

AIR TRAFFIC MANAGEMENT

Enroll no. – 101445

Name - Vibhor Saxena

Supervisor - Prof. Dr. Satya Prakash Ghrera



May, 2014

Submitted

In

Partial fulfillment of Degree of Bachelor of Technology

DEPARTMENT OF COMPUTER SCIENCE ENGINEERING

AND

INFORMATION TECHNOLOGY

**JAYPEE UNIVERSITY OF INFORMATION TECHNOLOGY ,
WAKNAGHAT.**

Table of Contents

Index	Page No.
1) INTRODUCTION	6
-What is Air Traffic Management?	
-Need for Air Traffic Management.	
2) PROBLEM STATEMENT AND MOTIVATION	7
3) LITERATURE REVIEW	8
-Fairness Landing Algorithm	
- Compromise immune algorithm (CIA)	
- Optimal take-off and landing algorithm	
- Flight Arrival Scheduling Strategy	
4) PROPOSED SOLUTION	22
5) SIMULATION	43
6) TESTING OUTCOMES	48
7) FUTURE WORKS	50
8) REFERENCES	52

CERTIFICATE

This is to certify that the work entitled-“**AIR TRAFFIC MANAGEMENT**” submitted by **VIBHOR SAXENA-101445** in partial fulfillment for award for degree of Bachelor of Technology in Information Technology of JAYPEE UNIVERSITY OF INFORMATION TECHNOLOGY has been carried out under my supervision. This work has not been submitted partially or wholly to any other University for any award of this or any other degree.

Prof. Dr. Satya Prakash Ghrera

(HOD computer science dept.)

Department of Computer Science Engineering and Information Technology

Jaypee University of Information Technology

Waknaghat

Acknowledgement

“It is not possible to prepare a project without the assistance & Encouragement of other people. This one is certainly no exception.”

On the very outset of this report, we would like to extend our sincere & heartfelt obligation towards all the personages who have helped us in this endeavor. Without their active guidance, help, cooperation & encouragement, we would not have made headway in the project.

We would like to show our greatest appreciation to **Prof. Dr. Satya Prakash Ghrera**. We feel motivated every time we get his encouragement. For his coherent guidance throughout the tenure of the project, we feel fortunate to be taught by **Prof. Dr. Satya Prakash Ghrera**, who gave us his unwavering support. Besides being our mentor, he taught us that there is no substitute for hard work.

We will be always in debt of Mr. Punit Gupta for providing us his timely help and guidance.

We owe our heartiest thanks to Brig. (Retd.) S.P. Ghrera (HOD, CES/IT Department) who has always inspired us to take initiatives and showed us the path for achieving our goal.

In the light of new developments and recent findings, we devote the task that was asked from us at Jaypee University of Information Technology to **“GPS BASED AIR TRAFFIC MANAGEMENT”**.

VIBHOR SAXENA-101445

Summary

Air traffic management systems are used to maintain the safe, orderly and expeditious flow of air traffic in the global air traffic control system. The position of air traffic system is one that requires highly specialized knowledge, skills, and abilities. System apply separation rules to keep aircraft at a safe distance from each other in their area of responsibility and move all aircraft safely and efficiently through their assigned sector of airspace, as well as on the ground. Because system have an incredibly large responsibility while on duty (often in aviation, "on position") and make countless split-second decisions on a daily basis, the ATC systems are consistently regarded around the world as one of the most mentally challenging careers, and can be notoriously stressful depending on many variables (equipment, configurations, weather, traffic volume, human factors, etc.). Also as we know that by time the air traffic is also increasing at very fast rate, and so is the delays of flight, and flight cancelation, because of human intervention the air traffic management is not very fast and accurate, so we are trying is to create a scheduling algorithm with minimum delay and minimum human intervention. So we are creating this algorithm by using multithreading, so that we could get minimum delays in our flight management. Currently we are taking a scenario from Delhi IGI Airport.

Signature of Student

Name: Vibhor Saxena
Ghrera

Date

Signature of Supervisor

Name: **Prof. Dr. Satya Prakash**

Date

Chapter - I

Introduction

What is Air Traffic Management?

Air traffic management is an aviation term encompassing all systems that assist aircraft to depart from an aerodrome, transit airspace, and land at a destination aerodrome, including Air traffic control (ATC), Air Space Management (ASM), Air Traffic Services (ATS), and Air Traffic Flow Management (ATFM) , Flight Scheduling and Capacity Management.

The increasing emphasis of modern ATM is on interoperable and harmonized systems that allow an aircraft to operate with the minimum of performance change from one airspace to another. ATC systems have traditionally been developed by individual States that concentrated on their own requirements, creating different levels of service and capability around the world. Many Air Navigation Service Providers (ANSPs) do not provide an ATC service that matches the capabilities of modern aircraft, so ICAO has developed the Aviation System Block Upgrade (ASBU) initiative in order to harmonize global planning of technology upgrades.

Need for Air Traffic Mananagement

- 1) To regulate planes on their desired paths
- 2) To prevent collision of planes in mid air
- 3) To give guidance and help to plane when it is in mid air.
- 4) To help the pilots locate the exact location of runway and helping in landing the plane.
- 5) To help automated updating of flight delay and suspension

Chapter-II

Problem Statement and Motivation

There are several algorithms which provide flight scheduling for Airport Traffic Control System(ATCS) in an undertaken scenario but are not valid for all the different scenarios of Air Traffic Control System .

To create an algorithm which optimize the air traffic management system and can work in different scenario like emergency landing , collision time of two planes , different planes according to their weight etc.Also to minimize the human intervention by creating this algorithm this algorithm cuts human work and increases efficiency and reduces human error.

Chapter-III

Literature Survey

We have studied four algorithms which implement different scheduling algorithms for scheduling flights in an airport . the **four algorithms which we studied are:-**

- 1) Fairness In Flight Landing Scheduling**
- 2) Compromise Immune Algorithm (CIA)**
- 3) Optimal Take off and Landing Algorithm**
- 4) Flight Arrival Scheduling Strategy**

[1] ALGORITHM 1:

Fairness In Flight Landing Scheduling

Yong Wang, Feng Wang, Dan Wang, Zhenghu Gong and Jiangchuan Liu “Revisit the Fairness Issues in Flight Landing Scheduling” 15th International IEEE Conference on Intelligent Transportation Systems, Anchorage, Alaska, USA, September 16-19, 2012

Flight landing optimization at the terminal area is an ongoing challenge for air traffic controllers. The current schedule scheme is first-come-first-served (FCFS). There are studies focusing on how to minimize the total cost or maximize the throughput. These schemes are short of fairness consideration. In this paper, we start from a real recent example to show that a lack of consideration of long-term fairness may cause significant problems in certain emergency situations. We propose new definitions of fairness between different airlines which consider the history fairness information. We then study the fairness in flight landing problem (FFLP). We show that the problem is NP-hard to solve. Therefore, we develop a fast heuristic. Our experiment results show that our algorithm achieves a 50.4% gain on long-term fairness.

Constraints

We consider that there are n flights to be scheduled and denote them as $F = \{f_1, \dots, f_n\}$, ordered by their ETA. Besides the ETA for flight f_i , which we denote as ETA_i , there are also three other constraints as follows:

1)Arrival Time Windows:

A flight must land between its earliest and latest possible landing time. This time window constraint can be dependent on the technical and operational reasons such as fuel limitation, maximum allowed delay, or maximum or minimum airspeed [9]. We use E_i and L_i to denote the earliest and latest possible time of arrival of the flight f_i respectively. So, the STA_i must satisfy $\forall f_i \in F, E_i \leq STA_i \leq L_i$.

2) Separation Requirements:-

The Federal Aviation Administration (FAA) regulates the minimum spacing between two successive landing flights in order to avoid the danger of wake turbulence [10]. The flights are classified into three weight classes (heavy, large, and small) based on the maximum takeoff weight by the FAA. The matrix of minimum separation times (in seconds) is shown in Table I. We use S to denote the matrix of minimum separation times and the minimum separation time of two consecutive flights f_i and f_j is thus $S[W_i, W_j]$, where W_i and W_j are the weight class of flight f_i and f_j respectively. The ST_{Ai} must satisfy $\forall f_i, f_j \in F$, if $ST_{Ai} < ST_{Aj}$, then $ST_{Aj} - ST_{Ai} \geq S[W_i, W_j]$.

3) Precedence Constraint:

The landing flights may be imposed on precedence over others due to overtaking constraints, airline preferences, or high priority flights [12]. We use $<$ to denote the precedence relations between two flights. f_i and f_j means f_i has higher priority than f_j and f_i must land before f_j . That is $\forall f_i, f_j \in F$, if $f_i < f_j$, then $ST_{Ai} < ST_{Aj}$. Obviously, if $f_i > f_j$ and $f_j < f_k$, then $f_i < f_k$.

Fairness Definition

Let $A = \{A_1, A_2, \dots, A_m\}$ be the set of airlines, A_j be the set of flights from airline j , where $A_j \in A$. $|A_j| = m$ represents the number of flights in A_j . Let $c_i(t_i)$ be the cost of f_i landing at time t_i , C_j be the total cost of all the flights from airline A_j , so $C_j = \sum_{f_i \in A_j} c_i(t_i)$.

In this paper, fairness in flight landing is measured by the difference between the *current cost* C_j and the *unaffected cost* C_{uj} . C_{uj} is defined as the cost of airline A_j being scheduled while ignoring all the other flights from other airlines. The corresponding *affected-extra-cost* Ca_j of airline A_j is defined as the cost caused by the presence of other airlines. According to the definitions above, we have $Ca_j = C_j - C_{uj}$.

Definition 1 (Fairness): the average affected extra cost of an airline.

Let F_j be the fairness value of airline A_j , then $F_j = C_{aj}/W_j = (C_i - C_{uj})/W_j$.

Here W_j denotes the weight of the flights in the airline. The definition of *fairness* in previous works is unfair as we have shown an example in the introduction. But under our new definition, the scheduling sequence of the two airlines could be viewed as being scheduled individually and there is no influence on each other. And therefore, the *fairness* of the two airlines are both equal to 0.

Advantages

- Consider aircrafts of variable sizes that is heavy, large & small.
- Consider aircrafts of different airlines.
- Algorithm is best for solving emergency scheduling scenarios .
- It shows 50.4% gain on total cost & fairness as compared to FCFS.

It consider history compensation in fairness scheduling scheme that is a if one flight from an airline is delayed at a time, compensation will be made in later scheduling on the flights from the same airline

Disadvantages

- Algorithm is only designed for landing of an aircraft thus misses flight take-off scenario.
- Does not include the scenario were a flight is delayed beyond its scheduling time.
- Assume the cost of flight is as linear function rather than a complex function.

