~0~~ | ~~0(1250)~~ | ~~0~~ | ~~0~~ | ~~0~~ | ~~0~~ | 0 | - |
| Demand | 550 | 930 | 0 | 620 | 900 | 800 | 0 | | |
| Columnpenalty | 19=54-35 | 2=14-12 | - | 0=19-19 | 19=39-20 | 4=51-47 | - | | |

Figure 2: Reduced Table

The max penalty is 23 which occurs in row (Mumbai).
The min $C_{ij}$ in this row is $C_{22}$ =12.
The max allotment in this cell is min(900,930) = 900
It satisfies supply of Mumbai and adjust the need of Sydney from 930 to 30(930-900=30)

| | Hongkong | Sydney | Amsterdam | Moscow | Toronto | Bangkok | Lisbon | Supply | Row penalty |
|---|---|---|---|---|---|---|---|---|---|
| Delhi | 116 | 31 | 71 | 19 | 49 | 47 | 48 | 1200 | 12=31-19 |
| ~~Mumbai~~ | ~~35~~ | ~~12(900)~~ | ~~108~~ | ~~70~~ | ~~49~~ | ~~89~~ | ~~90~~ | ~~0~~ | - |
| Chandigadh | 54 | 14 | 65 | 19 | 105 | 59 | 68 | 900 | 5=19-14 |
| Hydrabad | 74 | 109 | 58 | 48 | 39 | 51 | 20(700) | 100 | 9=48-39 |
| Ahemdabad | 100 | 102 | 90 | 30 | 20 | 58 | 68 | 700 | 10=30-20 |
| $S_{dummy}$ | 0 | 0 | 0(1250) | 0 | 0 | 0 | 0 | 0 | - |
| Demand | 550 | 30 | 0 | 620 | 900 | 800 | 0 | | |
| Columnpenalty | 20=74-54 | 17=31-14 | - | 0=19-19 | 19=39-20 | 4=51-47 | - | | |

Figure 3: Reduced Table

honkong has the max penalty, which is 20. In this queue, the min $C_{ij}$ is $C_{31} = 54$. In this cell, the max allotment is min(900,550) = 550. It meets Hongkong's need and reduces Chandigadh's supply from 900 to 350 (900 - 550=350).

| | Hongkong | Sydney | Amsterdam | Moscow | Toronto | Bangkok | Lisbon | Supply | Row penalty |
|---|---|---|---|---|---|---|---|---|---|
| Delhi | 116 | 31 | 71 | 19 | 49 | 47 | 48 | 1200 | 12=31-19 |
| ~~Mumbai~~ | ~~35~~ | ~~12(900)~~ | ~~108~~ | ~~70~~ | ~~49~~ | ~~89~~ | ~~90~~ | ~~0~~ | - |
| Chandigadh | 54(550) | 14 | 65 | 19 | 105 | 59 | 68 | 350 | 5=19-14 |
| Hydrabad | 74 | 109 | 58 | 48 | 39 | 51 | 20(700) | 100 | 9=48-39 |
| Ahemdabad | 100 | 102 | 90 | 30 | 20 | 58 | 68 | 700 | 10=30-20 |
| $S_{dummy}$ | 0 | 0 | 0(1250) | 0 | 0 | 0 | 0 | 0 | - |
| Demand | 0 | 30 | 0 | 620 | 900 | 800 | 0 | | |
| Columnpenalty | - | 17=31-14 | - | 0=19-19 | 19=39-20 | 4=51-47 | - | | |

Figure 4: Reduced Table

In queue Bangkok,the max penalty, 19, is found.
In this queue, the min $C_{ij}$ is $C_{55} = 20$.
In this cell, the max allocation is min(700,900) = 700.
It meets Ahmedabad's supply and reduces Toronto's need from 900 to 200 (900 - 700=200).

| | Hongkong | Sydney | Amsterdam | Moscow | Toronto | Bangkok | Lisbon | Supply | Row penalty |
|---|---|---|---|---|---|---|---|---|---|
| Delhi | 116 | 31 | 71 | 19 | 49 | 47 | 48 | 1200 | 12=31-19 |
| ~~Mumbai~~ | ~~35~~ | ~~12(900)~~ | ~~108~~ | ~~70~~ | ~~49~~ | ~~89~~ | ~~90~~ | ~~0~~ | - |
| Chandigadh | 54(550) | 14 | 65 | 19 | 105 | 59 | 68 | 350 | 5=19-14 |
| Hydrabad | 74 | 109 | 58 | 48 | 39 | 51 | 20(700) | 100 | 9=48-39 |
| ~~Ahemdabad~~ | ~~100~~ | ~~102~~ | ~~90~~ | ~~30~~ | ~~20~~ | ~~58~~ | ~~68~~ | ~~0~~ | - |
| $S_{dummy}$ | 0 | 0 | 0(1250) | 0 | 0 | 0 | 0 | 0 | - |
| Demand | 0 | 30 | 0 | 620 | 900 | 800 | 0 | | |
| Columnpenalty | - | 17=31-14 | - | 0=19-19 | 10=49-39 | 4=51-47 | - | | |

Figure 5: Reduced Table

In column Sydney, the max penalty, 17, is found.
In this queue, the min $C_{ij}$ is $C_{32} = 14$.
In this cell, the max allocation is $\min(350, 30) = 30$
It meets Sydney's need and reduces Chandigadh's supply from 350 to 320 (350 - 30=320).

| | Hongkong | Sydney | Amsterdam | Moscow | Toronto | Bangkok | Lisbon | Supply | Row penalty |
|---|---|---|---|---|---|---|---|---|---|
| Delhi | 116 | 31 | 71 | 19 | 49 | 47 | 48 | 1200 | 28=47-19 |
| ~~Mumbai~~ | ~~35~~ | ~~12(900)~~ | ~~108~~ | ~~70~~ | ~~49~~ | ~~89~~ | ~~90~~ | ~~0~~ | - |
| Chandigadh | 54(550) | 14(30) | 65 | 19 | 105 | 59 | 68 | 320 | 40=59-19 |
| Hydrabad | 74 | 109 | 58 | 48 | 39 | 51 | 20(700) | 100 | 9=48-39 |
| ~~Ahemdabad~~ | ~~100~~ | ~~102~~ | ~~90~~ | ~~30~~ | ~~20~~ | ~~58~~ | ~~68~~ | ~~0~~ | - |
| $S_{dummy}$ | 0 | 0 | 0(1250) | 0 | 0 | 0 | 0 | 0 | - |
| Demand | 0 | 0 | 0 | 620 | 900 | 800 | 0 | | |
| Columnpenalty | - | - | - | 0=19-19 | 10=49-39 | 4=51-47 | - | | |

Figure 6: Reduced Table

Row Chandigadh has the max penalty of 40. In this row, the $minC_{ij}$ is $C_{34}$=19
In this cell, the max allotment is min(320,620) = 320.
It meets Chandigadh's requirement and reduces Moscow's demand from 620 to 300 (620 - 320=300).

| | Hongkong | Sydney | Amsterdam | Moscow | Toronto | Bangkok | Lisbon | Supply | Row penalty |
|---|---|---|---|---|---|---|---|---|---|
| Delhi | 116 | 31 | 71 | 19 | 49 | 47 | 48 | 1200 | 28=47-19 |
| ~~Mumbai~~ | ~~35~~ | ~~12(900)~~ | ~~108~~ | ~~70~~ | ~~49~~ | ~~89~~ | ~~90~~ | ~~0~~ | - |
| ~~Chandigadh~~ | ~~54(550)~~ | ~~14(30)~~ | ~~65~~ | ~~19(320)~~ | ~~105~~ | ~~59~~ | ~~68~~ | ~~0~~ | - |
| Hydrabad | 74 | 109 | 58 | 48 | 39 | 51 | 20(700) | 100 | 9=48-39 |
| ~~Ahemdabad~~ | ~~100~~ | ~~102~~ | ~~90~~ | ~~30~~ | ~~20~~ | ~~58~~ | ~~68~~ | ~~0~~ | - |
| $S_{dummy}$ | 0 | 0 | 0(1250) | 0 | 0 | 0 | 0 | 0 | - |
| Demand | 0 | 0 | 0 | 300 | 200 | 800 | 0 | | |
| Columnpenalty | - | - | - | 29=48-19 | 10=49-39 | 4=51-47 | - | | |

Figure 7: Reduced Table

In column Moscow, the max penalty, 29, is found.
In this column, the min $C_{ij}$ is $C_{14} = 19$
In this cell, the max allotment is min(1200,300) = 300.
It meets Moscow's need and reduces Delhi's supply from 1200to900(1200 - 300=900)

In Row Htdrabad, the max penalty,12 is found.
In this column, the min $C_{ij}$ is $C_{45} = 39$
In this cell, the max allotment is min(100,200) = 100.
It meets Hydrabad's need and reduces Toronto's supply from 200 to 100 (200 - 100=100).

| | Hongkong | Sydney | Amsterdam | Moscow | Toronto | Bangkok | Lisbon | Supply | Row penalty |
|---|---|---|---|---|---|---|---|---|---|
| Delhi | 116 | 31 | 71 | 19(300) | 49 | 47 | 48 | 900 | 2=49-47 |
| ~~Mumbai~~ | ~~35~~ | ~~12(900)~~ | ~~108~~ | ~~70~~ | ~~49~~ | ~~89~~ | ~~90~~ | ~~0~~ | - |
| ~~Chandigadh~~ | ~~54(550)~~ | ~~14(30)~~ | ~~65~~ | ~~19(320)~~ | ~~105~~ | ~~59~~ | ~~68~~ | ~~0~~ | - |
| Hydrabad | 74 | 109 | 58 | 48 | 39 | 51 | 20(700) | 100 | 12=51-39 |
| ~~Ahemdabad~~ | ~~100~~ | ~~102~~ | ~~90~~ | ~~30~~ | ~~20~~ | ~~58~~ | ~~68~~ | ~~0~~ | - |
| $S_{dummy}$ | 0 | 0 | 0(1250) | 0 | 0 | 0 | 0 | 0 | - |
| Demand | 0 | 0 | 0 | 0 | 200 | 800 | 0 | | |
| Columnpenalty | - | - | - | - | 10=49-39 | 4=51-47 | - | | |

Figure 8: Reduced Table

| | Hongkong | Sydney | Amsterdam | Moscow | Toronto | Bangkok | Lisbon | Supply | Row penalty |
|---|---|---|---|---|---|---|---|---|---|
| Delhi | 116 | 31 | 71 | 19(300) | 49 | 47 | 48 | 900 | 2=49-47 |
| ~~Mumbai~~ | ~~35~~ | ~~12(900)~~ | ~~108~~ | ~~70~~ | ~~49~~ | ~~89~~ | ~~90~~ | 0 | - |
| ~~Chandigadh~~ | ~~54(550)~~ | ~~14(30)~~ | ~~65~~ | ~~19(320)~~ | ~~105~~ | ~~59~~ | ~~68~~ | 0 | - |
| ~~Hydrabad~~ | ~~74~~ | ~~109~~ | ~~58~~ | ~~48~~ | ~~39(100)~~ | ~~51~~ | ~~20(700)~~ | 0 | - |
| ~~Ahemdabad~~ | ~~100~~ | ~~102~~ | ~~90~~ | ~~30~~ | ~~20(700)~~ | ~~58~~ | ~~68~~ | 0 | - |
| ~~$S_{dummy}$~~ | ~~0~~ | ~~0~~ | ~~0(1250)~~ | ~~0~~ | ~~0~~ | ~~0~~ | ~~0~~ | 0 | - |
| Demand | 0 | 0 | 0 | 0 | 100 | 800 | 0 | | |
| Columnpenalty | - | - | - | - | 49 | 47 | - | | |

Figure 9: Reduced Table

In Toronto, the max penalty,49 is found.
In this queue, the min $C_{ij}$ is $C_{15} = 49$
In this cell, the max allocation is min(900,100) = 100.
It meets Toronto's need and reduces Delhi's supply from 900 to 800 (900 - 100=800).

In Row Delhi , the max penalty,47 is found.
In this queue, the min $C_{ij}$ is $C_{16} = 47$
In this cell, the max allotment is min(800,800) = 800.
It meets Delhi's supply and Bangkok's supply.

| | Hongkong | Sydney | Amsterdam | Moscow | Toronto | Bangkok | Lisbon | Supply | Row penalty |
|---|---|---|---|---|---|---|---|---|---|
| Delhi | 116 | 31 | 71 | 19(300) | 49(100) | 47 | 48 | 800 | 47 |
| ~~Mumbai~~ | ~~35~~ | ~~12(900)~~ | ~~108~~ | ~~70~~ | ~~49~~ | ~~89~~ | ~~90~~ | 0 | - |
| ~~Chandigadh~~ | ~~54(550)~~ | ~~14(30)~~ | ~~65~~ | ~~19(320)~~ | ~~105~~ | ~~59~~ | ~~68~~ | 0 | - |
| ~~Hydrabad~~ | ~~74~~ | ~~109~~ | ~~58~~ | ~~48~~ | ~~39(100)~~ | ~~51~~ | ~~20(700)~~ | 0 | - |
| ~~Ahemdabad~~ | ~~100~~ | ~~102~~ | ~~90~~ | ~~30~~ | ~~20(700)~~ | ~~58~~ | ~~68~~ | 0 | - |
| ~~$S_{dummy}$~~ | ~~0~~ | ~~0~~ | ~~0(1250)~~ | ~~0~~ | ~~0~~ | ~~0~~ | ~~0~~ | 0 | - |
| Demand | 0 | 0 | 0 | 0 | 0 | 800 | 0 | | |
| Columnpenalty | - | - | - | - | - | 47 | - | | |

Figure 10: Reduced Table

Initial feasible solution is

| | Hongkong | Sydney | Amsterdam | Moscow | Toronto | Bangkok | Lisbon | Supply | Row Penalty |
|---|---|---|---|---|---|---|---|---|---|
| Delhi | 116 | 31 | 71 | 19(300) | 49(100) | 47(800) | 48 | 1200 | 12\|12\|12\|12\|12\|12\|28\|28\|2\|2\|47 |
| Mumbai | 35 | 12(900) | 108 | 70 | 49 | 89 | 90 | 900 | 23\|23\|23\|–\|–\|–\|–\|–\|–\|–\|– |
| Chandigadh | 54(550) | 14(30) | 65 | 19(320) | 105 | 59 | 68 | 900 | 5\|5\|5\|5\|5\|5\|40\|–\|–\|–\|– |
| Hydrabad | 74 | 109 | 58 | 48 | 39(100) | 51 | 20(700) | 800 | 19\|19\|9\|9\|9\|9\|9\|9\|12\|–\|– |
| Ahmedabad | 100 | 102 | 90 | 30 | 20(700) | 58 | 68 | 700 | 10\|10\|10\|10\|10\|–\|–\|–\|–\|–\|– |
| $S_{dummy}$ | 0 | 0 | 0(1250) | 0 | 0 | 0 | 0 | 1250 | 0\|–\|–\|–\|–\|–\|–\|–\|–\|–\|– |
| Demand | 550 | 930 | 1250 | 620 | 900 | 800 | 700 | | |
| $v_j$ | $v_1 = 54$ | $v_2 = 14$ | $v_3 = 68$ | $v_4 = 19$ | $v_5 = 49$ | $v_6 = 47$ | $v_7 = 30$ | | |
| Demand | 550 | 930 | 1250 | 620 | 900 | 800 | 700 | | |
| 114em Column | | | | | | | | | |
| Penalty | 35 | 12 | 58 | 19 | 20 | 47 | 20 | | |
| | 19 | 2 | – | 0 | 19 | 4 | 28 | | |
| | 19 | 2 | – | 0 | 19 | 4 | 28 | | |
| | 20 | 17 | – | 0 | 19 | 4 | – | | |
| | – | 17 | – | 0 | 19 | 4 | – | | |
| | – | 17 | – | 0 | 10 | 4 | – | | |
| | – | – | – | 0 | 10 | 4 | – | | |
| | – | – | – | 29 | 10 | 4 | – | | |
| | – | – | – | – | 10 | 4 | – | | |
| | – | – | – | – | 49 | 47 | – | | |
| | – | – | – | – | – | 47 | – | | |

Table 4: Initial feasible

the lowest possible total transportation cost $= 19 \times 300 + 49 \times 100 + 47 \times 800 + 12 \times 900 + 54 \times 550 + 14 \times 30 + 19 \times 320 + 39 \times 100 + 20 \times 700 + 20 \times 700 + 0 \times 1250 = 127$

In this case, the number of allotted cells equals one, which is one less than m+n-1=6+7-1=12. As a result, this is a degenerate solution.

We utilise an artificial quantity to solve degeneracy (d). The amount d is given to the vacant cell with the lowest transportation expense.

The amount d is assigned to hydeabad amsterdam, with a minimum transit cost of 58.

| | HongKong | Sydney | Amsterdam | Moscow | Toronto | Bangkok | Lisbon | Supply | |
|---|---|---|---|---|---|---|---|---|---|
| Delhi | 116 | 31 | 71 | 19(300) | 49(100) | 47(800) | 48 | 1200 | |
| Mumbai | 35 | 12(900) | 108 | 70 | 49 | 89 | 90 | 900 | |
| Chandigadh | 54(550) | 14(30) | 65 | 19(320) | 105 | 59 | 68 | 900 | |
| Hydrabad | 74 | 109 | 58(d) | 48 | 39(100) | 51 | 20(700) | 800 | |
| Ahmedabad | 100 | 102 | 90 | 30 | 20(700) | 58 | 68 | 700 | |
| $S_{dummy}$ | 0 | 0 | 0(1250) | 0 | 0 | 0 | 0 | 1250 | |
| Demand | 550 | 930 | 1250 | 620 | 900 | 800 | 700 | | |

Table 5: Degenerate Solution

Optimality test using MODI METHOD

| | HongKong | Sydney | Amsterdam | Moscow | Toronto | Bangkok | Lisbon | Supply | |
|---|---|---|---|---|---|---|---|---|---|
| Delhi | 116 | 31 | 71 | 19(300) | 49(100) | 47(800) | 48 | 1200 | |
| Mumbai | 35 | 12(900) | 108 | 70 | 49 | 89 | 90 | 900 | |
| Chandigadh | 54(550) | 14(30) | 65 | 19(320) | 105 | 59 | 68 | 900 | |
| Hydrabad | 74 | 109 | 58(d) | 48 | 39(100) | 51 | 20(700) | 800 | |
| Ahmedabad | 100 | 102 | 90 | 30 | 20(700) | 58 | 68 | 700 | |
| $S_{dummy}$ | 0 | 0 | 0(1250) | 0 | 0 | 0 | 0 | 1250 | |
| Demand | 550 | 930 | 1250 | 620 | 900 | 800 | 700 | | |

Table 6: Allocation Table

Iteration -1 of optimality difficulty
1.Locate $d_y$ for all unoccupied sections$(i, j)$, where $d_y = c_{ij} - (u_i + v_j)$

1. Replacing, $u_1 = 0$, we get
2. $c_{14} = u_1 + v_4 \Rightarrow v_4 = c_{14} - u_1 \Rightarrow v_4 = 19 - 0 \Rightarrow v_4 = 19$
3. $c_{34} = u_3 + v_4 \Rightarrow u_3 = c_{34} - v_4 \Rightarrow u_3 = 19 - 19 \Rightarrow u_3 = 0$
4. $c_{31} = u_3 + v_1 \Rightarrow v_1 = c_{31} - u_3 \Rightarrow v_1 = 54 - 0 \Rightarrow v_1 = 54$
5. $c_{32} = u_3 + v_2 \Rightarrow v_2 = c_{32} - u_3 \Rightarrow v_2 = 14 - 0 \Rightarrow v_2 = 14$
6. $c_{22} = u_2 + v_2 \Rightarrow u_4 = c_{22} - v_1 \Rightarrow u_2 = 12 - 14 \Rightarrow u_2 = -2$
7. $c_{15} = u_1 + v_5 \Rightarrow v_5 = c_{15} - u_1 \Rightarrow v_5 = 49 - 0 \Rightarrow v_5 = 49$
8. $c_{45} = u_4 + v_5 \Rightarrow u_4 = c_{45} - v_5 \Rightarrow u_4 = 39 - 49 \Rightarrow u_4 = -10$
9. $c_{43} = u_4 + v_3 \Rightarrow v_3 = c_{43} - u_4 \Rightarrow v_3 = 58 + 10 \Rightarrow v_3 = 68$
10. $c_{63} = u_6 + v_3 \Rightarrow u_6 = c_{63} - v_3 \Rightarrow u_6 = 0 - 68 \Rightarrow u_6 = -68$
11. $c_{47} = u_4 + v_7 \Rightarrow v_7 = c_{47} - u_4 \Rightarrow v_7 = 20 + 10 \Rightarrow v_7 = 30$
12. $c_{55} = v_5 + u_5 \Rightarrow u_5 = c_{55} - v_5 \Rightarrow u_5 = 20 - 49 \Rightarrow u_5 = -29$
13. $c_{16} = u_1 + v_6 \Rightarrow v_6 = c_{16} - u_1 \Rightarrow v_6 = 47 - 0 \Rightarrow v_6 = 47$

| | HongKong | Sydney | Amsterdam | Moscow | Toronto | Bangkok | Lisbon | Supply | $u_i$ |
|---|---|---|---|---|---|---|---|---|---|
| Delhi | 116 | 31 | 71 | 19(300) | 49(100) | 47(800) | 48 | 1200 | $u_1 = 0$ |
| Mumbai | 35(550) | 12(350) | 108 | 70 | 49 | 89 | 90 | 900 | $u_2 = -2$ |
| Chandigadh | 54 | 14(580) | 65 | 19(320) | 105 | 59 | 68 | 900 | $u_3 = 0$ |
| Hydrabad | 74 | 109 | 58(d) | 48 | 39(100) | 51 | 20(700) | 800 | $u_4 = -10$ |
| Ahmedabad | 100 | 102 | 90 | 30 | 20(700) | 58 | 68 | 700 | $u_5 = -29$ |
| $S_{dummy}$ | 0 | 0 | 0(1250) | 0 | 0 | 0 | 0 | 1250 | $u_6 = -68$ |
| Demand | 550 | 930 | 1250 | 620 | 900 | 800 | 700 | | |
| $v_j$ | $v_1 = 37$ | $v_2 = 14$ | $v_3 = 68$ | $v_4 = 19$ | $v_5 = 49$ | $v_6 = 47$ | $v_7 = 30$ | | |

