Genetic Algorithm for Optimal Travel Plan Preparation

Dr. C. Muthu
Associate Professor & Head
Department of Computer Science
Loyola College, Chennai
&
M. Arun Rajesh
Data Analyst, Shalom InfoTech,
Bharathidasan University Technology Park
Tiruchirappalli

Abstract

Stochastic Optimization Algorithms, such as the Genetic Algorithm, are often used by the Business Organizations to obtain the best solutions for their business problems. In this paper, a Genetic Algorithm is used to determine an optimal travel plan for an entire Music Troupe.

Keywords*:* Stochastic Optimization Techniques, Genetic Algorithm

Introduction

The collaborative filtering techniques are now often used in the field of data science.[1] The business problems that are related to the big data analytics are at present tackled with the help of the Hadoop Ecosystem.[2] The Data Analysts now use the KNN algorithm for the formulation of Price Predictors.[3] Hierarchical Clustering algorithms are at present used for getting valuable business insights into the customer preferences.[4] In this paper, a Genetic Algorithm, which tries to intelligently modify the solutions in a way that is likely to improve them, is used to determine an optimal travel plan for an entire Music Troupe.

Data Ingestion

In this paper, an optimal travel plan is determined for the members of a Music Troupe, who are coming from all over the country by flight to Mumbai, where they wish to conduct a grand Music Concert. They will all arrive on the same day and leave on the same day, and they would like to share transportation to and from the airport. There are dozens of flights per day to Mumbai from any of the Music Troupe's members' locations, all leaving at different times. The flights also vary in fare and in duration.

Planning a trip for a group of people from different locations all arriving at the same place is always a challenge, and it often requires an optimal solution. A lot of different inputs are to be ingested by the Optimization Algorithm in order to arrive at an optimal solution. Some vital inputs are required, such as what every one's flight schedule should be, how many cars should be rented, and which airport is easiest. Many outputs should also be considered, such as the total cost, time spent waiting at airports, and time taken off work. Because the inputs cannot be mapped to the outputs with a simple formula, the need for obtaining an optimal solution arises in this case.

To begin, a new Python program file called **tripOptimization.py** is to be created and the following code is to be inserted into it:

```
import time
import random
import math
troupe          =  [(‘IlaiyaRaja’, ‘MAA’), (‘Yesudas’, ‘TRV’),
                    (‘SubhaMudgal’, ‘BLR’), (‘SonuNigam’, ‘HYD’),
                    (‘AnuShankar’, ‘DEL’), (‘ShreyaGhosal’, ‘CCU’)]
destination     =  ‘BOM’
```

The sample data for this study are stored as **trip.txt** and this data file contains origin, destination, departure time, arrival time, and fare for a set of flights in a comma-separated format:

```
DEL,    MAA,    20:27,  23:42,  1690
MAA,    DEL,    19:53,  22:21,  1730
DEL,    BOM,    6:39,   8:09,    860
BOM,    DEL,    6:17,   8:26,    890
DEL,    BLR,    8:23,   10:28,  1490
BLR,    DEL,    7:04,   9:11,   1280
```

These data are loaded into a dictionary with the origin and destination as the keys and a list of potential flight details as the values. The following code to ingest the data is to be added to tripOptimization.py:

```
flights = { }
for line in file (‘trip.txt.’):
    origin, destination, departure, arrival, fare = line.strip( ).split(‘,’ )
    flights.setdefault ((origin,destination), [ ])
    #Details are added to the list of possible flights
    flights [(origin, destination)].append ((departure, arrival, int (fare)))
```

The utility function getTimeInMinutes( ) is also defined, which calculates how many minutes into the day a given time is. This makes it easy to calculate flight times and waiting times. This function is added to tripOptimization.py:

```
def getTimeInMinutes(t):
x = time.strptime(t, ‘%H : %M’)
return x[3] * 60 + x[4]
```

The challenge now is to decide which flight each person in the Music Troupe should take. Even though keeping the total fare down is a goal, there are many other possible factors that the optimal solution will take into account and try to minimize, such as the total waiting time at the airport or total flight time. These factors will have to be taken in to account through an appropriate Cost Function.

When approaching an optimization problem like this one, it is necessary to determine how a potential solution will be represented. A simple representation that is possible in this study is a list of flight numbers. In this case, each number can represent which flight a troupe member chooses to take, where 0 is the first flight of the day, 1 is the second, and so on. Since each Music Troupe member needs an outbound flight and a return flight, the length of this list is twice the number of troupe members. For example, the list

[3. 5, 4, 2, 3, 6, 3, 7, 2, 3, 4, 1]

represents a solution in which Ilaiya Raja takes the fouth flight of the day from Chennai to Mumbai, and the sixth flight back to Chennai on the day he returns. Yesudas takes the fifth flight from Trivandrum to Mumbai, and the third flight back.

