# Project Statement

1. Implement the algorithm discussed in the article by C. Sriram, A. Haghani (2003) using the assumptions above in Python Code. Clearly describe any modification (if needed) of the algorithm in the article. The code should read the input Excel file attached (Project Data.xlsx) and print the final output of the solution in an easy-to-read format in the Jupyter Notebook file. On top of the flight and maintenance schedules for each aircraft, the solution should include the average weekly maintenance cost of each aircraft and the overall average maintenance cost (i.e. the objective function value). Run the code on the Excel file data and make sure to include the final solution output in your Jupyter Notebook file.
2. For 3 or more major data structures (input/output/intermediate) used in your project solution, discuss and justify your choice of an implemented data structure based on data manipulation requirements (access, deletion, …), runtime and memory complexity, and/or search and sort requirements of the project and its subroutines/functions/classes. Demonstrate the data structure with a small example and sample data.
3. For 3 or more major functions/subroutines/classes or major sections of the codes employed in your project, include a clear description of the idea behind the algorithm (one paragraph each), plus a clear, and easy-to-read algorithm, and its documented Python code. Select functions/subroutines/classes that best describe your understanding of the course material and the article by C. Sriram, A. Haghani (2003). For each item, include the growth of the function analysis and demonstrate its functionality using sample data.
4. Your submission files (Python Codes excluded) should transfer enough knowledge of the problem’s background, concepts, assumptions, and the solution approach in a way that a reader with prior knowledge of programming, is enabled to replicate your project codes and obtain similar results without any, or with minimum need to refer to the original article by C. Sriram, A. Haghani (2003), or any other sources.

# Background for Aircraft Maintenance problem

The aircraft maintenance schedule is one of the major decisions an airline has to make during its operation. Though maintenance scheduling comes as an end-stage in an airline operation, it has the potential for cost savings. Maintenance scheduling is easily understood but difficult to solve the problem. Given a flight schedule with aircraft assigned to it, the aircraft maintenance-scheduling problem is to determine which aircraft should fly which segment and when and where each aircraft should undergo different levels of maintenance check required by the Federal Aviation Administration. The objective is to minimize the maintenance cost and any costs incurred during the re-assignment of aircraft to the flight segments. Consideration of maintenance constraints has long been recognized to be a cornerstone in aircraft scheduling. The development of the aircraft maintenance schedule is a complicated task involving the synthesis of a range of economic, political, legal, and technical factors. Demand for service, aircraft utilization, and operational cost of aircraft are the principal drivers. The goal is to achieve a balanced pattern of flights that results in a timetable consistent with the FAA regulations and airline policies. The major airlines have witnessed significant changes in their operating environment after the airline deregulation act of 1978. As a result of fierce competition, the airlines had to cut their prices down and this led to more passengers flying than ever before. More than 80% of passengers are now traveling on tickets priced at less than the base fare. This accompanying downward pressure on revenues has led many carriers to focus their attention on controlling maintenance and personnel costs. From an operating point of view, the demand for service sets the daily flight schedule and determines which type of aircraft will be flown on a given route. This is the primary constraint faced by the maintenance planners who must schedule an inspection for each plane in the fleet in compliance with the FAA regulations. The possibility of assigning individual aircraft to different routes throughout the day offers the flexibility needed to meet this requirement.

In this project, we shall consider the problem faced by an airline needing to construct a 7-day planning horizon cyclic schedule with maintenance constraints for a heterogeneous fleet of aircraft. The maintenance checks on aircraft are to be scheduled based on the given flight schedule. The flight schedule consists of a sequence of flight legs to be carried out by an aircraft. So the maintenance scheduling problem is solved after the aircraft are assigned to the flight legs. Solving the maintenance-scheduling problem to optimality may cause a re-assignment of aircraft to the flight legs. Different aircraft assignments lead to different costs and revenues for an airline. For example, a flight leg that can be flown by two aircraft of different capacities can result in a loss of revenue if the smaller aircraft is chosen when the demand for the leg exceeds the smaller aircraft capacity. In contrast, a flight leg that can be flown by two aircraft of different capacity can result in higher operating cost if the larger aircraft is chosen when the demand for the leg is lower than the capacity of the larger aircraft. So the shuffling (re-assignment) of aircraft would result in loss of revenue or an increase in operating cost. This cost is taken into account by penalizing the assignment of inappropriate aircraft to the flight legs.