Table 7: degenerate solution

2. Find $d_y$ for all unoccupied sectiond $(i, j)$, where $d_y = c_{ij} - (u_i + v_j)$

1. $d_{11} = c_{11} - (u_1 + v_1) = 116 - (0 + 54) = 62$
2. $d_{12} = c_{12} - (u_1 + v_2) = 31 - (0 + 14) = 17$
3. $d_{13} = c_{13} - (u_1 + v_3) = 71 - (0 + 68) = 3$
4. $d_{17} = c_{17} - (u_1 + v_7) = 48 - (0 + 30) = 18$
5. $d_{21} = c_{21} - (u_2 + v_1) = 35 - (-2 + 54) = -17$
6. $d_{23} = c_{23} - (u_2 + v_3) = 108 - (-2 + 68) = 42$
7. $d_{24} = c_{24} - (u_2 + v_4) = 70 - (-2 + 19) = 53$
8. $d_{25} = c_{25} - (u_2 + v_5) = 49 - (-2 + 49) = 2$
9. $d_{26} = c_{26} - (u_2 + v_6) = 89 - (-2 + 47) = 44$
10. $d_{27} = c_{27} - (u_2 + v_7) = 90 - (-2 + 30) = 62$
11. $d_{33} = c_{33} - (u_3 + v_3) = 65 - (0 + 68) = -3$
12. $d_{35} = c_{35} - (u_3 + v_5) = 105 - (0 + 49) = 56$
13. $d_{36} = c_{36} - (u_3 + v_6) = 59 - (0 + 47) = 12$

14. $d_{37} = c_{37} - (u_3 + v_7) = 68 - (0 + 30) = 38$

15. $d_{41} = c_{41} - (u_4 + v_1) = 74 - (-10 + 54) = 30$

16. $d_{42} = c_{42} - (u_4 + v_2) = 109 - (-10 + 14) = 105$

17. $d_{44} = c_{44} - (u_4 + v_3) = 48 - (-10 + 19) = 39$

18. $d_{46} = c_{46} - (u_4 + v_6) = 51 - (-10 + 47) = 14$

19. $d_{51} = c_{51} - (u_5 + v_1) = 100 - (-29 + 54) = 75$

20. $d_{52} = c_{52} - (u_5 + v_2) = 102 - (-29 + 14) = 117$

21. $d_{53} = c_{53} - (u_5 + v_3) = 90 - (-29 + 68) = 51$

22. $d_{54} = c_{54} - (u_5 + v_4) = 30 - (-29 + 19) = 40$

23. $d_{56} = c_{56} - (u_5 + v_6) = 58 - (-29 + 47) = 40$

24. $d_{57} = c_{57} - (u_5 + v_7) = 68 - (-29 + 30) = 67$

25. $d_{61} = c_{61} - (u_6 + v_1) = 0 - (-68 + 54) = 14$

26. $d_{62} = c_{62} - (u_6 + v_2) = 0 - (-68 + 14) = 54$

27. $d_{64} = c_{64} - (u_6 + v_4) = 0 - (-68 + 19) = 49$

28. $d_{65} = c_{65} - (u_6 + v_5) = 0 - (-68 + 49) = 19$

29. $d_{66} = c_{66} - (u_6 + v_6) = 0 - (-68 + 47) = 21$

30. $d_{67} = c_{67} - (u_6 + v_7) = 0 - (-68 + 30) = 38$

3.Select the smallest negative value from all of them

$$d_{ij}(costofopportunity) = d_{21} = [-17]$$

and draw a shortest and fast path from Mumbai Hongkong
Closed path is Mumbai Hongkong $\rightarrow$ *MumbaiSydney* $\rightarrow$ *ChandigadhSydney*

| | HongKong | Sydney | Amsterdam | Moscow | Toronto | Bangkok | Lisbon | Supply | $u_i$ |
|---|---|---|---|---|---|---|---|---|---|
| Delhi | 116[62] | 31[17] | 71[3] | 19(300) | 49(100) | 47(800) | 48[18] | 1200 | $u_1 = 0$ |
| Mumbai | 35[−17] | 12(900) | 108[42] | 70[53] | 49[2] | 89[44] | 90[62] | 900 | $u_2 = -2$ |
| Chandigadh | 54(550) | 14(30) | 65[−3] | 19(320) | 105[56] | 59[12] | 68[38] | 900 | $u_3 = 0$ |
| Hydrabad | 74[30] | 109[105] | 58(*d*) | 48[39] | 39(100) | 51[14] | 20(700) | 800 | $u_4 = -10$ |
| Ahmedabad | 100[75] | 102[117] | 90[51] | 30[40] | 20(700) | 58[40] | 68[67] | 700 | $u_5 = -29$ |
| $S_{dummy}$ | 0[14] | 0[54] | 0(1250) | 0[49] | 0[19] | 0[21] | 0[38] | 1250 | $u_6 = -68$ |
| Demand | 550 | 930 | 1250 | 620 | 900 | 800 | 700 | | |
| $v_j$ | $v_1 = 54$ | $v_2 = 14$ | $v_3 = 68$ | $v_4 = 19$ | $v_5 = 49$ | $v_6 = 47$ | $v_7 = 30$ | | |

Table 8: Opportunity Cost

Fast path and plus/minus sign allotment...

| | Hongkong | Sydney | Amsterdam | Moscow | Toronto | Bangkok | Lisbon | Supply | $u_i$ |
|---|---|---|---|---|---|---|---|---|---|
| Delhi | 116[62] | 31[17] | 71[3] | 19(300) | 49(100) | 47(800) | 48[18] | 1200 | $u_1 = 0$ |
| Mumbai | 35[−17](+) | 12(900)(−) | 108[42] | 70[53] | 49[2] | 89[44] | 90[62] | 900 | $u_2 = -2$ |
| Chandigadh | 54(550)(−) | 14(30)(+) | 65[−3] | 19(320) | 105[56] | 59[12] | 68[38] | 900 | $u_3 = 0$ |
| Hydrabad | 74[30] | 109[105] | 58(*d*) | 48[39] | 39(100) | 51[14] | 20(700) | 800 | $u_4 = -10$ |
| Ahmedabad | 100[75] | 102[117] | 90[51] | 30[40] | 20(700) | 58[40] | 68[67] | 700 | $u_5 = -29$ |
| $S_{dummy}$ | 0[14] | 0[54] | 0(1250) | 0[49] | 0[19] | 0[21] | 0[38] | 1250 | $u_6 = -68$ |
| Demand | 550 | 930 | 1250 | 620 | 900 | 800 | 700 | | |
| $v_j$ | $v_1 = 54$ | $v_2 = 14$ | $v_3 = 68$ | $v_4 = 19$ | $v_5 = 49$ | $v_6 = 47$ | $v_7 = 30$ | | |

Table 9: Fast Path

4.The lowest assigned value among all negative position$(-)$ on a shortest and path is $= 550$

Subtract 550 from all (-) and it to all (+)

| | HongKong | Sydney | Amsterdam | Moscow | Toronto | Bangkok | Lisbon | Supply | |
|---|---|---|---|---|---|---|---|---|---|
| Delhi | 116 | 31 | 71 | 19(300) | 49(100) | 47(800) | 48 | 1200 | |
| Mumbai | 35(550) | 12(350) | 108 | 70 | 49 | 89 | 90 | 900 | |
| Chandigadh | 54(550) | 14(580) | 65 | 19(320) | 105 | 59 | 68 | 900 | |
| Hydrabad | 74 | 109 | 58(d) | 48 | 39(100) | 51 | 20(700) | 800 | |
| Ahmedabad | 100 | 102 | 90 | 30 | 20(700) | 58 | 68 | 700 | |
| $S_{dummy}$ | 0 | 0 | 0(1250) | 0 | 0 | 0 | 0 | 1250 | |
| Demand | 550 | 930 | 1250 | 620 | 900 | 800 | 700 | | |

Table 10: Allocated Value

5.Steps 1–4 should be repeated until an most favorable solution is reached.

Iteration -2 of optimality examination

1.Find $u_i$ and $v_j$ for all occupied cells $(i, j)$, where $c_i j = u_i + v_j$

1. Replacing, $u_1 = 0$, we get
2. $c_{14} = u_1 + v_4 \Rightarrow v_4 = c_{14} - u_1 \Rightarrow v_4 = 19 - 0 \Rightarrow v_4 = 19$
3. $c_{34} = u_3 + v_4 \Rightarrow u_3 = c_{34} - v_4 \Rightarrow u_3 = 19 - 19 \Rightarrow u_3 = 0$
4. $c_{32} = u_3 + v_2 \Rightarrow v_2 = c_{32} - u_3 \Rightarrow v_2 = 14 - 0 \Rightarrow v_2 = 14$
5. $c_{22} = u_2 + v_2 \Rightarrow u_2 = c_{22} - v_2 \Rightarrow u_2 = 12 - 14 \Rightarrow u_2 = -2$
6. $c_{21} = u_2 + v_1 \Rightarrow v_1 = c_{21} - u_2 \Rightarrow v_1 = 35 + 2 \Rightarrow v_1 = 37$
7. $c_{15} = u_1 + v_5 \Rightarrow v_5 = c_{15} - u_1 \Rightarrow v_5 = 49 - 0 \Rightarrow v_5 = 49$

8. $c_{45} = u_4 + v_5 \Rightarrow u_4 = c_{45} - v_5 \Rightarrow u_4 = 39 - 49 \Rightarrow u_4 = -10$
9. $c_{43} = u_4 + v_3 \Rightarrow v_3 = c_{43} - u_4 \Rightarrow v_3 = 58 + 10 \Rightarrow v_3 = 68$
10. $c_{63} = u_6 + v_3 \Rightarrow u_6 = c_{63} - v_3 \Rightarrow u_6 = 0 - 68 \Rightarrow u_6 = -68$
11. $c_{47} = u_4 + v_7 \Rightarrow v_7 = c_{47} - u_4 \Rightarrow v_7 = 20 + 10 \Rightarrow v_7 = 30$
12. $c_{55} = u_5 + v_5 \Rightarrow u_5 = c_{55} - v_5 \Rightarrow u_5 = 20 - 49 \Rightarrow u_5 = -29$
13. $c_{16} = u_1 + v_6 \Rightarrow v_6 = c_{16} - u_1 \Rightarrow v_6 = 47 - 0 \Rightarrow v_6 = 47$

| | Hongkong | Sydney | Amsterdam | Moscow | Toronto | Bangkok | Lisbon | Supply | $u_i$ |
|---|---|---|---|---|---|---|---|---|---|
| Delhi | 116 | 31 | 71 | 19(300) | 49(100) | 47(800) | 48 | 1200 | $u_1 = 0$ |
| Mumbai | 35(300) | 12(350) | 108 | 70 | 49 | 89 | 90 | 900 | $u_2 = -2$ |
| Chandigadh | 54 | 14(580) | 65 | 19(320) | 105 | 59 | 68 | 900 | $u_3 = 0$ |
| Hydrabad | 74 | 109 | 58($d$) | 48 | 39(100) | 51 | 20(700) | 800 | $u_4 = -10$ |
| Ahmedabad | 100 | 102 | 90 | 30 | 20(700) | 58 | 68 | 700 | $u_5 = -29$ |
| $S_{dummy}$ | 0 | 0 | 0(1250) | 0 | 0 | 0 | 0 | 1250 | $u_6 = -68$ |
| Demand | 550 | 930 | 1250 | 620 | 900 | 800 | 700 | | |
| $v_j$ | $v_1 = 54$ | $v_2 = 14$ | $v_3 = 68$ | $v_4 = 19$ | $v_5 = 49$ | $v_6 = 47$ | $v_7 = 30$ | | |

Table 11: Allocate Table

2. Find $d_y$ for all unoccupied sections $(i, j)$, where $d_y = c_{ij} - (u_i + v_j)$

1. $d_{11} = c_{11} - (u_1 + v_1) = 116 - (0 + 37) = 79$
2. $d_{12} = c_{12} - (u_1 + v_2) = 31 - (0 + 14) = 17$
3. $d_{13} = c_{13} - (u_1 + v_3) = 71 - (0 + 68) = 3$
4. $d_{17} = c_{17} - (u_1 + v_7) = 48 - (0 + 30) = 18$
5. $d_{23} = c_{23} - (u_2 + v_3) = 108 - (-2 + 68) = 42$
6. $d_{24} = c_{24} - (u_2 + v_4) = 70 - (-2 + 19) = 53$
7. $d_{25} = c_{25} - (u_2 + v_5) = 49 - (-2 + 49) = 2$
8. $d_{26} = c_{26} - (u_2 + v_6) = 89 - (-2 + 47) = 44$
9. $d_{31} = c_{31} - (u_3 + v_1) = 54 - (-0 + 47) = 44$

10. $d_{27} = c_{27} - (u_2 + v_7) = 90 - (-2 + 30) = 62$

11. $d_{33} = c_{33} - (u_3 + v_3) = 65 - (0 + 68) = -3$

12. $d_{35} = c_{35} - (u_3 + v_5) = 105 - (0 + 49) = 56$

13. $d_{36} = c_{36} - (u_3 + v_6) = 59 - (0 + 47) = 12$

14. $d_{37} = c_{37} - (u_3 + v_7) = 68 - (0 + 30) = 38$

15. $d_{41} = c_{41} - (u_4 + v_1) = 74 - (-10 + 54) = 30$

16. $d_{42} = c_{42} - (u_4 + v_2) = 109 - (-10 + 14) = 105$

17. $d_{44} = c_{44} - (u_4 + v_3) = 48 - (-10 + 19) = 39$

18. $d_{46} = c_{46} - (u_4 + v_6) = 51 - (-10 + 47) = 14$

19. $d_{51} = c_{51} - (u_5 + v_1) = 100 - (-29 + 54) = 75$

20. $d_{52} = c_{52} - (u_5 + v_2) = 102 - (-29 + 14) = 117$

21. $d_{53} = c_{53} - (u_5 + v_3) = 90 - (-29 + 68) = 51$

22. $d_{54} = c_{54} - (u_5 + v_4) = 30 - (-29 + 19) = 40$

23. $d_{56} = c_{56} - (u_5 + v_6) = 58 - (-29 + 47) = 40$

24. $d_{57} = c_{57} - (u_5 + v_7) = 68 - (-29 + 30) = 67$

25. $d_{61} = c_{61} - (u_6 + v_1) = 0 - (-68 + 54) = 14$

26. $d_{62} = c_{62} - (u_6 + v_2) = 0 - (-68 + 14) = 54$

27. $d_{64} = c_{64} - (u_6 + v_4) = 0 - (-68 + 19) = 49$

28. $d_{65} = c_{65} - (u_6 + v_5) = 0 - (-68 + 49) = 19$

29. $d_{66} = c_{66} - (u_6 + v_6) = 0 - (-68 + 47) = 21$

30. $d_{67} = c_{67} - (u_6 + v_7) = 0 - (-68 + 30) = 38$

| | HongKong | Sydney | Amsterdam | Moscow | Toronto | Bangkok | Lisbon | Supply | $u_i$ |
|---|---|---|---|---|---|---|---|---|---|
| Delhi | 116[79] | 31[17] | 71[3] | 19(300) | 49(100) | 47(800) | 48[18] | 1200 | $u_1 = 0$ |
| Mumbai | 35(550) | 12(350) | 108[42] | 70[53] | 49[2] | 89[44] | 90[62] | 900 | $u_2 = -2$ |
| Chandigadh | 54[17] | 14(580) | 65[−3] | 19(320) | 105[56] | 59[12] | 68[38] | 900 | $u_3 = 0$ |
| Hydrabad | 74[47] | 109[105] | 58($d$) | 48[39] | 39(100) | 51[14] | 20(700) | 800 | $u_4 = -10$ |
| Ahmedabad | 100[92] | 102[117] | 90[51] | 30[40] | 20(700) | 58[40] | 68[67] | 700 | $u_5 = -29$ |
| $S_{dummy}$ | 0[31] | 0[54] | 0(1250) | 0[49] | 0[19] | 0[21] | 0[38] | 1250 | $u_6 = -68$ |
| Demand | 550 | 930 | 1250 | 620 | 900 | 800 | 700 | | |
| $v_j$ | $v_1 = 37$ | $v_2 = 14$ | $v_3 = 68$ | $v_4 = 19$ | $v_5 = 49$ | $v_6 = 47$ | $v_7 = 30$ | | |

Table 12: Sign Allocation

3.Select the smallest negative value from all of them

$$d_{ij}(costofopportunity) = d_{33} = [-3]$$

and skecth a shortest and fast path from Chandigadh Amsterdam
Closed path is Chandigadh Amsterdam$\rightarrow ChandigadhMoscow \rightarrow DelhiMoscow \rightarrow DelhiToronto \rightarrow HydrabadToronto \rightarrow HydrabadChandigadh$

Fast path and plus/minus sign allocation...

| | Hongkong | Sydney | Amsterdam | Moscow | Toronto | Bangkok | Lisbon | Supply | $u_i$ |
|---|---|---|---|---|---|---|---|---|---|
| Delhi | 116[79] | 31[17] | 71[3] | 19(300)(+) | 49(100)(−) | 47(800) | 48[18] | 1200 | $u_1 = 0$ |
| Mumbai | 35(550) | 12(350) | 108[42] | 70[53] | 49[2] | 89[44] | 90[62] | 900 | $u_2 = -2$ |
| Chandigadh | 54(550)(−) | 14(30)(+) | 65[−3] | 19(320) | 105[56] | 59[12] | 68[38] | 900 | $u_3 = 0$ |
| Hydrabad | 74[30] | 109[105] | 58(*d*) | 48[39] | 39(100) | 51[14] | 20(700) | 800 | $u_4 = -10$ |
| Ahmedabad | 100[75] | 102[117] | 90[51] | 30[40] | 20(700) | 58[40] | 68[67] | 700 | $u_5 = -29$ |
| $S_{dummy}$ | 0[14] | 0[54] | 0(1250) | 0[49] | 0[19] | 0[21] | 0[38] | 1250 | $u_6 = -68$ |
| Demand | 550 | 930 | 1250 | 620 | 900 | 800 | 700 | | |
| $v_j$ | $v_1 = 54$ | $v_2 = 14$ | $v_3 = 68$ | $v_4 = 19$ | $v_5 = 49$ | $v_6 = 47$ | $v_7 = 30$ | | |

Table 13: Closed Path

dummy cell is selected, so end the procedure for current selected dummy cell. Try the optimal solution with next dummy cell count- 2
The quality d is assigned to *MumbaiAmsterdam*, which has the minimum transportation cost = 65.

| | Hongkong | Sydney | Amsterdam | Moscow | Toronto | Bangkok | Lisbon | Supply |
|---|---|---|---|---|---|---|---|---|
| Delhi | 116 | 31 | 71 | 19(300) | 49(100) | 47(800) | 48 | 1200 |
| Mumbai | 35(550) | 12(350) | 108 | 70 | 49 | 89 | 90 | 900 |
| Chandigarh | 54 | 14(580) | 65 | 19(320) | 105 | 59 | 68 | 900 |
| Hydrabad | 74 | 109 | 58(d) | 48 | 39(100) | 51 | 20(700) | 800 |
| Ahmedabad | 100 | 102 | 90 | 30 | 20(700) | 58 | 68 | 700 |
| $S_{dummy}$ | 0 | 0 | 0(1250) | 0 | 0 | 0 | 0 | 1250 |
| Demand | 550 | 930 | 1250 | 620 | 900 | 800 | 700 | |

Table 14: Dummy Cell

Optimality test using modi method..
Allocation table is
Iteration-1 of optimality test
1. Find $u_i$ and $v_j$ for all occupied cells$(i, j)$, where $c_{ij} = u_i + v_j$

1. Substituting, $u_3 = 0$, we get
2. $c_{31} = u_3 + v_1 \Rightarrow v_1 = c_{31} - u_3 \Rightarrow v_1 = 54 - 0 \Rightarrow v_1 = 54$
3. $c_{32} = u_3 + v_2 \Rightarrow v_2 = c_{32} - u_3 \Rightarrow v_2 = 14 - 0 \Rightarrow v_2 = 14$
4. $c_{22} = u_2 + v_2 \Rightarrow u_2 = c_{22} - v_2 \Rightarrow u_2 = 12 - 14 \Rightarrow u_2 = -2$
5. $c_{33} = u_3 + v_3 \Rightarrow v_3 = c_{33} - u_3 \Rightarrow v_3 = 65 - 0 \Rightarrow v_3 = 65$
6. $c_{63} = u_6 + v_3 \Rightarrow u_6 = c_{63} - v_3 \Rightarrow u_6 = 0 - 65 \Rightarrow u_6 = -65$
7. $c_{34} = u_3 + v_4 \Rightarrow v_4 = c_{34} - u_3 \Rightarrow v_4 = 19 - 0 \Rightarrow v_4 = 19$
8. $c_{14} = u_1 + v_4 \Rightarrow u_1 = c_{14} - v_4 \Rightarrow u_1 = 19 - 19 \Rightarrow u_1 = 0$
9. $c_{15} = u_1 + v_5 \Rightarrow v_5 = c_{15} - u_1 \Rightarrow v_5 = 49 - 0 \Rightarrow v_5 = 49$
10. $c_{45} = u_4 + v_5 \Rightarrow u_4 = c_{45} - v_5 \Rightarrow u_4 = 39 - 49 \Rightarrow u_4 = -10$
11. $c_{47} = u_4 + v_7 \Rightarrow v_7 = c_{47} - u_4 \Rightarrow v_7 = 20 + 10 \Rightarrow v_7 = 30$

| | Hongkong | Sydney | Amsterdam | Moscow | Toronto | Bangkok | Lisbon | Supply | $u_i$ |
|---|---|---|---|---|---|---|---|---|---|
| Delhi | 116 | 31 | 71 | 19(300) | 49(100) | 47(800) | 48 | 1200 | $u_1 = 0$ |
| Mumbai | 35 | 12(900) | 108 | 70 | 49 | 89 | 90 | 900 | $u_2 = -2$ |
| Chandigarh | 54(550) | 14(30) | 65(d) | 19(320) | 105 | 59 | 68 | 900 | $u_3 = 0$ |
| Hydrabad | 74 | 109 | 58 | 48 | 39(100) | 51 | 20(700) | 800 | $u_4 = -10$ |
| Ahmedabad | 100 | 102 | 90 | 30 | 20(700) | 58 | 68 | 700 | $u_5 = -29$ |
| $S_{dummy}$ | 0 | 0 | 0(1250) | 0 | 0 | 0 | 0 | 1250 | $u_6 = -65$ |
| Demand | 550 | 930 | 1250 | 620 | 900 | 800 | 700 | | |
| $v_j$ | $v_1 = 54$ | $v_2 = 14$ | $v_3 = 65$ | $v_4 = 19$ | $v_5 = 49$ | $v_6 = 47$ | $v_7 = 30$ | | |