[2] Algorithm 2

Compromise immune algorithm (CIA)

Jianli Ding, Jinling Ji, Heyi “Wang Optimization of Airport Flight Arrival and Departure Based on Compromise Immune Algorithm” *IEEE Transactions on Control Systems* 2007.

The essence of optimization of airport flight arrival and departure is to seek optimal sequencing of flight arrival and departure and reduce flight delay. In this paper according to the rule of “Flight arrival has priority over flight departure” and the limit of airport capacity, optimal schedule of airport flight arrival and departure is achieved through compromise immune algorithm (CIA), and flight delay is minimized. Experimental results show that CIA has great ability of problem solving and is especially suitable for solving practical problems of combinational optimization.

CIA is developed on the basis of combining the compromise method of optimizing several objectives. The compromise method decides the point closest to the ideal point using some kind of distance measuring method. To many complex problems, it is difficult to find the ideal point. CIA replaces the ideal point with the concept of proxy ideal point. The proxy ideal point is the point relating to current genetic generation instead of the ideal point of given problems. In other words, the proxy ideal point is computed according to part of the discovered solution space instead of the whole solution space. In each generation it is easy to acquire proxy ideal point. With advance of evolution process, the proxy ideal point will get close to the real ideal point.

The model is an integral programming model and uses CIA to seek solution. See Figure 1 for algorithm description. . In this paper’s algorithm, population size is 200; crossover probability is 0.3; mutation probability is 0.2; iteration times are 4. In the initial population, select the individual with minimum total time of delay as the proxy ideal point, and meanwhile set the regret value of every individual as $\square\square=1$; While population, update proxy ideal point, and meanwhile individual’s regret value varies with the change of individual’s delay time. Table 1 lists the results of flight

optimization within each time interval. For easy comparison, s the final traffic flow allocation approaches. In comparison with the initial

IMPORTANT POINTS

- CIA is developed on the basis of combing the compromise method of optimizing several objectives.
- In this, time is divided into several 15 min intervals . Flights within the same time interval are free of strict sequencing restriction, which actually allows great freedom to arrange flights.
- It uses two variable arrival capacity & departure capacity , that is the no. of planes that can be easily accommodated in a given time interval.

Advantages

- Airport's arrival and departure requirements and optimal allocation of arrival and departure capacities within given time are considered together.
- As compared to FCFS it produce least cost of flight delays.
- It take into consideration the that "Flight arrival has priority over flight departure".

- **Disadvantages**

- Does not consider any difference in types of plane.
- Does not provide any mechanism for dealing emergency situations.
- When compared with genetic algorithm it produces more number of delays.

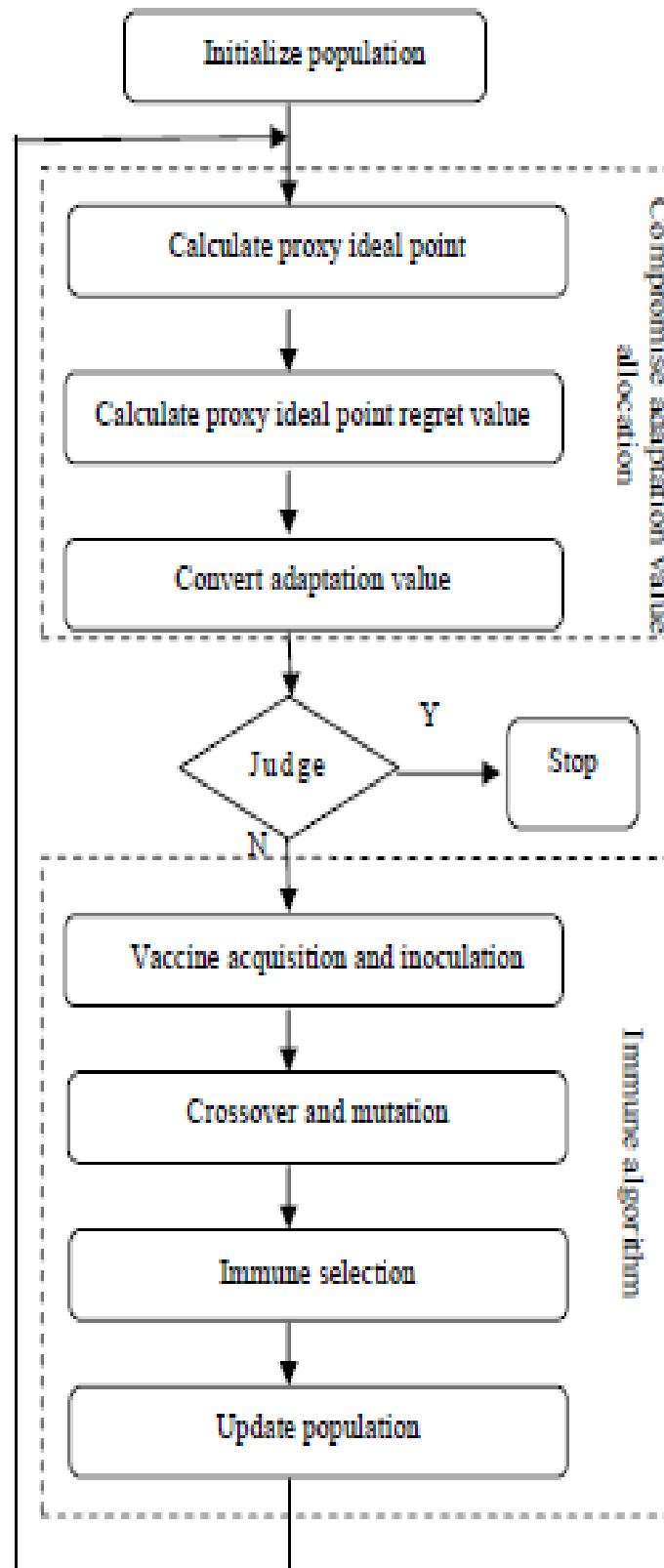


Figure 1. Flow chart of CLA

[3] **ALGORITHM 3**

Optimal take-off and landing algorithm.

Andrea D'Ariano, Paolo D'Urgolo, Dario Pacciarelli and Marco Pranzo" Optimal sequencing of aircrafts take-off and landing at a busy airport" 13th International IEEE Annual Conference on Intelligent Transportation Systems, Madeira Island, Portugal, September 19-22, 2010

This paper studies the problem of sequencing aircraft take-off and landing operations at congested airports. We introduce and analyze alternative detailed formulations and solution algorithms for scheduling arrival and departure times of the aircrafts, such that the delay with respect to the scheduled times is minimized. The aircraft scheduling problem (ASP) is viewed as an extension of the job shop scheduling problem with additional real-world constraints and formulated by using alternative graphs. Two alternative formulations model the required time separation among aircrafts in air segments and runways according to safety regulations and differ for the level of detail used to represent the holding circles. Scheduling rules, heuristic and exact methods are implemented and tested on practical size instances of the Fiumicino airport, the busiest airport in Italy. We show that two versions of an innovative branch and bound algorithm are always able to find good solutions in a few seconds and often improve the best solution computed by the scheduling heuristics. Optimality is proved in less than two minutes for more than half of the instances.

From a logical point of view, ATC decisions can be broadly divided into:

- 1) Routing decisions
- 2) Scheduling decisions

- Job Shop Scheduling
- Heuristics and Exact Algorithm

Sub-parts of aircraft scheduling algorithms evaluated in this algorithm

- 1)The Arc Greedy Heuristic (AGH)
- 2)Heuristic AMCC (Avoid Most Critical Completion time)
- 3)Heuristic AMSP (Avoid Most Similar Pair)
- 4)Branch and Bound (denoted as BB)
- 5)Job Greedy Heuristics

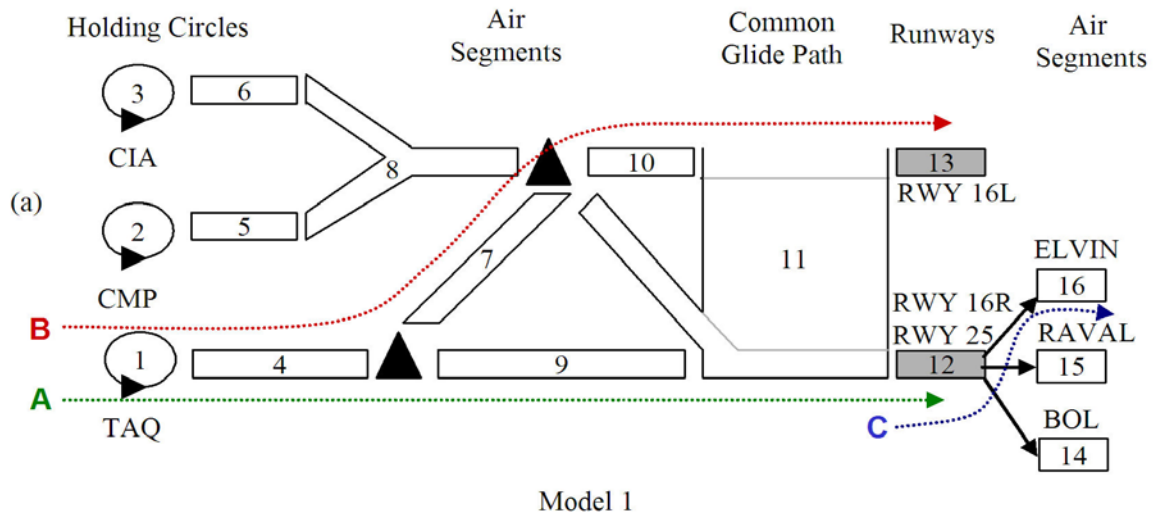


Figure – 2

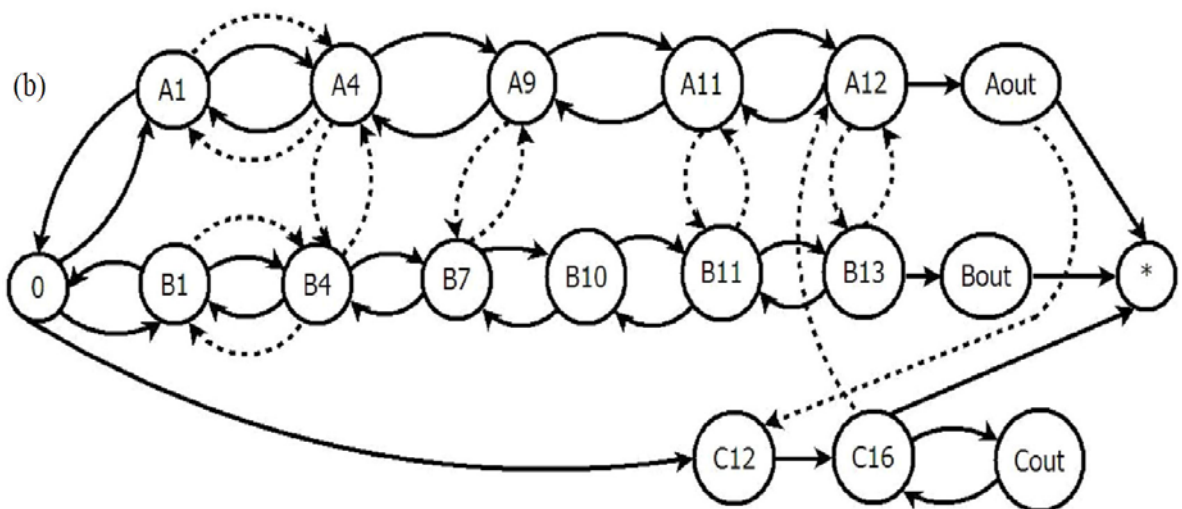


Figure- 3

ADVANTAGE

- Optimal/near-optimal solutions are found in a few seconds of computation for most of the instances and BB often outperforms the heuristics, including a practical scheduling technique.
- On the other hand, BB is always able to find a good solution within a few seconds.

