Constraint Satisfaction Problem (CSP) Based Implementation of Scheduling Aircraft at Runway

Ranjan Kumar Thakur[1], Ram Baksh[2], Arabind Kumar[3], Aditya Pratap Singh[4]

[1]Delhi University, Department of Mathematics, Deshbandhu College, Kalkaji, New Delhi - 110019

[2]Delhi University, Department of Mathematics, Dyal Singh College, Lodhi Road, New Delhi - 110003

[3]Delhi University, Department of Mathematics, Kamala Nehru College, August Kranti Marg, New Delhi - 110049

[4]Delhi University, Department of Mathematics, PGDAV College (Evening), Nehru Nagar, New Delhi - 110065

Abstract: ***The aim of this Paper is to implement a Constraint Satisfaction Problem (CSP) based solution for scheduling departure sequence of Aircraft at runways. Airports are getting more and more congested as they are operating a large number of Aircrafts at limited number of available runways. This is one of the most constraining factors encountered recently. A number of approaches are used to tackle this situation at major Airports. A possibility to alleviate this congestion is to assist controllers in the planning and scheduling process of aircraft. The prototype presented in this paper is aimed to offer such assistance in the establishment of an optimal departure schedule and the planning of initial climb phases for departing aircraft. In this paper, first the operational problem of departure management is addressed by describing briefly current practice and identifying the role of departure planning at airports. Second, a mapping of the departure management problem to constraint satisfaction is described. Third, the prototype is described in detail and an example solution is presented. Finally, some conclusions are drawn. C++ and Linux are used as an implementation environment.***

Keywords: CSP, Scheduling, Runway, Aircraft

1. Introduction

Controllers in airport control towers are responsible for the overall management of surface traffic at the airport. This is a difficult process: even under normal operating conditions at least three different controllers (one for each of the 'pre-flight', 'taxiways' and 'runways' areas) manage the aircraft on the airport. Each controller will try to establish an optimal plan for his/her own area and will try to provide the aircraft to the next controller in an efficient way. Departure management at the runways is the responsibility of the runway controller. The prototype featured in this paper intends to assist the runway controller in establishing optimal departure sequences, taking the plans of other controllers into account where necessary. The runway controller is the last planner in line and is dependent on the sequence of aircraft that is handed over by the previous (taxiway) controller. At the runway, typically only minor changes to the provided sequence can be made through the use of runway holdings and intersection take-offs. The current way of working leads to a sub-optimal use of the available runway capacity, since the provided departure sequences are for the largest part fixed. A way out to this problem follows from the recognition that the *runways* are the scarcer resource at airports. We will assume, therefore, that the runway controller (assisted by the prototype) determines the sequence of aircraft to obtain an optimal use of the runway capacity at the airport.

The departure management (runway scheduling) task entails the establishment of an optimal sequence of departing aircraft (the schedule) and the assignment of departure plans to these aircraft. Departure plans consist of start-up times at the gates, taxi plans for taxiing to the runways, and runway plans for take-off. The focus here is on the establishment of runway plans - start-up times and taxi plans can be derived from these plans. Runway plans specify which aircraft should use which runway for take-off, and at what time. Important for the establishment of runway plans is the so-called wake vortex separation, restricting departing aircraft at the same runway because of preceding aircraft that may be too close. Another relevant issue concerns the timeslot assigned to each aircraft. At most European airports, timeslots are co-ordinated time intervals of about 15 minutes in which aircraft should take off. Co-ordination is done with the CFMU (Central Flow Management Unit) in before the flight starts; the CFMU planning aims at obtaining a constant traffic flow through all sectors. For the airport controllers, this CFMU restriction ensures that the sectors are not overloaded by the feeders - the points where controllers hand over the flights to the next one.

1.2 Problem Statement

For the runway scheduling problem, one wants to find an efficient schedule for aircraft to use the runway. The output, then, to the runway scheduling problem is a time for each aircraft to begin using the runway, which will be referred to as a departure management (i.e. runway scheduling).

The runway scheduling problem is formally described as follows:
Airports can be said to provide a variety of resources used by all departing, arriving, and ground traffic. For the departure management problem at hand, the existence of runways, Standard Instrument Departure (SID) routes and Terminal Maneuvering Area (TMA) exit points are of specific importance. Runways connect to SID routes, which specify the route aircraft can take in airspace above the airport. SID routes lead to TMA exit points, marking the

boundaries of the airspace around the airport (see: [1], [4]). Figure 1.1 below schematically depicts part of the topology of an example airport: Prague.

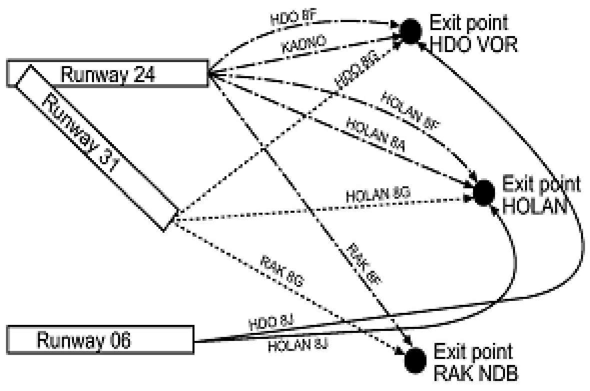


Figure 1.1: Schematic representation of part of Prague airport: runways, SID routes and exit points.