Cost Function Construction

The cost function is the key to solve any optimization problem. The goal of our optimization algorithm is to find a set of flights that minimizes the cost function. The cost function in our Group Travel Optimization problem will involve the following variables:

Fare	:	The total fare of all the plane tickets.
Travel Time	:	The total time that everyone has to spend on a plane.
Waiting Time	:	Time spent at the airport waiting for the other members of the troupe to arrive.
Departure Time	:	Flights that leave too early in the morning may impose an additional cost by requiring travelers to miss out on sleep.
Car Rental Period	:	If the troupe rents a car, they should return it earlier in the day than when they rented it or be forced to pay for a whole extra day.

We have to now determine how much money that time on the plane or time waiting in the airport is worth. The following **tripScheduleCost( )** function takes into account the total cost of the trip and the total time spent waiting at airports for the various members of the troupe. It also adds a penalty of Rs.250 if the car is returned at a later time of the day than when it was rented. The following **tripScheduleCost()** function is to be added to **tripOptimization.py**:

```
def tripScheduleCost (soln):
    totalfare = 0
    latestarrival = 0
    earleistdeparture = 24 * 60
    for d in range (len(soln)/2):
        #Get the inbound and outbound flights
         origin = troupe [d][1]
         outbound = flights [(origin, destination)][int(soln[d])]
         returnf = flights [(destination, origin)] [int(soln[d+1])]
        #Total fare is the fare of all outbound and return flights
         totalfare += outbound[2]
         totalfare += returnf[2]
        #Track the latest arrival and earliest departure
         if latestarrival < getTimeInMinutes (outbound [1]):
         latestarrival = getTimeInMinutes (outbound[1])
         if earliestdeparture> geTimeInMinutes (returnf[0]):
         earliestdeparture = getTimeInMinutes (return[0])
        #Every person should wait at the airport until the latest person
         arrives.
        #They also should arrive at the same time and wait for their
         flights.
         totalWaitingTime = 0
         for d in range (len(soln)/2):
         origin = troupe [d][1]
         outbound = flights[(origin, destination)] [int(soln[d])]
         returnf = flights[(destination, origin)] [int(soln[d+1])]
         totalWaitingTime += latestarrival – getTimeInMinutes
         (outbound[1])
         totalwaitingTime += getTimeInMinutes (returnf[0] – earliest
         departure
        #Check whether this solution requires an extra day of car
         rental
        #That will be Rs. 250!
         if latestarrival > earliestdeparture: totalfare += 250
         return totalfare + totalwaitingTime
```

Here, the total waiting time is calculated based on the assumption that all the Music Troupe members will leave the airport together when the last person arrives, and will all go to the airport for the earliest departure. This **tripScheduleCost()** function shall now be tried in the following Python session:

```
>>> reload (tripOptimization)
>>> tripOptimization.tripScheduleCost(s)
    52830
```

Optimal Travel Plan Determination

The goal of our optimal travel plan determination problem is to minimize the cost by choosing the correct set of flight numbers. Genetic Algorithms work by initially creating a set of random solutions known as the population. At each step of the optimization, the cost function for the entire population is calculated to get a ranked list of solutions. After the solutions are ranked, a new population - known as the next generation - is created. First, the top solutions in the current population are added to the new population as they are. This process is called elitism. The rest of the new population consists of completely new solutions that are created by modifying the best solutions.

There are two ways in which the solutions can be modified. The simpler of these is called mutation, which is usually a small, simple, random change to an existing solution. In our study, a mutation can be done simply by picking one of the numbers in the solution and increasing or decreasing it.