DISADVANTAGE

- This paper don't work on entrance delay in TMA

[4] ALGORITHM 4

Flight Arrival Scheduling Strategy

Shiwei Zhao ,Guo Wei , Huijuan Feng “Study on Optimizing Control of Flight Arrival Scheduling Strategy” International symposium on instrumentation & measurement , sensor networks and automation(IMSNA),2012

Flights arrival scheduling problem is stubborn problem in air traffic control at present. Through analyzing dynamic process of arriving flights, we make reasonable assumption and split the problem into two parts. First sequence all arriving flights in certain period, and adjust interval between arriving flights by model reference adaptive control. Simulation results show that arriving flights become steadier and smooth, which supply a new idea to deal with problem in ATC automation in future.

PROCESS OF FLIGHT ARRIVAL SCHEDULING

There are two parts in Process of flight arrival scheduling: flight sequencing and interval between flights adjusting.[2] It is main duty of Flight arrival scheduling controller that make the arrival flights from various airways to become one balanced and smooth stream of arrival flights by air traffic control rules, to guarantee flights landing well-ordered and peaceably. Air traffic controllers guide flights taking off or landing with standard flight course, as shown in figure 1, and adjust interval between flights according to maximum taking-off weight and minimum wake vortex separation. During flight arrival process, there are major factors which have effect on flights arrival scheduling, such as aircraft type, importance degree, urgency degree, and so on. According to maximum taking-off weight, there are three aircraft types: heavy aircraft, its maximum weight equal or above 136t; medium aircraft, its maximum weight between 7t and 136t; light aircraft, its maximum weight equal or below 7t. Importance degree is judged by property transport event by aircraft, such as special flight, official flight, charter flight, and passenger flight and cargo flight etc. there are some accident

events appearing inevitably in flight process, urgency degree ranks them according to impact of flight safety.

Adopting Fuzzy Reason Method and Building Flight Arrival Stream Model.

- 1)NST (Norm Separation Time)

$$NST = ETA(B) - ETA(A)$$

- 2)GSD (Groundspeed Difference)

$$GSD = GroundSpeed(A) - GroundSpeed(B)$$

- 3)SPI (Standard Position Interval)

$$SPI = D(B) - D(A)$$

- 4)IMD (Importance Degree)

$$IMD = IM(B) - IM(A)$$

- 5)ORDER (flight arrival serial number)

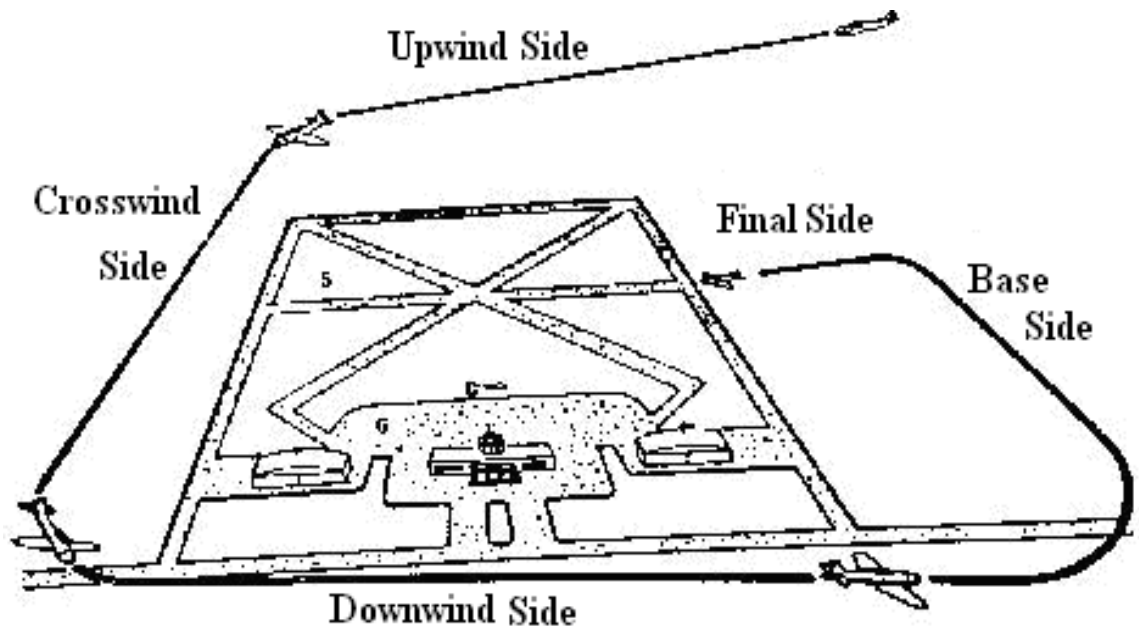


Figure-4

Characters of process of flight arrival scheduling

Summarizing above analysis and actual situation in arrival flight scheduling, we could reach following assumption, which could simplify analysis process.

Modern aircraft could execute control instructions completely; flights sequence could be adjusted by waiting in holding area or surpassing other aircraft, interval between flights could be adjusted flight parameters

Simulation aim is large and busy airport. To enhance efficiency, major aircrafts accessing these airports is medium or above type aircrafts, such as B737, A320 etc, light aircrafts taking off and landing on other neighboring smaller airports.

Advantage

- This method in sequencing arriving flights is verified by theory and proved reasonable; - Compared to FCFS algorithm, improved FCFS algorithm based on fuzzy reasoning is more efficient.

Disadvantage

- We only consider some impact factors, in actual situation, many impact factors should be considered, for example flight cancel ,route conditions, airport clearance environment, neighboring airports interaction, and the performance of the aircraft etc, those factors should be classified and dealt with in a reasonable way.

Comparison Of The Above Four Algorithms

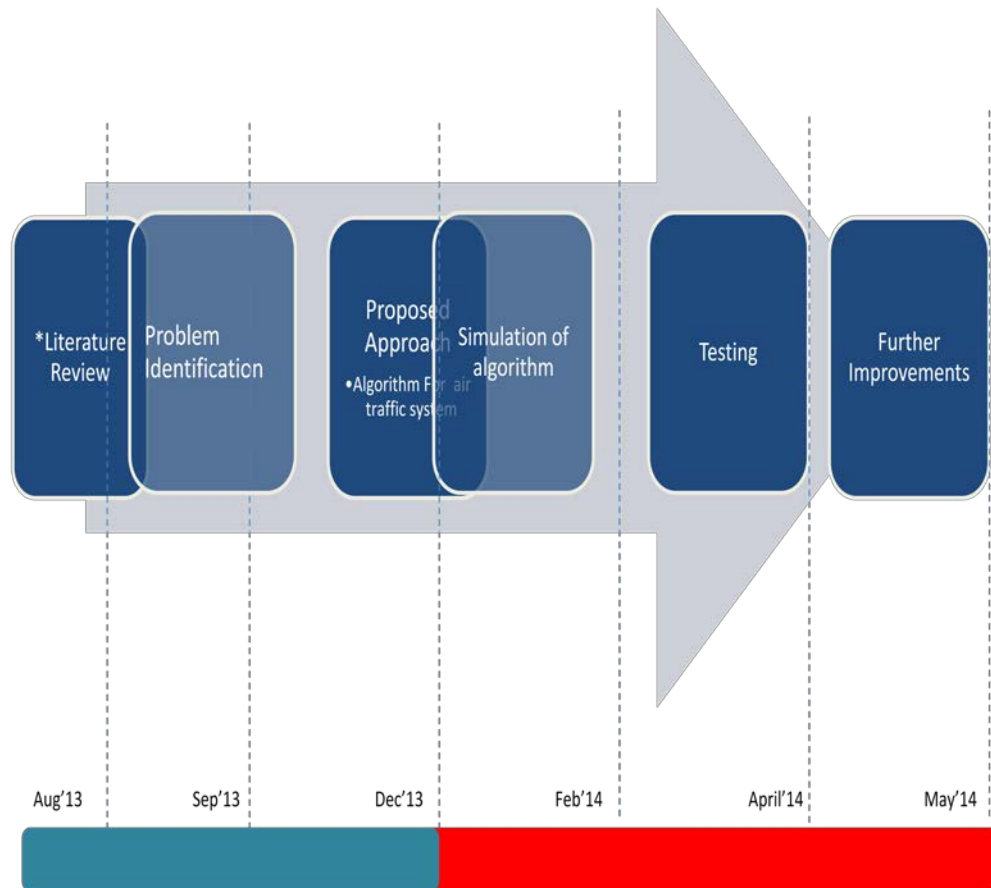
	ALGORITHM 1	ALGORITHM 2	ALGORITHM 3	ALGORITHM 4
Types of aeroplane	included	Not included	Not included	Included
Separation time	included	Included	Included	Included
Time of arrival of a plane	included	Included	Included	Included
Time of departure of a plane	Not included	Included	Included	Not included
*Emergency constraints	Included	Not included	Not included	Not included
Delay in flight schedule	Not included	Not included	Not included	Not included

TABLE 1

Here we have compared the all the four algorithms which we have studied and draw a very specific evaluations of them according to the parameters we have studied.

This evaluation show the which parameters are lacking in which algorithm and what are the disadvantages of these algorithms.

Project Coverage Plan



* To be continue till last

Figure 1

Chapter-IV

Proposed Solution

We are creating an algorithm where we have taken arrival time, departure time , strip length required for the flight to land and the weight of plane with weight which strip can bear .

This has resulted in the scheduling algorithm and the following code represents the simulation part of our algorithm.

For simulation we have used NETBEANS as tool for coding and providing classes for our front end.