Table 15: Allocation

12. $c_{55} = u_5 + v_5 \Rightarrow u_5 = c_{55} - v_5 \Rightarrow u_5 = 20 - 49 \Rightarrow u_5 = -29$
13. $c_{16} = u_1 + v_6 \Rightarrow v_6 = c_{16} - u_1 \Rightarrow v_6 = 47 - 0 \Rightarrow v_6 = 47$

2. Find $d_{ij}$ for all unoccupied cells(i,j), were $d_{ij} = c_{ij} - (u_i + v_j)$

1. $d_{11} = c_{11} - (u_1 + v_1) = 116 - (0 + 54) = 62$
2. $d_{12} = c_{12} - (u_1 + v_2) = 31 - (0 + 14) = 17$
3. $d_{13} = c_{11} - (u_1 + v_3) = 71 - (0 + 65) = 6$
4. $d_{17} = c_{17} - (u_1 + v_7) = 48 - (0 + 30) = 18$
5. $d_{21} = c_{21} - (u_2 + v_1) = 35 - (-2 + 54) = -17$
6. $d_{23} = c_{23} - (u_2 + v_3) = 108 - (-2 + 65) = 45$
7. $d_{24} = c_{24} - (u_2 + v_4) = 70 - (-2 + 19) = 53$
8. $d_{25} = c_{25} - (u_2 + v_5) = 49 - (-2 + 49) = 2$
9. $d_{26} = c_{26} - (u_2 + v_6) = 89 - (-2 + 47) = 44$
10. $d_{27} = c_{27} - (u_2 + v_7) = 90 - (-2 + 30) = 62$
11. $d_{33} = c_{33} - (u_3 + v_3) = 65 - (0 + 68) = -3$
12. $d_{35} = c_{35} - (u_3 + v_5) = 105 - (0 + 49) = 56$
13. $d_{36} = c_{36} - (u_3 + v_6) = 59 - (0 + 47) = 12$
14. $d_{37} = c_{37} - (u_3 + v_7) = 68 - (0 + 30) = 38$
15. $d_{41} = c_{41} - (u_4 + v_1) = 74 - (-10 + 54) = 30$
16. $d_{42} = c_{42} - (u_4 + v_2) = 109 - (-10 + 14) = 105$
17. $d_{43} = c_{33} - (u_4 + v_3) = 58 - (-10 + 65) = 3$
18. $d_{44} = c_{44} - (u_4 + v_4) = 48 - (-10 + 19) = 39$
19. $d_{46} = c_{46} - (u_4 + v_6) = 51 - (-10 + 47) = 14$
20. $d_{51} = c_{51} - (u_5 + v_1) = 100 - (-29 + 54) = 75$

21. $d_{52} = c_{52} - (u_5 + v_2) = 102 - (-29 + 14) = 117$
22. $d_{53} = c_{53} - (u_5 + v_3) = 90 - (-29 + 65) = 54$
23. $d_{54} = c_{54} - (u_5 + v_4) = 30 - (-29 + 19) = 40$
24. $d_{56} = c_{56} - (u_5 + v_6) = 58 - (-29 + 47) = 40$
25. $d_{57} = c_{57} - (u_5 + v_7) = 68 - (-29 + 30) = 67$
26. $d_{61} = c_{61} - (u_6 + v_1) = 0 - (-65 + 54) = 11$
27. $d_{62} = c_{62} - (u_6 + v_2) = 0 - (-65 + 14) = 51$
28. $d_{64} = c_{64} - (u_6 + v_4) = 0 - (-65 + 19) = 46$
29. $d_{65} = c_{65} - (u_6 + v_5) = 0 - (-65 + 49) = 16$
30. $d_{66} = c_{66} - (u_6 + v_6) = 0 - (-65 + 47) = 18$
31. $d_{67} = c_{67} - (u_6 + v_7) = 0 - (-65 + 30) = 35$

| | Hongkong | Sydney | Amsterdam | Moscow | Toronto | Bangkok | Lisbon | Supply | $u_i$ |
|---|---|---|---|---|---|---|---|---|---|
| Delhi | 116[62] | 31[17] | 71[6] | 19(300) | 49(100) | 47(800) | 48[18] | 1200 | $u_1 = 0$ |
| Mumbai | 35[-17] | 12(900) | 108[45] | 70[53] | 49[2] | 89[44] | 90[62] | 900 | $u_2 = -2$ |
| Chandigarh | 54(550) | 14(30) | 65(d) | 19(320) | 105[56] | 59[12] | 68[38] | 900 | $u_3 = 0$ |
| Hydrabad | 74[30] | 109[105] | 58[3] | 48[39] | 39(100) | 51[14] | 20(700) | 800 | $u_4 = -10$ |
| Ahmedabad | 100[75] | 102[117] | 90[54] | 30[40] | 20(700) | 58[40] | 68[67] | 700 | $u_5 = -29$ |
| $S_{dummy}$ | 0[11] | 0[51] | 0(1250) | 0[16] | 0[18] | 0[35] | 0[38] | 1250 | $u_6 = -65$ |
| Demand | 550 | 930 | 1250 | 620 | 900 | 800 | 700 | | |
| $v_j$ | $v_1 = 54$ | $v_2 = 14$ | $v_3 = 65$ | $v_4 = 19$ | $v_5 = 49$ | $v_6 = 47$ | $v_7 = 30$ | | |

Table 16: Opportunity Cost

3. We select the minimum negative value from all opportunity cost ($d_{ij}$) = $d_{21} = [-17]$ and draw a closed path from $MumbaiHonkong$.
Closed path is $MumbaiHonkong \rightarrow MumbaiSydney \rightarrow ChandigarhSydney \rightarrow ChandigarhHonkong$
Closed path and plus/minus sign allocation. 4. Minimum allocated value among all neg-

| | Hongkong | Sydney | Amsterdam | Moscow | Toronto | Bangkok | Lisbon | Supply | $u_i$ |
|---|---|---|---|---|---|---|---|---|---|
| Delhi | 116[62] | 31[17] | 71[6] | 19(300) | 49(100) | 47(800) | 48[18] | 1200 | $u_1 = 0$ |
| Mumbai | 35[-17](+) | 12(900)(-) | 108[45] | 70[53] | 49[2] | 89[44] | 90[62] | 900 | $u_2 = -2$ |
| Chandigarh | 54(550)(-) | 14(30)(+) | 65(d) | 19(320) | 105[56] | 59[12] | 68[38] | 900 | $u_3 = 0$ |
| Hydrabad | 74[30] | 109[105] | 58[3] | 48[39] | 39(100) | 51[14] | 20(700) | 800 | $u_4 = -10$ |
| Ahmedabad | 100[75] | 102[117] | 90[54] | 30[40] | 20(700) | 58[40] | 68[67] | 700 | $u_5 = -29$ |
| $S_{dummy}$ | 0[11] | 0[51] | 0(1250) | 0[16] | 0[18] | 0[35] | 0[38] | 1250 | $u_6 = -65$ |
| Demand | 550 | 930 | 1250 | 620 | 900 | 800 | 700 | | |
| $v_j$ | $v_1 = 54$ | $v_2 = 14$ | $v_3 = 65$ | $v_4 = 19$ | $v_5 = 49$ | $v_6 = 47$ | $v_7 = 30$ | | |

Table 17: Sign Allocation

ative position (-) on closed path = 550
Substract 550 from all (-) and Add it to all (+) 5. Iteration-2 of the optimality test
1. Find $u_i$ and $v_j$ for all occupied cells$(i, j)$, where $c_{ij} = u_i + v_j$

1. Substituting, $u_3 = 0$, we get

| | Hongkong | Sydney | Amsterdam | Moscow | Toronto | Bangkok | Lisbon | Supply |
|---|---|---|---|---|---|---|---|---|
| Delhi | 116 | 31 | 71 | 19(300) | 49(100) | 47(800) | 48 | 1200 |
| Mumbai | 35(550) | 12(350) | 108 | 70 | 49 | 89 | 90 | 900 |
| Chandigarh | 54 | 14(580) | 65(d) | 19(320) | 105 | 59 | 68 | 900 |
| Hydrabad | 74 | 109 | 58 | 48 | 39(100) | 51 | 20(700) | 800 |
| Ahmedabad | 100 | 102 | 90 | 30 | 20(700) | 58 | 68 | 700 |
| $S_{dummy}$ | 0 | 0 | 0(1250) | 0 | 0 | 0 | 0 | 1250 |
| Demand | 550 | 930 | 1250 | 620 | 900 | 800 | 700 | |

Table 18: Closed Path

2. $c_{14} = u_1 + v_4 \Rightarrow v_4 = c_{14} - u_1 \Rightarrow v_4 = 19 - 0 \Rightarrow v_4 = 19$
3. $c_{34} = u_3 + v_4 \Rightarrow u_3 = c_{34} - v_4 \Rightarrow u_3 = 19 - 19 \Rightarrow u_3 = 0$
4. $c_{32} = u_3 + v_2 \Rightarrow v_2 = c_{32} - u_3 \Rightarrow v_2 = 14 - 0 \Rightarrow v_2 = 14$
5. $c_{22} = u_3 + v_3 \Rightarrow u_2 = c_{22} - v_2 \Rightarrow u_2 = 12 - 42 \Rightarrow u_3 = -2$
6. $c_{21} = u_6 + v_3 \Rightarrow v_1 = c_{21} - u_2 \Rightarrow v_1 = 35 + 2 \Rightarrow v_1 = -37$
7. $c_{33} = u_3 + v_4 \Rightarrow v_3 = c_{33} - u_3 \Rightarrow v_3 = 65 - 0 \Rightarrow v_3 = 65$
8. $c_{63} = u_1 + v_4 \Rightarrow u_6 = c_{63} - v_3 \Rightarrow u_6 = 0 - 65 \Rightarrow u_6 = -65$
9. $c_{15} = u_1 + v_5 \Rightarrow v_5 = c_{15} - u_1 \Rightarrow v_5 = 49 - 0 \Rightarrow v_5 = 49$
10. $c_{45} = u_4 + v_5 \Rightarrow u_4 = c_{45} - v_5 \Rightarrow u_4 = 39 - 49 \Rightarrow u_4 = -10$
11. $c_{47} = u_4 + v_7 \Rightarrow v_7 = c_{47} - u_4 \Rightarrow v_7 = 20 + 10 \Rightarrow v_7 = 30$
12. $c_{55} = u_5 + v_5 \Rightarrow u_5 = c_{55} - v_5 \Rightarrow u_5 = 20 - 49 \Rightarrow u_5 = -29$
13. $c_{16} = u_1 + v_6 \Rightarrow v_6 = c_{16} - u_1 \Rightarrow v_6 = 47 - 0 \Rightarrow v_6 = 47$

| | Hongkong | Sydney | Amsterdam | Moscow | Toronto | Bangkok | Lisbon | Supply | $u_i$ |
|---|---|---|---|---|---|---|---|---|---|
| Delhi | 116 | 31 | 71 | 19(300) | 49(100) | 47(800) | 48 | 1200 | $u_1 = 0$ |
| Mumbai | 35(550) | 12(350) | 108 | 70 | 49 | 89 | 90 | 900 | $u_2 = -2$ |
| Chandigarh | 54 | 14(580) | 65(d) | 19(320) | 105 | 59 | 68 | 900 | $u_3 = 0$ |
| Hydrabad | 74 | 109 | 58 | 48 | 39(100) | 51 | 20(700) | 800 | $u_4 = -10$ |
| Ahmedabad | 100 | 102 | 90 | 30 | 20(700) | 58 | 68 | 700 | $u_5 = -29$ |
| $S_{dummy}$ | 0 | 0 | 0(1250) | 0 | 0 | 0 | 0 | 1250 | $u_6 = -65$ |
| Demand | 550 | 930 | 1250 | 620 | 900 | 800 | 700 | | |
| $v_j$ | $v_1 = 37$ | $v_2 = 14$ | $v_3 = 65$ | $v_4 = 19$ | $v_5 = 49$ | $v_6 = 47$ | $v_7 = 30$ | | |

Table 19: Closed Path

2. Find $d_{ij}$ for all unoccupied cells(i,j), were $d_{ij} = c_{ij} - (u_i + v_j)$

1. $d_{11} = c_{11} - (u_1 + v_1) = 116 - (0 + 37) = 79$
2. $d_{12} = c_{12} - (u_1 + v_2) = 31 - (0 + 14) = 17$
3. $d_{13} = c_{11} - (u_1 + v_3) = 71 - (0 + 65) = 6$
4. $d_{17} = c_{17} - (u_1 + v_7) = 48 - (0 + 30) = 18$

5. $d_{23} = c_{23} - (u_2 + v_3) = 108 - (-2 + 65) = 45$
6. $d_{24} = c_{24} - (u_2 + v_4) = 70 - (-2 + 19) = 53$
7. $d_{25} = c_{25} - (u_2 + v_5) = 49 - (-2 + 49) = 2$
8. $d_{26} = c_{26} - (u_2 + v_6) = 89 - (-2 + 47) = 44$
9. $d_{27} = c_{27} - (u_2 + v_7) = 90 - (-2 + 30) = 62$
10. $d_{31} = c_{31} - (u_3 + v_1) = 54 - (0 + 37) = 17$
11. $d_{35} = c_{35} - (u_3 + v_5) = 105 - (0 + 49) = 56$
12. $d_{36} = c_{36} - (u_3 + v_6) = 59 - (0 + 47) = 12$
13. $d_{37} = c_{37} - (u_3 + v_7) = 68 - (0 + 30) = 38$
14. $d_{41} = c_{41} - (u_4 + v_1) = 74 - (-10 + 37) = 30$
15. $d_{42} = c_{42} - (u_4 + v_2) = 109 - (-10 + 14) = 105$
16. $d_{43} = c_{33} - (u_4 + v_3) = 58 - (-10 + 65) = 3$
17. $d_{44} = c_{44} - (u_4 + v_4) = 48 - (-10 + 19) = 39$
18. $d_{46} = c_{46} - (u_4 + v_6) = 51 - (-10 + 47) = 14$
19. $d_{51} = c_{51} - (u_5 + v_1) = 100 - (-29 + 37) = 92$
20. $d_{52} = c_{52} - (u_5 + v_2) = 102 - (-29 + 14) = 117$
21. $d_{53} = c_{53} - (u_5 + v_3) = 90 - (-29 + 65) = 54$
22. $d_{54} = c_{54} - (u_5 + v_4) = 30 - (-29 + 19) = 40$
23. $d_{56} = c_{56} - (u_5 + v_6) = 58 - (-29 + 47) = 40$
24. $d_{57} = c_{57} - (u_5 + v_7) = 68 - (-29 + 30) = 67$
25. $d_{61} = c_{61} - (u_6 + v_1) = 0 - (-65 + 37) = 28$
26. $d_{62} = c_{62} - (u_6 + v_2) = 0 - (-65 + 14) = 51$
27. $d_{64} = c_{64} - (u_6 + v_4) = 0 - (-65 + 19) = 46$
28. $d_{65} = c_{65} - (u_6 + v_5) = 0 - (-65 + 49) = 16$
29. $d_{66} = c_{66} - (u_6 + v_6) = 0 - (-65 + 47) = 18$
30. $d_{67} = c_{67} - (u_6 + v_7) = 0 - (-65 + 30) = 35$

Since all $d_{ij} = 0$.
So Final optimal solution is arrived.

| | Hongkong | Sydney | Amsterdam | Moscow | Toronto | Bangkok | Lisbon | Supply | $u_i$ |
|---|---|---|---|---|---|---|---|---|---|
| Delhi | 116[79] | 31[17] | 71[6] | 19(300) | 49(100) | 47(800) | 48[18] | 1200 | $u_1 = 0$ |
| Mumbai | 35(550) | 12(350) | 108[45] | 70[53] | 49[2] | 89[44] | 90[62] | 900 | $u_2 = -2$ |
| Chandigarh | 54[17] | 14(580) | 65(d) | 19(320) | 105[56] | 59[12] | 68[38] | 900 | $u_3 = 0$ |
| Hydrabad | 74[47] | 109[105] | 58[3] | 48[39] | 39(100) | 51[14] | 20(700) | 800 | $u_4 = -10$ |
| Ahmedabad | 100[92] | 102[117] | 90[54] | 30[40] | 20(700) | 58[40] | 68[67] | 700 | $u_5 = -29$ |
| $S_{dummy}$ | 0[28] | 0[51] | 0(1250) | 0[46] | 0[16] | 0[18] | 0[35] | 1250 | $u_6 = -65$ |
| Demand | 550 | 930 | 1250 | 620 | 900 | 800 | 700 | | |
| $v_j$ | $v_1 = 37$ | $v_2 = 14$ | $v_3 = 65$ | $v_4 = 19$ | $v_5 = 49$ | $v_6 = 47$ | $v_7 = 30$ | | |

Table 20: Optimal Solution

| | Hongkong | Sydney | Amsterdam | Moscow | Toronto | Bangkok | Lisbon | Supply |
|---|---|---|---|---|---|---|---|---|
| Delhi | 116 | 31 | 71 | 19(300) | 49(100) | 47(800) | 48 | 1200 |
| Mumbai | 35(550) | 12(350) | 108 | 70 | 49 | 89 | 90 | 900 |
| Chandigarh | 54 | 14(580) | 65(d) | 19(320) | 105 | 59 | 68 | 900 |
| Hydrabad | 74 | 109 | 58 | 48 | 39(100) | 51 | 20(700) | 800 |
| Ahmedabad | 100 | 102 | 90 | 30 | 20(700) | 58 | 68 | 700 |
| $S_{dummy}$ | 0 | 0 | 0(1250) | 0 | 0 | 0 | 0 | 1250 |
| Demand | 550 | 930 | 1250 | 620 | 900 | 800 | 700 | |

Table 21: Final Optimal Solution

The minimum total transportation cost
= 19*300+49*100+47*800+35*550+12*350+14*580+19*320+39*100+20*700+0*1250=117750

### 3.1.1 Python Pseudo code is Given Below:

```
a = [[3,1,7,4],[2,6,5,9],[8,3,3,2]]\\cost  matrix
s = [300,400,500]\\supply matrix
d = [250,350,400,200]\\demand matrix
n = 3\\ no of sources with  supply
m = 4\\ no of destinations with demands
cost  = 0\\cost variable

r = a.copy()
c = [[0 for x in range(n)] for y in range(m)]\\matrix with columns as supply and rows as demand
for i in range(m):
    for j in range(n):
        c[i][j]=a[j][i]
\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\

def vogel(a,r,c,s,d,n,m,cost):\\function vogel
    rowcoldiff(a,r,c,s,d,n,m,cost)\\calling function rowcoldiff
def rowcoldiff(a,r,c,s,d,n,m,cost):\\function rowcoldiff
    rd = [];//set of mindiff(function) of all the rows
    cd = [];//set of mindiff(function) of all the columns

    if(n > 1 and m > 1):\\ checking the dimensions of the current matrix

        for i in range(n):
            rd.append(mindiff(r[i]))//updating the row diff matrix
        for j in range(m):
            cd.append(mindiff(c[j]))//updating the columnn diff matrix

        l1=max(rd)// finding the maximun value of row differences
        l2=max(cd)// finding the maximun value of column differences
        if l1 >= l2:

            k = rd.index(l1)#rowindex//finding the index of maximum element of row column differences
            k1 = min(r[k])//finding the min value of the row(kth)
            k2 = r[k].index(k1)#columnindex//finding the column index of k1
            if s[k] >= d[k2]:// comparing supply and demand of the cell and fullfing the demand
                s[k]=s[k]-d[k2]// upating the supply
                cost = cost + k1*d[k2]//updating the cost
                d[k2]=0//updating rhe demand
                for i in range(n):
                    r[i].remove(r[i][k2])// removing the column from r matrix after fulfilling its demand
                c.remove(c[k2]) //removing column from c matrix
                d.remove(d[k2])// removing the fullfilled demand
                m=m-1//changing the size of matrix since a column is removed

                print(rd)
                print(cd)
                print(d,1)
                print(s,1)

            else:
                d[k2]=d[k2]-s[k]//updating the demand all the supply is over in that row
                cost = cost + k1*s[k]//updating cost
                s[k]=0//updating completed supply
                for i in range(m):
                    c[i].remove(c[i][k])//removinng the row from c matrix
                r.remove(r[k])//removing row from r matrix
                s.remove(s[k])//removing completed supply
                n=n-1//changing the size ofmatrix since a row is removed
```

Figure 11: Pseudo code

```
            print(rd)
            print(cd)
            print(d,2)
            print(s,2)
    else://when the maximum of mindiff value is in the cd array.rest of code under this else statement is same

        k = cd.index(l2)#column index

        k1 = min(c[k])

        k2 = c[k].index(k1)


        if s[k2] >= d[k]:
            s[k2]=s[k2]-d[k]
            cost = cost + k1*d[k]
            d[k]=0
            for i in range(n):
                r[i].remove(r[i][k])
            c.remove(c[k])
            d.remove(d[k])
            m=m-1

            print(rd)
            print(cd)
            print(d,1)
            print(s,1)


        else:
            d[k]=d[k]-s[k2]
            cost = cost + k1*s[k2]
            s[k2]=0
            for i in range(m):
                c[i].remove(c[i][k2])
            r.remove(r[k2])
            s.remove(s[k2])
            n=n-1

            print(rd)
            print(cd)
            print(d,2)
            print(s,2)

elif (n == 1 and  m > 1):

    k = min(r[0])//only one row is left
    k1 = r[0].index(k)#columnindex//finding the element in the cell
    k2 = 0//only one row is left so all cells  in all columns has same row value
    if s[0] >= d[k1]:
            s[0]=s[0]-d[k1]
            cost = cost + k*d[k2]
            d[k2]=0
            for i in range(n):
                r[i].remove(r[i][k1])
            c.remove(c[k1])
            d.remove(d[k1])
            m=m-1

            print(rd)
            print(cd)
            print(d,1)
            print(s,1)
```

Figure 12: Pseudo Code

# 4 Chapter 4

## 4.1 Case2: Crew Assignment

We must identify an assigned task of the requisite technical crew members to these flight usefulness after generating all valid flight services cover at least once each flight segment. Each planned flight segment has an appropriate technical crew to run it. This plan aims to find a possible low-cost assignment such that assigns each crew to flights with the lowest possible cost.
Operations Research has aided the aviation sector in transforming itself in order to compete effectively in the marketplace and meet the complicated expectations of customers.