Since most of the maintenance checks are done during the night the problem of concern is where the aircraft spends the night each day in a 7-day cyclic schedule rather than intermediate stops. So a sequence of flight legs to which an aircraft is assigned for any given day can be considered as one trip. In the course of re-assignment, an aircraft is assigned to a trip rather than a single flight leg. The penalty cost for a trip would be the sum of the penalties for all flight legs that constitute the trip. Hereafter a sequence of flight legs that make a trip will be identified as an Origin-Destination (OD) pair.

Aircraft maintenance takes place in a series of checks of increasing diligence with the exception of unscheduled fixes. The frequency of these checks depends on the combination of flight hours and a number of take-off and landing cycles and may be performed at any site appropriately equipped. Because each aircraft type has different inventory requirements, little savings can be achieved by combining facilities for different fleets. To be in compliance with the Federal Aviation Administration constraints, some companies have adopted maintenance policies that call for routine inspections at least every four days. There are four major types of checks mandated by the FAA that each aircraft has to undergo. These vary in scope, duration, and frequency. We shall be checking only for the first major check (denoted as Type A) actually mandated by the FAA occurs at every 65 flight-hours, or about once a week. Type A checks involve inspection of all major systems such as landing gear, engines, and control surfaces.

### The objective of the Project

The objective of this project is to present an innovative mathematical formulation and an effective methodology to solve the aircraft maintenance-scheduling problem. The formulation and the solution method are to assign aircraft to the OD pairs to minimize the maintenance cost. The scope of the problem discussed here is limited to weekly domestic flight schedules. During the period of flight inactivity that is usually in the late evening to early morning, the maintenance is routinely performed. Given a flight schedule, the aircraft reassignment and maintenance-scheduling problems are to determine which aircraft should fly which OD pair and when and where each aircraft should undergo maintenance checks of Type A and B. The objective is to minimize the maintenance cost and the penalty incurred during the re-assignment of aircraft to the OD pairs. This problem can be solved at different levels. A typical one would consider two important levels of maintenance requirement (Type A and B), heterogeneity in the fleet, the location of maintenance base facility for different aircraft types, and the cyclical schedule of an n-day planning horizon. The scope of the problem with which we are dealing in this research is as follows:

* Only domestic airline operations are considered.
* Aircraft assignment is made before the maintenance schedule.
* Only Type-A maintenance constraints are being considered.
* Only the existing maintenance bases are considered and no recommendations are made for the construction of new feasible and optimal location of maintenance bases.
* Unexpected maintenance requirements are not being considered.