Proposed Algorithm working

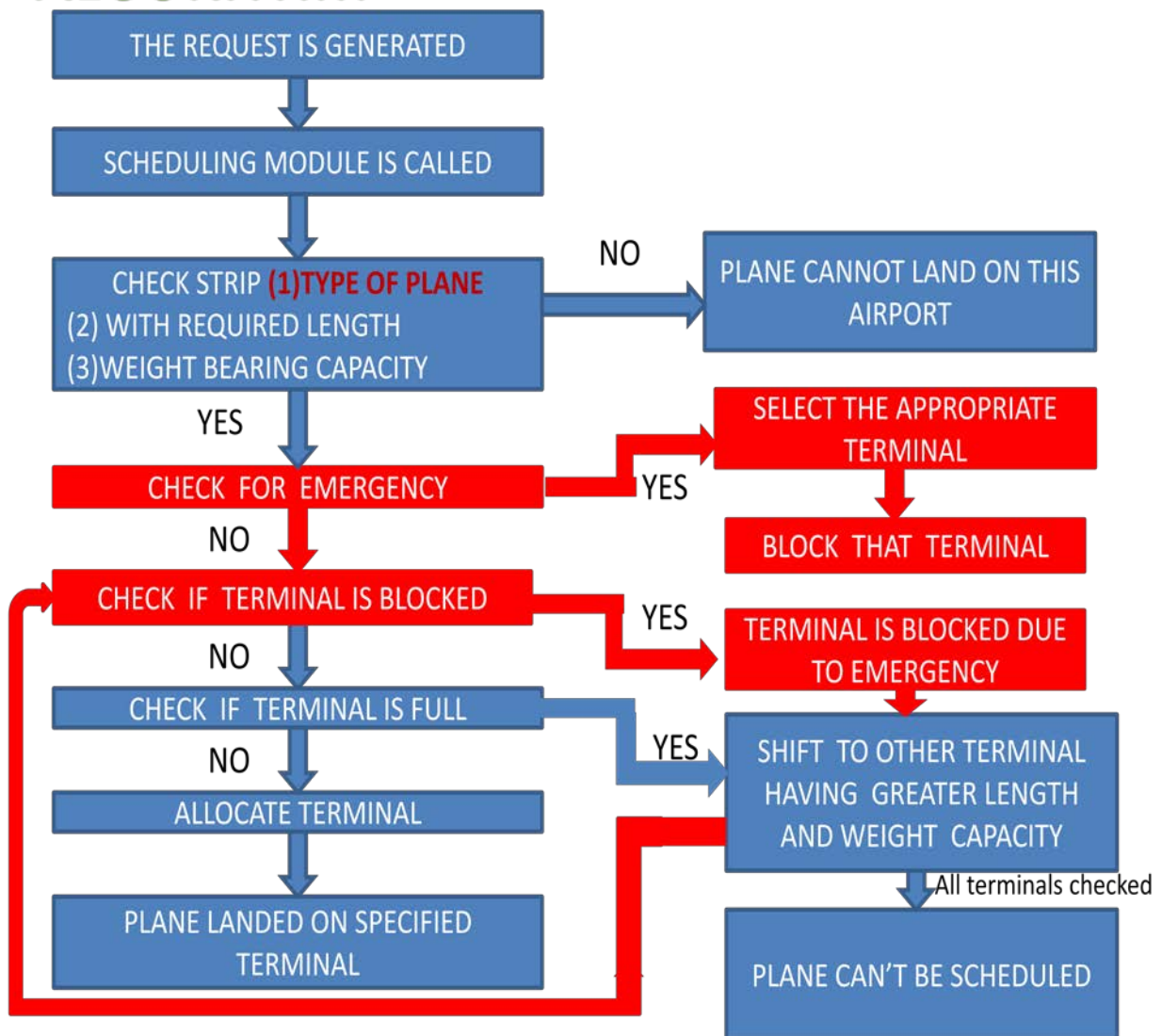
Anytime a plane arrives to the airport it sends a landing request to the algorithm, well before the time of its arrival, so as we all know that at a particular time there could be many requests arriving to the algorithm, so we have distinguished these requests on the basis of time of arrival on nano seconds difference so at a particular time only one request is coming at the algorithm, we have then put these requests in the queue using FIRST COME FIRST SERVE basis,

So our algorithm is working on multithreading, so whenever a planes requests its arrival onto the airport it checks for all the possible landing option for that airplane, and if the runway is busy, it then shifts its landing to other runway which is free or having lesser queue, so in this way we can reduce the delay time, and increase the efficiency for our algorithm.

But in case of an emergency , whenever a plane is having an emergency landing , then till the plane gets landed properly and removed from the runway till then we shuts down that runway for other flights, and try to accommodate them on other runways or if not then we diverge them to nearby airport to reduce the amount of delay.

We have designed this algorithm also to reduce the amount of human intervention, all this work is now done by humans which raise an human error also , so in order to increase the safety and decrease the delays and flight cancelation we have created this algorithm , because in future the amount of air traffic is only going to be increased.

ALGORITHM



FLOW CHART -1

CODE

Faults.java

```
package javaapplication2;

import java.util.ArrayList;
import java.util.List;
import javax.annotation.Resources;
import java.util.logging.Level;
import java.util.logging.Logger;
import java.sql.*;
import javax.swing.JTable;

public class Faults {
    Terminal s1=new Terminal();
    Terminal s2=new Terminal();
    Terminal s3=new Terminal();

    public void func()
    {

        try
        {
            Class.forName("org.apache.derby.jdbc.ClientDriver");
            // Class.forName("org.apache.derby.jdbc.ClientDriver");
            Connection con = DriverManager.getConnection("jdbc:derby://localhost:1527/db",
"vibhor", "vibhor");
            Statement statement=con.createStatement();
            ResultSet rs=statement.executeQuery("select * from untitled4 order by AR asc ");
```

```

req re3=new req();
while(rs.next())
{
requirement r1=new requirement();
//System.out.print(rs.getString("AR"));

r1.arrival_time=Integer.parseInt(rs.getString("AR"));
System.out.print(rs.getString("DP"));
r1.depart_time=Integer.parseInt(rs.getString("DP"));
//r1.delay=1;
r1.weight=Integer.parseInt(rs.getString("SWR"));
r1.lenght=Integer.parseInt(rs.getString("SLR"));


r1.id=Integer.parseInt(rs.getString("FN"));
System.out.print(r1.id);
re3.submit (r1);
// update_table();
}
}catch(Exception e)
{

}}


public void add_resource()
{
resource r=new resource();
r.name= "Indra Gandhi";
r.capacity=70;
s1.name="terminal1";
s1.striplenght=12500;
s1.weight=350;
s1.terminal=20;
s2.name="terminal2";
s1.al_terminals=0;
s2.striplenght=13780;
s2.weight=500;

```

```

        s2.terminal=25;
        s2.al_terminals=0;
        s3.name="terminal3";
        s3.striplenght=14534;
        s3.weight=700;
        s3.terminal=25;
        s3.al_terminals=0;
        r.strip.add(s1);
        r.strip.add(s2) ;
        r.strip.add(s3);
        req r2=new req();
        r2.submit(r);
    }

    /*
    public void add_requirement()
    {System.out.println("Initializing Simulation");
    System.out.println("request generating");
        for(int i=0;i<70;i++)
    {
        requirement r1=new requirement();
        r1.arrival_time=10;
        r1.depart_time=5;
        r1.delay=1;
        r1.weight=50;
        r1.lenght=8750;
        req re3=new req();
        r1.id=i;
        re3.submit (r1);

    }

    }

    */
    public void ss()

```

```

{
    func();

    add_resource();

    schedule s=new schedule();
    s.start();

}

```

```

public static void main(String args[]) {
    Faults f=new Faults();
    f.ss();
}
}

```

Schedule.java

```

package javaapplication2;

import java.beans.Statement;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.text.SimpleDateFormat;
import java.util.Calendar;
import java.util.LinkedList;
import java.util.Queue;
import java.util.logging.Level;

```

```

import java.util.logging.Logger;


public class schedule extends Thread
{
    req q=new req();
    requirement r;

    @Override
    public void run()
    {
        System.out.println("scheduling algorithm started");
        while(true)
        {
            if(!q.queue.isEmpty())
            {
                r=q.queue.poll();
                shedule_algo(r);
                try {
                    ///
                    Thread.sleep(2000);
                } catch (InterruptedException ex) {
                    Logger.getLogger(schedule.class.getName()).log(Level.SEVERE, null,
ex);
                }

                ///
                if(r.sched==0)
                {
                    q.queue.add(r);
                }
            }
            else if(!q.to.isEmpty())

```

```

        {departure();
        }
        else
        {
            try {
                finalise(q);
            } catch (ClassNotFoundException ex) {
                Logger.getLogger(schedule.class.getName()).log(Level.SEVERE, null,
ex);
            } catch (SQLException ex) {
                Logger.getLogger(schedule.class.getName()).log(Level.SEVERE, null,
ex);
            }
            this.stop();
        }
    }
}

```

```

public void shedule_algo (requirement r1)
{

    //System.out.println("scheduling started");
    int re;

    for(int i=0;i<req.conn.get(0).strip.size();i++)
    { // System.out.println("scheduling "+" id no. "+r1.id+" plane");
        Terminal t=req.conn.get(0).strip.get(i);
        re=check(t,q,r);
        if (re==1)
        {
            Calendar cal = Calendar.getInstance();

```

```

        cal.getTime();
        SimpleDateFormat sdf = new SimpleDateFormat("HH:mm:ss");

        System.out.println("");
        t.wait=1;
        thread o=new thread(t);

        System.out.println("plane id "+r.id+ " landed on" + " "+ t.name);

        t.al_terminals++;
        re= 1;
        r.s=t;
        r.sched=1;
        // cal.add(Calendar.MINUTE, +2);
        r.time = sdf.format(cal.getTime()) ;
        System.out.println(r.time );

        cal.add(Calendar.HOUR, +10);
        r.out_time=sdf.format(cal.getTime()) ;
        q.submit_allocate(r);
        q.submit_to(r);
        break;
    }

}

// while(!q.queue1.isEmpty())
//System.out.println("plane"+r.id);
//System.out.println("plane can't land on this airport due to its weight
requirments");

}

public int check(Terminal t,req q,requirement r)
{
    int slenght1=t.striplenght;

