ModSim Project 1

October 4, 2019

Modeling Flight Delays

The Question

What is the best way to increase the number of flights without delays? We will model airplane traffic between several airports and test two different modeling strategies to avoid flight delays and maintain flight turnaround efficiency.

In [1]:
```
# Configure Jupyter so figures appear in the notebook
%matplotlib inline

# Configure Jupyter to display the assigned value after an assignment
%config InteractiveShell.ast_node_interactivity='last_expr_or_assign'

# import functions from the modsim library
from modsim import *

# set the random number generator
np.random.seed(7)
import random

import pandas as pd
import datetime
from dateutil.parser import parse
import math
import numpy as np
```

Below is data collected in 2008 which details flights and delays. This data was narrowed to include only Delta (DL) and United (UA) flights between airports LAX, JFK, ATL, IAD, SEA. By using only flights between specific airports, we reduce the likelihood that the data is influenced primarily by the airport or the airline.

In [2]:
```
trips = pd.read_csv('2008.csv')
```

Out[2]:

	Year	Month	DayofMonth	DayOfWeek	DepTime	ArrTime	UniqueCarrier	\
0	2008	1	1	2	613.0	1407.0	UA	
1	2008	1	2	3	615.0	1435.0	UA	
2	2008	1	3	4	607.0	1454.0	UA	
3	2008	1	4	5	618.0	1523.0	UA	
4	2008	1	5	6	615.0	1416.0	UA	

```
...    ...   ...        ...        ...      ...      ...          ...
10152  2008     2         29          5   2128.0   2311.0           DL
10153  2008     2         29          5   1858.0   2041.0           DL
10154  2008     2         29          5   1455.0   1646.0           DL
10155  2008     2         29          5    824.0   1002.0           DL
10156  2008     2         29          5    957.0   1147.0           DL

       ActualElapsedTime  AirTime  ArrDelay  ...  Origin Dest Distance  \
0                  294.0    278.0     -24.0  ...     LAX  JFK     2475
1                  320.0    298.0       4.0  ...     LAX  JFK     2475
2                  347.0    299.0      23.0  ...     LAX  JFK     2475
3                  365.0    284.0      52.0  ...     LAX  JFK     2475
4                  301.0    282.0     -15.0  ...     LAX  JFK     2475
...                  ...      ...       ...  ...     ...  ...      ...
10152              103.0     77.0      -2.0  ...     ATL  IAD      533
10153              103.0     79.0       0.0  ...     ATL  IAD      533
10154              111.0     78.0       5.0  ...     ATL  IAD      533
10155               98.0     78.0      -5.0  ...     ATL  IAD      533
10156              110.0     82.0      -2.0  ...     ATL  IAD      533

       TaxiIn  TaxiOut  CarrierDelay  WeatherDelay  NASDelay  SecurityDelay  \
0         3.0     13.0           NaN           NaN       NaN            NaN
1         3.0     19.0           NaN           NaN       NaN            NaN
2         8.0     40.0           0.0           0.0      23.0            0.0
3         3.0     78.0           0.0           0.0      52.0            0.0
4         4.0     15.0           NaN           NaN       NaN            NaN
...       ...      ...           ...           ...       ...            ...
10152     8.0     18.0           NaN           NaN       NaN            NaN
10153     7.0     17.0           NaN           NaN       NaN            NaN
10154     5.0     28.0           NaN           NaN       NaN            NaN
10155     4.0     16.0           NaN           NaN       NaN            NaN
10156     7.0     21.0           NaN           NaN       NaN            NaN

       LateAircraftDelay
0                    NaN
1                    NaN
2                    0.0
3                    0.0
4                    NaN
...                  ...
10152                NaN
10153                NaN
10154                NaN
10155                NaN
10156                NaN

[10157 rows x 21 columns]
```

The Model

To model flights and delays, we will use a state object which keeps a list of planes and also keeps track of ticks with the time variable. These variables are global but change throughout, so putting them in the state object makes sense. To simulate the planes themselves, a Plane class is created, which contains any variables for the planes and several functions to update them.

Our model, obviously, is more simple than a real-life airport system. We have limited our traffic to only a few airports, and a small number of planes. We have also decided to focus on airport delays–effectively ignoring in-flight delays due to weather, diversions, or other spontaneous circumstances.