# Formulation of the Problem

The maintenance-scheduling problem is most naturally modeled as a closed-loop network. Using OD schedule as input, the maintenance-scheduling problem is formulated as a min-cost multi-commodity network flow model with integer restrictions on the variables, as such each plane represents a separate commodity. Usually, a fleet of different aircraft is considered rather than individual aircraft in scheduling, but since maintenance requirements are considered in this scheduling problem, each aircraft should be considered as a separate commodity. Each OD pair segment has an upper and lower capacity of one unit of flow. Now define nd as the planning horizon and nc as the number of cities in the OD schedule. The total number of nodes in the underlying graph is nd\*nc. But not necessarily each node should have input and output arcs. If np is the total number of planes in the fleet, then the total number of arcs is nd\*np. This is because each aircraft is assigned to exactly one OD segment, and each OD segment is covered by exactly one aircraft. This produces a large graph. Origin is the airport where an aircraft leaves in the early morning after spending the previous night. The destination is the airport where the aircraft spends the night for that day. Each arc in the network represents a unique OD trip assigned to an aircraft in the flight schedule. In case of two aircraft having the same Origin and Destination for any given day, then each trip of those aircraft for that day is identified by the subscript r. For example, an aircraft is assigned to a trip, which starts, from Washington to Austin to Las Vegas to Los Angeles and another aircraft is assigned to a trip from Washington to Chicago to Arizona to Los Angeles in the original flight schedule. In this case for both trips Washington is the Origin and Los Angeles is the Destination. Since the proposed formulation only needs to consider the Origin and Destination (intermediate stops are not considered), to make these two trips a unique subscript r is introduced. In this formulation, we do not need to consider the intermediate stops because they have no effect on the maintenance schedule. The flight schedule including the sequence of flight legs that are to be flown by aircraft during a day is an input to the problem and we refer to it as an OD trip. The proposed formulation is only concerned with the assignment of aircraft to these OD trips.

Over the planning horizon, aircraft can be assigned a variety of routes depending on their maintenance needs and the availability of facilities. In effect, the planning horizon is balanced. That is, the in-degree and out-degree for each node that represents a city on a particular day are the same (the number of a particular type of aircraft that are going into a node is the same as those going out of that node). This implies that the graph is Eulerian so removing any cycle leaves the Eulerian property. Among the factors considered in maintenance, scheduling is passenger demand, revenue, seating capacity, fuel costs, crew size, and maintenance costs. Many of these factors are captured in the objective coefficients of the decision variables and others are captured by the constraints. For example, the potential revenue generated by flights can be determined by forecasting the demand for seats on those flights and multiplying the minimum of the forecasted demand and the seating capacity by the average fare. This model satisfies balance constraints that force the aircraft to circulate through the network of flights. The assumptions that are made in the proposed formulation are as follows:

* Aircraft maintenance checks are performed only during the night.
* There is no aircraft operation during the night.
* The maintenance bases are located at the airport.
* The objective function is to minimize the average weekly maintenance cost of the

aircraft to carry the Origin-Destination (OD) pairs in the attached Excel file.

* Homogeneous fleet (all aircraft can fly all OD’s)
* Operation costs of aircraft are assumed to be identical.
* Maintenance types:
  + A: Every 4 days
  + There is no Maintenance type B here
* Maintenance costs of aircraft in all cities are listed in the Excel file attached.
* Only aircraft cycles of one week will be investigated.
* For Type-A maintenance an aircraft maintenance schedule will happen twice a week. (once on the fourth day and once on the 7th day)

## Why is optimization not possible?

The aircraft optimization formulation results in a very large integer programming problem that cannot be solved exactly in reasonable computation time. The development of heuristic procedures for solving this problem is left for future research. In the project, we focus on presenting an efficient solution procedure for the original formulation of the problem presented that is based on the limitations on the number of days between two consecutive maintenance checks. This formulation is more appropriate because airlines normally impose upon themselves restrictions on the number of days between consecutive maintenance checks that are generally more stringent than those mandated by the FAA.

# Development of the algorithm

This search technique is a combination of a depth-first search and random search. First, a list of nodes in a given day is made. The first aircraft and the first node are chosen from the respective list. Then an exhaustive depth-first search is performed from the node to find the best cyclic schedule for the chosen aircraft. Then the assigned links are removed from the network. Now the second aircraft is chosen from the list, if the number of outgoing arcs in the first node is not equal to zero, again a depth-first search is performed to find the best cyclic schedule for the second aircraft. If the number of outgoing arcs is equal to zero, then the second node from the node list is chosen and an exhaustive search is performed from this node to find the best cyclic schedule for the aircraft. This procedure is repeated until there are no more aircraft or nodes in the corresponding list.