```



```

        int sweight1=t.weight;
        int re=0;

        //System.out.println( sdf.format(cal.getTime()) );
        if(t.wait==0)
        {
            if (r.lenght<=t.striplenght && r.weight<=t.weight)
            {
                System.out.println("trying plane "+r.id+" to accomodated on "+t.name);

                if(t.alert==1 )
                {
                    System.out.println("plane "+r.id+" can't be accommodated on "+t.name+
"due to emergency");
                    // q.submit(r);
                    re=0;
                }
                else if(t.al_terminals==t.terminal )
                {
                    System.out.println("plane "+r.id+" can't be accommodated on "+t.name+
"due to unavailability of space");
                    // q.submit(r);
                    re=0;
                }

            //
            //         else if(t.emg==r.flag)
            //         {
            //             // t.al_terminals=t.terminal;
            //             System.out.println("plane id "+r.id+ " landed on" +" "+ t.name);
            //             System.out.println(t.name+" blocked");
            //
            //             t.alert=1;
            //             r.s=t;
            //
            //

```

```

//          q.submit_allocate(r);
//          re=1;
//      }
      else
      {
          re= 1;
      }
    }else
    {
        System.out.println(r.id+" cannot be accomodated on "+t.name+" due to
parameters\n");
    }

}
else
{
    System.out.println(r.id+" cannot be accomodated on "+t.name+" busy\n");
}

    return re;
}

```

```

public void finalise (req q) throws ClassNotFoundException, SQLException
{String s=" ";

```

```

System.out.println("plane"+s+"terminal"+s+"time"+s+"strip_lenght_req"+s+"weight");
    requirement r= new requirement();
    Class.forName("org.apache.derby.jdbc.ClientDriver");
    // Class.forName("org.apache.derby.jdbc.ClientDriver");
    Connection con =
DriverManager.getConnection("jdbc:derby://localhost:1527/db", "vibhor", "vibhor");
    java.sql.Statement st=con.createStatement();
    while(true)

```

```

        {
            if(!req.queue1.isEmpty())
            {

                r=req.queue1.poll();
                System.out.println(" "+r.id+s+r.s.name+s+r.time+s+r.lenght+s+s+r.weight);
                String          d="insert          into          VIBHOR.UNTITLED7
VALUES("+r.id+"",""+r.s.name+"",""+r.time+"",""+r.lenght+"",""+r.weight+"");

                int rs=st.executeUpdate(d);
            }
            if(!req.dep.isEmpty())
            {
                r=req.dep.poll();
                System.out.println("
"+r.id+s+r.s.name+s+r.out_time+s+r.lenght+s+s+r.weight);
                String          d="insert          into          VIBHOR.UNTITLED8
VALUES("+r.id+"",""+r.s.name+"",""+r.out_time+"",""+r.lenght+"",""+r.weight+"");
                int rs=st.executeUpdate(d);
            }
            else
            {
                break;
            }

        }

        output o=new output();
        o.setVisible(true);
    };

    public void departure ()
    {
        int re;
        requirement r1;
        for(int j= 0 ; j<req.to.size();j++)

```

```

{r1=req.to.poll();

try {
    ///
    Thread.sleep(2000);
} catch (InterruptedException ex) {
    Logger.getLogger(schedule.class.getName()).log(Level.SEVERE, null,
ex);
}

for(int i=0;i<req.conn.get(0).strip.size();i++)
{
Terminal t=req.conn.get(0).strip.get(i);
re=check_dep(t,r1);
    if (re==1)
    {break;
    }
}
}
}

public int check_dep(Terminal t,requirement r)
{ Calendar cal = Calendar.getInstance();
req q=new req();
    cal.getTime();
    SimpleDateFormat sdf = new SimpleDateFormat("HH:mm:ss");
int slenght1=t.striplenght;
int sweight1=t.weight;
int re=0;
if (r.lenght<=t.striplenght && r.weight<=t.weight)

{
    // if(t.al_terminals==t.terminal)

```

```

// {
// System.out.println("plane "+r.id+" can't be departed on "+t.name);
//q.submit(r);
// re=0;

System.out.println("");
System.out.println("plane id "+r.id+ " departed on" +" "+ t.name);
t.al_terminals--;
re= 1;
r.s=t;
cal.add(Calendar.HOUR, +10);
r.out_time = sdf.format(cal.getTime()) ;
q.submit_dep(r);
}

return re;
}

}

```

Req.java

```
package javaapplication2;
```

```
/*
```

* To change this template, choose Tools | Templates

* and open the template in the editor.

*/

```
import javaapplication2.requirement;
```

```
import javaapplication2.resource;
```

```
import java.util.ArrayList;
```

```
import java.util.LinkedList;
```

```
import java.util.List;
```

```
import java.util.Queue;
```

```
public class req {
```

```
    static Queue<requirement> queue = new LinkedList<requirement>();
```

```
    static Queue<requirement> queue1 = new LinkedList<requirement>();
```

```
    static List<resource> conn=new ArrayList<resource>();
```

```
    static Queue<requirement> to = new LinkedList<requirement>();
```

```
    static Queue<requirement> dep = new LinkedList<requirement>();
```

```
    public void submit( requirement e)
```

```
{System.out.print(e.id+"\n");
```

```
queue.add(e);
```

```
}
```

```
    public void submit( resource e)
    {
        System.out.print(e.name+"\n");
        conn.add(e);
    }
```

```
    public void submit_allocate( requirement f)
    {
        queue1.add(f);
    }
```

```
    public void submit_dep( requirement r1)
    {
        dep.add(r1);
    }
```

```
    public void submit_to( requirement r1)
    {
        to.add(r1);
    }
```

```
}
```

Requirement.java

```
public class requirement {  
    String type;  
    int arrival_time;  
    int depart_time;  
    int delay;  
    int weight;  
    int lenght;  
    Terminal s;  
    int id;  
    String time;  
    String out_time;  
    int flag=0;  
    int sched=0;  
}
```

Resource.java

```
public class resource {  
    String name;  
    int capacity;  
    List<Terminal> strip=new ArrayList<Terminal>();  
}
```

Waiting.java

```
package javaapplication2;
```

```
import java.util.logging.Level;
```



```

import java.util.logging.Logger;

/**
 *
 * @author Vibhor
 */
public class waiting extends Thread {
    private requirement r1;
    private req q;
    public waiting (requirement r)
    {this.r1 = r;
    this.start();
    }

    @Override
    public void run()
    {
        System.out.println("is processing");
        // Thread.sleep(1);

        System.out.println("plane ready for take off");
        q.submit_to(r1);
    }
}

```

Terminal.java

```

public class Terminal {
    String name;
    int striplenght;
    int weight ;
    int terminal;
    int al_terminals=0;
    int emg=1;
}

```

```
int alert=0;
static int wait=0;
}
```

Thread.java

```
public class thread implements Runnable{

Terminal t1;
public thread ( Terminal t)
{t.wait=1;
t1=t;
Thread thread = new Thread(this);
thread.start();
t.wait=0;
}
public void run()
{

System.out.println(t1.name + "under process");

try
{
Thread.sleep(5000);
System.out.println("This is completed");
t1.wait=0;
}
catch (InterruptedException e)
{
Thread.currentThread().interrupt();
}

}
```

```
}
```

Currenttime.java

```
import java.text.SimpleDateFormat;
```

```
import java.util.Calendar;
```

```
public class currentTime {
```

```
    public static void main(String[] args) {
```

```
        Calendar cal = Calendar.getInstance();
```

```
        cal.getTime();
```

```
        SimpleDateFormat sdf = new SimpleDateFormat("HH:mm:ss");
```

```
        System.out.println( sdf.format(cal.getTime()) );
```

```
    }
```

```
}
```

Chapter-V

SIMULATION WORKFLOW

- **An Request Is Generated By Aircraft Stating Various Requirements For Landing**
 - **Priority Queue**
 - **FCFS**
- **Scheduling Module Is Called**
 - **Parameter Check**
 - **Strip Allotment**
- **A Check For Any Emergency Constraint**
 - **Block Terminal**
 - **Put Flights On Hold For That Terminal**
- STEP 1
- Flights that cannot land except the blocked terminal , are put on hold or diverted to nearby airport
- STEP 2
- Other terminals will still be functioning
- STEP 3
- Now ,the finalise function is called , entry of log into database
- STEP 4
- Retrieval of value from database and put into another priority queue
- STEP 5
- This queue is for the flights waiting for departure and who has sent the departure request.
- STEP 6
- Check function is also used , in order to allot strip for take off
- STEP 7

- It checks if the strip is free ,if not then assign to other free strip of higher capacity
- STEP 8
- Finally as the plane takes off it is entered into another queue called take off
- STEP 9
- An automated output is generated.

SIMULATION WORKFLOW

Screenshot 1 – opening of project



Screenshot 2 – requests for takeoff and landing

Air Traffic Management System

Requests Resources Simulation

Request's For Landing

ADD


Type

Flight No.	Arrival Time	Type	Emergency	Strip Length Req.	Strip Weight Req.	Status
1	1	1	0	12000	344	1
13000	1	1	0	11000	400	1
3	2	2	0	10000	400	2
4	32	2	0	10000	350	2
6	2	2	0	11000	600	2

Request's For Take off

Add

Flight No.	Departure Time	Type	Emergency	Strip Length Req.	Strip Weight Req.	Status
1	1	2	0	12000	345	1
3	3	3	0	13000	333	3
4	4	4	0	14000	450	3
12	2	2	0	10000	234	2
14	5	4	0	14000	600	5



Windows taskbar: 13:04 02-05-2014

Screenshot 3 – landing request generator

Aircraft Landing Request

Landing Request

Flight No.

Arrival Time


Type (1,2,3)

Status (L,O,B)

Emergency (Y,N)

Strip Length Req.

Strip Weight Req.



Windows taskbar: 13:05 02-05-2014

Screenshot 4 – take-off request generator

Aircraft Takeoff Request

Take Off Request

Flight No.

Type (1,2,3)


Departure time

Emergency (Y,N)

Strip Length Req.

Strip weight Req.

Status (L,O,B)



The screenshot shows a Windows desktop environment. The taskbar at the bottom includes icons for Windows, Internet Explorer, Firefox, Google Chrome, and several other applications. The system tray on the right shows the time as 13:06 and the date as 02-05-2014.

