Project Report

DIAL A RIDE CAB-SHARING SCHEDULING PROBLEM

November 11, 2013

Ashutosh Trivedi
MT2013030
IIIT Bangalore
`ashutosh.trivedi@iiitb.org`

Algorithm for Dial a Ride

Abstract

Dial a ride is a cab sharing problem in which the system has to give the complete schedule of the given cabs to maximize the total revenue.Cabs can share the passengers according to the capacity given to the system. The input to the system are nodes of the city in form of matrix, location of cabs at midnight,capacity of cabs and all the requests. The system has been developed for off line request processing so the total requests are already known to system and no dynamic request allowed. The output of the system will give the path of the cab as locations where it has served a request and time when for the same. Following is the report on the implementation with all the data structure and methods used for developing the algorithm.The programming language used for developing the algorithm is Java.

Introduction

A high level Introduction of the algorithm is as following. The actual functionality of each function is explained in later sections.

```
1.Read the input file and load into respective data structures

2.Find the shortest path between each node
  2a.calculate shortest path matrix by Floyd Warshall algorithm
  2b.Also get the path matrix which return all the nodes in the
      shortest path

3.Calculate revenue of each  request

4.Mark all the long (high revenue) requests and sort the requests
    according to it

start the time of clock
currentTime = 1;

5.while(currentTime <maxTime)
{
  6.get all the valid request of this time

  7.check for which cab these requests will suit
    7a.For each request identify the available cabs in
    7b.Based on filters and different criteria select the best cab
    7b.If no cab satisfy the criteria than leave the request

  8.update details of each cab

  9.update current time
  currentTime = currentTime+1;

}
```

Listing 1: Overview of the Implemented Algorithm

As we can see that after getting shortest path matrix the main two functions are how to match request to a cab and what are the criteria to select and leave a request based on cab details.

Data Structures

There are three data structures has been developed for the algorithm. First one to keeps track of all the details with respect to request second one keeps all the details of cabs and third one is consist of both the cab and request data structures. It will keeps all the details of sharing, as which cab has served which request it.

- **Request Model** -Following are the details of the Request model:

```
reqID          Request ID based on the revenue of the request.
origin         origin of the request
dest           destination of the request
tofReq         time of request
tofExp         time of expiary
length         length of the request default 2
rev            revenue of the request default 0
status         status of the request (open and close) "O"/ "C"
pic            the time at which request is picked up at source
drop           the time at which it is dropped at destination
```

Listing 2: Request model data structure

- **Cab Model** -Following are the details of the Cab model:

```
CabID               A number given to each cab starting from zero.
location            Location of the cab at it Timestamp
timestamp           TimeStamp of the cab
occupancy           No of passanges in the cab
Next Destination    It stores the node where cab will go next
Final Destination   Towards which destination the cab is headed to.
timer               The time for which cab is sitting ideal
```

Listing 3: Cab model data structure

- **Sharing Model** -Following are the details of the Sharing model:

```
This data structure has both request and cab structure to store the
    details of which cab has served which request

Occupancy       Occupancy of the cab at the time of picking up the
                request.
status          R: Running C: Closed

Request Model
Cab Model
```

Listing 4: Sharing Model data structure

It is clear from the above that schedule of each cab will be stored in sharing cab data structure. which will store that which cab has served which request and at what time and what was the occupancy of the cab at the timing of picking that request.

Methods and Algorithms

Following are the important algorithms used for the implementation of the project.

- **Floyd Warshall** - Floyd Warshall algorithm is used for finding shortest path matrix and nodes between source and destination. Function **pathNodes** takes input as start node, end node, and path matrix calculated during floyed Warshall algorithm and return the array of all the nodes in the shortest path.

- **Get Valid Request** - Following is the method to get a valid requests at current time

```

For each request {

if ( (Time of request < currentTime) && (Time of Exp > currentTime )
&& (request Status != 'C')    )
Than add to valid request List

}
//return all the valid request of current time
return List
```