After processing all the aircraft in the above-mentioned procedure, a feasible schedule and the objective function value is found. This objective value is compared with the objective value obtained from the previous iteration and the better solution is saved. For the next iteration, the node list and aircraft list are perturbed (shuffled). The procedure is performed for the new list of nodes and aircraft. The steps of the algorithm are as follows:

* Step 0: Make a list of aircraft in any order. Make another list of nodes (cities) for any given day. Initialize a number of iterations = 1.
* Step 1: Let n - 1.
* Step 2: Pick the nth aircraft from the list of aircraft.
* Step 3: Let K = 1.
* Step 4: Pick the Kth node from the list of nodes.
* Step 5: If there are no more nodes available for the allocation in the Kth node, let K = K + 1 and go to step 4, otherwise go to step 6.
* Step 6: Do a depth-first search to find the best possible cyclic schedule for the nth aircraft. If a feasible cyclic schedule exists go to step 7, otherwise let K = K + 1, go to step 5.
* Step 7: Add the route to the schedule. Delete the arcs from the network that are assigned to the nth aircraft.
* Step 8: If n = number of aircraft,
  + (a) reconstruct the network (in step 7 arcs are removed from the network, for the new iteration, these arcs need to be placed back in the network),
  + (b) perturb the aircraft list randomly (construct a list by choosing each aircraft at random from the list of aircraft),
  + (c) perturb the node list randomly (construct a list by choosing each node at random from the list of nodes that belongs to any given day). If a feasible solution was found in the previous iterations, compare the current solution with the existing one, and if the current one has a lower objective function value save the current solution and delete the previous solution. Otherwise, let n = n + 1 and go to step 2.
* Step 9: If the number of iterations is less than the maximum number of iterations, increment the number of iterations and go to step 2, otherwise stop. In all test problems, the maximum number of iterations is set at 5000.

# Data structure analysis

These are the variables used in the notebook:

'i' is the index for planes (1 to n\_p)

'J', 'k' is the index for cities (1 to n\_c)

'd' is the index for days (1 to n\_d), here (0 to 6)

'j(d)' is the node for city j on day d

'j(d-1)k(d)r' is the edge connecting city j on day d-1 to city k on day d via route r

'G(k(d))' is the set of nodes connected to k(d)

'F(k(d))' is the set of nodes k(d) is connected to

'L' is set of all edges

'N' is a set of all nodes

'g(i,j)' is the cost for plane i at city j

'p(j)' is the number of planes that can take maintenance at city j

'y(i)'=1, is the number of days until the due date for maintenance

'w(i,j,d)'=1, if maintenance taken, else 0

'C(i,j(d-1)k(d)r)' is the cost for plane i via j(d-1)k(d)r edge

'x(i,j(d-1)k(d)r)'=1, if plane i is taken via j(d-1)k(d)r edge, else 0

We used a pandas DataFrame for storing the data of the maintenance Type A cost and also the OD pairs for each day. Accessing a single row by index (index is sorted and unique) should have runtime O(m) where m << n\_rows. Accessing a single row by index (index is NOT unique and is NOT sorted) should have runtime O(n\_rows). Accessing a single row by index (index is NOT unique and is sorted) should have runtime O(m) where m < n\_rows). Accessing row(s) (independently of an index) by boolean indexing should have runtime O(n\_rows).

The second data structure we have used is the NumPy array. g is a 2D array that provides the maintenance cost for a plane at a city. w is a 2D array where 1 represents the maintenance check of a plane in a city on a day. Here the assumption is made that on every 4th day of the week the plane takes a maintenance check. We have used the numpy array because of its convenience and the various functions it has. Pandas DataFrame are widely used in data analysis.