The usage of the Hungarian Assignment issue for crew assignments on Air India and Indigo flights is highlighted in the article.
Our goal is to provide readers with valuable insights into the portion of the aviation business that effectively employs Operations Research methodology to reduce costs, increase revenues, and improve the efficiency of the airline sector. We'll discuss a few industry issues and then explain and cite the solution that's frequently employed to profit on these opportunistic opportunities.
We also want to broaden your horizons by covering a real-world, practical problem, the Air Crew Assignment Problem, and how to solve it using a simple solution technique called the Hungarian Method to save time and costs while increasing profits. This would allow a logical and factual understanding of the OR method employed in flying.
The aircrew assignment problem is a subproblem of the airline crew scheduling problem. The purpose of the crew assignment problem is to figure out the most efficient way to assign crew members to a set of crew pairs.
For any airline, air crew management is a critical duty. Flights should be arranged to maximise both cost and time efficiency. To address the crew assignment problem for AIRINDIA Airlines, we employed Harold Kuhn's Hungarian Assignment technique, which he invented in 1955.
The to-and-fro timetable for flights between Delhi and Mumbai is displayed in the table below. We've anticipated that the crew will be given at least 6 hours of layover time (rest time). The flight numbers have been changed to make it easier for readers to understand.

## 4.2 Crew Assignment Problem

| Delhi-Mumbai | | | Mumbai-Delhi | | |
|---|---|---|---|---|---|
| Flight No | Departure | Arrival | Flight No | Departure | Arrival |
| a | 06.00 | 12.00 | 1 | 05.30 | 11.30 |
| b | 07.30 | 13.30 | 2 | 09.00 | 15.00 |
| c | 11.30 | 17.30 | 3 | 15.00 | 21.00 |
| d | 19.00 | 01.00 | 4 | 18.30 | 00.30 |
| e | 00.30 | 06.30 | 5 | 00.00 | 06.00 |
| f | 06.30 | 11.30 | 6 | 06.00 | 11.30 |
| g | 08.30 | 13.00 | 7 | 09.30 | 15.30 |
| h | 11.00 | 17.00 | 8 | 15.30 | 21.00 |
| i | 19.30 | 01.30 | 9 | 19.30 | 00.30 |
| j | 12.30 | 07.30 | 10 | 00.30 | 06.00 |

**Solution:**
To select optimal positions, we first calculate layover times from the above table.
Computing values for table 1(layover time)
$1^{st}$ Row

| sections | A.Time(Mumbai) | D.time(Mumbai) | Resting time(diff.) |
|---|---|---|---|
| 1 | 12:00 | 5:30 | 17.5 |
| 2 | 12:00 | 09:00 | 21 |
| 3 | 12:00 | 15:00 | 27 |
| 4 | 12:00 | 18:30 | 6.5 |
| 5 | 12:00 | 00:00 | 12 |
| 6 | 12:00 | 06:00 | 18 |
| 7 | 12:00 | 09:30 | 21.5 |
| 8 | 12:00 | 15:30 | 27.5 |
| 9 | 12:00 | 19:30 | 7.5 |
| 10 | 12:00 | 00:30 | 12.5 |

2nd row

| sections | A.Time(Mumbai) | D.time(Mumbai) | Resting time(diff.) |
|---|---|---|---|
| 1 | 13:30 | 5:30 | 16 |
| 2 | 13:30 | 09:00 | 19.5 |
| 3 | 13:30 | 15:00 | 25.5 |
| 4 | 13:30 | 18:30 | 5 |
| 5 | 13:30 | 00:00 | 10.5 |
| 6 | 13:30 | 06:00 | 16.5 |
| 7 | 13:30 | 09:30 | 20 |
| 8 | 13:30 | 15:30 | 26 |
| 9 | 13:30 | 19:30 | 6 |
| 10 | 13:30 | 00:30 | 11 |

3rd row

| sections | A.Time(Mumbai) | D.time(Mumbai) | Resting time(diff.) |
|---|---|---|---|
| 1 | 17:30 | 5:30 | 12 |
| 2 | 17:30 | 09:00 | 15.5 |
| 3 | 17:30 | 15:00 | 21.5 |
| 4 | 17:30 | 18:30 | 25 |
| 5 | 17:30 | 00:00 | 6.5 |
| 6 | 17:30 | 06:00 | 12.5 |
| 7 | 17:30 | 09:30 | 16 |
| 8 | 17:30 | 15:30 | 22 |
| 9 | 17:30 | 19:30 | 26 |
| 10 | 17:30 | 00:30 | 7 |

4th row

| sections | A.Time(Mumbai) | D.time(Mumbai) | Resting time(diff.) |
|---|---|---|---|
| 1 | 01:00 | 5:30 | 4.5 |
| 2 | 01:00 | 09:00 | 8 |
| 3 | 01:00 | 15:00 | 14 |
| 4 | 01:00 | 18:30 | 17.5 |
| 5 | 01:00 | 00:00 | 23 |
| 6 | 01:00 | 06:00 | 5 |
| 7 | 01:00 | 09:30 | 8.5 |
| 8 | 01:00 | 15:30 | 14.5 |
| 9 | 01:00 | 19:30 | 18.5 |
| 10 | 01:00 | 00:30 | 23.5 |

5th row

| sections | A.Time(Mumbai) | D.time(Mumbai) | Resting time(diff.) |
|---|---|---|---|
| 1 | 06:30 | 5:30 | 23 |
| 2 | 06:30 | 09:00 | 26.5 |
| 3 | 06:30 | 15:00 | 8.5 |
| 4 | 06:30 | 18:30 | 12 |
| 5 | 06:30 | 00:00 | 17.5 |
| 6 | 06:30 | 06:00 | 23.5 |
| 7 | 06:30 | 09:30 | 27 |
| 8 | 06:30 | 15:30 | 9 |
| 9 | 06:30 | 19:30 | 13 |
| 10 | 06:30 | 00:30 | 18 |

6th row

| sections | A.Time(Mumbai) | D.time(Mumbai) | Resting time(diff.) |
|---|---|---|---|
| 1 | 11:30 | 5:30 | 18 |
| 2 | 11:30 | 09:00 | 21.5 |
| 3 | 11:30 | 15:00 | 27.5 |
| 4 | 11:30 | 18:30 | 7 |
| 5 | 11:30 | 00:00 | 12.5 |
| 6 | 11:30 | 06:00 | 18.5 |
| 7 | 11:30 | 09:30 | 22 |
| 8 | 11:30 | 15:30 | 28 |
| 9 | 11:30 | 19:30 | 8 |
| 10 | 11:30 | 00:30 | 13 |

7th row

| sections | A.Time(Mumbai) | D.time(Mumbai) | Resting time(diff.) |
|---|---|---|---|
| 1 | 13:00 | 5:30 | 16.5 |
| 2 | 13:00 | 09:00 | 20 |
| 3 | 13:00 | 15:00 | 26 |
| 4 | 13:00 | 18:30 | 5.5 |
| 5 | 13:00 | 00:00 | 11 |
| 6 | 13:00 | 06:00 | 17 |
| 7 | 13:00 | 09:30 | 20.5 |
| 8 | 13:00 | 15:30 | 26.5 |
| 9 | 13:00 | 19:30 | 6.5 |
| 10 | 13:00 | 00:30 | 11.5 |

8th row

| sections | A.Time(Mumbai) | D.time(Mumbai) | Resting time(diff.) |
|---|---|---|---|
| 1 | 17:00 | 5:30 | 12.5 |
| 2 | 17:00 | 09:00 | 16 |
| 3 | 17:00 | 15:00 | 22 |
| 4 | 17:00 | 18:30 | 25.5 |
| 5 | 17:00 | 00:00 | 7 |
| 6 | 17:00 | 06:00 | 13 |
| 7 | 17:00 | 09:30 | 16.5 |
| 8 | 17:00 | 15:30 | 22.5 |
| 9 | 17:00 | 19:30 | 26.5 |
| 10 | 17:00 | 00:30 | 7.5 |

9th row

| sections | A.Time(Mumbai) | D.time(Mumbai) | Resting time(diff.) |
|---|---|---|---|
| 1 | 01:30 | 5:30 | 28 |
| 2 | 01:30 | 09:00 | 7.5 |
| 3 | 01:30 | 15:00 | 13.5 |
| 4 | 01:30 | 18:30 | 17 |
| 5 | 01:30 | 00:00 | 22.5 |
| 6 | 01:30 | 06:00 | 4.5 |
| 7 | 01:30 | 09:30 | 8 |
| 8 | 01:30 | 15:30 | 14 |
| 9 | 01:30 | 19:30 | 18 |
| 10 | 01:30 | 00:30 | 23 |

10th row

| sections | A.Time(Mumbai) | D.time(Mumbai) | Resting time(diff.) |
|---|---|---|---|
| 1 | 07:30 | 5:30 | 22 |
| 2 | 07:30 | 09:00 | 25.5 |
| 3 | 07:30 | 15:00 | 7.5 |
| 4 | 07:30 | 18:30 | 11 |
| 5 | 07:30 | 00:00 | 16.5 |
| 6 | 07:30 | 06:00 | 22.5 |
| 7 | 07:30 | 09:30 | 26 |
| 8 | 07:30 | 15:30 | 8 |
| 9 | 07:30 | 19:30 | 12 |
| 10 | 07:30 | 00:30 | 17 |

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| a | 17.5 | 21 | 27 | 6.5 | 12 | 18 | 21.5 | 27.5 | 7.5 | 12.5 | |
| b | 16 | 19.5 | 25.5 | 5 | 10.5 | 16.5 | 20 | 26 | 6 | 11 | |
| c | 12 | 15.5 | 21.5 | 25 | 6.5 | 12.5 | 16 | 22 | 26 | 7 | |
| d | 4.5 | 8 | 14 | 17.5 | 23 | 5 | 8.5 | 14.5 | 18.5 | 23.5 | |
| e | 23 | 26.5 | 8.5 | 12 | 17.5 | 23.5 | 27 | 9 | 13 | 18 | |
| f | 18 | 21.5 | 27.5 | 7 | 12.5 | 18.5 | 22 | 28 | 8 | 13 | |
| g | 16.5 | 20 | 26 | 5.5 | 11 | 17 | 20.5 | 26.5 | 6.5 | 11.5 | |
| h | 12.5 | 16 | 22 | 25.5 | 7 | 13 | 16.5 | 22.5 | 26.5 | 7.5 | |
| i | 28 | 7.5 | 13.5 | 17 | 22.5 | 4.5 | 8 | 14 | 18 | 23 | |
| j | 22 | 25.5 | 7.5 | 11 | 16.5 | 22.5 | 26 | 8 | 12 | 17 | |
| | | | | | | | | | | | |

Calculating values for table 2 (layover time)
1st Column

| sections | A.Time(Mumbai) | D.time(Mumbai) | Resting time(diff.) |
|---|---|---|---|
| 1 | 11:30 | 06:00 | 18.5 |
| 2 | 15:00 | 06:00 | 15 |
| 3 | 21:00 | 06:00 | 9 |
| 4 | 00:30 | 06:00 | 5.5 |
| 5 | 06:00 | 06:00 | 24 |
| 6 | 11:30 | 06:00 | 18.5 |
| 7 | 15:30 | 06:00 | 14.5 |
| 8 | 21:00 | 06:00 | 9 |
| 9 | 00:30 | 06:00 | 5.5 |
| 10 | 06:00 | 06:00 | 24 |

2nd Column

| sections | A.Time(Mumbai) | D.time(Mumbai) | Resting time(diff.) |
|---|---|---|---|
| 1 | 11:30 | 07:30 | 20 |
| 2 | 15:00 | 07:30 | 16.5 |
| 3 | 21:00 | 07:30 | 10.5 |
| 4 | 00:30 | 07:30 | 7 |
| 5 | 06:00 | 07:30 | 25.5 |
| 6 | 11:30 | 07:30 | 20 |
| 7 | 15:30 | 07:30 | 16 |
| 8 | 21:00 | 07:30 | 10.5 |
| 9 | 00:30 | 07:30 | 7 |
| 10 | 06:00 | 07:30 | 25.5 |

3rd Column

| sections | A.Time(Mumbai) | D.time(Mumbai) | Resting time(diff.) |
|---|---|---|---|
| 1 | 11:30 | 11:30 | 24 |
| 2 | 15:00 | 11:30 | 20.5 |
| 3 | 21:00 | 11:30 | 14.5 |
| 4 | 00:30 | 11:30 | 11 |
| 5 | 06:00 | 11:30 | 5.5 |
| 6 | 11:30 | 11:30 | 24 |
| 7 | 15:30 | 11:30 | 20 |
| 8 | 21:00 | 11:30 | 14.5 |
| 9 | 00:30 | 11:30 | 11 |
| 10 | 06:00 | 11:30 | 5.5 |

4th Column

| sections | A.Time(Mumbai) | D.time(Mumbai) | Resting time(diff.) |
|---|---|---|---|
| 1 | 11:30 | 19:00 | 7.5 |
| 2 | 15:00 | 19:00 | 28 |
| 3 | 21:00 | 19:00 | 22 |
| 4 | 00:30 | 19:00 | 18.5 |
| 5 | 06:00 | 19:00 | 13 |
| 6 | 11:30 | 19:00 | 7.5 |
| 7 | 15:30 | 19:00 | 27.5 |
| 8 | 21:00 | 19:00 | 22 |
| 9 | 00:30 | 19:00 | 18.5 |
| 10 | 06:00 | 19:00 | 13 |

5th Column

| sections | A.Time(Mumbai) | D.time(Mumbai) | Resting time(diff.) |
|---|---|---|---|
| 1 | 11:30 | 00:30 | 13 |
| 2 | 15:00 | 00:30 | 9.5 |
| 3 | 21:00 | 00:30 | 27.5 |
| 4 | 00:30 | 00:30 | 24 |
| 5 | 06:00 | 00:30 | 18.5 |
| 6 | 11:30 | 00:30 | 13 |
| 7 | 15:30 | 00:30 | 9 |
| 8 | 21:00 | 00:30 | 27.5 |
| 9 | 00:30 | 00:30 | 24 |
| 10 | 06:00 | 00:30 | 18.5 |

6th Column

| sections | A.Time(Mumbai) | D.time(Mumbai) | Resting time(diff.) |
|---|---|---|---|
| 1 | 11:30 | 06:30 | 19 |
| 2 | 15:00 | 06:30 | 15.5 |
| 3 | 21:00 | 06:30 | 9.5 |
| 4 | 00:30 | 06:30 | 6 |
| 5 | 06:00 | 06:30 | 24.5 |
| 6 | 11:30 | 06:30 | 19 |
| 7 | 15:30 | 06:30 | 15 |
| 8 | 21:00 | 06:30 | 9.5 |
| 9 | 00:30 | 06:30 | 6 |
| 10 | 06:00 | 06:30 | 24.5 |

7th Column

| sections | A.Time(Mumbai) | D.time(Mumbai) | Resting time(diff.) |
|---|---|---|---|
| 1 | 11:30 | 08:30 | 21 |
| 2 | 15:00 | 08:30 | 17.5 |
| 3 | 21:00 | 08:30 | 11.5 |
| 4 | 00:30 | 08:30 | 8 |
| 5 | 06:00 | 08:30 | 26.5 |
| 6 | 11:30 | 08:30 | 21 |
| 7 | 15:30 | 08:30 | 17 |
| 8 | 21:00 | 08:30 | 11.5 |
| 9 | 00:30 | 08:30 | 8 |
| 10 | 06:00 | 08:30 | 26.5 |

8th Column

| sections | A.Time(Mumbai) | D.time(Mumbai) | Resting time(diff.) |
|---|---|---|---|
| 1 | 11:30 | 11:00 | 23.5 |
| 2 | 15:00 | 11:00 | 20 |
| 3 | 21:00 | 11:00 | 14 |
| 4 | 00:30 | 11:00 | 10.5 |
| 5 | 06:00 | 11:00 | 5 |
| 6 | 11:30 | 11:00 | 23.5 |
| 7 | 15:30 | 11:00 | 19.5 |
| 8 | 21:00 | 11:00 | 14 |
| 9 | 00:30 | 11:00 | 10.5 |
| 10 | 06:00 | 11:00 | 5 |

9th Column

| sections | A.Time(Mumbai) | D.time(Mumbai) | Resting time(diff.) |
|---|---|---|---|
| 1 | 11:30 | 19:30 | 8 |
| 2 | 15:00 | 19:30 | 4.5 |
| 3 | 21:00 | 19:30 | 22.5 |
| 4 | 00:30 | 19:30 | 19 |
| 5 | 06:00 | 19:30 | 13.5 |
| 6 | 11:30 | 19:30 | 8 |
| 7 | 15:30 | 19:30 | 28 |
| 8 | 21:00 | 19:30 | 22.5 |
| 9 | 00:30 | 19:30 | 19 |
| 10 | 06:00 | 19:30 | 13.5 |

10th Column

| sections | A.Time(Mumbai) | D.time(Mumbai) | Resting time(diff.) |
|---|---|---|---|
| 1 | 11:30 | 12:30 | 25 |
| 2 | 15:00 | 12:30 | 21.5 |
| 3 | 21:00 | 12:30 | 15.5 |
| 4 | 00:30 | 12:30 | 12 |
| 5 | 06:00 | 12:30 | 6.5 |
| 6 | 11:30 | 12:30 | 25 |
| 7 | 15:30 | 12:30 | 21 |
| 8 | 21:00 | 12:30 | 15.5 |
| 9 | 00:30 | 12:30 | 12 |
| 10 | 06:00 | 12:30 | 6.5 |

Table-2:Crew Based at Mumbai airport

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| a | 18.5 | 15 | 9 | 5.5 | 24 | 18.5 | 14.5 | 9 | 5.5 | 24 | |
| b | 20 | 16.5 | 10.5 | 7 | 25.5 | 20 | 16 | 10.5 | 7 | 25.5 | |
| c | 24 | 20.5 | 14.5 | 11 | 5.5 | 24 | 20 | 14.5 | 11 | 5.5 | |
| d | 7.5 | 28 | 22 | 18.5 | 13 | 7.5 | 27.5 | 22 | 18.5 | 13 | |
| e | 13 | 9.5 | 27.5 | 24 | 18.5 | 13 | 9 | 27.5 | 24 | 18.5 | |
| f | 19 | 15.5 | 9.5 | 6 | 24.5 | 19 | 15 | 9.5 | 6 | 24.5 | |
| g | 21 | 17.5 | 11.5 | 8 | 26.5 | 21 | 17 | 11.5 | 8 | 26.5 | |
| h | 23.5 | 20 | 14 | 10.5 | 5 | 23.5 | 19.5 | 14 | 10.5 | 5 | |
| i | 8 | 4.5 | 22.5 | 19 | 13.5 | 8 | 28 | 22.5 | 19 | 13.5 | |
| j | 25 | 21.5 | 15.5 | 12 | 6.5 | 25 | 21 | 15.5 | 12 | 6.5 | |
| | | | | | | | | | | | |

The combined layover time matrix (Table-3) is created by selecting the smallest element from the 2 matched elements in Table-2. The Crew is Based in Mumbai if the (*) present; else, The Crew is Based in Delhi.

Table-3

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| a | 17.5 | 15* | 9* | 5.5* | 12 | 18 | 14.5* | 9* | 5.5* | 12.5 | |
| b | 16 | 16.5* | 10.5* | 5 | 10.5 | 16.5 | 16* | 10.5* | 6 | 11 | |
| c | 12 | 15.5 | 14.5* | 11* | 5.5* | 12.5 | 16 | 14.5* | 11* | 5.5* | |
| d | 4.5 | 8 | 14 | 17.5 | 13* | 5 | 8.5 | 14.5 | 18.5* | 13* | |
| e | 13* | 9.5* | 8.5 | 12 | 17.5 | 13* | 9* | 9 | 13 | 18 | |
| f | 18 | 15.5* | 9.5* | 6* | 12.5 | 18.5 | 15* | 9.5* | 6* | 13 | |
| g | 16.5 | 15.5* | 11.5* | 5.5 | 11 | 17 | 17* | 11.5* | 6.5 | 11.5 | |
| h | 12.5 | 16 | 14* | 10.5* | 5* | 13 | 16.5 | 14* | 10.5* | 5* | |
| i | 8* | 4.5* | 13.5 | 17 | 13.5* | 4.5 | 8 | 14 | 18 | 13.5* | |
| j | 22 | 21.5* | 7.5 | 11 | 6.5* | 22.5 | 21* | 8 | 12* | 6.5* | |
| | | | | | | | | | | | |

now the above problem simply solvre by Hungarian Method.