Screenshot 5 – resources available at airport

Air Traffic Management System

Requests Resources Simulation

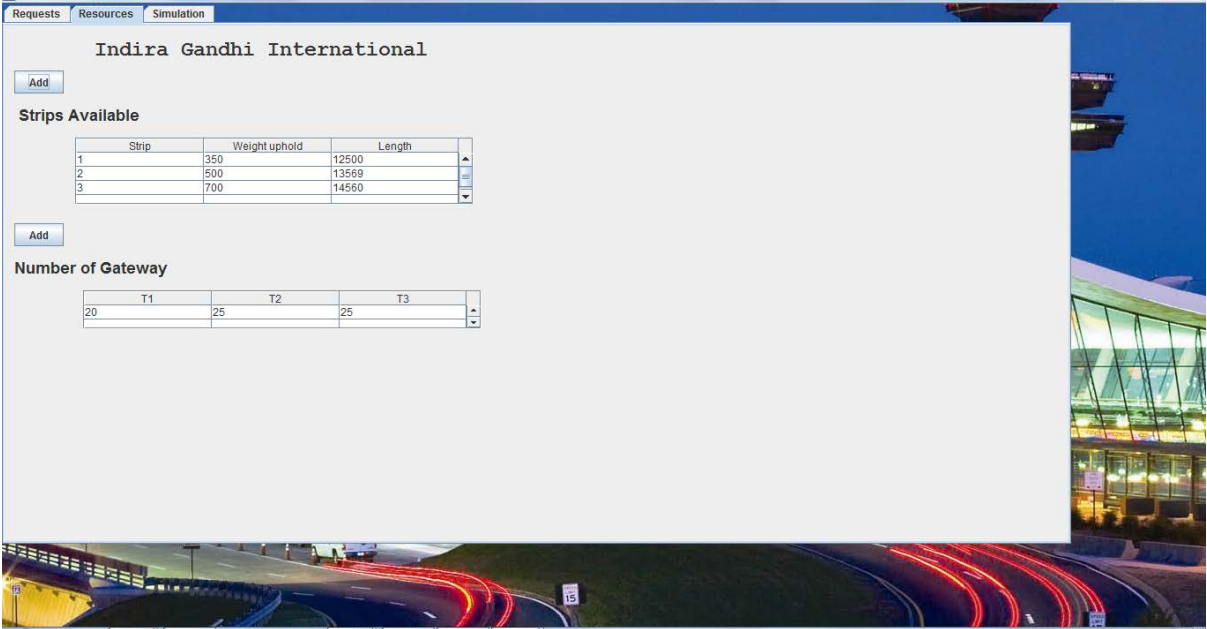
Indira Gandhi International

Strips Available

Strip	Weight uphold	Length
1	350	12500
2	500	13569
3	700	14560

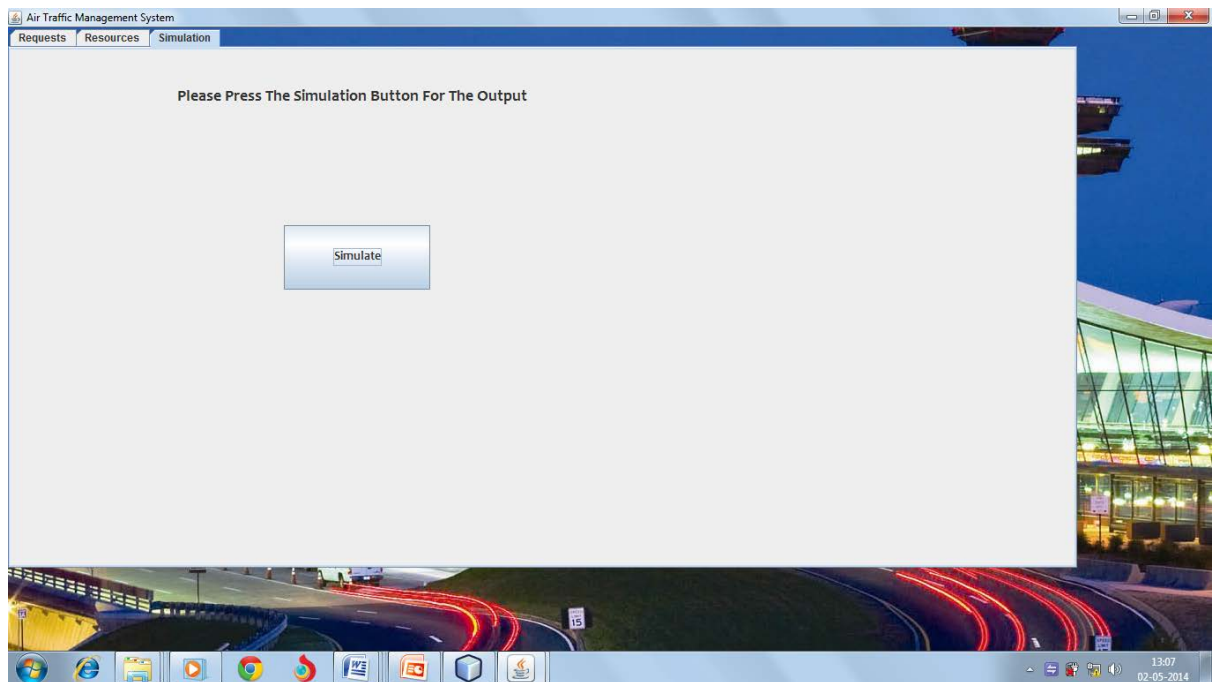
Number of Gateway

T1	T2	T3
20	25	25



The screenshot shows a Windows desktop environment. The taskbar at the bottom includes icons for Windows, Internet Explorer, Firefox, Google Chrome, and several other applications. The system tray on the right shows the time as 13:07 and the date as 02-05-2014.

Screenshot 6 – simulation button , simulation starts



Screenshot 7 – final output

The screenshot shows the 'Aircraft Landing Takeoff Request Output' window. It contains two tables. The first table, 'Aircraft's Landed', lists aircraft that have landed with columns for Pid, Terminal, Time, Slr, and Swr. The second table, 'Aircraft's Take-off's', lists aircraft that have taken off with columns for Pid, Strip, Time, Slr, and Swr.

Pid	Terminal	Time	Slr	Swr
123	terminal1	13:20:37	12000	334
11	terminal2	13:20:39	10000	450
13000	terminal2	13:20:41	11000	400
1	terminal1	13:20:43	12000	344
567	terminal1	13:20:45	12000	345
234	terminal2	13:20:47	13500	455
55	terminal3	13:20:49	14000	450
22	terminal1	13:20:51	11000	345
69	terminal3	13:20:53	14000	450
14	terminal3	13:20:55	14000	400
3	terminal2	13:20:57	10000	400
78	terminal2	13:20:59	12400	444
89	terminal3	13:21:01	14500	444
33	terminal2	13:21:03	12400	450
6	terminal2	13:21:05	11000	500

Pid	Strip	Time	Slr	Swr
123	terminal1	23:21:13	12000	334
11	terminal2	23:21:15	10000	450
13000	terminal2	23:21:17	11000	400
1	terminal1	23:21:19	12000	344
567	terminal1	23:21:21	12000	345
234	terminal2	23:21:23	13500	455
55	terminal3	23:21:25	14000	450
22	terminal1	23:21:27	11000	345
69	terminal3	23:21:29	14000	450
14	terminal3	23:21:31	14000	400
3	terminal2	23:21:33	10000	400
78	terminal2	23:21:35	12400	444
89	terminal3	23:21:37	14500	444
33	terminal2	23:21:39	12400	450
6	terminal2	23:21:41	11000	500
4	terminal1	23:21:43	10000	350
345	terminal3	23:21:45	14500	560

Chapter-VI

Testing outcomes

→IGI Airport New Delhi

We have tried to test our algorithm with a data of 100 aircrafts from the real time based data of IGI AIRPORT DELHI , and calculated the amount of delays and how our algorithm is performing when given a real scenario.

We are considering 3 runways from IGI AIRPORT. With specifications



Figure -5

→Runway Statistics

Runway No.	Direction	Length
1	10/28	12,000ft
2	9/27	9,229ft
3	11/29	14,534ft

Table 2

→Terminal 1 – domestic

Terminal 1 facilities are currently used by GoAir, IndiGo, SpiceJet and are split into two areas, Terminal 1D for departures and Terminal 1C for arrivals.

→Terminal 2

Terminal 2 will work in tandem with T3, until the proposed T4 terminal is built, upon which it will be demolished as per the proposed master plan.

→Terminal 3 – domestic and international

It occupies an area of 502,000 m² (5,400,000 sq ft), with a capacity to handle 34 million passengers annually.

→Capacity

Terminal	NO. of Flight
T-1	20
T-2	25
T-3	25

Table 3

→Beijing Capital International Airport

Runway Statistics

Runway No.	Direction	Length
1	18L/36R	12,500ft
2	18R/36L	11,300ft
3	01/19	12,500ft

Table 4

→Terminal 1

Terminal 1 reopened for a second time on June 27, 2008, and became the operational base for all domestic flights operated by the HNA Group

→Terminal 2

Domestic and international flights.

→Terminal 3

For International Flights Only.

Outcome –

Anytime a plane arrives to the airport it sends a landing request to the algorithm, well before the time of its arrival, so as we all know that at a particular time there could be many requests arriving to the algorithm, so we have distinguished these requests on the basis of time of arrival on nano seconds difference so at a particular time only one request is coming at the algorithm, we have then put these requests in the queue using FIRST COME FIRST SERVE basis,

So our algorithm is working on multithreading, so whenever a planes requests its arrival onto the airport it checks for all the possible landing option for that airplane, and if the runway is busy, it then shifts its landing to other runway which is free or having lesser queue, so in this way we can reduce the delay time, and increase the efficiency for our algorithm.

But in case of an emergency , whenever a plane is having an emergency landing , then till the plane gets landed properly and removed from the runway till then we shuts down that runway for other flights, and try to accommodate them on other runways or if not then we diverge them to nearby airport to reduce the amount of delay.

We a=have designed this algorithm also to reduce the amount of human intervention, all this work is now done by humans which raise an human error also , so in order to increase the safety and decrease the delays and flight cancelation we have created this algorithm , because in future the amount of air traffic is only going to be increased.

TESTING 1 –

We tested our algorithm on an dataset of 20 aircrafts.

Dataset →

Flight No.	Arrival time	Type	Departure Time	Emergency Cons.	Strip length Req.	Strip Weight Req.
1	1	1	1	0	10000	234
2	2	2	2	0	11000	345
3	2	2	2	0	11500	453
4	2	2	2	0	12000	450
5	3	3	3	0	12000	564
123	1	1	1	0	12000	323
124	2	2	2	0	13000	432
125	2	2	2	0	14000	444
126	1	1	1	0	13000	345
127	1	1	1	0	11000	334
6	1	1	1	0	11500	345
7	1	11	1	0	12000	453
8	1	1	1	0	12900	345
9	1	1	1	0	12500	345
10	1	1	1	0	13000	344
234	2	2	2	0	14000	232
235	2	2	2	0	12300	432
236	2	2	2	0	11000	342
237	1	1	1	0	9000	232
238	2	2	2	0	11323	342

The algorithm worked fine and landed all the aircrafts in very less time, with very less delays.