## Demonstration of Pandas Dataframe

The DataFrame can be created using a single list or a list of lists.

import pandas as pd

data = [1,2,3,4,5]

df = pd.DataFrame(data)

print df

ts output is as follows −

0

0 1

1 2

2 3

3 4

4 5

### Example 2

import pandas as pd

data = [['Alex',10],['Bob',12],['Clarke',13]]

df = pd.DataFrame(data,columns=['Name','Age'])

print(df)

Its output is as follows −

Name Age

0 Alex 10

1 Bob 12

2 Clarke 13

### Example 3

import pandas as pd

data = [['Alex',10],['Bob',12],['Clarke',13]]

df = pd.DataFrame(data,columns=['Name','Age'],dtype=float)

print(df)

Its output is as follows −

Name Age

0 Alex 10.0

1 Bob 12.0

2 Clarke 13.0

We will understand this by selecting a column from the DataFrame.

import pandas as pd

d = {'one' : pd.Series([1, 2, 3], index=['a', 'b', 'c']),

'two' : pd.Series([1, 2, 3, 4], index=['a', 'b', 'c', 'd'])}

df = pd.DataFrame(d)

print df ['one']

We will understand this by adding a new column to an existing data frame.

import pandas as pd

d = {'one' : pd.Series([1, 2, 3], index=['a', 'b', 'c']),

'two' : pd.Series([1, 2, 3, 4], index=['a', 'b', 'c', 'd'])}

df = pd.DataFrame(d)

# Adding a new column to an existing DataFrame object with column label by passing new series

print ("Adding a new column by passing as Series:")

df['three']=pd.Series([10,20,30],index=['a','b','c'])

print df

print ("Adding a new column using the existing columns in DataFrame:")

df['four']=df['one']+df['three']

print df

Columns can be deleted or popped; let us take an example to understand how.

# Using the previous DataFrame, we will delete a column

# using del function

import pandas as pd

d = {'one' : pd.Series([1, 2, 3], index=['a', 'b', 'c']),

'two' : pd.Series([1, 2, 3, 4], index=['a', 'b', 'c', 'd']),

'three' : pd.Series([10,20,30], index=['a','b','c'])}

df = pd.DataFrame(d)

print ("Our dataframe is:")

print df

# using del function

print ("Deleting the first column using DEL function:")

del df['one']

print df

# using pop function

print ("Deleting another column using POP function:")

df.pop('two')

print df

Row Selection, Addition, and Deletion

We will now understand row selection, addition and deletion through examples. Let us begin with the concept of selection.

Selection by Label

Rows can be selected by passing the row label to a loc function.

import pandas as pd

d = {'one' : pd.Series([1, 2, 3], index=['a', 'b', 'c']),

'two' : pd.Series([1, 2, 3, 4], index=['a', 'b', 'c', 'd'])}

df = pd.DataFrame(d)

print df.loc['b']

## Demonstration of NumPy Arrays

In this code we initialized the array and printed it. We can create a NumPy ndarray object by using the array() function.

import numpy as np

arr = np.array([1, 2, 3, 4, 5])

print(arr)

print(type(arr))

To create an ndarray, we can pass a list, tuple or any array-like object into the array() method, and it will be converted into an ndarray:

import numpy as np

arr = np.array((1, 2, 3, 4, 5))

print(arr)

0-D Arrays

0-D arrays, or Scalars, are the elements in an array. Each value in an array is a 0-D array.

import numpy as np

arr = np.array(42)

print(arr)

1-D Arrays

An array that has 0-D arrays as its elements is called uni-dimensional or 1-D array.

These are the most common and basic arrays.

import numpy as np

arr = np.array([1, 2, 3, 4, 5])

print(arr)

2-D Arrays

An array that has 1-D arrays as its elements is called a 2-D array.

These are often used to represent matrix or 2nd order tensors.

NumPy has a whole sub module dedicated towards matrix operations called numpy.mat

import numpy as np

arr = np.array([[1, 2, 3], [4, 5, 6]])

print(arr)

3-D arrays

An array that has 2-D arrays (matrices) as its elements is called 3-D array.