Listing 5: get Valid request method

- **Get the best cab for each request** - This method is one of the most important or we can say the engine of the whole system which select the best cab to serve the request. If no cab is free at that time or Request does not suit to any of the cab the request will be dropped. The algorithm is as follows.

```
//The idea is that for each request system will search all the
cab which can reach at request location within 60 min.

suitedcab <- List of cabs which can reach at the request location
    within 60 min

For each valid request {

  if ( ( TimeOfExp - CabTimeStamp)< 60min ) and ( cabOcuupancy <
      Occupancylimt )
  add to suited cab List
  }

 Now for each suitedCab {

  // if cab is empty and request is Marked Large or Request originate
      at the  location of cab at that time than its the best suited cab
       for the request

  if( CabOccupancy ==0 )
    if(request marked 'Long') or (request originate at the cab
        location)
     best cab for the request

  if(Cab is more than half filled)
    if(request marked 'Long') or (time to reach at request location
        <20min)
    best cab for the request

  //if cab is already occupied, than it has to check weather it will
      be good to serve the request or not
  if(CabOccupancy >0 )
```

```
    if cab->destination  lies in the shortest path of new request
      best cab for the request

    if new req->destination lies the shortest path of cab
      best cab for the request

    if cab is marked 'Long' means a large request with large revenue
      Serve the request
```

Listing 6: Finding the best cab for the request

Any cab and request fall under the above criteria will be matched together.It is 2 way filtering, suppose a cab can be very near to the request location but it may not be the optimum. So if we have find some closest cabs we have to also check that if the request is worth picking up at that time or not.

- **Update cab details**- This part of the algorithm is actually managing the each cab in different time and at different place. As the algorithm is running for all the cabs every time. How we update the cab details is as follows.

```
//as soon as the cab is matched to a request

t <-- Time for a cab to reach at the request location

//add t to Timestamp of the cab
Currentcabtime = t+CabTimestamp

//if request was valid at CurrentCabTime  than we can just pick up the
     request at CurrentCabTime because cab is there at this time

update cab timestamp as CurrentCabTime

//If request was not valid at CurrentCabTime It means cab has reached
    the request location before the request has occure

update CurrentCabTime as the requestTime

//Cab has waited at that location untill the request has been occure

update cab location

update cab occupancy

update next and final destination

update revenue of the cab

close the request

Add to sharing model
```

Listing 7: Update the details of the cab

The most important thing in above method is how we update the TimeStamp of a cab. we get the valid requests in current time but a cab's TimeStamp can go beyond the current time or lag behind. In this way we can see the requests ahead in time and reach there in well advance if cab's time is behind the current time. As we have all the request available so we can use that information to model this idea.

- **Moving a idle Cab**- If a cab is unoccupied for 60 min at one place. we can check this condition well in ahead because cab time and current time is different. So we can check that till current time that there is no request for the cab, so we can move the cab in some suitable place without waiting for 60 min. Algorithm for this task is as follows.

```
start a timer for the cab
timer=0

if (cab Occupancy == 0)
{
  //update timer
  timer++
}

if(Timer>60 )
{
  //Move cab to a better place
  MoveCab(cab)
}

Movecab()
{
  check which requests are valid for cab Timestamp

  if(request are close (within 100 min ) && cab can reach there in
      time

    send cab to that direction

    update next and final destination
}
```

Listing 8: Moving a idle sitting Cab to some other location

- **Other Methods-** There are some more methods and conditions have been added to increase the outcome of the algorithm. Like the cab will not go to drop the passengers until the capacity of the cab is not reached a certain limit. Or cab is not waiting at a same place not more than 60 min. Marking 'long' requests and giving them preferences will also help in increasing revenue.

- **Input / Output Display-** Output will be given as the schedule of each cab revenue generated by each cab and total revenue generated. For each cab the output will in format **(Location, Time, Pick/Drop)** sorted based on time. It will give spatial-temporal of each cab. The first argument is input file name. Output file name can be given as second argument to the program. A sample output file is also attached with the document.

Discussion

This problem is dependent on various constraints and also data dependent upto some extent. For off-line processing of the requests the idea of running the cabs in some time lag with the current time really gives an edge over dynamic request processing. As we can really get to know weather a request is going to be there in future at a particular location. If not we can move the cab to some better location for maximizing the revenue. This notion can be extended as 2 fold or 3 fold or multiple fold to get the optimum solution. There can also be improvements on moving a idle cab to a better locations.

Reference

1 T-Share: A Large-Scale Dynamic Taxi Ridesharing Service Shuo Ma, Yu Zheng, Ouri Wolfson University of Illinois at Chicago, Chicago, USA sma21@uic.edu wolfson@cs.uic.edu Microsoft Research Asia, Beijing, China

2 CABSHARING: AN EFFECTIVE, DOORTODOOR,ONDEMAND TRANSPORTATION SERVICE Gyozo Gidofalvi1 , Torben Bach Pedersen Geomatic Aalborg ApS, Nybrogade 20, 1203 Copenhagen K, Denmark, tbp@cs.aau.dk