Step-1: Locate the least element in each row subtract it

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| a | 12 | 9.5 | 3.5 | 0 | 6.5 | 12.5 | 9 | 3.5 | 0 | 7 | (-5.5) |
| b | 11 | 11.5 | 5.5 | 0 | 5.5 | 11.5 | 11 | 5.5 | 1 | 6 | (-5) |
| c | 6.5 | 10 | 9 | 5.5 | 0 | 7 | 10.5 | 9 | 5.5 | 0 | (-5.5) |
| d | 0 | 3.5 | 9.5 | 13 | 8.5 | 0.5 | 4 | 10 | 14 | 8.5 | (-4.5) |
| e | 4.5 | 1 | 0 | 3.5 | 9 | 4.5 | 0.5 | 0.5 | 4.5 | 9.5 | (-8.5) |
| f | 12 | 9.5 | 3.5 | 0 | 6.5 | 12.5 | 9 | 3.5 | 0 | 7 | (-6) |
| g | 11 | 12 | 6 | 0 | 5.5 | 11.5 | 11.5 | 6 | 1 | 6 | (-5.5) |
| h | 7.5 | 11 | 9 | 5.5 | 0 | 8 | 11.5 | 9 | 5.5 | 0 | (-5) |
| i | 3.5 | 0 | 9 | 12.5 | 9 | 0 | 3.5 | 9.5 | 13.5 | 9 | (-4.5) |
| j | 15.5 | 15 | 1 | 4.5 | 0 | 16 | 14.5 | 1.5 | 5.5 | 0 | (-6.5) |
| | | | | | | | | | | | |

Step-2: Locate the least element in each column subtract it

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| a | 12 | 9.5 | 3.5 | 0 | 6.5 | 12.5 | 8.5 | 3 | 0 | 7 | |
| b | 11 | 11.5 | 5.5 | 0 | 5.5 | 11.5 | 10.5 | 5 | 1 | 6 | |
| c | 6.5 | 10 | 9 | 5.5 | 0 | 7 | 10 | 8.5 | 5.5 | 0 | |
| d | 0 | 3.5 | 9.5 | 13 | 8.5 | 0.5 | 3.5 | 9.5 | 14 | 8.5 | |
| e | 4.5 | 1 | 0 | 3.5 | 9 | 4.5 | 0 | 0 | 4.5 | 9.5 | |
| f | 12 | 9.5 | 3.5 | 0 | 6.5 | 12.5 | 8.5 | 3 | 0 | 7 | |
| g | 11 | 12 | 6 | 0 | 5.5 | 11.5 | 11 | 5.5 | 1 | 6 | |
| h | 7.5 | 11 | 9 | 5.5 | 0 | 8 | 11 | 8.5 | 5.5 | 0 | |
| i | 3.5 | 0 | 9 | 12.5 | 9 | 0 | 3 | 9 | 13.5 | 9 | |
| j | 15.5 | 15 | 1 | 4.5 | 0 | 16 | 14 | 1 | 5.5 | 0 | |
| | (-0) | (-0) | (-0) | (-0) | (-0) | (-0) | (-0.5) | (-0.5) | (-0) | (-0) | |

Step-3: With the fewest number of lines possible, cover all zeros. Determine the smallest number of lines necessary to cover all of the zeros in the matrix. Because the number of lines required to cover all zeros is fewer than the size of the matrix (10), go to step 4.

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| a | 12 | 9.5 | 3.5 | 0 | 6.5 | 12.5 | 8.5 | 3 | 0 | 7 | |
| b | 11 | 11.5 | 5.5 | 0 | 5.5 | 11.5 | 10.5 | 5 | 1 | 6 | |
| c | 6.5 | 10 | 9 | 5.5 | 0 | 7 | 10 | 8.5 | 5.5 | 0 | |
| d | 0 | 3.5 | 9.5 | 13 | 8.5 | 0.5 | 3.5 | 9.5 | 14 | 8.5 | |
| e | 4.5 | 1 | 0 | 3.5 | 9 | 4.5 | 0 | 0 | 4.5 | 9.5 | ✓ |
| f | 12 | 9.5 | 3.5 | 0 | 6.5 | 12.5 | 8.5 | 3 | 0 | 7 | |
| g | 11 | 12 | 6 | 0 | 5.5 | 11.5 | 11 | 5.5 | 1 | 6 | |
| h | 7.5 | 11 | 9 | 5.5 | 0 | 8 | 11 | 8.5 | 5.5 | 0 | |
| i | 3.5 | 0 | 9 | 12.5 | 9 | 0 | 3 | 9 | 13.5 | 9 | ✓ |
| j | 15.5 | 15 | 1 | 4.5 | 0 | 16 | 14 | 1 | 5.5 | 0 | |
| | ✓ | | | ✓ | ✓ | | | | ✓ | ✓ | |

Step-4: Increase the number of zeros. Select the smallest element among the cells not covered by any line (say k=0.5) to create the revised table. Subtract k=0.5 from every element not covered by a line in the cell. In the intersection cell of two lines, multiply each element by 0.5.

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| a | 12 | 9 | 3 | 0 | 6.5 | 12 | 9 | 2.5 | 0 | 7 | |
| b | 11 | 11 | 5 | 0 | 5.5 | 11 | 10 | 4.5 | 1 | 6 | |
| c | 6.5 | 9.5 | 8.5 | 5.5 | 0 | 7 | 9.5 | 8 | 5.5 | 0 | |
| d | 0 | 3 | 9 | 13 | 8.5 | 0.5 | 3 | 9 | 14 | 8.5 | |
| e | 5 | 1 | 0 | 4 | 9.5 | 4.5 | 0 | 0 | 5 | 10 | |
| f | 12 | 9 | 3 | 0 | 6.5 | 12.5 | 8 | 2.5 | 0 | 7 | |
| g | 11 | 11.5 | 5.5 | 0 | 5.5 | 11.5 | 10.5 | 5 | 1 | 6 | |
| h | 7.5 | 10.5 | 8.5 | 5.5 | 0 | 8 | 10.5 | 8 | 5.5 | 0 | |
| i | 4 | 0 | 9 | 13 | 9.5 | 0 | 3 | 9 | 14 | 9.5 | |
| j | 15.5 | 14.5 | 0.5 | 4.5 | 0 | 16 | 13.5 | 0.5 | 5.5 | 0 | |
| | | | | | | | | | | | |

Step-3: Cover all zeros with a minimum number of lines Determine the minimum number of lines, required to cover all zeros in the matrix. There are 7 lines required to cover all zeros, which is less than size of matrix (10), so go to step-4

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| a | 12 | 9 | 3 | 0 | 6.5 | 12 | 9 | 2.5 | 0 | 7 | |
| b | 11 | 11 | 5 | 0 | 5.5 | 11 | 10 | 4.5 | 1 | 6 | |
| c | 6.5 | 9.5 | 8.5 | 5.5 | 0 | 7 | 9.5 | 8 | 5.5 | 0 | |
| d | 0 | 3 | 9 | 13 | 8.5 | 0.5 | 3 | 9 | 14 | 8.5 | ✓ |
| e | 5 | 1 | 0 | 4 | 9.5 | 4.5 | 0 | 0 | 5 | 10 | ✓ |
| f | 12 | 9 | 3 | 0 | 6.5 | 12.5 | 8 | 2.5 | 0 | 7 | |
| g | 11 | 11.5 | 5.5 | 0 | 5.5 | 11.5 | 10.5 | 5 | 1 | 6 | |
| h | 7.5 | 10.5 | 8.5 | 5.5 | 0 | 8 | 10.5 | 8 | 5.5 | 0 | |
| i | 4 | 0 | 9 | 13 | 9.5 | 0 | 3 | 9 | 14 | 9.5 | ✓ |
| j | 15.5 | 14.5 | 0.5 | 4.5 | 0 | 16 | 13.5 | 0.5 | 5.5 | 0 | |
| | | | | ✓ | ✓ | | | | ✓ | ✓ | |

Step-4: Create additional zeros, Develop the revised table by selecting the smallest element, among the cells not covered by any line (say k=0.5) Subtract k=0.5 from every element in the cell not covered by a line. Add k=0.5 to every element in the intersection cell of two lines.

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| a | 11.5 | 8.5 | 2.5 | 0 | 6.5 | 11.5 | 7.5 | 2 | 0 | 7 | |
| b | 10.5 | 10.5 | 4.5 | 0 | 5.5 | 10.5 | 9.5 | 4 | 1 | 6 | |
| c | 6 | 9 | 8 | 5.5 | 0 | 6 | 9 | 7.5 | 5.5 | 0 | |
| d | 0 | 3 | 9 | 13.5 | 9 | 0 | 3 | 9 | 14.5 | 9 | |
| e | 5 | 1 | 0 | 4.5 | 10 | 4.5 | 0 | 0 | 5.5 | 10.5 | |
| f | 11.5 | 8.5 | 2.5 | 0 | 6.5 | 11.5 | 7.5 | 2 | 0 | 7 | |
| g | 10.5 | 11 | 5 | 0 | 5.5 | 10.5 | 10 | 4.5 | 1 | 6 | |
| h | 7 | 10 | 8 | 5.5 | 0 | 7 | 10 | 7.5 | 5.5 | 0 | |
| i | 4 | 0 | 9 | 13.5 | 10 | 0 | 3 | 9 | 14.5 | 10 | |
| j | 15 | 14 | 0 | 4.5 | 0 | 15 | 13 | 0 | 5.5 | 0 | |
| | | | | | | | | | | | |

Step-3: With the fewest number of lines possible, cover all zeros. Calculate the smallest number of lines needed to cover all of the zeros in the matrix. To cover all zeros, nine lines are necessary, which is fewer than the size of the matrix (10); thus, go to step-4.

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| a | 9.5 | 6.5 | 0.5 | 0 | 6.5 | 9.5 | 5.5 | 0 | 0 | 7 | |
| b | 8.5 | 8.5 | 2.5 | 0 | 5.5 | 8.5 | 7.5 | 2 | 1 | 6 | |
| c | 4 | 7 | 6 | 5.5 | 0 | 4 | 7 | 5.5 | 5.5 | 0 | |
| d | 0 | 3 | 9 | 15.5 | 11 | 0 | 3 | 9 | 16.5 | 11 | ✓ |
| e | 5 | 1 | 0 | 6.5 | 12 | 4.5 | 0 | 0 | 7.5 | 12.5 | |
| f | 9.5 | 6.5 | 0.5 | 0 | 6.5 | 9.5 | 5.5 | 0 | 0 | 7 | |
| g | 8.5 | 9 | 3 | 0 | 5.5 | 8.5 | 8 | 2.5 | 1 | 6 | |
| h | 5 | 8 | 6 | 5.5 | 0 | 5 | 8 | 5.5 | 5.5 | 0 | |
| i | 4 | 0 | 9 | 15.5 | 12 | 0 | 3 | 9 | 16.5 | 12 | ✓ |
| j | 15 | 14 | 0 | 6.5 | 2 | 15 | 13 | 0 | 7.5 | 2 | |
| | | | ✓ | ✓ | ✓ | | ✓ | ✓ | ✓ | ✓ | |

Step-4: Increase the number of zeros. Select the minor element among the cells not covered by any line (say k=1) to create the revised table. Subtract k=1 from every element not covered by a line in the cell. Every element in the intersection cell of two lines should have k=1.

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| a | 8.5 | 5.5 | 0.5 | 0 | 6.5 | 8.5 | 5.5 | 0 | 0 | 7 | |
| b | 7.5 | 7.5 | 2.5 | 0 | 5.5 | 7.5 | 7.5 | 2 | 1 | 6 | |
| c | 3 | 6 | 6 | 5.5 | 0 | 3 | 7 | 5.5 | 5.5 | 0 | |
| d | 0 | 3 | 10 | 16.5 | 12 | 0 | 4 | 10 | 17.5 | 12 | |
| e | 4 | 0 | 0 | 6.5 | 12 | 3.5 | 0 | 0 | 7.5 | 12.5 | |
| f | 8.5 | 5.5 | 0.5 | 0 | 6.5 | 8.5 | 5.5 | 0 | 0 | 7 | |
| g | 7.5 | 8 | 3 | 0 | 5.5 | 7.5 | 8 | 2.5 | 1 | 6 | |
| h | 4 | 7 | 6 | 5.5 | 0 | 4 | 8 | 5.5 | 5.5 | 0 | |
| i | 4 | 0 | 10 | 16.5 | 13 | 0 | 4 | 10 | 17.5 | 13 | |
| j | 14 | 13 | 0 | 6.5 | 2 | 14 | 13 | 0 | 7.5 | 2 | |
| | | | | | | | | | | | |

Step-3: Cover all zeros using the fewest number of lines feasible. Calculate the fewest number of lines required to cover all of the matrix's zeros. Proceed to step 4 because the number of lines required to cover all zeros is less than the size of the matrix (10).

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| a | 8.5 | 5.5 | 0.5 | 0 | 6.5 | 8.5 | 5.5 | 0 | 0 | 7 | |
| b | 7.5 | 7.5 | 2.5 | 0 | 5.5 | 7.5 | 7.5 | 2 | 1 | 6 | |
| c | 3 | 6 | 6 | 5.5 | 0 | 3 | 7 | 5.5 | 5.5 | 0 | |
| d | 0 | 3 | 10 | 16.5 | 12 | 0 | 4 | 10 | 17.5 | 12 | ✓ |
| e | 4 | 0 | 0 | 6.5 | 12 | 3.5 | 0 | 0 | 7.5 | 12.5 | ✓ |
| f | 8.5 | 5.5 | 0.5 | 0 | 6.5 | 8.5 | 5.5 | 0 | 0 | 7 | |
| g | 7.5 | 8 | 3 | 0 | 5.5 | 7.5 | 8 | 2.5 | 1 | 6 | |
| h | 4 | 7 | 6 | 5.5 | 0 | 4 | 8 | 5.5 | 5.5 | 0 | |
| i | 4 | 0 | 10 | 16.5 | 13 | 0 | 4 | 10 | 17.5 | 13 | ✓ |
| j | 14 | 13 | 0 | 6.5 | 2 | 14 | 13 | 0 | 7.5 | 2 | |
| | | | ✓ | ✓ | ✓ | | | ✓ | ✓ | ✓ | |

Step-4:Cover all zeros using the fewest number of lines feasible. Calculate the fewest number of lines required to cover all of the matrix's zeros. Proceed to step 4 because the number of lines required to cover all zeros is less than the size of the matrix (10).

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| a | 5.5 | 2.5 | 0.5 | 0 | 6.5 | 5.5 | 2.5 | 0 | 0 | 10 | |
| b | 4.5 | 4.5 | 2.5 | 0 | 5.5 | 4.5 | 4.5 | 2 | 1 | 6 | |
| c | 3 | 3 | 6 | 5.5 | 0 | 0 | 4 | 5.5 | 5.5 | 0 | |
| d | 0 | 3 | 13 | 19.5 | 15 | 0 | 4 | 13 | 20.5 | 15 | |
| e | 4 | 0 | 3 | 9.5 | 15 | 3.5 | 0 | 3 | 10.5 | 15.5 | |
| f | 5.5 | 2.5 | 0.5 | 0 | 6.5 | 5.5 | 2.5 | 0 | 0 | 7 | |
| g | 4.5 | 5 | 3 | 0 | 5.5 | 4.5 | 5 | 2.5 | 1 | 6 | |
| h | 1 | 4 | 6 | 5.5 | 0 | 1 | 5 | 5.5 | 5.5 | 0 | |
| i | 4 | 0 | 13 | 19.5 | 16 | 0 | 4 | 13 | 20.5 | 16 | |
| j | 11 | 10 | 0 | 6.5 | 2 | 11 | 10 | 0 | 7.5 | 2 | |
| | | | | | | | | | | | |

Step-3:With the fewest number of lines possible, cover all zeros. Calculate the least number of lines needed to cover all of the zeros in the matrix. Because the number of lines required to cover all zeros is fewer than the size of the matrix (10), go to step 4.

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| a | 8.5 | 5.5 | 0.5 | 0 | 6.5 | 8.5 | 5.5 | 0 | 0 | 7 | |
| b | 7.5 | 7.5 | 2.5 | 0 | 5.5 | 7.5 | 7.5 | 2 | 1 | 6 | |
| c | 3 | 6 | 6 | 5.5 | 0 | 3 | 7 | 5.5 | 5.5 | 0 | ✓ |
| d | 0 | 3 | 10 | 16.5 | 12 | 0 | 4 | 10 | 17.5 | 12 | ✓ |
| e | 4 | 0 | 0 | 6.5 | 12 | 3.5 | 0 | 0 | 7.5 | 12.5 | ✓ |
| f | 8.5 | 5.5 | 0.5 | 0 | 6.5 | 8.5 | 5.5 | 0 | 0 | 7 | |
| g | 7.5 | 8 | 3 | 0 | 5.5 | 7.5 | 8 | 2.5 | 1 | 6 | |
| h | 4 | 7 | 6 | 5.5 | 0 | 4 | 8 | 5.5 | 5.5 | 0 | ✓ |
| i | 4 | 0 | 10 | 16.5 | 13 | 0 | 4 | 10 | 17.5 | 13 | ✓ |
| j | 14 | 13 | 0 | 6.5 | 2 | 14 | 13 | 0 | 7.5 | 2 | |
| | | | ✓ | ✓ | | | | ✓ | ✓ | | |

Step-4: Increase the number of zeros. Select the smallest element among the cells not covered by any line (say k=2) to create the revised table. Subtract k=2 from every element not covered by a line in the cell.
To each element in the intersection cell of two lines, multiply by 2.

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| a | 3.5 | 0.5 | 0.5 | 0 | 4.5 | 3.5 | 0.5 | 0 | 0 | 5 | |
| b | 2.5 | 2.5 | 2.5 | 0 | 3.5 | 2.5 | 2.5 | 2 | 1 | 4 | |
| c | 3 | 3 | 8 | 7.5 | 0 | 0 | 4 | 7.5 | 7.5 | 0 | |
| d | 0 | 3 | 15 | 21.5 | 15 | 0 | 4 | 15 | 22.5 | 15 | |
| e | 4 | 0 | 5 | 11.5 | 15 | 3.5 | 0 | 5 | 12.5 | 15.5 | |
| f | 3.5 | 0.5 | 0.5 | 0 | 4.5 | 3.5 | 0.5 | 0 | 0 | 5 | |
| g | 2.5 | 3 | 3 | 0 | 3.5 | 2.5 | 3 | 2.5 | 1 | 4 | |
| h | 1 | 4 | 8 | 7.5 | 0 | 1 | 5 | 7.5 | 7.5 | 0 | |
| i | 4 | 0 | 15 | 21.5 | 16 | 0 | 4 | 15 | 22.5 | 16 | |
| j | 9 | 8 | 0 | 6.5 | 2 | 9 | 8 | 0 | 7.5 | 0 | |
| | | | | | | | | | | | |

Step-3: With the fewest number of lines possible, cover all zeros. Calculate the least number of lines needed to cover all of the zeros in the matrix. To cover all zeros, 9 lines are necessary, which is fewer than the size of the matrix (10), thus go to step-4.

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| a | 3.5 | 0.5 | 0.5 | 0 | 4.5 | 3.5 | 0.5 | 0 | 0 | 5 | |
| b | 2.5 | 2.5 | 2.5 | 0 | 3.5 | 2.5 | 2.5 | 2 | 1 | 4 | |
| c | ~~3~~ | ~~3~~ | ~~8~~ | ~~7.5~~ | ~~0~~ | ~~0~~ | ~~4~~ | ~~7.5~~ | ~~7.5~~ | ~~0~~ | ✓ |
| d | ~~0~~ | ~~3~~ | ~~15~~ | ~~21.5~~ | ~~15~~ | ~~0~~ | ~~4~~ | ~~15~~ | ~~22.5~~ | ~~15~~ | ✓ |
| e | ~~4~~ | ~~0~~ | ~~5~~ | ~~11.5~~ | ~~15~~ | ~~3.5~~ | ~~0~~ | ~~5~~ | ~~12.5~~ | ~~15.5~~ | ✓ |
| f | 3.5 | 0.5 | 0.5 | 0 | 4.5 | 3.5 | 0.5 | 0 | 0 | 5 | |
| g | 2.5 | 3 | 3 | 0 | 3.5 | 2.5 | 3 | 2.5 | 1 | 4 | |
| h | ~~1~~ | ~~4~~ | ~~8~~ | ~~7.5~~ | ~~0~~ | ~~1~~ | ~~5~~ | ~~7.5~~ | ~~7.5~~ | ~~0~~ | |
| i | ~~4~~ | ~~0~~ | ~~15~~ | ~~21.5~~ | ~~16~~ | ~~0~~ | ~~4~~ | ~~15~~ | ~~22.5~~ | ~~16~~ | |
| j | ~~9~~ | ~~8~~ | ~~0~~ | ~~6.5~~ | ~~0~~ | ~~9~~ | ~~8~~ | ~~0~~ | ~~7.5~~ | ~~0~~ | |
| | | | | ✓ | | | | ✓ | ✓ | | |

Step-4: Increase the number of zeros. Select the smallest element among the cells not covered by any line (say k=0.5) to create the revised table. Subtract k=0.5 from every element not covered by a line in the cell. In the intersection cell of two lines, multiply each element by k=0.5.

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| a | 3 | 0 | 0 | 0 | 4 | 3 | 0 | 0 | 0 | 4.5 | |
| b | 2 | 2 | 2 | 0 | 3 | 2 | 2 | 2 | 1 | 3.5 | |
| c | 0 | 3 | 8 | 8 | 0 | 0 | 4 | 8 | 8 | 0 | |
| d | 0 | 3 | 15 | 22 | 15 | 0 | 4 | 15.5 | 23 | 15 | |
| e | 4 | 0 | 5 | 12 | 15 | 3.5 | 0 | 5.5 | 13 | 15.5 | |
| f | 3 | 0 | 0 | 0 | 4 | 3 | 0 | 0 | 0 | 4.5 | |
| g | 2 | 2.5 | 2.5 | 0 | 3 | 2 | 2.5 | 2.5 | 1 | 3.5 | |
| h | 1 | 4 | 8 | 8 | 0 | 1 | 5 | 8 | 8 | 0 | |
| i | 4 | 0 | 15 | 22 | 16 | 0 | 4 | 15.5 | 23 | 16 | |
| j | 9 | 8 | 0 | 7 | 0 | 9 | 8 | 0.5 | 8 | 0 | |
| | | | | | | | | | | | |

Step-3: With the fewest number of lines possible, cover all zeros. Determine the smallest number of lines necessary to cover all of the zeros in the matrix.
To cover all zeros, 9 lines are necessary, which is fewer than the size of the matrix (10), thus go to step-4.