→The output window we get is

Aircraft Landing Takeoff Request Output

Aircraft's Landed

Pid	Terminal	Time	Str	Swr
237	terminal1	12:22:38	9000	232
10	terminal2	12:22:38	13000	344
9	terminal1	12:22:40	12500	345
8	terminal2	12:22:42	12900	345
7	terminal2	12:22:44	12000	453
6	terminal1	12:22:46	11500	345
127	terminal1	12:22:48	11000	334
126	terminal2	12:22:50	13000	345
123	terminal1	12:22:52	12000	323
1	terminal1	12:22:54	10000	234
238	terminal1	12:22:56	11323	342
236	terminal1	12:22:58	11000	342
235	terminal2	12:23:00	12300	432
234	terminal3	12:23:02	14000	232
125	terminal3	12:23:04	14000	444

Aircraft's Take-off's

Pid	Strip	Time	Str	Swr
237	terminal1	22:23:18	9000	232
10	terminal2	22:23:20	13000	344
9	terminal1	22:23:22	12500	345
8	terminal2	22:23:24	12900	345
7	terminal2	22:23:26	12000	453
6	terminal1	22:23:28	11500	345
127	terminal1	22:23:30	11000	334
126	terminal2	22:23:32	13000	345
123	terminal1	22:23:34	12000	323
1	terminal1	22:23:36	10000	234
238	terminal1	22:23:38	11323	342
236	terminal1	22:23:40	11000	342
235	terminal2	22:23:42	12300	432
234	terminal3	22:23:44	14000	232
125	terminal3	22:23:46	14000	444
124	terminal2	22:23:48	13000	432
4	terminal2	22:23:50	12000	450
3	terminal2	22:23:52	11500	453

Output-1

TESTING 2 – now we tested our algorithm on a dataset of 55 aircrafts.

Dataset→

Flight No.	Arrival time	Type	Departure Time	Emergency Cons.	Strip length Req.	Strip Weight Req.
1	1	1	1	0	10000	234
2	2	2	2	0	11000	345
3	2	2	2	0	11500	453
4	2	2	2	0	12000	450
5	3	3	3	0	12000	564
123	1	1	1	0	12000	323
124	2	2	2	0	13000	432
125	2	2	2	0	14000	444
126	1	1	1	0	13000	345
127	1	1	1	0	11000	334
6	1	1	1	0	11500	345
7	1	11	1	0	12000	453
8	1	1	1	0	12900	345
9	1	1	1	0	12500	345
10	1	1	1	0	13000	344

234	2	2	2	0	14000	232
235	2	2	2	0	12300	432
236	2	2	2	0	11000	342
237	1	1	1	0	9000	232
238	2	2	2	0	11323	342
11	1	1	1	0	12000	333
12	1	1	1	0	13000	432
13	1	1	1	0	11000	222
14	1	1	1	0	14000	332
15	1	1	1	0	9000	123
345	1	1	1	0	12345	123
346	1	1	1	0	8900	223
347	1	1	1	0	7890	232
348	1	1	1	0	13453	343
349	1	1	1	0	10000	343
11111	1	1	1	0	11111	223
1122	1	1	1	0	12000	333
16	1	1	1	0	12300	221
17	1	1	1	0	9000	223
18	1	1	1	0	10000	212
19	1	1	1	0	13000	332
20	2	2	1	0	11500	432
21	1	1	1	0	10500	123
22	2	2	2	0	9000	232
23	2	2	2	0	9400	332
24	1	1	1	0	12000	334
25	2	2	2	0	7890	221
567	1	1	1	0	12345	112
568	2	2	2	0	12322	224
569	1	1	1	0	12400	234
678	1	1	1	0	12000	443
679	1	1	1	0	12390	332
666	2	2	2	0	10900	221
667	1	1	1	0	8900	232
668	1	1	1	0	12000	345
669	1	1	1	0	13000	224
778	1	1	1	0	11000	342
779	1	1	1	0	14001	211

The output generated was →

Aircraft's Landed

Pid	Terminal	Time	Str	Swr
568	terminal1	12:45:19	12322	224
25	terminal1	12:45:21	7890	221
23	terminal1	12:45:23	9400	332
22	terminal1	12:45:25	9000	232
20	terminal2	12:45:27	11500	432
238	terminal1	12:45:29	11323	342
236	terminal1	12:45:31	11000	342
235	terminal2	12:45:33	12300	432
234	terminal3	12:45:35	14000	232
125	terminal3	12:45:37	14000	444
124	terminal2	12:45:39	13000	432
4	terminal2	12:45:41	12000	450
3	terminal2	12:45:43	11500	453
2	terminal1	12:45:45	11000	345
0	terminal3	12:45:47	12000	564

Aircraft's Take-off's

Pid	Strip	Time	Str	Swr
123	terminal1	22:47:05	12000	323
1	terminal1	22:47:07	10000	234
568	terminal1	22:47:09	10900	221
23	terminal1	22:47:11	12322	224
25	terminal1	22:47:13	7890	221
23	terminal1	22:47:15	9400	332
22	terminal1	22:47:17	9000	232
20	terminal2	22:47:19	11500	432
238	terminal1	22:47:21	11323	342
236	terminal1	22:47:23	11000	342
235	terminal2	22:47:25	12300	432
234	terminal3	22:47:27	14000	232
125	terminal3	22:47:29	14000	444
124	terminal2	22:47:31	13000	432
4	terminal2	22:47:33	12000	450
3	terminal2	22:47:35	11500	453
2	terminal1	22:47:37	11000	345
0	terminal3	22:47:39	12000	564

Output -1

In this we can see all 55 aircrafts were landed onto the terminal .

```

terminalUnder process
This is completed
234 cannot be accommodated on terminal1 due to parameters
234 cannot be accommodated on terminal2 due to parameters
trying plane 234 to accommodated on terminal3
plane id 234 landed on terminal3
12:45:35
terminalUnder process
This is completed
125 cannot be accommodated on terminal1 due to parameters
125 cannot be accommodated on terminal2 due to parameters
trying plane 125 to accommodated on terminal3
plane id 125 landed on terminal3
12:45:37
terminalUnder process
This is completed
124 cannot be accommodated on terminal1 due to parameters
trying plane 124 to accommodated on terminal2
plane 124 can't be accommodated on terminal2 due to unavailability of space
trying plane 124 to accommodated on terminal3
plane id 124 landed on terminal3
12:45:39
terminalUnder process
This is completed
4 cannot be accommodated on terminal1 due to parameters
trying plane 4 to accommodated on terminal2
plane 4 can't be accommodated on terminal2 due to unavailability of space
trying plane 4 to accommodated on terminal3

```

Output -2

In this manner our algorithm works , while landing 55 aircrafts.

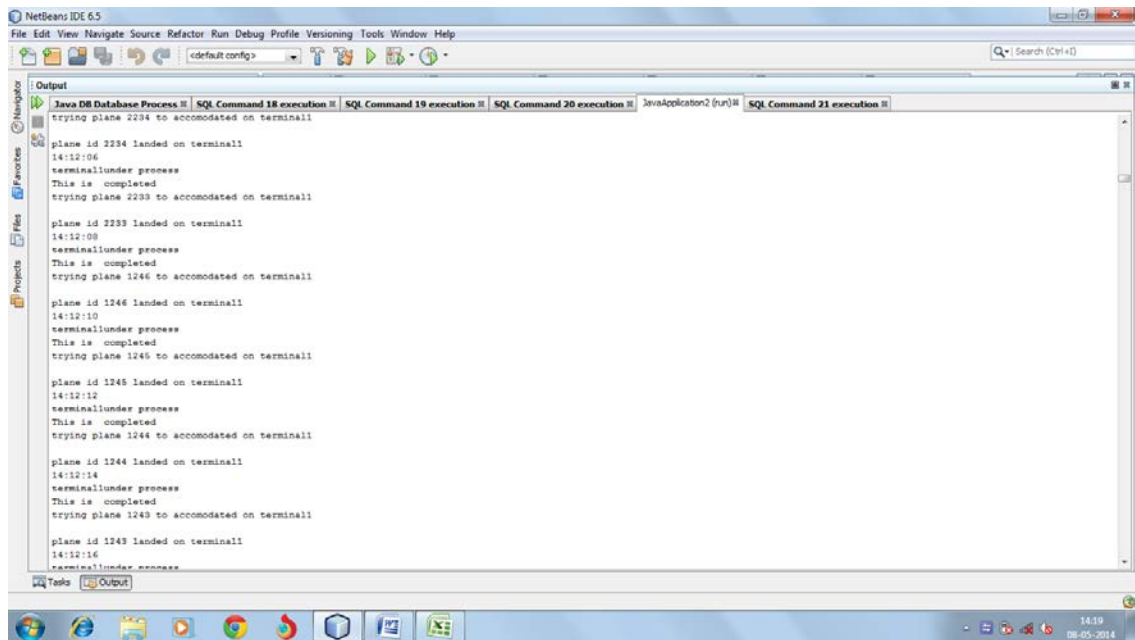
TESTING 3 – Here we try to accommodate 80 aircrafts on the Airport.

Dataset →

Flight No.	Arrival time	Type	Departure Time	Emergency Cons.	Strip length Req.	Strip Weight Req.
1	1	1	1	0	10000	234
2	2	2	2	0	11000	345
3	2	2	2	0	11500	453
4	2	2	2	0	12000	450
5	3	3	3	0	12000	564
123	1	1	1	0	12000	323
124	2	2	2	0	13000	432
125	2	2	2	0	14000	444
126	1	1	1	0	13000	345
127	1	1	1	0	11000	334
6	1	1	1	0	11500	345
7	1	11	1	0	12000	453
8	1	1	1	0	12900	345
9	1	1	1	0	12500	345
10	1	1	1	0	13000	344
234	2	2	2	0	14000	232
235	2	2	2	0	12300	432
236	2	2	2	0	11000	342
237	1	1	1	0	9000	232
238	2	2	2	0	11323	342
11	1	1	1	0	12000	333
12	1	1	1	0	13000	432
13	1	1	1	0	11000	222
14	1	1	1	0	14000	332
15	1	1	1	0	9000	123
345	1	1	1	0	12345	123
346	1	1	1	0	8900	223
347	1	1	1	0	7890	232
348	1	1	1	0	13453	343
349	1	1	1	0	10000	343
11111	1	1	1	0	11111	223
1122	1	1	1	0	12000	333
16	1	1	1	0	12300	221
17	1	1	1	0	9000	223
18	1	1	1	0	10000	212
19	1	1	1	0	13000	332
20	2	2	1	0	11500	432
21	1	1	1	0	10500	123
22	2	2	2	0	9000	232
23	2	2	2	0	9400	332
24	1	1	1	0	12000	334
25	2	2	2	0	7890	221