Given an airport with its runways, SID routes, and exit points, the departure management problem consists of allocating these resources and a suitable timetable to each flight to be scheduled. Suppose that *F1, F2 ,.., Fn* is the set of flights to be scheduled. Assume, furthermore, that for each flight *Fj* is given:

- The aircraft with its corresponding properties (e.g., its speed and weight class).
- The destination point, which is the TMA exit point.
- The CFMU (Central Flow Management Unit) time interval within which the flight needs to take-off.

Then for each flight *Fj* , the following will need to be planned:

- A take-off time: the time at which the aircraft should start its take-off roll at the runway.
- A runway, leading to the SID route.
- A total flight time, stating the time at which the aircraft should start flying its SID route (depending on the take-off time) and when it should complete the route.
- A SID route, leading to the TMA exit point.
- An exit time, the time at which the aircraft should pass the TMA exit point (depending on the total flight time).

2. Design of Aircraft Scheduling using CSPs

2.1 Introduction

To solve the Aircraft Scheduling problem using constraint satisfaction, we need to formulate it as a Constraint satisfaction Programming (CSP) problem. A CSP-problem can be defined as:

- A set of *variables* X={x1,...,xn}.
- For each variable x_i, a finite set D_i of possible values (its *domain*).
- A set of *constraints* restricting the values that the variables can simultaneously take.

2.2 Variables and Domains in Aircraft Scheduling

In order to describe the planning problem as a CSP-problem, we need to distinguish the variables, their associated domains and relevant constraints in the problem description. The variables in the problem space can then be identified with the flights that need to be scheduled. A flight will be defined as the total path of an aircraft from the gate via take-off to its exit point: the destination at which the aircraft leaves the airport's airspace. Flights are then constructed of the following parts:

- Take-off at the runway.
- SID at the SID route.
- Exit at the exit point.

The gate itself, it should be noted, as well as the various taxi routes an aircraft could take to reach a runway are currently excluded from the problem space. The values to which all constituents of flights need to be assigned fall into two categories:

- The time point or time range, stating when a particular part of a flight needs to start.
- The resources (runways, SID routes, exit points) needed by that part of the flight.

2.3 Constraints in Aircraft Scheduling

Constraints in a CSP-problem restrict the combinations of values assigned to the variables in the domain. For the departure management problem, a number of constraints *C1, C2,…, Cm* can be formulated to restrict the combinations of assigned times and allocated resources to all parts of the flights to be scheduled. Given its problem space, the following types of constraints can be distinguished:

- Resource constraints, specifying which resources a flight (each part of it) requires.
- Order constraints, restricting the time-order of the parts constituting a flight.
- Timeslot constraints, stating that flights need to take-off within their CFMU-timeslot.
- Separation constraints, formulating minimum separation times between aircraft of different speed and weight class for runways and exit points.
- Topology constraints, describing which runways connect to which flight-routes and which exit points.
- Additional tower-control constraints, reflecting controller decisions to let aircraft depart in a specific order or at specific time-intervals.

Having formulated the variables, values, and constraints relevant to the departure management problem, the task to be accomplished now is to find an assignment of times and allocation of resources to each sub-path of each flight, such that none of the constraints is violated. This task can be described as a scheduling task (being a specific CSP problem): a process of allocating scarce resources to activities over a period of time.

3. The Scheduling Model

3.1 Flights

Every Flight is divided into three sub-paths: a take-off, flight and exit. The Flight has been assigned its value as a quadruplet (runway for takeoff, SID-Route for flight, Exit point for Exit, take-off time). So domain for a Flight will be R×S×E×T, where R is the domain for runways, S is the domain for SID routes, E is the domain for Exit points and T is the domain for take-off time.

If we consider Prague airport as an example (*figure 1.1*) then
R = {runway24, runway31, runway06}
S = {HDO8F, KADNO, HDO8G, HOLAN8F, HOLAN8A, HOLAN8G, RAK8F, RAK8G, HOLAN8J, HDO8J}
E = {HDO, HOLAN, RAK}
T = {between allotted CFMU time slot for the given flight}.

3.2 Resource Constraint

Resource constraint specifies that which Flight require which runway, SID route and exit point for its takeoff. A file “resource requirement” is being updated according to the requirements for each flight. Suppose F_1 requires runways Runway24 or Runway06, SID routes HOLAN8A or RAK 8F OR HDO, exit point HDO OR HOLAN then in the file the information is expressed as in the table below:

Table 1: Resource Constraint

Flight No	*No of Runways*	*Runways*	*No of Sidroutes*	*Sidroutes*	*No of Exit Points*	*Exit Point*
1001	2	R24 R31	3	HOLAN8G RAK8F HDO	3	HDO HOLA N RAK

3.3 Timeslot Constraint

Timeslot constraint specifies the CFMU-timeslot flights need to use for its take-off. As an example, Flight number 3124 (F_1) has been allotted the CFMU-timeslot as 10:15 – 10:30, then the given Flight need to takeoff between 10:15 to 10:30. To codify that flight 1 should take-off in timeslot [1015, 1030] the prototype implements:

F_1.getTakeOffTime() > = F_1.getCFMUSTime() ;
F_1.getTakeOffTime() < = F_1.getCEMUETime() ;

Where F_1.getCFMUSTime() returns 10:15 and F_1.getCFMUETime() returns 10:30.