These are often used to represent a 3rd order tensor.

import numpy as np

arr = np.array([[[1, 2, 3], [4, 5, 6]], [[1, 2, 3], [4, 5, 6]]])

print(arr)

Check Number of Dimensions

NumPy Arrays provides the ndim attribute that returns an integer that tells us how many dimensions the array has.

Check how many dimensions the arrays have:

import numpy as np

a = np.array(42)

b = np.array([1, 2, 3, 4, 5])

c = np.array([[1, 2, 3], [4, 5, 6]])

d = np.array([[[1, 2, 3], [4, 5, 6]], [[1, 2, 3], [4, 5, 6]]])

print(a.ndim)

print(b.ndim)

print(c.ndim)

print(d.ndim)

We import the files using pandas libraries and we visualize the dataframes

xls = pd.ExcelFile('ProjectData.xlsx')

df1 = pd.read\_excel(xls, 'Project Data')

df2 = pd.read\_excel(xls, 'Maintenance Type A Cost')

print (df1)

print (df2)

We create a 2 d array for w

'''

w is 2D array where 1 represents the maintenance check of a plane at a city on

a day. Here the assumption is made that on every 4th day of the week the plane

takes a main check

'''

w=np.zeros([n\_p,n\_c+1,n\_d])

w[:,:,4]=1

print(w.shape)

#w

# = 1, if plane i takes the maintenance check at city j on day d (here d=4), 0 otherwise

#rows=plane

#cols=city + added 0 city

#depth=day

We create a 2 d array for g

## '''

## g is an 2D array that provides the maintenance cost for a plane at a city.

## '''

## g=df2.drop(columns=['Aircraft']).to\_numpy()

## print(g.shape)

## g

## 

## Demonstrations of the functions

We also use a subroutine for creating a network

def create\_network(df\_new):

A=np.zeros([n\_c+1,n\_c+1,n\_d])

for i in range(n\_d):

df=df\_new[df\_new['Day of Week']==i].copy()

df=df.reset\_index(drop=True)

for i,li in enumerate(df['Origin City']):

A[df['Origin City'].iloc[i],df['Destination City'].iloc[i]]=1

return A

The time complexity for this is O(num\_cities\*num\_days). This is because we are creating a data structure by running 2 for loops running from the length of number of cities and number of days.

We create a subroutine called dfs.

'''

dfs function outputs a cyclicity, cost and route path for a week's schedule. It

performs recursive depth first search optimization algorithm to minimize the

cyclic cost of the airplane's schedule.

Here the cost for scheduling a plane on any route is considered homogenous, with

a constant value of 0.

'''

INF = 10e10

def dfs(n\_p,ori,k,d,network\_array):

if d==6:

path=[]

if network\_array[ori][k][d]==1:

path.append(k)

cost=0

return True,cost,path

else:

return False,INF,path

else:

curr\_cost = INF

is\_cyclic = False

curr\_path = []

for i in range(15):

ind = i

if network\_array[k,i+1,d]:

#print(network\_array[k,i,d])

p\_cyclic, p\_cost, p\_path = dfs(n\_p,ori,i+1,d+1,network\_array)

if p\_cyclic == True:

is\_cyclic = True

if p\_cost <= curr\_cost:

ind = i

curr\_cost = p\_cost

curr\_path = []

curr\_path = p\_path

curr\_path.append(i+1)

if d==4 and is\_cyclic==True:

curr\_cost += g[n\_p,ind-1]

return is\_cyclic, curr\_cost , curr\_path

Time Complexity of the algorithm is O((number of correct paths available in the network) \* 7 ). This changes from network to network. This is justified because we have to check for each and every possibility and come up with a best path.