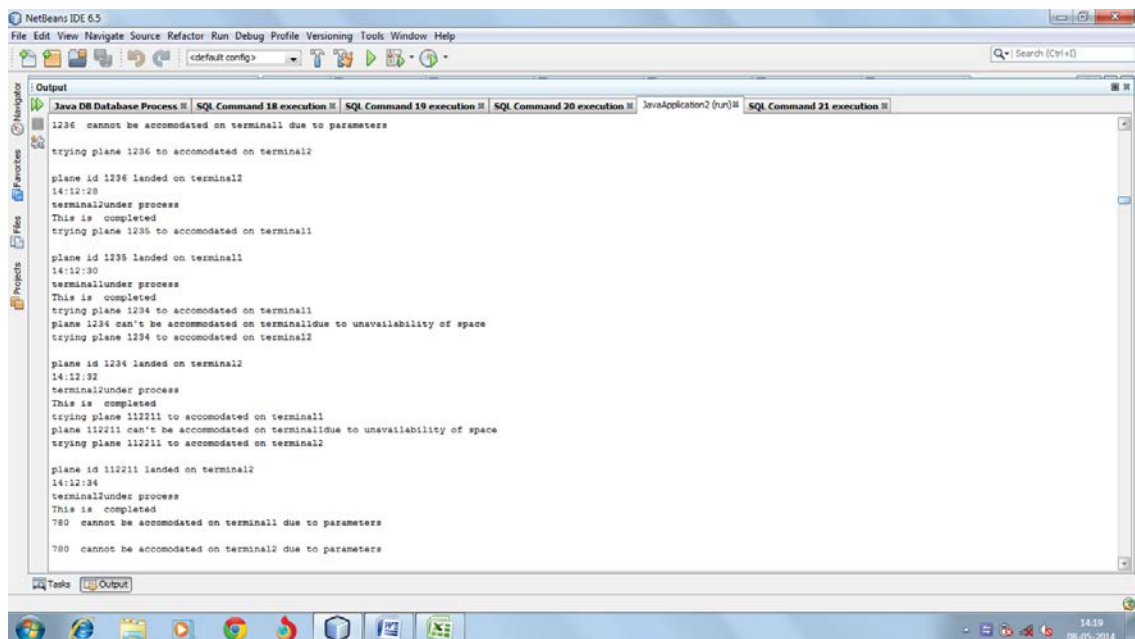
567	1	1	1	0	12345	112
568	2	2	2	0	12322	224
569	1	1	1	0	12400	234
678	1	1	1	0	12000	443
679	1	1	1	0	12390	332
666	2	2	2	0	10900	221
667	1	1	1	0	8900	232
668	1	1	1	0	12000	345
669	1	1	1	0	13000	224
778	1	1	1	0	11000	342
779	1	1	1	0	14001	211
780	1	1	1	0	14002	223
112211	1	1	1	0	10000	224
1234	1	1	1	0	10101	231
1235	1	1	1	0	12002	332
1236	1	1	1	0	10000	443
1237	1	1	1	0	9000	331
1238	1	1	1	0	8900	231
1240	1	1	1	0	10900	392
1241	1	1	1	0	9999	123
1242	1	1	1	0	13999	339
1243	1	1	1	0	12000	331
1244	1	1	1	0	9922	122
1245	1	1	1	0	8000	133
1246	1	1	1	0	9898	299
2233	1	1	1	0	12300	224
2234	1	1	1	0	8909	332
2235	1	1	1	0	12000	233
2236	1	1	1	0	10000	332
2237	1	1	1	0	7890	122
1011	1	1	1	0	12000	220
111	1	1	1	0	12000	332
232	1	1	1	0	11000	229
1010	1	1	1	0	10000	331
11001	1	1	1	0	10200	221
899	1	1	1	0	12900	221
8877	1	1	1	0	8900	221
9999	1	1	1	0	11111	111

The output generated showed that the capacity of the Airport was only 70 aircrafts,as soon as the terminal fills up, the flights are diverted to other terminals and as all the terminals get filled , the flights were put on hold.



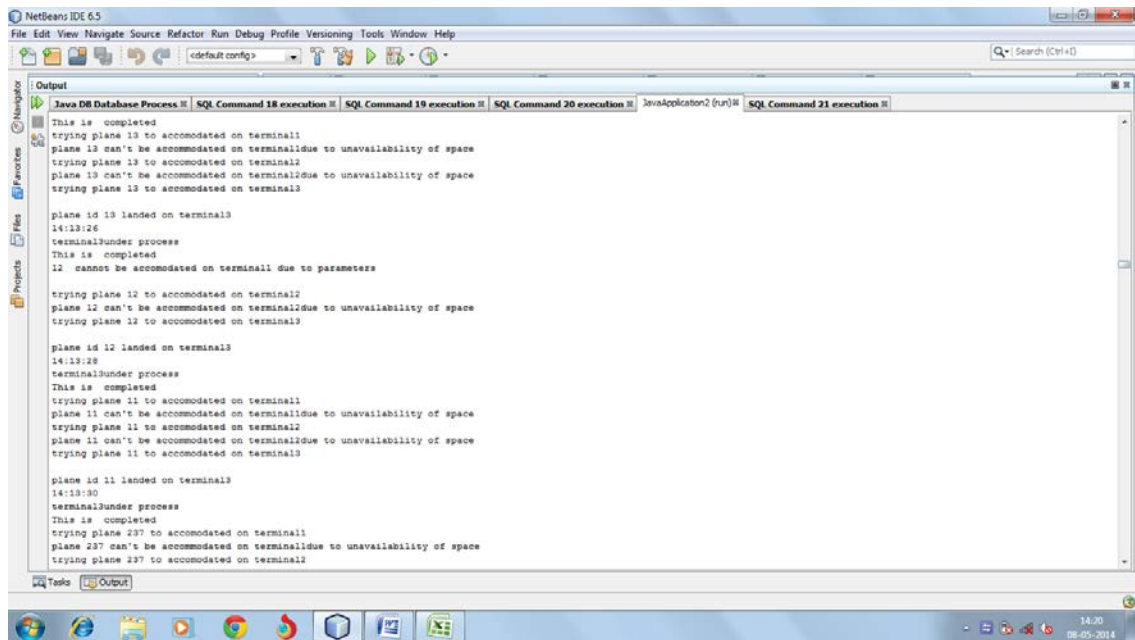
Output – 1

Flights are landing onto the Airport



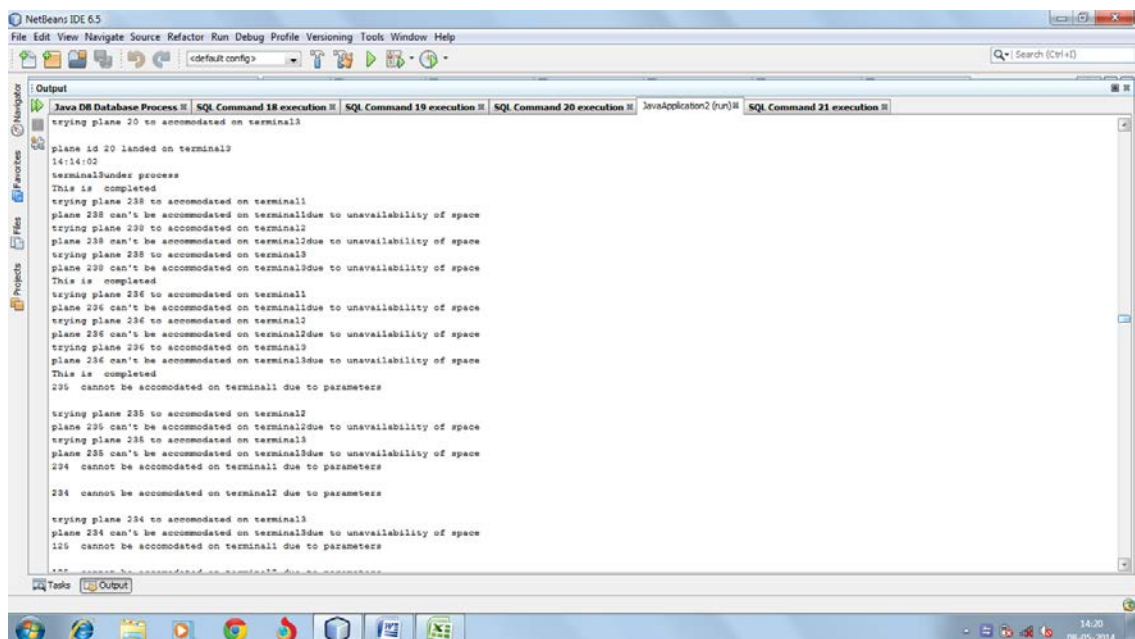
Output – 2

Terminal 1 gets full , trying to accommodate on other Terminals



Output – 3

Other Terminals landing flights on them.



Output – 4

All the Terminals get filled up and no more planes can't be accommodated onto the Airport.

Future work and conclusion

As far as we know that this system is based on continuous betterment, and this field itself is very huge, and not a project of a semester, so we would like to move it further by doing the various testing on our algorithm to check its feasibility, whether is useful or not, so a lot of testing is required, and yes we can say that this project doesn't ends here, so a lot of testing has to be done and comparing it with other ongoing algorithms. I will try my best to do it.

Tools and Technique Used :

NETBEANS:- it is an application platform framework for Java desktop applications and others. The NetBeans IDE is written in Java and can run on Windows, OS X, Linux, Solaris and other platforms supporting a compatible JVM. it is used for simulation purpose of our algorithm with dummy data and provide us with generic classes.

Derby Database: is a relational database management system (RDBMS) developed by the Apache Software Foundation that can be embedded in [Java](#) programs and used for online transaction processing. It has a 2.6 MB disk-space footprint. for pushing and popping our data from the database.

CHAPTER VII

References

[1] Yong Wang, Feng Wang, Dan Wang, Zhenghu Gong and Jiangchuan Liu “Revisit the Fairness Issues in Flight Landing Scheduling” 15th International IEEE Conference on Intelligent Transportation Systems, Anchorage, Alaska, USA, September 16-19, 2012

[2] Jianli Ding, Jinling Ji, Heyi “Wang Optimization of Airport Flight Arrival and Departure Based on Compromise Immune Algorithm” *IEEE Transactions on Control Systems* 2007.

[3] Andrea D’Ariano, Paolo D’Urgolo, Dario Pacciarelli and Marco Pranzo” Optimal sequencing of aircrafts take-off and landing at a busy airport” 13th International IEEE Annual Conference on Intelligent Transportation Systems, Madeira Island, Portugal, September 19-22, 2010

[4] Shiwei Zhao ,Guo Wei , Huijuan Feng “Study on Optimizing Control of Flight Arrival Scheduling Strategy” International symposium on instrumentation & measurement , sensor networks and automation(IMSNA),2012

→Web-references

<http://www.newdelhiaairport.in/>

http://en.wikipedia.org/wiki/Indira_Gandhi_International_Airport