The other way to modify solutions is called crossover or breeding. This method involves taking two of the best solutions and combining them in some way. In this study, a simple way to do crossover is to take a random number of elements from one solution and the rest of the elements from another solution. A new population, usually the same size as the old one, is created by randomly mutating and heading the best solutions. Then the process repeats - the new population is ranked and another population is created. This continues either for a fixed number of iterations or until there has been no improvement over several generations. The **geneticOptimization( )** function shall now be added to **tripOptimization.py**:

```
def geneticOptimization (domain, costf, populationSize = 50, step = 1,
                         mutationProbability = 0.2, elite = 0.2
                         maximumIterations = 100):
        #Mutation Operation is performed
         def doMutation (vec) :
         i = random.randint (0, len (domain) – 1)
         if random.random ( ) < 0.5 and vec[i] > domain[i][0] :
         return vec[0:i] + [vec[i] – step] + vec[i+1:]
         elif vec[i] < domain[i][1] :
         return vec[0:i] + [vec[i] + step] + vec[i+1:]
        #Crossover Operation is performed
         def doCrossover(r1, r2):
         i = random.randint(1, len(domain) –2)
         return r1[0:i] + r2[i:]
```

```
#               The initial population is built
                population = [ ]
                for i in range (populationSize):
                vec = [random.randint(domain[i][0], domain[i][1])
                for i in range (len(domain))]
                population.append(vec)
               #To find the number of winners from each generation
                topelite = int (elite * populationSize)
               #Main loop starts here
                for i in range (maximumIterations):
                scores = [(costf (v), v) for v in population]
                scores.sort ( )
                ranked = [v for (s,v) in scores]
               #Starting with the pure winners
                population = ranked [0 : topelite]
               #Adding mutated and breed forms of the winners
                while len(population) < populationSize:
                if random.random( ) < mutationProbability:
               #Mutation is done
                c=random.randint (0, topelite)
                population.append(doMutation(ranked[c]))
                else:
               #Crossover is done
                C1 = random.randint (0, topelite)
                C2 = random.randint (0, topelite)
                population.append (doCrossover(ranked[c1], ranked[c2]))
               #Displaying current best score
                print scores[0][0]
                return scores[0][1]
```

In order to display all the flights that the Music Troupe members need to take in order to ensure the above mentioned optimum cost, the following **displayOptimalSchedule( )** function is to be added to **tripOptimization.py:**

```
def dispalyOptimalSchedule(r) :
for d in range (len(r)/2):
    name = troupe [d] [0]
    origin = troupe [d] [1]
    out = flights [(origin, destination)] [r[d]]
    ret = flights [(destination, origin)] [r[d+1]]
```

```
print '%10s %10s %5s – %5s $%3s %5s %5s $%3s'
      % (name, origin, out[0], out[1], out[2], ret[0], ret[1],
      ret[2]) '
```

The **geneticOptimization()** function shall now be executed in order to optimize the group travel plan of the Music Troupe by using the Genetic Algorithm.

```
>>> reload (tripOptimization)
>>> s = tripOptimization.geneticOptimization (domain,
    tripOptimization.tripScheduleCost)
>>> tripOptimization.tripScheduleCost(s)
    22870
```

The optimal group travel plan for the entire Music Troupe obtained by using the Genetic Algorithm shall now be displayed by executing the **displayOptimalSchedule( )** function in a Python session:

```
>>> tripOptimization.dispalyOptimalSchedule(s)
    Ilaiya Raja Chennai 11:44-14:12 Rs 1090 10:53-12:23 Rs 7300
    Yesudas Trivandrum 11:32-15:59 Rs 2900 11:54-15:19 Rs 2660
    Shubha Mudgal Bengaluru 10:57-13:40 Rs 1890 10:42-13:26 Rs 1290
    Sonu Nigam Hyderabad 10:29-13:41 Rs 2480 11:47-14:15 Rs 1800
    Anu Shankar Delhi 10:54-12:27 Rs 1340 10:53-13:31 Rs 1220
    Shereya Ghosal Kolkata 11:28-13:27 Rs 1750 14:17-16:31 Rs 1390
```

References

1. Jacques Bughin, “Big Data, Big Bang?”, *Journal of Big Data*, 2016,Vol.3, Iss. 2, pp. 1-14.
2. Muthu, C. and Prakash, M.C., “Impact of Hadoop Ecosystem on Big Data Analytics”, *International Journal of Exclusive Management Research*, Special Issue, 2015, Vol. 1, Iss. 1, pp. 88-90.
3. Muthu, C. and Prakash, M.C.,“Building a Price Predictor for an Auctioning Website”, *ReTeLL*, 2015, Vol. 15, Iss. 1, pp. 135-137.
4. Muthu, C. and Prakash, M.C., “Hierarchical Clustering of Users’ Preferences”, *ReTeLL*, 2016, Vol. 16, Iss. 1, pp. 135-136.
5. Muthu, C. and Prakash, M.C., “Matching the users of a Website using SVM Technique”, *ReTeLL*, 2017, Vol. 17, Iss. 1, pp. 53-56.
6. Muthu, C. and Prakash, M.C., “Using Bayesian Classifier for Email Sorting”, *ReTeLL*, 2017, Vol. 17, Iss. 1, pp. 57-60.