We created a last iteration for the final planning schedule:

num\_iter = 5000

for iter in range(num\_iter):

start = time.time()

network\_array\_copy\_1 = network\_array

final\_array\_copy\_1 = final\_array

objective\_func\_1 = 0

planning\_schedule\_1=[]

for n in range(0,n\_p):

cyclic=False

k=1

while cyclic!=True:

d = 0

while np.sum(network\_array\_copy\_1[k,:,d])==0:

k = k+1

cyclic,cost,path = dfs(n,k,k,d,network\_array\_copy\_1)

if(cyclic):

planning\_schedule\_1.append(path)

for i in range(len(path)-1):

final\_array\_copy\_1[n,path[i],path[i+1],d]=1

network\_array\_copy\_1[path[i],path[i+1],d]=0

d+=1

objective\_func\_1 += cost

network\_array\_copy\_2 = network\_array

final\_array\_copy\_2 = final\_array

objective\_func\_2 = 0

planning\_schedule\_2=[]

temp = np.array([i for i in range(n\_p)])

random.shuffle(temp)

for n in temp:

cyclic=False

k=1

while cyclic!=True:

d = 0

while np.sum(network\_array\_copy\_2[k,:,d])==0:

k = k+1

cyclic,cost,path = dfs(n,k,k,d,network\_array\_copy\_2)

if(cyclic):

planning\_schedule\_2.append(path)

for i in range(len(path)-1):

final\_array\_copy\_2[n,path[i],path[i+1],d]=1

network\_array\_copy\_2[path[i],path[i+1],d]=0

d+=1

objective\_func\_2 += cost

if objective\_func\_1 <= objective\_func\_2 :

objective\_function = objective\_func\_1

final\_array = final\_array\_copy\_1

network\_array = network\_array\_copy\_1

planning\_schedule = planning\_schedule\_1

else:

objective\_function = objective\_func\_2

planning\_schedule = planning\_schedule\_2

final\_array = final\_array\_copy\_2

network\_array = network\_array\_copy\_2

end =time.time()

print('Ran successfully for iteration ',i,"and time taken till this iteration is ",(end-start))

# Final results

The result of this is the final planning schedule for the planes

Here the rows are the planes and the columns are the days of the week.

**1 2 3 4 5 6 7**

**1 11 11 14 13 14 12 13**

**2 8 5 6 3 13 9 12**

**3 13 7 1 9 11 5 8**

**4 12 5 10 4 6 8 11**

**5 5 4 9 6 10 7 8**

**6 1 13 14 0 9 10 8**

**7 4 14 11 8 1 12 10**

**8 5 6 7 0 10 12 13**

**9 13 8 5 4 9 10 12**

**10 9 10 0 7 12 8 1**

**11 9 1 0 5 8 14 11**

**12 8 7 6 10 2 1 3**

**13 10 2 6 1 12 14 9**

**14 11 6 8 2 4 13 1**

**15 14 8 4 7 6 1 2**

**16 5 2 11 0 1 8 13**

**17 9 4 14 0 13 12 10**

**18 6 14 0 13 12 1 2**

**19 0 11 7 10 5 13 6**

**20 6 5 14 8 7 2 9**

**21 8 4 12 3 10 5 9**

**22 6 11 13 4 2 1 5**

**23 7 10 9 14 8 0 1**

**24 0 2 4 5 9 6 13**

**25 1 14 12 4 6 10 5**

The weekly maintenance for each aircraft is as follows:

1 Aircraft = 1,

2 Aircraft =2,

3 Aircraft =14,

4 Aircraft =8,

5 Aircraft =5,

6 Aircraft =3,

7 Aircraft =7,

8 Aircraft =2,

9 Aircraft =8,

10 Aircraft =14,

11 Aircraft =5,

12 Aircraft =6,

13 Aircraft =7,

14 Aircraft =14,

15 Aircraft =7,

16 Aircraft =8,

17 Aircraft =6,

18 Aircraft =2,

19 Aircraft =10,

20 Aircraft =3,

21 Aircraft =5,

22 Aircraft =8,

23 Aircraft =7,

24 Aircraft =5,

25 Aircraft =8

The final value of the objective function or the total overall weekly cost for all the is:

**207**