3.4 Separation Constraint

For a given airport, certain separation rules are prescribed. Separation settings define the minimum separation times aircraft at the runways or exit points need to obey. Here, we enables the user to configure separation criteria for runways and exit points for all relevant combinations of aircraft types.

For runways, one can set the minimum separation times for aircraft. In our prototype, two minimum separations can be distinguished: a large separation and a default separation. The large separation will be observed when an aircraft taking off after another aircraft is either in a larger speed class or in a smaller weight class. Under these circumstances, separation should be larger due to the stronger impact the wake vortex (i.e., the air turbulence) of the preceding aircraft can have on the following aircraft. For example, when a Cessna Citation (weight class Light) or a Fokker Friendship F27/500 (weight class Medium) takes off after a Boeing 747-400 (weight class Heavy) on the same runway, the large separation time needs to be observed. In any other situation, the default separation time should be adhered to.

This scheme is implemented by creating a transition times table, specifying the amounts of time that must elapse between two consecutive aircraft that takeoff. The table is then associated with the runway resources and filled with the minimum separation times to be observed for any two aircraft taking off. In our solution, the function GetTransitionTime returns the separation time between any aircraft with speed class *sp1* and weight class *wt1* that takes off before any aircraft with speed class *sp2* and weight class *wt2*:

```
Int GetTransitionTime(const SpeedClass sp1, const
SpeedClass sp2,
const WeightClass wt1, const WeighClass wt2)
   {
   if ((sp1 < sp2) || (wt1 > wt2))
   return maxRunwaySeparation;
   else
      return defaultRunwaySeparation;
      }
```

This function is called when filling the transition time table *TT* with minimum separations for all combinations of speed and weight classes:

```
for (SpeedClass sp1=A; sp1<=E; sp1++)
    for (WeightClass wt1=Light; wt1<=Heavy; wt1++)
      for (SpeedClass sp2=A; sp2<=E; sp2++)
        for (WeightClass wt2=Light; wt2<=Heavy; wt2=++)
          (*TT).setValue((sp1*wt1),(sp2*wt2),
             GetTransitionTime(sp1, sp2, wt1, wt2));
```

Thus, the transition time table is created and filled, effectively constraining any planning solution to the minimum separations times to be observed for aircraft taking off. For exit points, other separation times may and typically do apply. To separate aircraft on exit points, a minimum separation time can be set in a fashion similar to that for runways. In this way, it can be assured that sector via the exit points will be sufficiently spaced in time.

3.5 Topology Constraint

In our solution, matrixes are defined to lay down the connections between runways, SID routes, and exit points corresponding to the airport topology used. For example, for Prague airport constraints are defined to indicate that runway

R06 connects to SID routes HDO8J, HOLAN8J, leading to exit points HOLAN, HDO. For Prague airport topology constraints are defined by two matrices as follows:

Table 2: Runway SID – Route Connection

	HDO8F	KADNO	HDO8G	HOLAN8F	HOLAN8A	HOLAN8G	RAK8F	RAK8G	HOLAN8J	HDO8J
Runway24	1	1	0	1	1	0	1	0	0	0
Runway31	0	0	1	0	0	1	0	1	0	0
Runway06	0	0	0	0	0	0	0	0	1	1

Table 3: Exit-Point SID-Route Connection

	HDO8F	KADNO	HDO8G	HOLAN8F	HOLAN8A	HOLAN8G	RAK8F	RAK8G	HOLAN8J	HDO8J
HDO	1	1	1	0	0	0	0	0	0	1
HOLAN	0	0	0	1	1	1	0	0	1	0
RAK	0	0	0	0	0	0	1	1	0	0

3.6 Additional Tower Control Constraints

Additional tower-control constraints can be added to reflect controller decisions to influence the schedule. The prototype allows the tower-control to change the order or time interval in which specific flights should take-off. For example, to force flight 2 to take-off before flight 1, the prototype implements:

F_2.getTakeOffTime() < = F_2.getTakeOffTime();

Similarly, to force flight 1 to take-off at t=1015 :

F_1.setTakeofTime(1015);

4. Implementation

4.1 Layout

In the prototype presented here we opt C++ as the programming environment and made a few relaxations in the constraint area. These include inbound traffic constraint relaxation and additional tower control constraint relaxation. These constraints can easily be implemented with minor changes in the presented program.