In [3]:
```
planes = []
time = 0
state = State(planes = planes,time = time)
```

Out[3]:
```
planes    []
time       0
dtype: object
```

In [4]:
```
class Plane:

    def __init__(self, airline, inFlight, distance, target):          ## Initializes an instance of the Plane class
        self.airline = airline
        self.inFlight = inFlight
        self.distance = distance
        self.target = target
        self.wait = 0
        self.data = []

    def move(self):           ##the plane's movement tracker, which moves the plane towards its target by or
        if self.distance > 0:
            self.data.append(str(self.distance))
            self.distance -= 1
            return True
        else:
            return False

    def delay(self):           ##the plane's delay timer at airports, which counts down tick by one second if i
        if self.wait > 0:
            self.data.append(0)
            self.wait -= 1
            return True
        else:
            return False

    def go_to(self, target):      ##sets a new target airport for the plane, while also calculating the distance
        temp = self.target
        self.target = target
        self.distance = flight_time(temp,target)
```

```
    ##--------Getters---------##
    def getAirline(self):
        return self.airline
    def getInFlight(self):
        return self.inFlight
    def getDistance(self):
        return self.distance
    def getTarget(self):
        return self.target
    def getData(self):
        return self.data
    def getWait(self):
        return self.wait

    ##--------Setters---------##
    def setAirline(self,airline):
        self.airline = airline
    def setInFlight(self,inFlight):
        self.inFlight = inFlight
    def setDistance(self,distance):
        self.distance = distance
    def setTarget(self,target):
        self.target = target
    def setWait(self, wait):
        self.wait = wait

def flight_time(x, y):      #Outside the plane class, flight time calculates the time/distance in ticks between
    if (x == "ATL" and y == "LAX") or (y == "ATL" and x == "LAX"):
        return 51
    elif (x == "ATL" and y == "IAD") or (y == "ATL" and x == "IAD"):
        return 21
    elif (x == "ATL" and y == "JFK") or (y == "ATL" and x == "JFK"):
        return 28
    elif (x == "ATL" and y == "SEA") or (y == "ATL" and x == "SEA"):
        return 57
    elif (x == "LAX" and y == "IAD") or (y == "LAX" and x == "IAD"):
        return 59
    elif (x == "LAX" and y == "SEA") or (y == "LAX" and x == "SEA"):
        return 35
    elif (x == "LAX" and y == "JFK") or (y == "LAX" and x == "JFK"):
        return 66
    elif (x == "IAD" and y == "JFK") or (y == "IAD" and x == "JFK"):
        return 17
    elif (x == "IAD" and y == "SEA") or (y == "IAD" and x == "SEA"):
        return 70
    elif (x == "JFK" and y == "SEA") or (y == "JFK" and x == "SEA"):
        return 76
```

```
    else:
        return False

def delay_factor(baseNum, margin):    ##Adds an element of randomness to the delay, which can be adju
    rnd = random.randint(1,margin*2)
    return int((baseNum - (margin)) + rnd)
```

In [5]:
```
plane1 = Plane("UA",False,0,"LAX")
plane2 = Plane("DL",False,0,"ATL")
plane3 = Plane("UA",False,0,"LAX")
plane4 = Plane("DL",False,0,"ATL")
plane5 = Plane("UA",False,0,"LAX")
plane6 = Plane("DL",False,0,"ATL")
plane7 = Plane("UA",False,0,"LAX")
plane8 = Plane("DL",False,0,"LAX")
plane9 = Plane("UA",False,0,"ATL")
plane10 = Plane("DL",False,0,"LAX")
plane11 = Plane("UA",False,0,"ATL")
state.planes.append(plane1)
state.planes.append(plane2)
state.planes.append(plane3)
state.planes.append(plane4)
state.planes.append(plane5)
state.planes.append(plane6)
state.planes.append(plane7)
state.planes.append(plane8)
state.planes.append(plane9)
state.planes.append(plane10)
state.planes.append(plane11)
```

For comparison we are using two different models for airlines, assuming each has only 2 planes, going between 2 airports.

Delta Airlines (DL) will be using a model where 1 plane is kept in reserve. Any time delta experiences a significant delay (variable maxDelay), the reserve plane will be called in to replace the original, instantly resetting the delay to 0.