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| a | ~~3~~ | ~~0~~ | ~~0~~ | ~~0~~ | ~~4~~ | ~~3~~ | ~~0~~ | ~~0~~ | ~~0~~ | ~~4.5~~ | ✓ |
| b | 2 | 2 | 2 | 0 | 3 | 2 | 2 | 2 | 1 | 3.5 | |
| c | ~~0~~ | ~~3~~ | ~~8~~ | ~~8~~ | ~~0~~ | ~~0~~ | ~~4~~ | ~~8~~ | ~~8~~ | ~~0~~ | ✓ |
| d | ~~0~~ | ~~3~~ | ~~15~~ | ~~22~~ | ~~15~~ | ~~0~~ | ~~4~~ | ~~15.5~~ | ~~23~~ | ~~15~~ | ✓ |
| e | ~~4~~ | ~~0~~ | ~~5~~ | ~~12~~ | ~~15~~ | ~~3.5~~ | ~~0~~ | ~~5.5~~ | ~~13~~ | ~~15.5~~ | ✓ |
| f | ~~3~~ | ~~0~~ | ~~0~~ | ~~0~~ | ~~4~~ | ~~3~~ | ~~0~~ | ~~0~~ | ~~0~~ | ~~4.5~~ | ✓ |
| g | 2 | 2.5 | 2.5 | 0 | 3 | 2 | 2.5 | 2.5 | 1 | 3.5 | |
| h | ~~1~~ | ~~4~~ | ~~8~~ | ~~8~~ | ~~0~~ | ~~1~~ | ~~5~~ | ~~8~~ | ~~8~~ | ~~0~~ | ✓ |
| i | ~~4~~ | ~~0~~ | ~~15~~ | ~~22~~ | ~~16~~ | ~~0~~ | ~~4~~ | ~~15.5~~ | ~~23~~ | ~~16~~ | ✓ |
| j | ~~9~~ | ~~8~~ | ~~0~~ | ~~7~~ | ~~0~~ | ~~9~~ | ~~8~~ | ~~0.5~~ | ~~8~~ | ~~0~~ | ✓ |
| | | | | ✓ | | | | | | | |

Step-4: Increase the number of zeros. Select the smallest element among the cells not covered by any line (say k=1) to create the revised table. Subtract k=1 from every element not covered by a line in the cell.
Every element in the intersection cell of two lines should have k= 1.

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| a | 3 | 0 | 0 | 1 | 4 | 3 | 0 | 0 | 0 | 4.5 | |
| b | 1 | 1 | 1 | 0 | 2 | 1 | 1 | 1 | 0 | 2.5 | |
| c | 0 | 3 | 8 | 9 | 0 | 0 | 4 | 8 | 8 | 0 | |
| d | 0 | 3 | 15 | 23 | 15 | 0 | 4 | 15.5 | 23 | 15 | |
| e | 4 | 0 | 5 | 13 | 15 | 3.5 | 0 | 5.5 | 13 | 15.5 | |
| f | 3 | 0 | 0 | 1 | 4 | 3 | 0 | 0 | 0 | 4.5 | |
| g | 1 | 1.5 | 1.5 | 0 | 2 | 1 | 1.5 | 1.5 | 1 | 2.5 | |
| h | 1 | 4 | 8 | 9 | 0 | 1 | 5 | 8 | 8 | 0 | |
| i | 4 | 0 | 15 | 23 | 16 | 0 | 4 | 15.5 | 23 | 16 | |
| j | 9 | 8 | 0 | 8 | 0 | 9 | 8 | 0.5 | 8 | 0 | |
| | | | | | | | | | | | |

Step-3:With the fewest number of lines possible, cover all zeros. Determine the smallest number of lines necessary to cover all of the zeros in the matrix.
To cover all zeros, 9 lines are necessary, which is fewer than the size of the matrix (10), thus go to step-4.

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| a | 3 | 0 | 0 | 1 | 4 | 3 | 0 | 0 | 0 | 4.5 | |
| b | 1 | 1 | 1 | 0 | 2 | 1 | 1 | 1 | 0 | 2.5 | |
| c | 0 | 3 | 8 | 9 | 0 | 0 | 4 | 8 | 8 | 0 | |
| d | 0 | 3 | 15 | 23 | 15 | 0 | 4 | 15.5 | 23 | 15 | |
| e | 4 | 0 | 5 | 13 | 15 | 3.5 | 0 | 5.5 | 13 | 15.5 | |
| f | 3 | 0 | 0 | 1 | 4 | 3 | 0 | 0 | 0 | 4.5 | |
| g | 1 | 1.5 | 1.5 | 0 | 2 | 1 | 1.5 | 1.5 | 1 | 2.5 | |
| h | 1 | 4 | 8 | 9 | 0 | 1 | 5 | 8 | 8 | 0 | |
| i | 4 | 0 | 15 | 23 | 16 | 0 | 4 | 15.5 | 23 | 16 | |
| j | 9 | 8 | 0 | 8 | 0 | 9 | 8 | 0.5 | 8 | 0 | |
| | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | |

The Optimal assignments are:

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **a** | 3 | 0 | 0 | 1 | 4 | 3 | 0 | **[0]** | 0 | 4.5 | |
| **b** | 1 | 1 | 1 | 0 | 2 | 1 | 1 | 1 | **[0]** | 2.5 | |
| **c** | 0 | 3 | 8 | 9 | **[0]** | 0 | 4 | 8 | 8 | 0 | |
| **d** | **[0]** | 3 | 15 | 23 | 15 | 0 | 4 | 15.5 | 23 | 15 | |
| **e** | 4 | 0 | 5 | 13 | 15 | 3.5 | **[0]** | 5.5 | 13 | 15.5 | |
| **f** | 3 | **[0]** | 0 | 1 | 4 | 3 | 0 | 0 | 0 | 4.5 | |
| **g** | 1 | 1.5 | 1.5 | **[0]** | 2 | 1 | 1.5 | 1.5 | 1 | 2.5 | |
| **h** | 1 | 4 | 8 | 9 | 0 | 1 | 5 | 8 | 8 | **[0]** | |
| **i** | 4 | 0 | 15 | 23 | 16 | **[0]** | 4 | 15.5 | 23 | 16 | |
| **j** | 9 | 8 | **[0]** | 8 | 0 | 9 | 8 | 0.5 | 8 | 0 | |
| | | | | | | | | | | | |

Optimal solution is:

| Work | Job | Cost |
|---|---|---|
| **a** | 8 | 9 |
| **b** | 9 | 6 |
| **c** | 5 | 5.5 |
| **d** | 1 | 4.5 |
| **e** | 7 | 9 |
| **f** | 2 | 15.5 |
| **g** | 4 | 5.5 |
| **h** | 10 | 5 |
| **i** | 6 | 4.5 |
| **j** | 3 | 7.5 |
| | Total | 72 |

### 4.2.1 Python Pseudo code of Hungarian Algorithm Given Below:

```
DEFINE CLASS Vertex:
    DEFINE FUNCTION __init__(self, key):
        SET self.key TO key
        SET self.label TO None
        SET self.neighbors TO set()
        SET self.indecent_edges TO set()
        SET self.in_left TO None
    DEFINE FUNCTION get_edge(self, neighbor):
        FOR e IN self.indecent_edges:
            IF neighbor IN e.vertices:
                RETURN e
        RETURN False
    DEFINE FUNCTION set_label(self, label):
        SET self.label TO label
    DEFINE FUNCTION set_in_left(self, in_left):
        SET self.in_left TO in_left
    DEFINE FUNCTION filter_neighbors(self):
        SET new_neighbors TO set()
        FOR v IN self.neighbors:
            FOR e IN self.indecent_edges:
                IF v EQUALS e.vertices[0] or v EQUALS e.vertices[1]:
                    new_neighbors.add(v)
                    break
        SET self.neighbors TO new_neighbors
DEFINE CLASS Edge:

    DEFINE FUNCTION __init__(self, v1, v2, weight=0):

        SET self.vertices TO [v1, v2]

        SET self.weight TO weight
    DEFINE FUNCTION __eq__(self, e):

        RETURN (self.vertices EQUALS e.vertices

                and self.weight EQUALS e.weight)

    DEFINE FUNCTION __hash__(self):

        RETURN hash((frozenset(self.vertices), self.weight))
DEFINE CLASS Graph:
    DEFINE FUNCTION __init__(self, g=None, negate=False):
        IF g is None:
            SET g TO {}
        SET self.vertices TO {}
        FOR v1 IN g:
            FOR v2 IN g[v1]:
                IF type(g[v1]) is dict:
                    self.add_edge(v1, v2, g[v1][v2], negate)
```

```
        FOR v1 IN g:
            FOR v2 IN g[v1]:
                IF type(g[v1]) is dict:
                    self.add_edge(v1, v2, g[v1][v2], negate)
                ELSE:
                    self.add_edge(v1, v2)
    DEFINE FUNCTION add_vertex(self, key):
        SET self.vertices[key] TO Vertex(key)
    DEFINE FUNCTION add_edge(self, v1, v2, weight=1, negate=False):
        IF v1 not IN self.vertices:
            self.add_vertex(v1)
        IF v2 not IN self.vertices:
            self.add_vertex(v2)
        IF negate:
            SET e TO Edge(v1, v2, -weight)
        ELSE:
            SET e TO Edge(v1, v2, weight)
        self.vertices[v1].neighbors.add(v2)
        self.vertices[v2].neighbors.add(v1)
        self.vertices[v1].indecent_edges.add(e)
        self.vertices[v2].indecent_edges.add(e)
    DEFINE FUNCTION is_bipartite(self, start_vertex):
        IF start_vertex is None:
            RETURN True
        self.clear_labeling()
        self.vertices[start_vertex].set_label(1)
        SET queue TO [start_vertex]
        WHILE queue:
            SET v TO queue.pop()
            FOR w IN self.vertices[v].neighbors:
                IF self.vertices[w].label is None:
                    self.vertices[w].set_label(1 - self.vertices[v].label)
                    queue.append(w)
                ELSEIF self.vertices[w].label EQUALS self.vertices[v].label:
                    RETURN False
        RETURN True
    DEFINE FUNCTION make_complete_bipartite(self, start_vertex):
        IF start_vertex is None:
            RETURN True
        self.clear_labeling()
        self.generate_feasible_labeling(start_vertex)
        FOR x IN self.vertices:
            IF self.vertices[x].in_left:
                FOR y IN self.vertices:
                    IF (not self.vertices[y].in_left
                            and y not IN self.vertices[x].neighbors):
                        self.add_edge(x, y, 0)
        self.clear_labeling()
    DEFINE FUNCTION feasibly_label(self, v):
```

```
    DEFINE FUNCTION feasibly_label(self, v):
        SET _max TO None
        FOR e IN self.vertices[v].indecent_edges:
            IF _max is None or e.weight > _max:
                SET _max TO e.weight
        self.vertices[v].set_label(_max)
        self.vertices[v].set_in_left(True)
    DEFINE FUNCTION generate_feasible_labeling(self, start_vertex):
        IF start_vertex is None:
            RETURN True
        self.feasibly_label(start_vertex)
        SET queue TO [start_vertex]
        WHILE queue:
            SET v TO queue.pop()
            FOR w IN self.vertices[v].neighbors:
                IF self.vertices[w].label is None:
                   IF self.vertices[v].label EQUALS 0:
                        self.feasibly_label(w)
                    ELSE:
                        self.vertices[w].set_label(0)
                        self.vertices[w].set_in_left(False)
                    queue.append(w)
                ELSEIF ((self.vertices[w].label EQUALS 0 and self.vertices[v].label EQUALS 0) or
                      (self.vertices[w].label != 0 and self.vertices[v].label != 0)):
                    RETURN False
        RETURN True
    DEFINE FUNCTION clear_label(self, v):
        self.vertices[v].set_label(None)
    DEFINE FUNCTION clear_labeling(self):
        FOR v IN self.vertices:
            self.vertices[v].set_label(None)
    DEFINE FUNCTION edge_in_equality_subgraph(self, e):
        SET e_endpoints TO list(e.vertices)
        IF (self.vertices[e_endpoints[0]].label is None or
                self.vertices[e_endpoints[1]].label is None):
            RETURN False
        RETURN e.weight EQUALS (self.vertices[e_endpoints[0]].label +

                            self.vertices[e_endpoints[1]].label)
    DEFINE FUNCTION equality_subgraph(self):
        SET eq_H TO copy.deepcopy(self)
        FOR v IN eq_H.vertices:
            SET eq_H.vertices[v].indecent_edges TO list(filter(
                self.edge_in_equality_subgraph,
                eq_H.vertices[v].indecent_edges))
            eq_H.vertices[v].filter_neighbors()
        RETURN eq_H
DEFINE FUNCTION generate_feasible_labeling(_g, start_vertex):
    IF start_vertex is None:
        RETURN True
```

```
    IF start_vertex is None:
        RETURN True
    _g.feasibly_label(start_vertex)
    SET queue TO [start_vertex]
    WHILE queue:
        SET v TO queue.pop()
        FOR w IN _g.vertices[v].neighbors:
            IF _g.vertices[w].label is None:
                IF _g.vertices[v].label EQUALS 0:
                    _g.feasibly_label(w)
                ELSE:
                    _g.vertices[w].set_label(0)
                queue.append(w)
            ELSEIF _g.vertices[w].label EQUALS _g.vertices[v].label:
                RETURN False
    RETURN True
DEFINE FUNCTION vertex_saturated(v, _m):
    FOR e IN _m:
        IF v EQUALS e.vertices[0]:
           RETURN e.vertices[1]
        ELSEIF v EQUALS e.vertices[1]:
            RETURN e.vertices[0]
    RETURN False
DEFINE FUNCTION find_matching(_g, matching_type='max', RETURN_type='list'):
    SET negate TO False IF matching_type EQUALS 'max' else True
    SET G TO Graph(_g, negate)
    SET start_vertex TO list(G.vertices.keys())[0]
    G.make_complete_bipartite(start_vertex)
    SET is_bipartite TO G.generate_feasible_labeling(start_vertex)
    IF not is_bipartite:
        RETURN False
    SET eq_G TO G.equality_subgraph()
    SET M TO set()
    FOR x IN eq_G.vertices:
        IF eq_G.vertices[x].in_left and not vertex_saturated(x, M):
            SET max_edge TO None
            FOR y IN eq_G.vertices[x].neighbors:
                IF not vertex_saturated(y, M):
                   IF max_edge is None or eq_G.vertices[x].get_edge(y).weight > max_edge.weight:
                        SET max_edge TO eq_G.vertices[x].get_edge(y)
            IF max_edge is not None:
                M.add(max_edge)
    SET S TO set()
    SET T TO set()
    SET path_end TO None
    WHILE len(M) < int(len(eq_G.vertices) / 2):
        IF path_end is None:
            FOR x IN eq_G.vertices:
               IF eq_G.vertices[x].in_left and not vertex_saturated(x, M):
                    S.add(x)
```

```
        IF eq_G.vertices[x].in_left and not vertex_saturated(x, M):
            S.add(x)
            SET path_end TO x
            break
SET S_nbs TO set()
FOR v IN S:
    SET S_nbs TO S_nbs | eq_G.vertices[v].neighbors
IF S_nbs EQUALS T:
    SET alpha TO None
    FOR x IN S:
        FOR y IN G.vertices.keys() - T:
            IF not G.vertices[y].in_left and y IN G.vertices[x].neighbors:
                SET new_alpha TO G.vertices[x].label + G.vertices[y].label - G.vertices[x].get_edge(y).weight
                SET alpha TO new_alpha IF alpha is None or new_alpha < alpha else alpha
    IF alpha is not None:
        FOR u IN S:
            SET G.vertices[u].label TO G.vertices[u].label - alpha
        FOR v IN T:
            SET G.vertices[v].label TO G.vertices[v].label + alpha
        SET eq_G TO G.equality_subgraph()
SET S_nbs TO set()
FOR v IN S:
    SET S_nbs TO S_nbs | eq_G.vertices[v].neighbors
IF S_nbs != T:
    SET y TO list(S_nbs - T)[0]
    SET z TO vertex_saturated(y, M)
    IF not z:
        T.add(y)
        SET y_path_last TO y
        SET y_path_curr TO y
        SET matched_nbs TO True
        WHILE matched_nbs:
            SET matched_nbs TO False
            FOR x IN S & eq_G.vertices[y_path_curr].neighbors:
                SET y_matched_nb TO vertex_saturated(x, M)
                IF y_matched_nb and y_matched_nb != y_path_last:
                    SET matched_nbs TO True
                    M.add(eq_G.vertices[y_path_curr].get_edge(x))
                    M.remove(eq_G.vertices[x].get_edge(y_matched_nb))
                    SET y_path_last TO y_path_curr
                    SET y_path_curr TO y_matched_nb
                    break
            IF not matched_nbs:
                M.add(eq_G.vertices[y_path_curr].get_edge(path_end))
        SET S TO set()
        SET T TO set()
        SET path_end TO None
    ELSE:
        S.add(z)
```

```
                    IF not matched_nbs:
                        M.add(eq_G.vertices[y_path_curr].get_edge(path_end))
                SET S TO set()
                SET T TO set()
                SET path_end TO None
            ELSE:
                S.add(z)
                T.add(y)
    SET edge_multiple TO -1 IF matching_type EQUALS 'min' else 1
    IF RETURN_type EQUALS 'list':
        RETURN list(map(lambda ele: ((ele.vertices[0], ele.vertices[1]), edge_multiple * ele.weight), M))
    ELSEIF RETURN_type EQUALS 'total':
        SET total TO 0
        FOR e IN M:
            SET total TO total + (edge_multiple * e.weight)
        RETURN total
# Calling the algorithm
SET _G TO {
    'Ann': {'RB': 3, 'CAM': 2, 'GK': 1},
    'Ben': {'LW': 3, 'S': 2, 'CM': 1},
    'Cal': {'CAM': 3, 'RW': 2, 'SWP': 1},
    'Dan': {'S': 3, 'LW': 2, 'GK': 1},
    'Ela': {'GK': 3, 'LW': 2, 'F': 1},
    'Fae': {'CM': 3, 'GK': 2, 'CAM': 1},
    'Gio': {'GK': 3, 'CM': 2, 'S': 1},
    'Hol': {'CAM': 3, 'F': 2, 'SWP': 1},
    'Ian': {'S': 3, 'RW': 2, 'RB': 1},
    'Jon': {'F': 3, 'LW': 2, 'CB': 1},
    'Kay': {'GK': 3, 'RW': 2, 'LW': 1, 'LB': 0}
}
OUTPUT(find_matching(_G, matching_type='max', RETURN_type='list'))  # 1
OUTPUT(find_matching(_G, matching_type='max', RETURN_type='total'))  # 2
SET _H TO {
    'A': {'#191': 22, '#122': 14, '#173': 120, '#121': 21, '#128': 4, '#104': 51},
    'B': {'#191': 19, '#122': 12, '#173': 172, '#121': 21, '#128': 28, '#104': 43},
    'C': {'#191': 161, '#122': 122, '#173': 2, '#121': 50, '#128': 128, '#104': 39},
    'D': {'#191': 19, '#122': 22, '#173': 90, '#121': 11, '#128': 28, '#104': 4},
    'E': {'#191': 1, '#122': 30, '#173': 113, '#121': 14, '#128': 28, '#104': 86},
    'F': {'#191': 60, '#122': 70, '#173': 170, '#121': 28, '#128': 68, '#104': 104},
}
OUTPUT(find_matching(_H, matching_type='min', RETURN_type='list'))  # 3
```

# 5 Chapter 5

## 5.1 Case3 : Air India-Transporting numerous items in a one plane

The problem consists of a transportation network, as well as a collection of diverse commodities, such as cargo goods, passenger classes, and so on, all of which are distinguished by their physical characteristics. When different types of goods are transmitted across the same network,which will benefit the airline business by lowering costs and increasing revenues, as well as allowing them to transport multiple items at once, shortening travel time. This process will take longer if they opt to deliver cargo in one plane and passengers in different flights.
Let us define the model in following manner. The first is the Indra Gandhi International Airport in Delhi, and the second is the Chhatrapati Shivaji Maharaj International Airport in Mumbai. From Delhi to Mumbai, we must transport both cargo goods and passengers.
There are many destinations between Delhi and Mumbai, thus we must choose the quickest route between them so that cargo and people are transported in less time, which is beneficial to the airline sector.Let's call Indira Gandhi International Airport 1 and Chatterpati Shivaji Maharaj International Airport 9 , respectively. and the different notations used to express different locations between them are as follows:
2 = Jaipur
3 = Kota
4 = Ahemdabad
5 = Indore
6 = Vadodara
7 = Surat
8 = Nasik
Below is the route map with all of the distances between each place for the above problem.
The main aim is to find the shortest route between Airport 1 and Airport 9 such that the cost is minimized.
we have assumed distance in hundreds.

## 5.2 Solution Approach

What if you were given a graph of nodes in which each node is connected to a number of other nodes at varying distances? What is the shortest path to every other node in the network if you start from one of the nodes in the graph? Dijkstra's Algorithm is a simple algorithm that is used to find the shortest distance, or path, between a beginning node to a target node in a weighted network. This technique is being used to find the quickest way between two airports.
Dijkstra's algorithm is an iterative algorithm that finds the shortest path from a single beginning node to all other nodes in a graph. This algorithm is used to solve the model, and the final result is shown below.