4.2 Flight

We consider Flight as an object with data Member and member functions in the Flight class as below:

```
#include<iostream>
#include<string>
#include<cstring>
using namespace std;
class Flight{
 string runway;
 string sidRoute;
 string exitPoint;
 string speedClass;
 string weightClass;
 int CFMUSTime,CFMUETime;//CFMU interval
 int flightNo;
 int takeOffTime;
 public :
//-------------constructor for the class------------------
 Flight(string sc,string wc,string exit,int cfmus,int
cfmue,int fn)
 { setSpeedClass(sc);
 setWeightClass(wc);
 setExitPoint(exit);
 setCFMUSTime(cfmus);
 setCFMUETime(cfmue);
 setFlightNo(fn); }
//-------------set and get function for takeoff time---------
---
 void setTakeOffTime(int n)
 { takeOffTime=n; }
 int getTakeOffTime()
 { return takeOffTime; }

//----------set and get function for flight number-----------
-------
 int setFlightNo(int n)
 { flightNo=n; }
 int getFlightNo()
 {return flightNo;}
//----------set and get function for runway------------------
----
 void setRunway(string RW)
 { runway=RW; }
 string getRunway()
 {return runway;}
//------------set and get function for sid route---------------
 void setSidRoute(string s)
 {sidRoute=s;}
 string getSidRoute()
 {return sidRoute;}
//---------------set and get function for exit point-----------
--
 void setExitPoint(string E)
 {exitPoint=E;}
 string getExitPoint()
 {return exitPoint;}

//----------------set and get function for speed class--------
-----
 void setSpeedClass(string SC)
 {speedClass=SC;}
 string getSpeedClass()
 {return speedClass;}
//---------------set and get function for weight class-------
-------
 void setWeightClass(string WC)
 {weightClass=WC;}
 string getWeightClass()
 {return weightClass;}
//----------------set and get function for CFMU start time-
----------
 void setCFMUSTime(int ST)
 {CFMUSTime=ST;}
 int getCFMUSTime()
 {return CFMUSTime;}
//--------------------set and get function for CFMU end
time-------
 void setCFMUETime(int FT)
 {CFMUETime=FT;}
 int getCFMUETime()
 {return CFMUETime;}
//-------------------display function---------------------------
----
 void showData(){ cout<<"\nFlight Number :
"<<flightNo;
 cout<<"\nTakeOff runway : "<<getRunway()<<"\t\tSid
route : "<<getSidRoute()
 <<"\t\tTMA exit point : "<<getExitPoint();
 cout<<"\nSpeed Class : "<<getSpeedClass()
                                         <<"\t\tWeight class :
                          "<<getWeightClass();
 cout<<"\nCFMU Strat time of flight :
"<<getCFMUSTime()
                                         <<"\t\tCFMU End time of
                          flight : "<<getCFMUETime();
 cout<<"\nTakeoff time : "<<getTakeOffTime()<<endl;
}
};
```

4.3 Resource Constraint

To take care of the resource constraint we write a function named ResourceConstraint() and placed it into a header file Resource Constraint.h . This function is being called in main() for each Flight and it will update a file RC.dat in the follwing manner;

FlightNo	No of Runways	Runways	No of Sidroutes	Sidroutes	No of Exit Points	Exit Points

```
2134 2 runway24 runway31 3 hdo8f kadno hdo8g hdo
3124 3 runway24 runway31 runway06 3 holan8a holan8g holan8j holan
1234 3 runway24 runway31 runway06 3 holan8f holan8g holan8j holan
2314 2 runway24 runway31 2 rak8f rak8g rak
4321 2 runway31 runway06 2 hdo8g hdo8j hdo
```

To accomplish this scenario, C++ code is as below:void ResourceConstraint()

```
{
 ofstream outRCFile("RC.dat",ios::out|ios::app);
 if(!outRCFile)
 { cerr<<"File could not be opened"<<endl;
 exit(1); }
 cout<<"Enter Flight_No, No_Of_R, Runways,
No_Of_S, SIDRoutes, exitPoint"<<endl
 <<"Enter end-of-file to end the input .\n ? ";

 int flight_No;
 int NR;
 int NS;
 string runway;
 string SIDRoute;
 string exitPoint;

 cin>>flight_No;
 cin>>NR;
 outRCFile<<flight_No<<" "<<NR<<" ";
 for(int i=0;i<NR;i++)
 { cin>>runway;
 outRCFile<<runway<<" "; }
 cin>>NS;
 outRCFile<<NS<<" ";
 for(int i=0;i<NS;i++)
 { cin>>SIDRoute;
 outRCFile<<SIDRoute<<" "; }
 cin>>exitPoint;
 outRCFile<<exitPoint<<"\n";
 outRCFile.close();
}
```

4.4 Checking of Resource Constraints

During the run of backtrack () function in main () the file "RC.dat" will behave as a domain for resource constraint checking. To check resource constraint we write a function for each Runway, SID route and exit point. These functions are placed in the header files checkRunwayResource.h checkSIDRouteRresource.h and checkExitPointResource.h, and the C++ code for these are as below:

- checkRunway()
- Similarly for checkSIDRoute() and checkExitPoint().

```
int checkRunway(int flightNumber,string
runway)
{
 int fn,nr,i,ns;
 char str[15],xtra[256];
 ifstream infile("RC.dat",ios::in);
 if(!infile)
 { cerr<<"File could not be opened."<<endl;
 exit(1); }
 while(!infile.eof())
 { infile>>fn;
 if(fn==flightNumber)
 {infile>>nr;
 for(i=0;i<=nr;i++)
 {infile.getline(str,14,' ');
 if(runway==str)
 {infile.close();
 return 1;}
 }
 }
 infile.getline(xtra,255);
 }
 infile.close();
 return 0;
}
```

4.5 Domain for Runway, SID Route and Exit Point

It's been require assigning runways, SID routes and exit points to all flights during the whole course of program run. So, prior to the execution of the function main() Runway Domain, SID Route Domain and exit point domain is to be fixed. That part is taken care of by the functions DomainRunway(), DomainSIDRoute() and DomainExitPoint(). These functions are placed in the header files DomainRunway.h, DomainSIDRoute.h and DomainExitPoint.h respectively. Basically these functions create files Runway.dat, SIDRoute.dat and ExitPoint.dat containing the all possible values for runways, SID routes and exit points respectively. The C++ code for this task is as below:

- DomainRunway

```
void DomainRunway()
{
 ofstream
outRunwayFile("Runway.dat",ios::out);
 if(!outRunwayFile)
 {
 cerr<<"File could not be opened"<<endl;
 exit(1);
 }
 string Runway;
 cout<<"Enter Runways one by one :
"<<endl
 <<"Enter end-of-file to end input \n? ";
 while(cin>>Runway)
 {
 outRunwayFile<<Runway<<endl;
 cout<<"? ";
 }
}
```

- Similarly for DomainSIDRoute() and DomainExitPoint()

4.6 Assignment from Different Domains

We write two functions namely FromDomainRunway() and FromDomainSIDRoute() that returns the runway and SID route from their respective domains when being called by the Flight object. These functions are being placed in AssigningFromRunway.h and AssigningFromSIDRoute.h header files.

For each Flight object a pointer is assigned and its value is changed whenever an assignment is being made to the Flight. C++ code for this is as under:

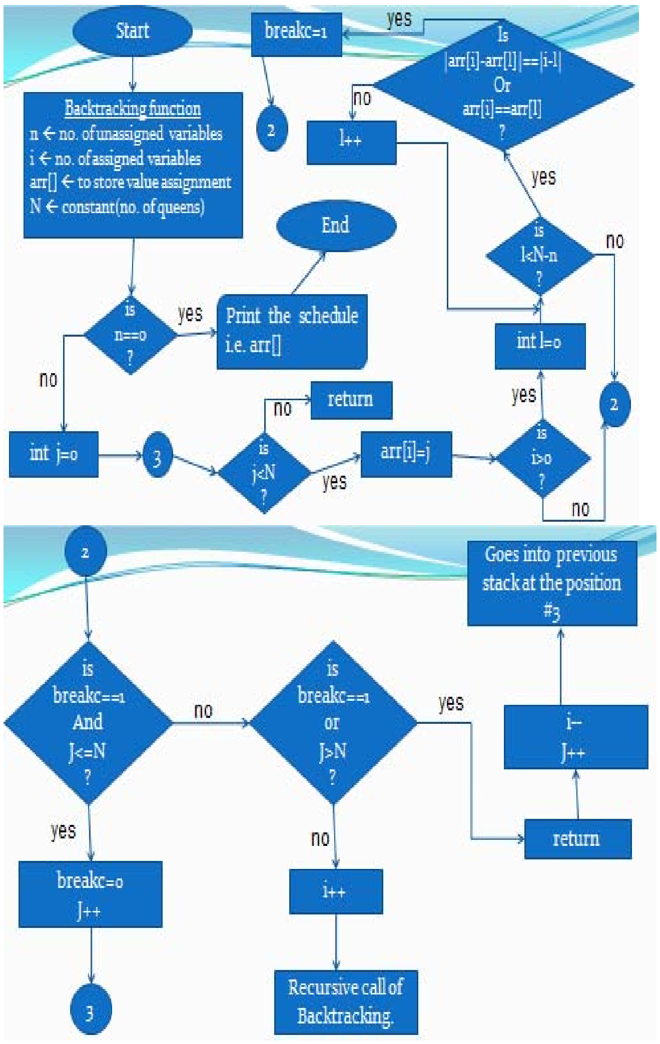
1. Assigning from runway

```
string FromDomainRunway(int *Rptr)
{
 int End;
 ifstream
inRunwayFile("Runway.dat",ios::in);

 if(!inRunwayFile)
 {
 cerr<<"File could not be opened"<<endl;
 exit(1);
 }
 string Runway;
 inRunwayFile.seekg(*Rptr);
 inRunwayFile>>Runway;
 *Rptr=inRunwayFile.tellg();
 inRunwayFile.seekg(0,ios::end);
 End=inRunwayFile.tellg();
 inRunwayFile.seekg(*Rptr);
 if(*Rptr==(End-1))
 *Rptr=0;
 inRunwayFile.close();
 return Runway;
}
```