United Airlines (UA) will be using a model where all planes are always in service, flying opposite directions between the 2 airports. Since there is no reserve plane, United makes turnarounds longer to maintain planes and reduce the impact of delays. However, if one of their planes exceeds a significant delay (variable maxDelay), the flight is cancelled, and the plane must wait until the next scheduled flight. Since the planes fly between two airports, this means two previously scheduled flights are cancelled.

The data will be obtained in the form of a ratio, comparing the number of successful flights for each airline. The variables for maximum delays and turnarounds are designed to be as close to real life as possile based on research. Running the simulation usually takes upwards of 2 minutes because of the vast quantity of data being processed. We experimented with smaller time scales and numbers but this resulted in very varied outputs.

In [6]:
```
def run_simulation(numPlanes, air1,
                   air2):  # The run_simulation function runs the simulation
```

```
    state.time = 0
    DL = 0
    UA = 0
    planes = state.planes[0:((numPlanes * 2) - 1)]
    for x in range(100000):
        state.time += 1
        DL += sim1(planes, air1, air2, 22)
        UA += sim2(planes, air1, air2, 22, 5)
    return [DL, UA, DL / UA]


def sim1(planes, air1, air2,
         maxDelay):  # Sim1 implements Delta's reserve plane model
    success = 0
    for plane in planes:
        if plane.getAirline() == "DL":
            if not (plane.delay()):
                if plane.getWait() > maxDelay:
                    plane.setWait(0)
                if not (plane.move()):
                    success += 1
                    if (plane.getTarget() == air1):
                        plane.go_to(air2)
                    else:
                        plane.go_to(air1)
                    plane.setWait(delay_factor(15, 9))
    return success


def sim2(planes, air1, air2, maxDelay,
         addTurn):  # Sim2 implements United's model
    success = 0  # that operates without reserve planes
    for plane in planes:
        if plane.getAirline() == "UA":
            if not (plane.delay()):
                if not (plane.move()):
                    success += 1
                    if (plane.getTarget() == air1):
                        plane.go_to(air2)
                    else:
                        plane.go_to(air1)
                    delay = delay_factor(15, 9)
                    plane.setWait(delay + addTurn)
                    if delay > maxDelay:
                        success -= 2
    return success
```

```python
test1 = run_simulation(2, "IAD",
                       "JFK")  # This section collects data from run_simulation
test2 = run_simulation(
    2, "ATL", "LAX")  # and creates lists to store all the different datasets
test3 = run_simulation(2, "JFK", "SEA")
test4 = run_simulation(3, "IAD", "JFK")
test5 = run_simulation(3, "ATL", "LAX")
test6 = run_simulation(3, "JFK", "SEA")
test7 = run_simulation(4, "IAD", "JFK")
test8 = run_simulation(4, "ATL", "LAX")
test9 = run_simulation(4, "JFK", "SEA")
test10 = run_simulation(5, "IAD", "JFK")
test11 = run_simulation(5, "ATL", "LAX")
test12 = run_simulation(5, "JFK", "SEA")
test13 = run_simulation(6, "IAD", "JFK")
test14 = run_simulation(6, "ATL", "LAX")
test15 = run_simulation(6, "JFK", "SEA")

tests = [
    test1, test2, test3, test4, test5, test6, test7, test8, test9, test10,
    test11, test12, test13, test14, test15
]

DL_Flights = []
for test in tests:
    DL_Flights.append(test[0])

UA_Flights = []
for test in tests:
    UA_Flights.append(test[1])

ratio = []
for test in tests:
    ratio.append(test[2])

num_planes = [2, 2, 2, 3, 3, 3, 4, 4, 4, 5, 5, 5, 6, 6, 6]
flight_length = [1, 3, 5, 1, 3, 5, 1, 3, 5, 1, 3, 5, 1, 3, 5]
```

Out[6]: [1, 3, 5, 1, 3, 5, 1, 3, 5, 1, 3, 5, 1, 3, 5]

The Results

The ratios of Delta's successful flights versus United's succesful flights are shown below. For each flight path, there are four ratios–each representing a test with a different number of planes. For reference, the flight paths are in order of shortest time to longest.

In [7]:
```python
print(ratio)
```

[0.749, 0.6914498141263941, 0.6605504587155964, 0.986452998513134, 0.9151234567901234, 0.8929752066115703, 1