**analysis matrix**

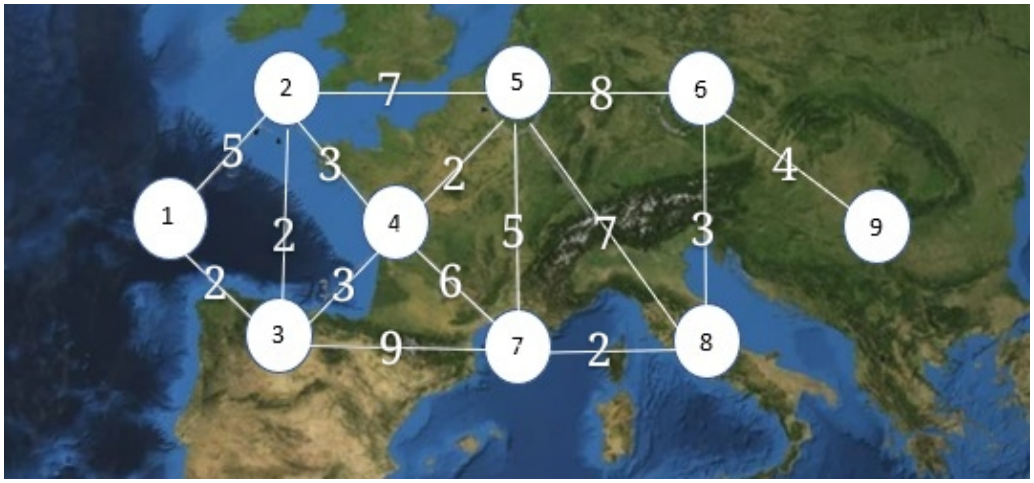



Figure 13: Air Map

| | A | B | C | D | E | F | G | H | I |
|---|---|---|---|---|---|---|---|---|---|
| **A** | 0 | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ |
| **B** | 0 | 5 | 2 | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ |
| **C** | 0 | 4 | 2 | ∞ | ∞ | ∞ | 11 | ∞ | ∞ |
| **D** | 0 | 4 | 2 | 7 | 11 | ∞ | 11 | ∞ | ∞ |
| **E** | 0 | 4 | 2 | 7 | 9 | ∞ | 11 | ∞ | ∞ |
| **F** | 0 | 4 | 2 | 7 | 9 | 17 | 11 | 16 | ∞ |
| **G** | 0 | 4 | 2 | 7 | 9 | 17 | 11 | 13 | ∞ |
| **H** | 0 | 4 | 2 | 7 | 9 | 16 | 11 | 13 | ∞ |
| **I** | 0 | 4 | 2 | 7 | 9 | 16 | 11 | 13 | ∞ |

**The Shortest Path Is From :** $Delhi \rightarrow Kota \rightarrow Surat \rightarrow Nasik \rightarrow Vadodara \rightarrow Mumbai$

### 5.2.1 Python Pseudo code of Dijkstra Algorithm Given Below:

```
dijkstra algorithm

function stra(matrix,unvsett,shortdistance,node,source):
    s = node
    print(node)
    unvsett.remove(s)//remove current node from unvisited set
    print(unvsett)//print unvisited set

for i in unvsett:// running the loop in unvisited set
if  matrix[s][i] != 0:// checking if s and i are connected in the graph
if  shortdistance[s]+matrix[s][i] < matrix[source][i] ://if distance from the source(edge length) is greater than distance
(through path from s(current node)
shortdistance[i] = shortdistance[s]+matrix[s][i];//update the shortdistance array with the newly found short distance
elif shortdistance[s]+matrix[s][i] > matrix[source][i] & matrix[source][i] == 0://if the node i is not connected to the source node
                shortdistance[i] = shortdistance[s]+matrix[s][i];//updating shortest distance with only possible path distance
    print(shortdistance)
    if len(unvsett) > 0://checking  if unvisited set is empty
        k = len(unvsett)

        l = 0;
        for i in range(k):// finding the node with least possible shortdistance from the source
            if shortdistance[unvsett[l]] > shortdistance[unvsett[i]] & shortdistance[unvsett[i]] > 0:
                l = i;
        node = unvsett[l];//the node with least distance from source

stra(matrix,unvsett,shortdistance,node,source);//recalling the function with new updated unvisited set and new current node
if unvisited set is not empty
    else:
        print(shortdistance);//printing the shortdistance array if all the nodes are visited

Main function
m = [[0, 4, 0, 0, 0, 0, 0, 8, 0], //// matrix
        [4, 0, 8, 0, 0, 0, 0, 11, 0],
        [0, 8, 0, 7, 0, 4, 0, 0, 2],
        [0, 0, 7, 0, 9, 14, 0, 0, 0],
        [0, 0, 0, 9, 0, 10, 0, 0, 0],
        [0, 0, 4, 14, 10, 0, 2, 0, 0],
        [0, 0, 0, 0, 0, 2, 0, 1, 6],
        [8, 11, 0, 0, 0, 0, 1, 0, 7],
        [0, 0, 2, 0, 0, 0, 6, 7, 0]
        ];
u = [0,1,2,3,4,5,6,7,8];//nodes
s = 0;//source node
sd = m[s];//edge set of source node
stra(m,u,sd,0,0)//function
```

# 6 Chapter 6

## 6.1 Case4: Aircraft Maintenance Problem

As their levels of operational performance, complexity, and expense continue to rise, modern aircraft turbine engines need new and increasingly severe management, operations, and maintenance difficulties. Repair and overhaul of aircraft gas turbine engines Repairing and overhauling gas turbine engines, rebuilding gas turbine engines, balancing components and assemblies, testing and troubleshooting gas turbine engines, and inspecting gas turbine engine components and assemblies are all jobs for technicians. As a consequence, depending on business standards, the scope of the analysis may be separated into multi-level abilities,all technicians are employed needed to do only a subset of the duties stated in the analysis.
This aviation engine specialist's responsibilities include troubleshooting and inspecting gas turbine engine components and assemblies. This can entail dismantling and rebuilding engines, as well as balancing components in these complex, contemporary engines. Engine overhauls are carried out in specialised shops where engines are dismantled, issues are detected, and engine components are cleaned, repaired, or rebuilt. The engine is then reassembled and tested.
The tolerances and performance standards for aviation gas turbine engines are extremely tight, necessitating the use of highly qualified experts with excellent mechanical skills. Workers in this position must be able to decipher technical texts and drawings. Materials management data bases have been built in engine repair facilities, and keyboarding abilities are becoming increasingly crucial.
Any maintenance job entails a variety of tasks.In this case,A model of aircraft maintenance is built, which aids in the prioritization of the most important aircraft maintenance activity. Here, actions are represented by nodes in a network. The activities and their durations are presented inside the node when showing activities on an Activity-On-Node or CPM network.The network "critical route," which consists of the sequence of project operations that determines the lowest necessary project time, is computed using the CPM approach.
Consequently.
All of the duration are in the project outlined in Table 1

tableThe activity time in the aircraft gas turbine engine repair/overhaul operations network

| No. | Code | Activity Description | Precedence Activity | Activity time |
|---|---|---|---|---|
| 1 | A | Check stand by pump | - | Around 32 man-hours |
| 2 | B | Calibrate all ganges | A | Around 80 man-hours |
| 3 | C | Dismantle pump cover and remove rotor | A | Around 8 man-hours |
| 4 | D | Dismantle Turbine cover and remove rotor | A | Around 16 man-hours |
| 5 | E | Clean all ganges and line | B | Around 32 man-hours |
| 6 | F | Replace ganges | E | Around 16 man-hours |
| 7 | G | Repair lubrication system | C | Around 28 and 40 man-hours |
| 8 | H | Rebuild impeller | C | Around 256 and 352 man-hours |
| 9 | I | Clear pump casting | C | Around 32 man-hours |
| 10 | J | Fix pump bearings | G | Around 16 man-hours |
| 11 | K | balance impeller | H | Around 32 man-hours |
| 12 | L | Reinstall impeller | I,J,K | Around 16 man-hours |
| 13 | M | Rebuild turbine rotor | D | Around 260 and 400 man-hours |
| 14 | N | Check turbine bearings | D | Around 16 man-hours |
| 15 | O | Balance turbine rotor | M | Around 64 man-hours |
| 16 | P | Fix turbine bearings | N | Around 16 man-hours |
| 17 | Q | Fix turbine rotor | O,P | Around 16 and 28 man-hours |
| 18 | R | Fix turbine cover | Q | Around 20 and 32 man-hours |
| 19 | S | Text components | R | Around 12 and 18 man-hours |
| 20 | T | Check clearance | R | Around 16 man-hours |
| 21 | U | Fix pump bearings | L | Around 16 man-hours |
| 22 | V | Fix pump cover | U | Around 18 and 24 man-hours |
| 23 | W | Install shaft packing | V | Around 16 man-hours |
| 24 | X | Final test | S,T,W,F | Around 48 and 72 man-hours |

**Critical path,Total float,Free float,Independent float**

| | | |
|---|---|---|
| A | - | 32 |
| B | A | 80 |
| C | A | 8 |
| D | A | 16 |
| E | B | 32 |
| F | E | 15 |
| G | C | 40 |
| H | C | 32 |
| I | C | 90 |
| J | G | 70 |
| K | H | 40 |
| L | I,J,K | 70 |
| M | D | 30 |
| N | D | 60 |
| O | M | 50 |
| P | N | 30 |
| Q | O,P | 50 |
| R | Q | 40 |
| S | R | 60 |
| T | R | 50 |
| U | L | 40 |
| V | U | 50 |
| W | V | 10 |
| X | S,T,W,F | 60 |

**Solution:**

| Activity | Immediate predecessors | Duration |
|---|---|---|
| A | - | 32 |
| B | A | 80 |
| C | A | 8 |
| D | A | 16 |
| E | B | 32 |
| F | E | 15 |
| G | C | 40 |
| H | C | 32 |
| I | C | 90 |
| J | G | 70 |
| K | H | 40 |
| L | I,J,K | 70 |
| M | D | 30 |
| N | D | 60 |
| O | M | 50 |
| P | N | 30 |
| Q | O,P | 50 |
| R | Q | 40 |
| S | R | 60 |
| T | R | 50 |
| U | L | 40 |
| V | U | 50 |
| W | V | 10 |
| X | S,T,W,F | 60 |

**Edge and its preceded,succeed node:**

| **Edge** | **Node1 $\rightarrow$ Node2** |
|---|---|
| A | $1 \rightarrow 2$ |
| B | $2 \rightarrow 3$ |
| C | $2 \rightarrow 4$ |
| D | $2 \rightarrow 5$ |
| E | $3 \rightarrow 6$ |
| G | $4 \rightarrow 8$ |
| H | $4 \rightarrow 9$ |
| I | $4 \rightarrow 10$ |
| M | $5 \rightarrow 12$ |
| N | $5 \rightarrow 13$ |
| F | $6 \rightarrow 7$ |
| X | $7 \rightarrow 20$ |
| J | $8 \rightarrow 10$ |
| K | $9 \rightarrow 10$ |
| L | $10 \rightarrow 11$ |
| U | $11 \rightarrow 18$ |
| O | $12 \rightarrow 14$ |
| P | $13 \rightarrow 14$ |
| Q | $14 \rightarrow 15$ |
| R | $15 \rightarrow 16$ |
| T | $16 \rightarrow 7$ |
| S | $16 \rightarrow 17$ |
| d | $17 \rightarrow 7$ |
| V | $18 \rightarrow 19$ |
| W | $19 \rightarrow 7$ |

### 6.1.1 The network diagram for the project, along with activity time, is

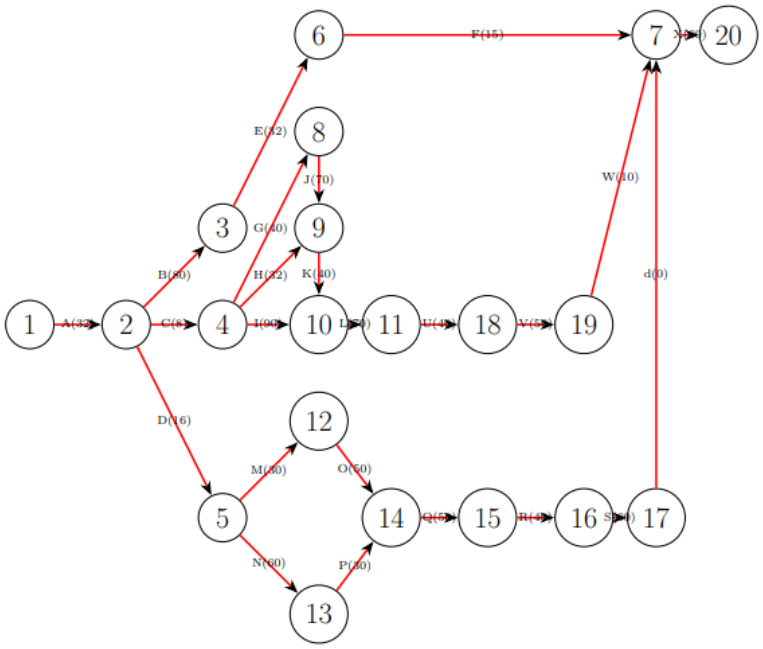



Figure 14: Network Diagram

**Forward pass method** $E_1 = 0$

$E_2 = E_1 + t_{1,2}[t_{1,2} = A = 32] = 0 + 32 = 32$

$E_3 = E_2 + t_{2,3}[t_{2,3} = B = 80] = 32 + 80 = 112$

$E_4 = E_2 + t_{2,4}[t_{2,4} = C = 8] = 32 + 8 = 40$

$E_5 = E_2 + t_{2,5}[t_{2,5} = D = 16] = 32 + 16 = 48$

$E_6 = E_3 + t_{3,6}[t_{3,6} = E = 32] = 112 + 32 = 144$

$E_7 = Max\ E_i + t_{i,7}\ [i = 616, 17, 19]$

= Max $\{E_6 + t_{6,7}; E_16 + t_{16,7}; E_17 + t_{17,7}; E - 19 + t_{19,7}\}$

= Max {144+15;228+50;288+0;310+10}

= Max{159;278;288;320}

= 320

$E_8 = E_4 + t_{4,8}[t_{4,8} = G = 40] = 40 + 40 = 80$

$E_9 = E_4 + t_{4,9}[t_{4,9} = H = 32] = 40 + 32 = 72$

$E_{10} = Max\{E_i + t_{i,10}\}[i = 48, 9]$

= Max$\{E_4 + t_{4,10}; E_8 + t_8; E_9 + t_{9,10}\}$

= Max{40+90;80+70;72+40}

= Max{130;150;112}

= 150

$E_{11} = E_{10} + t_{10,11}[t_{10,11} = L = 70] = 150 + 70 = 220$

$E_{12} = E_5 + t_{5,12}[t_{5,12} = M = 30] = 48 + 30 = 78$

$E_{13} = E_5 + t_{5,13}[t_{5,13} = N = 60] = 48 + 60 = 108$

$E_{14} = Max\{E_i + t_{i,14}\}[i = 1213]$

= Max$\{E_{12} + t_{12,14}; E_{13} + t_{13,14}\}$

= Max{78+50;108+30}

= 138

$E_{15} = E_{14} + t_{14,15}[t_{14,15} = Q = 50] = 138 + 50 = 188$
$E_{16} = E_{15} + t_{15,16}[t_{15,16} = R = 40] = 188 + 40 = 228$
$E_{17} = E_{16} + t_{16,17}[t_{16,17} = S = 60] = 228 + 60 = 288$
$E_{18} = E_{11} + t_{11,18}[t_{11,18} = U = 40] = 220 + 40 = 260$
$E_{19} = E_{18} + t_{18,19}[t_{18,19} = V = 50] = 260 + 50 = 310$
$E_{20} = E_7 + t_{7,20}[t_{7,20} = X = 60] = 320 + 60 = 380$

**Backward pass method**

$L_{20} = E_{20} = 380$
$L_{19} = L_7 - t_{19,7}[t_{19,7} = W = 10] = 320 - 10 = 310$
$L_{18} = L_{19} - t_{17,7}[t_{17,7} = V = 0] = 320 - 0 = 260$
$L_{17} = L_7 - t_{17,7}[t_{17,7} = d = 0] = 320 - 0 = 320$
$L_{16} = Min\{L_j - t_{16,j}\}[j = 177]$
= Min$\{L_{17} - t_{16,17}; L_7 - t_{16,7}\}$
= Min{320-60;320-50}
= Min{260,270}
= 260
$L_{15} = L_{16} - t_{15,16}[t_{15,16} = R = 40] = 260 - 40 = 220$
$L_{14} = L_{15} - t_{14,15}[t_{14,15} = Q = 40] = 220 - 50 = 170$
$L_{13} = L_{14} - t_{13,14}[t_{13,14} = P = 40] = 170 - 30 = 140$
$L_{12} = L_{14} - t_{12,14}[t_{12,14} = O = 40] = 170 - 50 = 120$
$L_{11} = L_{18} - t_{11,18}[t_{11,18} = U = 40] = 260 - 40 = 220$
$L_{10} = L_{11} - t_{10,11}[t_{10,11} = L = 40] = 220 - 70 = 150$
$L_9 = L_{10} - t_{9,10}[t_{9,10} = K = 40] = 150 - 40 = 110$
$L_8 = L_{10} - t_{8,10}[t_{8,10} = J = 40] = 150 - 70 = 80$
$L_7 = L_{20} - t_{7,20}[t_{7,20} = X = 40] = 380 - 60 = 320$
$L_6 = L_7 - t_{6,7}[t_{6,7} = F = 40] = 320 - 15 = 305$
$L_5 = Min\{L_j - t_{5,j}\}[j = 1312]$
= Min$\{L_{13} - t_5; L_{12} - t_{5,12}\}$
= Min{140-60;120-30}
= Min{80,90}
= 80
$L_4 = Min\{L_j - t_{4,j}\}[j = 109,8]$
= Min$\{L_{10} - t_{4,10}; L_9 - t_{4,9}; L_8 - t_{4,8}\}$
= Min{150-90;110-32;80-40}
= Min{60;78;40}
= 40
$L_3 = L_6 - t_{3,6}[t_{3,6} = E = 32] = 305 - 32 = 273$
$L_2 = Min\{L_5 - t_{2,j}\}[j = 54,3]$
= Min{80-16;40-8;273-80}
= Min{64;32;193}
= 32
$L_1 = L_2 - t_{1,2}[t_{1,2} = A = 32] = 32 - 32 = 0$

**The Critical Path is 1-2-4-8-10-11-18-19-7-20 and Critical activities are A,C,G,J,L,U,V,W,X**
**The total time is 380**

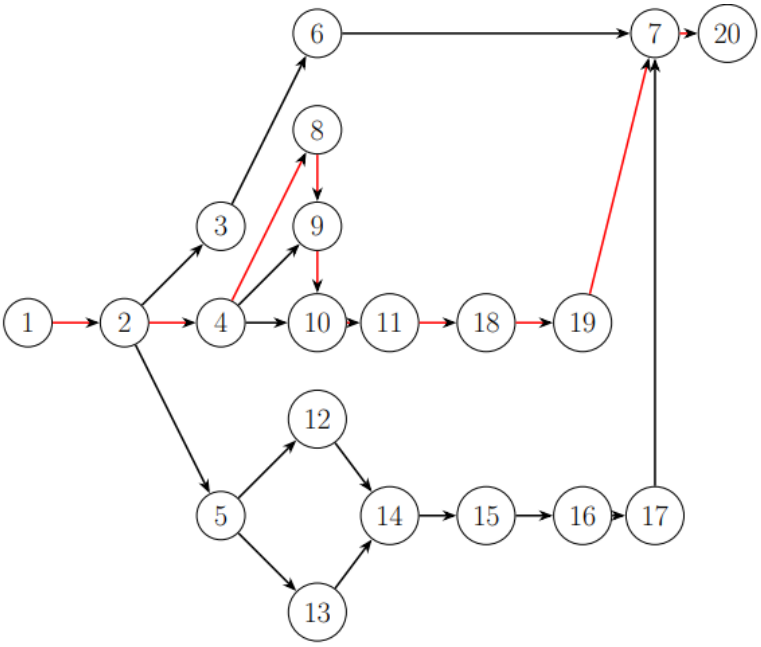



Figure 15: Network Diagram

tableFor each non-critical activity, the total float,free float and independent float calculations are shown

| **Activity** | **Duration** | **Earliest time start** | | | **Latest time finish** | **Earliest time finish** | **latest time start** | **total float** | **Free float** | **Independent Float** |
|---|---|---|---|---|---|---|---|---|---|---|
| $(i,j)$ | $(t_{ij})$ | $(E_i)$ | $(E_j)$ | $(L_i)$ | $(L_j)$ | $(E_i+t_{ij})$ | $(L_j-t_{ij})$ | $(L_j-t_{ij})-E_i$ | $(E_j-E_i)-t_{ij}$ | $(E_j-L_i)-t_{ij}$ |
| **(1)** | **(2)** | **(3)** | **(4)** | **(5)** | **(6)** | **(7)=(3)+(2)** | **(8)=(6)-(2)** | **(9)=(8)-(3)** | **(10)=(4)-(3)-(2)** | **(11)=(4)-(5)-(2)** |
| 2-3 | 80 | 32 | 112 | 32 | 273 | 112 | 193 | 161 | 0 | 0 |
| 2-5 | 16 | 32 | 48 | 32 | 80 | 48 | 64 | 32 | 0 | 0 |
| 3-6 | 32 | 112 | 144 | 273 | 305 | 144 | 273 | 161 | 0 | -161 |
| 4-9 | 32 | 40 | 72 | 40 | 110 | 72 | 78 | 38 | 0 | 0 |
| 4-10 | 90 | 40 | 150 | 40 | 150 | 130 | 60 | 20 | 20 | 20 |
| 5-12 | 30 | 48 | 78 | 80 | 120 | 78 | 90 | 42 | 0 | -32 |
| 5-13 | 60 | 48 | 108 | 80 | 140 | 108 | 80 | 32 | 0 | -32 |
| 6-7 | 15 | 144 | 320 | 305 | 320 | 159 | 305 | 161 | 161 | 0 |
| 9-10 | 40 | 72 | 150 | 110 | 150 | 112 | 110 | 38 | 38 | 0 |
| 12-14 | 50 | 78 | 138 | 120 | 170 | 128 | 120 | 42 | 10 | -32 |
| 13-14 | 30 | 108 | 138 | 140 | 170 | 138 | 140 | 32 | 0 | -32 |
| 14-15 | 50 | 138 | 188 | 170 | 220 | 188 | 170 | 32 | 0 | -32 |
| 15-16 | 40 | 188 | 228 | 220 | 260 | 228 | 220 | 32 | 0 | -32 |
| 16-7 | 50 | 228 | 320 | 260 | 320 | 278 | 270 | 42 | 42 | 10 |
| 16-17 | 60 | 228 | 288 | 260 | 320 | 288 | 260 | 32 | 0 | -32 |
| 17-7 | 0 | 288 | 320 | 320 | 320 | 288 | 320 | 32 | 32 | 0 |