2. Assigning from SID route

```
string FromDomainSIDRoute(int *Sptr)
{
 int End;
 ifstream
inSIDRouteFile("SIDRoute.dat",ios::in);

 if(!inSIDRouteFile)
 {
 cerr<<"File could not be opened"<<endl;
 exit(1);
 }
 string sidroute;
 inSIDRouteFile.seekg(*Sptr);
 inSIDRouteFile>>sidroute;
 *Sptr=inSIDRouteFile.tellg();
 inSIDRouteFile.seekg(0,ios::end);
 End=inSIDRouteFile.tellg();
 inSIDRouteFile.seekg(*Sptr);
 if(*Sptr==(End-1))
 *Sptr=0;
 inSIDRouteFile.close();
 return sidroute;
}
```

4.7.1 Backtracking Flowchart


Start
breakc=1
yes
Is |arr[i]-arr[l]|==|i-l| Or arr[i]==arr[l] ?
no
Backtracking function
n ← no. of unassigned variables
i ← no. of assigned variables
arr[] ← to store value assignment
N ← constant(no. of queens)
2
l++
yes
End
is l<N-n ?
no
is n==0 ?
yes
Print the schedule i.e. arr[]
int l=0
no
return
no
yes
2
int j=0
3
is j<N ?
yes
arr[i]=j
is i>0 ?
no
2
Goes into previous stack at the position #3
is breakc==1 And J<=N ?
no
is breakc==1 or J>N ?
yes
i--
J++
yes
no
return
breakc=0
J++
i++
3
Recursive call of Backtracking.

4.7.2 Backtracking Function

We use recursion to do backtracking.

```
void backtrackFunction(int unassigned,int i,Flight F[N],int
TCT1[][10],int TCT2[][10],int TT[][N],int Rptr[],int Sptr[])
{
int j,l,breakc,ex;
char ch;
int bt=0;
int statusR,statusS,statusTC,statusTOT;
int bforR=0;
if(unassigned==0)
{
cout<<"\n======================================
============================================
======================\n";
for(int k=0;k<N;k++)
{
F[k].showData();
cout<<"Rptr["<<k<<"] = "<<Rptr[k]<<"\tSptr["<<k<<"] =
"<<Sptr[k]<<endl<<endl;
}
cout<<"======================================
============================================
======================\n";
cout<<"For another result press a ";
cin>>ch;
return;
}
if(unassigned>0)
{
while(1)
{
F[i].setRunway(FromDomainRunway(&Rptr[i]));
statusR=checkRunway(F[i].getFlightNo(),F[i].getRunway());
cout<<"status of "<<F[i].getRunway()<<" for F["<<i<<"] =
"<<statusR<<" Value of Rptr = "<<Rptr[i]<<endl;
if(statusR==1)
{
while(1)
{
F[i].setSidRoute(FromDomainSIDRoute(&Sptr[i]));
statusS=checkSIDRoute(F[i].getFlightNo(),F[i].getSidRoute()
);
cout<<"status of "<<F[i].getSidRoute()<<" for F["<<i<<"] =
"<<statusS<<"Value of Sptr = "<<Sptr[i]<<endl;
if(statusS==1)
{
//--------------checking topology constraint I----------------------
----------------------
statusTC=checkFromTCT1(F[i].getRunway(),F[i].getSidRout
e(),TCT1);
cout<<"status of TC1 for F["<<i<<"] = "<<statusTC<<endl;
if(statusTC==1)
{
//----------checking topology constraint II-------------------------
-----------------------
statusTC=checkFromTCT2(F[i].getExitPoint(),F[i].getSidRou
te(),TCT2);
cout<<"status of TC2 for F["<<i<<"] = "<<statusTC<<endl;
//--------------------------------------------------------------------
----------------
if(statusTC==1)
{
cout<<endl<<"Allotment of Take off time and checking of
separation constraint"<<endl;
//--------------------------------------------------------------------
------------------
for(j=F[i].getCFMUSTime();j<=F[i].getCFMUETime();j++)
{
F[i].setTakeOffTime(j);//allotment of takeoff time
statusTOT=1;
cout<<"Take off time for F["<<i<<"] =
"<<F[i].getTakeOffTime()<<endl;
if(i>0)
{
for(l=0;l<i;l++)
{
//checking of separation constraint
if((F[i].getRunway())==(F[l].getRunway()))
{
if(F[l].getTakeOffTime()>F[i].getTakeOffTime())
{
if(diff(F[l].getTakeOffTime(),F[i].getTakeOffTime())>=TT[i]
[l])
{
cout<<diff(F[l].getTakeOffTime(),F[i].getTakeOffTime());
cout<<" "<<TT[i][l]<<endl;
statusTOT=1;
}
else
{
cout<<diff(F[l].getTakeOffTime(),F[i].getTakeOffTime());
cout<<" "<<TT[i][l];statusTOT=0;
// cout<<" -> Separation constraint get voilated\n";
break;//break out of separation constraint checking
}
}
else
{
if(diff(F[i].getTakeOffTime(),F[l].getTakeOffTime())>=TT[l]
[i])
{
cout<<diff(F[i].getTakeOffTime(),F[l].getTakeOffTime());
cout<<" "<<TT[l][i]<<endl;
statusTOT=1;
}
else
{
cout<<diff(F[i].getTakeOffTime(),F[l].getTakeOffTime());
cout<<" "<<TT[l][i]<<endl;
// cout<<" -> Separation constraint get voilated\n";
statusTOT=0;
break;//break out of separation constraint checking
}
}
}

}//end of for loop for separation constraint checking

}//end of (i>0) if condition

if(statusTOT==1)
{
F[i].showData();
cout<<"======================================
==========================================
===\n";
i++;unassigned--;
backtrackFunction(unassigned,i,F,TCT1,TCT2,TT,Rptr,Sptr);
i--;unassigned++;
}

}//end of for loop for takeoff time allotement

}//end of if condition for topology constraint II

if(statusTC==0&&Sptr[i]==0)
break;//go out of loop to change the runway

}//end of if condition for topology constraint I

if(statusTC==0&&Sptr[i]==0)
break;//go out of loop to change the runway

}//end of if condition for SIDROUTE Resource constraint
checking i.e. if(statusS==1) end here

if(statusS==0&&Sptr[i]==0)
break;
if((statusTOT==0)&&(j>F[i].getCFMUETime())&&(Rptr[i]!
=0)&&(Sptr[i]==0))
{
cout<<"\nRunway is getting changed skipping backtrack
since for F["<<i<<"] runways are available\n";
break;
}
if((statusTOT==0)&&(j>F[i].getCFMUETime())&&(Rptr[i]=
=0))
{
Sptr[i]=0;
cout<<"\nbacktrack as no possible takeoff time can be
obtained for any runway for F["<<i<<"]\n";
return;
}
}//end of while loop for sid route

}//end of if condition for RUNWAY Resource constraint
checking i.e. if(statusR==1) end here

if((i>0)&&(statusR==0)&&(Rptr[i]==0))
{
Sptr[i]=0;
cout<<"\nbacktrack as no possible takeoff time can be
obtained for any runway for F["<<i<<"]\n";
return;
}
if((i==0)&&(statusR==0)&&(Rptr[i]==0))
{
cout<<"\n No possible departure schedule can be obtained
\n";
return;
}

}//end of while loop for runway
}
#
```

