Busy Buses

Agustin Iniguez

November 14, 2021

This documents presents the solution taken to solve the problem Busy Buses. First it introduces the objective of the problem, the rules and the desired input data. Then it shows how it solves the problem and presents the results. Finally it briefly discusses the scalability of the algorithm

Objective

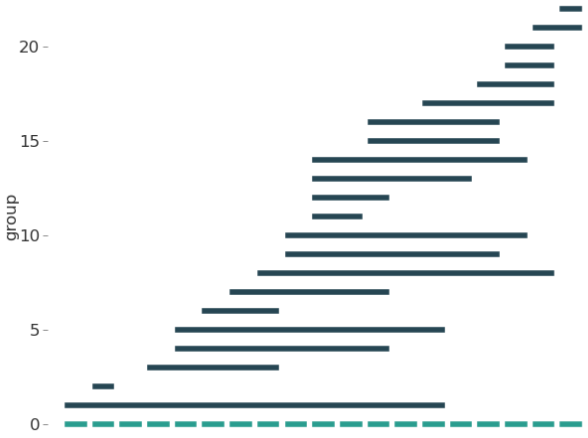
Given a certain bus route and groups of people that want to board the bus, the objective is to schedule the groups in order to maximize the ticket income. The ticket cost can be a fixed price per person or a variable price depending on the distance traveled.

Rules

The bus goes in only one direction, so groups cannot go back in the route. The groups cannot be divided. There is in average more people than what the bus can handle, so people cannot board the bus if the total amount of people in the group fills the bus above its capacity.

Input data

The input data is a group list. In this list the starting stop and exit stop is mentioned together with the amount of people in that group. These groups are selected one by one. First a group gets assigned with a random size of group (up to 10 persons in a group). Then the starting stop is selected from a uniform distribution of all stops. Afterwards, the exit stop is chosen from a uniform distribution of the remaining stops. This is repeated until there are in average more people per stop than the capacity. This gives a group distribution along the route shown in the graph.

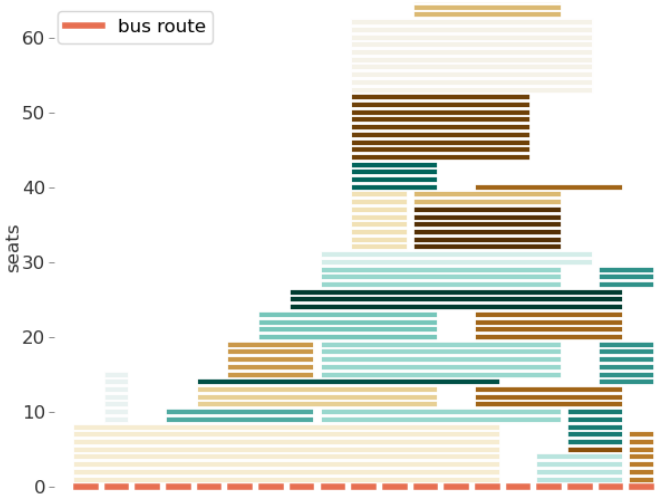


Solution

For the solution I considered different approaches, but in the end I decided for a simple greedy approach. The greedy algorithm first creates a schedule with all the groups as if the bus was infinite big. This is a interval partitioning scheduling. After, I select the seat with the lowest income from tickets and remove that seat. I repeat this until I have less or equal seats than the capacity.

Interval Partitioning Schedule

The schedule is done by looping through all the groups. For each group I check if there are available seats for them to seat. If not, I add seats to the bus in order to fit them. This creates a schedule for all the groups in an infinite large bus. This schedule is presented in the graph on the right.



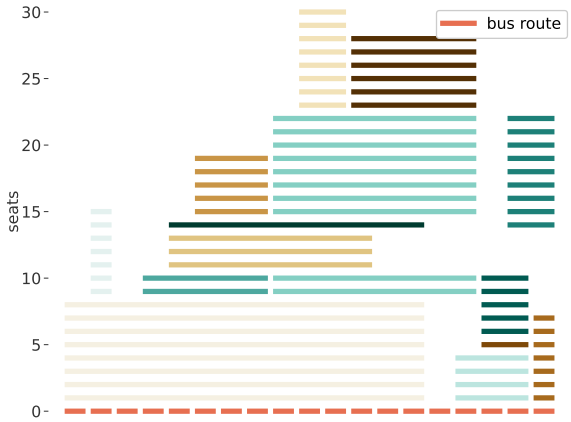
Removing seats

After creating a schedule with all the groups, I look at the seat that produces the lowest income. I select all the groups that are sitting in that seat and remove them from the schedule. Afterwards I remove all the empty seats. This of course can result in the removal of more than one seat.

After the removal of the seats is done, I loop through all the groups that are not allowed to board the bus and see if there is a gap in the remaining seats for them to board the bus. If there is, I add the group back to the schedule.

I repeat these two steps until I have less or equal amount of seats than the ideal capacity of the bus. Since the removal of the seats depends in the income from the tickets, and there are two ways that the tickets price varies, I obtain two different schedules for this bus route and this group list.

fixed ticket price



varying ticket price

Discussion

As the plots show, the schedule for both prices of the tickets are more or less equal. The difference lies on the two groups on top of the graphs. With fixed price the rout needs to take as many groups as it can, while with the varying price, the route needs to be as longer occupied as it can. I also plot the groups that are allowed to board vs the groups that are not for both ticket prices.

These graphs show precisely what we expect. The left one prefers to take groups that do not go far, while the one on the right prefers to take groups that travel longer distances.

Scalability

This algorithm, since it is a greedy algorithm, takes more time. Making the interval partitioning schedule takes O(n log(n)) where (n is the number of groups). Afterwards, removing seats and comparing the rejected groups in the schedule is another O(n log(n)) time algorithm, This leaves with a total (n^2 log (n)) time algorithms. With few groups it is not bad, but with the increase of the group number this becomes very demanding.

A way to solve this can be using different approaches. One could be by creating a graph, where the vertices are the groups and the connections exist if there is a schedule conflict between two groups. From this graph one can select the maximum independent set of vertices that are not connected. However, this will schedule only one seat.

Another way to solve it is to create a flow graph. The vertices are the stops and the directed connections are the groups and the bus route. This results in a flow problem that by weighting the connections with the ticket price, a maximum cost flow can be obtain. The downside is that it allows for partial flows, meaning division of groups.