Dial a Ride

Project Report

Author: Joshi Dnyanesh Madhav (MT2013061)

I. Prologue

This project is implemented in C. The design is object-based (structure). It is made modular which facilitates maintenance and makes future changes to the scheduling algorithm easy as it is quite separated from the rest of the program.

II. Data Structures

i. Every location has an array of 1440 linked lists. Each of the 1440 lists at a location holds requests active at the minute-of-day identified by the index of the list in the array. The requests in a list are kept sorted based on their fares. (The first node would hold the request with the largest fare.)
ii. Every vehicle has a queue of LocationMinuteSeatsTriple objects each holding the location, the minute-of-day and the list of passengers that the vehicle would pick at the location at the minute-of-day (in addition to any passengers it is carrying already.) This queue basically holds the path the vehicle is going to take.

III. The Shortest Path Finding Algorithm

The Bellman Ford algorithm is used to find the shortest paths between all nodes (location) to all nodes. One reason for the choice is that the graph is sparse, so time complexity isn't an issue. Another reason is it facilitated populating the location objects.

IV. The Scheduling Algorithm

Note 1: The term passenger is used synonymously with a request that's served.
Note 2: A vehicle always takes the shorest path between any two given locations. It will never deviate. So all passengers will be carried always by the shortest path to their destination.
Note 3: A vehicle will serve only that request a path from whose source to destination exists and which has not already been served by some other vehicle.
Note 4: Every vehicle has a queue of LocationMinuteSeatsTriple objects which gives the path the vehicle is going to take along with the related details. The queue ADT is used here only from an intutivity and convenience perspective. It does not serve any other purpose. I could as well have used a linked list.
Note 5: The algorithm is recursive. After it finishes, the queue of LocationMinuteSeatsTriple objects of each vehicle holds the details of the path the vehicle will take. The algorithm also populates other objects such as requests, vehicles etc. with their relevent details.

The core idea behind the scheduling algorithm is as follows:

> *A vehicle, on its way to a particular must-visit-location (which is decided by the passenger it is currently carrying who is going the farthest), will pick passengers on the way the path to whose destination coincides with the vehicle's current path if there are no path conflicts. As the vehicle picks each passenger, its must-visit-location gets updated accordingly if needed.*
>
> *An empty vehicle is allowed to halt at the location where it becomes empty in order to catch up just in time with a profitable nearby passenger. (The vehicle may not have to go anywhere else if it can get a profitable passenger at the current location itself after a reasonable halt time.) It is this passenger who gives the empty vehicle its next must-visit-location.*

The scheduling algorithm given below concentrates on the scheduling aspect and leaves out any finer details that can be found in the code. The algorithm is as follows:

visit(vehicle, current-location, minute-of-day, must-visit-location)

{

1. Check the tail of the vehicle's LocationMinuteSeatsTriple queue to find out the last LocationMinuteSeatsTriple object that was enqueued. This object holds the list of passengers the vehicle would be carrying as it arrives at the current-location.
2. Create a new LocationMinuteSeatsTriple object corresponding to the current-location. Populate it with the current-location, the minute-of-day of the vehicle's arrival at the current-location and the passengers the vehicle is carrying who are not to be dropped at the current-location (so it would continue to carry them) and any suitable passengers that can be picked up at the current-location. Passsengers who are to be dropped off at the current-location are not added, so they are dropped automatically. Suitability of the passengers to be picked up at the current-location is decided by the following factors:
   i. Pick up a passenger if the vehicle isn't full, his destination can be reached before the 1440^{th} minute by the shortest path and the must-visit-location of the vehicle falls in the shortest path to his destination. If such a passenger is picked up, update the must-visit-location=his destination.
   ii. Pick up a passenger if the vehicle isn't full and his destination falls in the shortest path to the must-visit-location of the vehicle.
3. Enqueue the newly created LocationMinuteSeatsTriple object into the vehicle's queue.
4. If the must-visit-location is same as the current-location, that means the vehicle is going to become empty at the current-location. To find new passengers such that the unproductive time of the vehicle is minimized, do the following:
   i. Scan all locations (including the current-location) to find passengers whose start minute (when his request becomes active) is greater than or equal to the minute the vehicle would reach there from the current-location and who can be dropped before the 1440^{th} minute. Find out the passenger with the maximum (most positive) 'fare minus time to start minute' value among these. Set the must-visit-location=his pick up location. (Note that the passenger so found may have the current-location itself as his pick up location in which case the vehicle would simply have to halt at the current-location itself until his start minute.)
   ii. Halt the vehicle at the current-location itself for a time period equal to the time the vehicle would have to wait at the must-visit-location for the passenger after it reached there. After the halt time, send the vehicle toward the must-visit-location. Obviously, it would reach there exactly at the minute-of-day equal to the passenger's start minute. (Basically, the vehicle waits at the current-location instead of the must-visit-location for the same amount of time. The must-visit-location may very well be the current-location itself as mentioned in 4.i in which case the travelling time from the current to the must-visit-location will be 0.)
5. If must-visit-location!=current-location and all parameters to be passed are valid, call
   visit(vehicle,
   the next location on the shortest path from the current-location to the must-visit-location,
   the minute-of-day when the vehicle would reach the next location to pick the passenger,
   must-visit-location)
   else if must-visit-location==current-location and all parameters to be passed are valid, call
   visit(vehicle,
   current-location,
   the suitable passenger's start minute at the current-location,
   current-location)

}

A call to the function visit(...) for every vehicle should be made as follows:
visit(vehicle, vehicles's-base-location, start_minute, vehicles's-base-location)

V. Classmates with whom the project was discussed

None

VI. References

None