4.8 Driver Program

```
int main()
{
Flight F[N]=
{Flight("high","medium","hdo",1000,1015,2134),Flight("lo
w","medium","holan",1005,1020,3124),Flight("medium","lo
w","holan",1010,1025,1234),Flight("low","low","rak",1005,
1020,2314),Flight("medium","medium","hdo",1005,1020,43
21)};

int
TransitionTable[N][N],TopologyConstraintTable1[3][10],To
pologyConstraintTable2[3][10],i=0;

int Rptr[N]={0},Sptr[N]={0};

constructTT(F,TransitionTable);

for(int a=0;a<N;a++)
{
for(int b=0;b<N;b++)
cout<<" "<<TransitionTable[a][b];
cout<<endl;
}

constructTCT1(TopologyConstraintTable1);
constructTCT2(TopologyConstraintTable2);

cout<<"\nGive Resource Constraint for each Flight one by one :\n";
for(i=0;i<N;i++)
{
ResourceConstraint();
cout<<endl<<endl;
}

backtrackFunction(N,i,F,TopologyConstraintTable1,TopologyConstraintTable2,TransitionTable,Rptr,Sptr);
}
```

5. Results and Discussion

Considering the Prague airport topology, and a hypothetical set of Flights as under:
F1 F2 F3 F4 F5
Speed Class High Low Medium Low Medium
Weight Class Medium Medium Low Low Medium
Exit point Hdo Holan Holan Rak Hdo
CFMU
Time Interval
10:00-10:15 10:05-10:20 10:10-10:25 10:05-10:20 10:10-10:25
Flight
Number
2134 3124 1234 2314 4321
With default separation time as 10 min and large separation time as 15 min.

The solution of this example on the execution of the program will be as below:

```
Transition table is :
15 10 15 10 10
15 10 15 10 10
15 10 15 10 10
15 10 15 10 10
15 10 15 10 10
Fill the table for connection between runways and sid routes
.
1 1 0 1 1 0 1 0 0 0
0 0 1 0 0 1 0 1 0 0
0 0 0 0 0 0 0 0 1 1
Fill the table for connection between exit point and sid routes
1 1 1 0 0 0 0 0 0 1
0 0 0 1 1 1 0 0 1 0
0 0 0 0 0 0 1 1 0 0
Give resource constraints for the flights:
2134 2 runway24 runway31 3 hdo8f kadno hdo8g hdo
3124 3 runway24 runway31 runway06 3 holan8a holan8g holan8j holan
1234 3 runway24 runway31 runway06 3 holan8f holan8g holan8j holan
2314 2 runway24 runway31 2 rak8f rak8g rak
4321 2 runway31 runway06 2 hdo8g hdo8j hdo
49
Solution is
Flight Number : 2134
TakeOff runway : runway24 Sid route : hdo8f TMA exit point : hdo
Speed Class : high Weight class : medium
CFMU Strat time of flight : 1000 CFMU End time of flight : 1015
Takeoff time : 1000

Flight Number : 3124
TakeOff runway : runway24 Sid route : holan8a TMA exit point : holan
Speed Class : low Weight class : medium
CFMU Strat time of flight : 1005 CFMU End time of flight : 1020
Takeoff time : 1010

Flight Number : 1234
TakeOff runway : runway31 Sid route : holan8g TMA exit point : holan
Speed Class: medium Weight class : low
CFMU Strat time of flight : 1010 CFMU End time of flight : 1025
Takeoff time : 1010

Flight Number: 2314
TakeOff runway: runway24 Sid route : rak8f TMA exit point : rak
Speed Class: low Weight class : low
CFMU Strat time of flight: 1005 CFMU End time of flight : 1020
Takeoff time: 1020

Flight Number: 4321
TakeOff runway: runway31 Sid route : hdo8g TMA exit point : hdo
Speed Class: medium Weight class : medium
CFMU Strat time of flight: 1005 CFMU End time of flight: 1020
Takeoff time: 1020
```