## 6.2 Python Pseudo code of Critical path Algorithm Given Below:

```
DEFINE FUNCTION analysis(rows):
    FOR eachTask IN rows.keys():
        SET rows[eachTask]['ES'] TO 0
        SET rows[eachTask]['EF'] TO 0
        SET rows[eachTask]['LS'] TO 0
        SET rows[eachTask]['LF'] TO 0
        SET rows[eachTask]['float'] TO 0
        SET rows[eachTask]['isCritical'] TO False
    FOR taskFW IN rows:
        IF '-1' IN rows[taskFW]['$d']:
            SET rows[taskFW]['ES'] TO 1
            SET rows[taskFW]['EF'] TO (rows[taskFW]['duration'])
        ELSE:
            FOR k IN rows.keys():
                FOR eachDependency IN rows[k]['$d']:
                    IF (eachDependency != '-1' and len(
                            rows[k]['$d']) EQUALS 1):
                        SET rows[k]['ES'] TO int(rows['task' + eachDependency]['EF']) + 1
                        SET rows[k]['EF'] TO int(rows[k]['ES']) + int(rows[k]['duration']) - 1
                    ELSEIF eachDependency != '-1':
                        IF int(rows['task' + eachDependency]['EF']) > int(rows[k]['ES']):
                            SET rows[k]['ES'] TO int(rows['task' + eachDependency]['EF']) + 1
                            SET rows[k]['EF'] TO int(rows[k]['ES']) + int(rows[k]['duration']) - 1
    SET tempList TO list()
    FOR element IN rows.keys():
```

```
FOR element IN rows.keys():
    tempList.append(element)
SET targetList TO list()
WHILE len(tempList) > 0:
    targetList.append(tempList.pop())
FOR eachItem IN targetList:
    IF targetList.index(eachItem) EQUALS 0:
        SET rows[eachItem]['LF'] TO rows[eachItem]['EF']
        SET rows[eachItem]['LS'] TO rows[eachItem]['ES']
    FOR eachDependency IN rows[eachItem]['$d']:
        IF eachDependency != '-1':
            IF rows['task' + eachDependency]['LF'] EQUALS 0:
                SET rows['task' + eachDependency]['LF'] TO int(rows[eachItem]['LS']) - 1
                SET rows['task' + eachDependency]['LS'] TO int(rows['task' + eachDependency]['LF']) - int(
                    rows['task' + eachDependency]['duration']) + 1
                SET rows['task' + eachDependency]['float'] TO int(rows['task' + eachDependency]['LF']) - int(
                    rows['task' + eachDependency]['EF'])
            IF (int(rows['task' + eachDependency]['LF']) > int(
                    rows[eachItem]['LS'])):
                SET rows['task' + eachDependency]['LF'] TO int(rows[eachItem]['LS']) - 1
                SET rows['task' + eachDependency]['LS'] TO int(rows['task' + eachDependency]['LF']) - int(
                    rows['task' + eachDependency]['duration']) + 1
                SET rows['task' + eachDependency]['float'] TO int(rows['task' + eachDependency]['LF']) - int(
                    rows['task' + eachDependency]['EF'])
OUTPUT('Task id, Task name, Duration, ES, EF, LS, LF, float, isCritical')
```

```
    OUTPUT('Task id, Task name, Duration, ES, EF, LS, LF, float, isCritical')
    FOR task IN rows:
        IF rows[task]['float'] EQUALS 0:
            SET rows[task]['isCritical'] TO True
        OUTPUT(
            str(rows[task]['id']) + ', ' + str(rows[task]['name']) + ', ' + str(rows[task]['duration']) + ', ' + str(

                rows[task]['ES']) + ', ' + str(rows[task]['EF']) + ', ' + str(rows[task]['LS']) + ', ' + str(

                rows[task]['LF']) + ', ' + str(rows[task]['float']) + ', ' + str(rows[task]['isCritical']))
# Calling the algorithm
# $d is dependency
SET _tasks TO {'task1': {'id': '1', 'name': 'A', 'duration': '32', '$d': ['-1']},
'task2': {'id': '2', 'name': 'B', 'duration': '80', '$d': ['1']},
'task3': {'id': '3', 'name': 'C', 'duration': '8', '$d': ['1']},
'task4': {'id': '4', 'name': 'D', 'duration': '16', '$d': ['1']},
'task5': {'id': '5', 'name': 'E', 'duration': '32', '$d': ['2']},
'task6': {'id': '6', 'name': 'F', 'duration': '16', '$d': ['5']},
'task7': {'id': '7', 'name': 'G', 'duration': '28', '$d': ['3']},
'task8': {'id': '8', 'name': 'H', 'duration': '256', '$d': ['3']},
'task9': {'id': '9', 'name': 'I', 'duration': '32', '$d': ['3']},
'task10': {'id': '10', 'name': 'J', 'duration': '16', '$d': ['7']},
'task11': {'id': '11', 'name': 'K', 'duration': '32', '$d': ['8']},
'task12': {'id': '12', 'name': 'L', 'duration': '16', '$d': ['9','10','11']},
'task13': {'id': '13', 'name': 'M', 'duration': '260', '$d': ['4']},
'task14': {'id': '14', 'name': 'N', 'duration': '16', '$d': ['4']},
'task15': {'id': '15', 'name': 'O', 'duration': '64', '$d': ['13']},
'task16': {'id': '16', 'name': 'P', 'duration': '16', '$d': ['14']},
'task17': {'id': '17', 'name': 'Q', 'duration': '16', '$d': ['15','16']},
'task18': {'id': '18', 'name': 'R', 'duration': '20', '$d': ['17']},
'task19': {'id': '19', 'name': 'S', 'duration': '12', '$d': ['18']},
'task20': {'id': '20', 'name': 'T', 'duration': '16', '$d': ['18']},
'task21': {'id': '21', 'name': 'U', 'duration': '16', '$d': ['12']},
'task22': {'id': '22', 'name': 'V', 'duration': '18', '$d': ['21']},
'task23': {'id': '23', 'name': 'W', 'duration': '16', '$d': ['22']},
'task24': {'id': '24', 'name': 'X', 'duration': '48', '$d': ['19','20','23','6']}}
analysis(_tasks)
```

# 7 Chapter 7

## 7.1 Conclusion

Operations Research approaches can be highly helpful in route optimization and enhancement of resource utilisation. It gives easy and convenient solution for better decision-making, saving time, resources and cost for enterprises. In this study, we used the Hungarian technique of assignment algorithm to re-route flights between Mumbai and Delhi in order to improve crew rest time for a certain airline operator As a result, OR methods can be particularly useful in determining flight times and routes in order to save operational costs. After analysing the aircraft maintenance activities of the airline using Critical Path Analysis, it can be concluded that CPM is a valuable tool for reducing time elapsed and increasing flying hours, which in turn increases the airline's profits, and we also solve the airline problem with various cases. Any aviation industry can benefit from the algorithms we mentioned above by saving time and money while increasing profits.

# 8 References

## References

[1] HARVEY H. SHORE. (2007). The Transportation Problem and the Vogel Approximation Method. Decision Sciences 13(4), (441–457). DOI: 10.1111/j.1540-5915.1970.tb00792.x

[2] Krishan Kasad,Kavisha Doshi, Kavya Kaushik, Khushboo Balani, Krish-Nathany(2021) Application of Operations Research in the Airline Industry. International Journal of Scientific Research and Engineering Development 4(5), (827-837). Retrieved March 2, 2022, from: http://www.ijsred.com/volume4/issue5/IJSRED-V4I5P86.pdf.

[3] Cynthia Barnhart, Peter Belobaba and Amedeo R. Odoni.(2003).Application of Operations Research in the Air Transport Industry.Transportation Science.37(4).Issue on Aviation Operations Research: Commemorating100 Years of Aviation (November 2003),368-391.Retrieved March 8 ,2022, from: https://www.jstor.org/stable/25769164?seq=1cid=pdf

[4] Varun, Mittal. Vidisha, Diwan. Anand, Baid.(2017). A Study on The Application of Operational Research in The Airline Industry. International Journal of Innovative Science and Research Technology. 2(10), (228-241), Retrieved March10, 2022, from: https://ijisrt.com/wp-content/uploads/2017/10/A-STUDY-ON-THE-APPLICATION-OF-OPERATIONS-RESEARCH-IN-THE-AIRLINE-INDUSTRY-1-ilovepdf-compressed.pdf

[5] F. M. ZeghalM. Minoux. (2004).Modeling and solving a Crew Assignment Problem in air transportation. European Journal of Operational Research 175 (2006), 187–209.Doi:10.1016/j.ejor.2004.11.028

[6] Sonakshi Khanna, Trusha Dholakiya, Suraj Prakash, Shubhang Gupta, Udit Jain. (2021) Application of the Hungarian Assignment Method in the Aviation Industry. International Journal of Innovative Science and Research Technology.6(10), (300-305). Retrieved March 18, 2022, from: https://ijisrt.com/assets/upload/files/IJISRT21OCT220.pdf

[7] Solai Rani P. (2021). Application of Graph Theory in Air-Transportation Network. J PurAppl Math. 2021; 5(1), (1-4). Retrieved March 22, 2022, from:https://www.pulsus.com/scholarly-articles/application-of-graph-theory-in-airtransportation-network.pdf.

[8] M. A. Javaid, (2013). Understanding Dijkstra algorithm, SSRN Electronic Journal, (1-27) . Retrieved March 28, 2022 from: Doi: 10.2139/ssrn.2340905

[9] Israa Ezzat Salem, Maad M. Mijwil, Alaa Wagih, Marwa M. Ismaeel. (2022), Flight-schedule using Dijkstra's algorithm with comparison of routes findings. International Journal of Electrical and Computer Engineering 12(2). (1675-1682). Retrieved April 1, 2022, from: DOI: 10.11591/ijece.v12i2.pp1675-1682

[10] Indrawaty.Y. Marit, Ellysa Nursanti, Prima Vitasari. (2022). Critical Path Method to Accelerate Automotive Maintenance Duration. INTERNATIONAL JOURNAL OF SCIENTIFIC TECHNOLOGY RESEARCH. 9(3), (6777-6782). Retrieved April 4, 2022, from: https://www.ijstr.org/final-print/mar2020/Critical-Path-Method-To-Accelerate-Automotive-Maintenance-Duration.pdf.

[11] Tzeu-Chen Han, Cheng-Chi Chung, Gin-Shuh Liang. (2006). Application of fuzzy critical path method to airport's cargo ground operation systems. Journal of Marine Science and Technology,14(3), (139-146). Retrieved April 8, 2022 from: DOI: 10.51400/2709-6998.2067

[12] Alam, M.A. and Faruq, M.O. 2019. Finding Shortest Path for Road Network Using Dijkstra's Algorithm. Bangladesh Journal of Multidisciplinary Scientific Research. 1, 2 (Jul. 2019), 41-45.DOI:10.46281/bjmsr.v1i2.366.

[13] Javaid, Adeel. (2013). Understanding Dijkstra Algorithm. SSRN Electronic Journal.DOI:10.2139/ssrn.2340905

[14] Junqueira, V.S.V. Nagano, M.S. Miyata, H. H. (2018) Procedure Structuring for Programming Aircraft Maintenance Activities. Revista de Gestão.27(2), (2-20). Retrieved April 9, 2022. From: www.emeraldinsight.com/2177-8736.htm, 2018 DOI: 10.1108/REGE-02-2018-0026

**List of Conferences / publications**

**Conference -** International Conference on Applied Mathematics and Computer Sciences-(ICAMCS-22)
**Organisers:** INSTITUTE FOR RESEARCH AND ACADEMIC JOURNALS (IRAJ)
**Title of Paper:** Applications Of Operational Research In Airline Management And Scheduling.
**Authors:** Vedika Dixit, L.N.Das
**Date of paper submission:** 19 April,2022
**Date of paper Acceptance:** 22 April,2022
**Date of paper conference:** 23 April,2022
**Name of Publication :** International Journal of Mechanical Engineering
**Link of Publication :** https://www.scopus.com/sourceid/21101016918


International Conference on Applied Mathematics and Computer Sciences- (ICAMCS-22)

**23rd April 2022,Patna,India**

This is to certify that ..... **Vedika Dixit** .....

of..... *Delhi Technological University, Delhi, India* .....

has done his/her excellence in presenting the research paper titled....."*Applications Of Operational Research In Airline Management And Scheduling*"....

.....

.....

on 23rd April 2022 at Patna,India.

Dr. D. Kirubakaran
President

Vinoth Vikram
Convener

**EVENT ACCEPTANCE LETTER**

**International Conference on Applied Mathematics and Computer Sciences-ICAMCS 23RD APRIL 2022 PATNA, INDIA**

Dear Vedika Dixit,

The reviewers committee of IRAJ congratulates you for acceptance of your research paper titled **"APPLICATIONS OF RESEARCH IN AIRLINE MANAGEMENTAND SCHEDULING OPERATIONAL"** bearing the paper Id **IRAJCONF_152304** and is for **International Conference on Applied Mathematics and Computer Sciences-ICAMCS** on **23RD APRIL 2022 PATNA, INDIA**. You are cordially invited to convene the event by presenting your research paper through Power Point presentation.

**For registration:** https://iraj.co.in/conf/reg.php?id=1609602

**Registration Procedure:**
To complete your registration procedure kindly deposit the registration fees which includes

| CATEGORIES | INDIAN | VIRTUAL PRESENTATION (INDIAN DELEGATES) |
|---|---|---|
| Student (UG / PG) | INR 5000.00 | INR 4500.00 |
| Student (UG / PG) with Scopus publication | INR 15000.00 | INR 14500.00 |
| PhD/Research Scholar | INR 5500.00 | INR 5000.00 |
| PhD/Research Scholar with Scopus publication | INR 15500.00 | INR 15000.00 |
| Academician / Industrial Professionals | INR 6000.00 | INR 5500.00 |
| Academician / Industrial Professionals with Scopus publication | INR 16000.00 | INR 15500.00 |
| Listeners | INR 2500.00 | INR 1200.00 |

**Mode of Payments:**

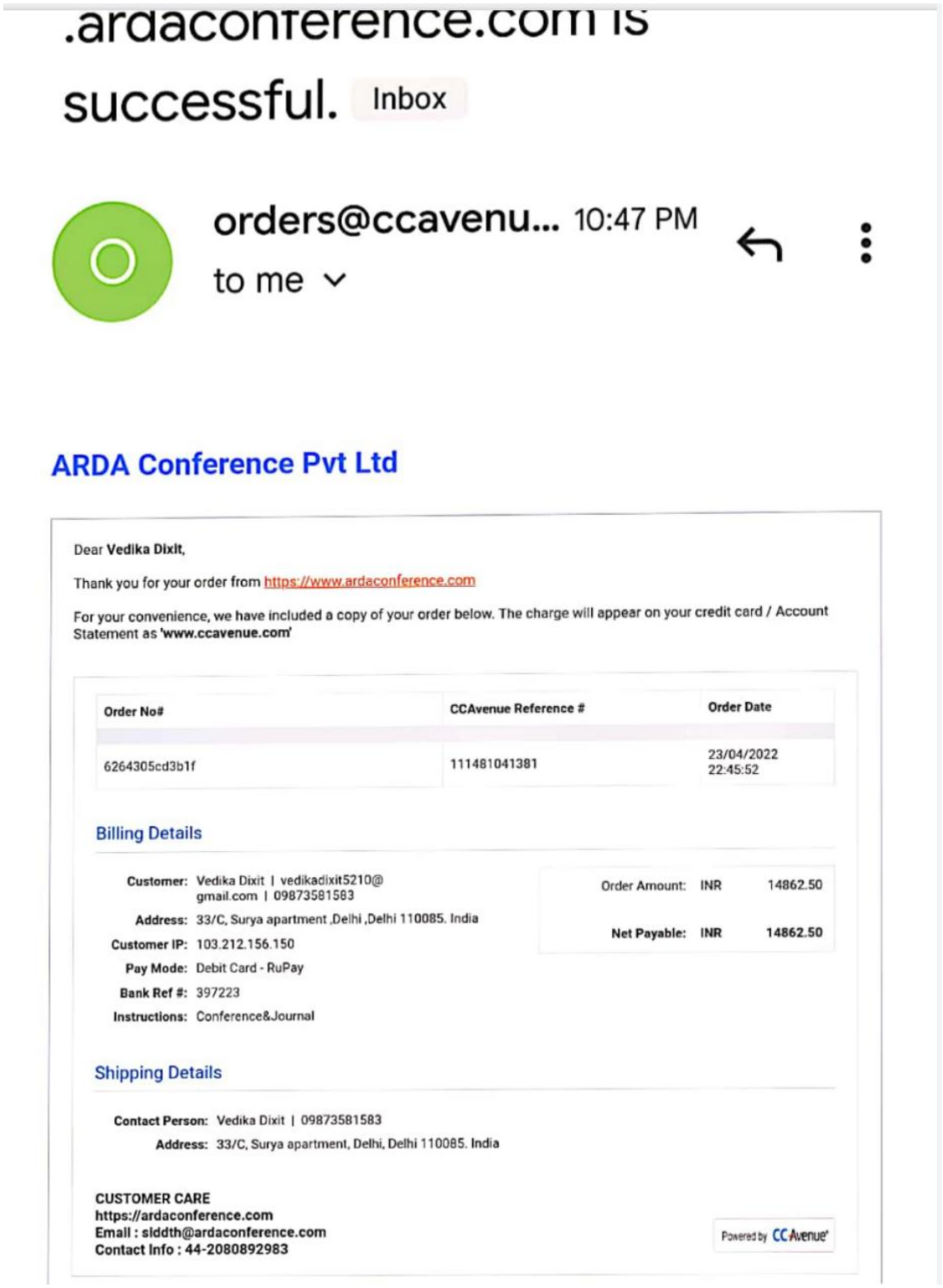
**Online Payment Link:** https://www.ardaconference.com/payment/

| Bank Details (For Offline Payment) | |
|---|---|
| Bank Account Name | **ARDA CONFERENCE PRIVATE LIMITED** |
| Bank Name | **IDFC FIRST Bank** |
| Account Number | **10043762624** |
| IFSC Code | **IDFB0080101** |
| Swift Code | **DFBINBBMUM** |
| Branch | **Nungambakkam, Chennai** |
| Country | **India** |

**You are requested to release the payment and mail us the screen of successful payment release with your name and title of paper to confirm your registration.**

**Camera ready paper submission:**
Kindly send your final camera ready paper for publication in our conference proceeding and Journal (Optional) for final printing purpose. Further subsequent mails regarding any modification will be sent to you by our reviewer's panel.

**Regards,**
**Conference Coordinator**
**Institute for Research and Academic Journal (IRAJ)**
**Contact no: +91-9677007228**



.ardaconference.com is successful. Inbox

orders@ccavenu... 10:47 PM

to me

## ARDA Conference Pvt Ltd

Dear **Vedika Dixit**,

Thank you for your order from https://www.ardaconference.com

For your convenience, we have included a copy of your order below. The charge will appear on your credit card / Account Statement as **'www.ccavenue.com'**

| Order No# | CCAvenue Reference # | Order Date |
|---|---|---|
| 6264305cd3b1f | 111481041381 | 23/04/2022 22:45:52 |

### Billing Details

**Customer:** Vedika Dixit | vedikadixit5210@gmail.com | 09873581583
**Address:** 33/C, Surya apartment ,Delhi ,Delhi 110085. India
**Customer IP:** 103.212.156.150
**Pay Mode:** Debit Card - RuPay
**Bank Ref #:** 397223
**Instructions:** Conference&Journal

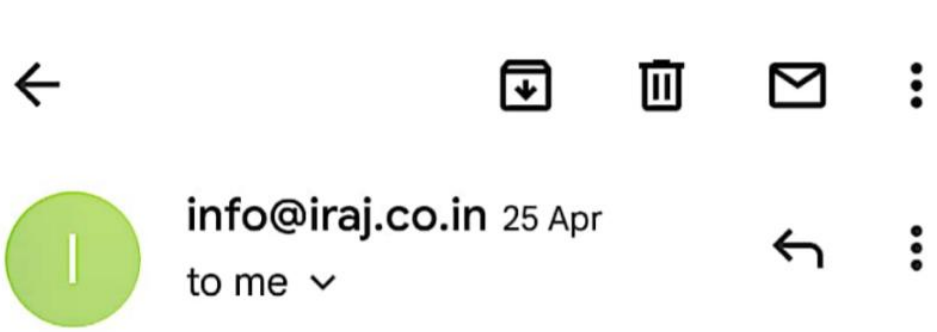
**Order Amount:** INR 14862.50
**Net Payable:** INR 14862.50

### Shipping Details

**Contact Person:** Vedika Dixit | 09873581583
**Address:** 33/C, Surya apartment, Delhi, Delhi 110085. India

**CUSTOMER CARE**
**https://ardaconference.com**
**Email : siddth@ardaconference.com**
**Contact Info : 44-2080892983**

Powered by CCAvenue



info@iraj.co.in 25 Apr

to me

Dear Vedika Dixit,

Greetings!!!

Thanks for registering with us. We have received the registration amount of 14500 INR for the International Conference on Applied Mathematics and Computer Sciences-ICAMCS. We will share the next update soon.

For more queries, please don't hesitate to draft us an e-mail. We will be happy to help you.