The departure scheduling prototype presented in this paper provides runway controllers with a decision support tool to establish optimal departure sequences for aircraft. As a consequence, runway capacity will be effectively enhanced without any physical changes to the airport infrastructure. The major advantage of the prototype lies in its flexibility: any airport topology can be used, inbound traffic is taken into account and certain take-off times or - orders can be fixed while others can be scheduled. The performance our solution achieves is acceptable for practical application at airports such as Prague. A future step might be to reduce the problem into Distributed Constraint Satisfaction Problem (DCSP) in order to get more efficiency. We expect that the acceptance of the tool will be high, since runway controllers will remain involved in the scheduled process. They will be able to impose restrictions on a schedule beforehand, and make modifications to calculated solutions afterwards by replanning parts of the generated schedule.

References

[1] S. Zelinkski and T. Romer, “An Airspace Concept Evaluation System Characterization of National Airspace System Delay,” AIAA 4th Aviation Technology, Integration and Operations (ATIO), September 2004.

[2] Idris, H., Delcaire, B., Anagnostakis, I., Hall, W., Pujet, N., Feron, E., Hansman, R., Clarke, J., and Odoni, A., \Identi_cation of Flow Constraint and Control Points in Departure Operations at Airport Systems," AIAA Guidance, Navigation, and Control Conference, Boston, MA, Aug. 10-12, 1998.

[3] Wood Z., Kistler M., Rathinam S., and Jung Y., \A Simulator for Modelling Aircraft Surface Operations at Airports," AIAA Modeling and Simulation Technologies Conference, Chicago, IL., August 2009.

[4] Gupta, G., Malik, W., and Jung, Y. \Incorporating Active Runway Crossings in Airport Departure Scheduling," AIAA Guidance, Navigation and Control Conference, Toronto, Canada, August 2-5, 2010.

[5] Brinton, C., Atkins, S., Cook, L., Lent, S., and Prevost, T., \Ration by Schedule for airport arrival and departure planning and scheduling," Integrated Communications Navigation and Surveillance Conference (ICNA), Herndon, VA, 11-13 May 2010

[6] Gupta, G., Malik., W., and Jung, Y., \A Mixed Integer Linear Program for the Airport Departure Scheduling Problem," 9th AIAA Aiviation Technoology, Integration, and Operations Conference (ATIO), Hilton Head, SC, September 2009

[7] H.N., Psaraftis, \A dynamic programming approach for sequencing groups of identical jobs," Operations Research 28 (1980), 13471359

[8] Balakrishnan, H. and Chandran, B., \E_cient and Equitable Departure Scheduling In Real-Time: New Approaches To Old Problems," 7th USA/Europe Air Tra_c Management R&D Seminar, July 2007, Barcelona, Spain

[9] Stiverson, P., Rathinam, S., \Heuristics for a runway-queue management problem," Proceedings of the Institution of Mechanical Engineers, Part G: Journal of Aerospace Engineering, August 2010

[10] Rathinam, S., Wood, Z., Sridhar, B., and Jung, Y. C., \A Generalized Dynamic Programming Approach for a Departure Scheduling Problem," AIAA Guidance, Navigation, and Control Conference, Chicago, Illinois, August 10-13, 2009

[11] Anonymous, "Aircraft Operations," *Aerospace America*, Dec. 1994, pp. 29

[12] Federal Aviation Administration, *Air Traffic Control* , 1994

[13] Federal Aviation Administration, *Airman's Information Manual*, Nov. 1995

[14] Carr, G., Erzberger, H., and Neuman, F., "Airline Arrival Prioritization in Sequencing and Scheduling," *Second USA/ Europe Air Traffic Management R&D Seminar*, December 1-4 1998

[15] P. J. Deitel and H. M. Deitel, C++ How to program , 6th edition

[16] P. van Leeuwen, H.H. Hesselink and J.H.T. Rohling," Scheduling aircraft using constraint satisfaction"

Author Profile

Ranjan Kumar Thakur received the M. Tech. degree in Computer Science and Data Processing from Indian Institute of Technology Kharagpur 2012.