Ride sharing problem

Vojtech Udrzal
Thursday 11:00
Open Informatics
udrzalv@gmail.com

***Abstract*—This school project looks into a ride sharing problem, proposes exact and greedy algorithms and by running simulations compares ride sharing with single rides on a real taxi rides data.**

I. Assignment

A. Problem Statement

We are given a list of 2-tuples indicating pickup and destination location (ride requests), a list of locations (current taxis locations) and a maximum factor by which a ride can be extended. Our goal is to combine rides in such a way, that the total amount of kilometers driven by all taxis is minimum possible. Shared rides can have different destination and pickup. We expect the input data not to change during the computation.

Additionally, we will consider complicating the problem in several ways. One could be by specifying a number of passenger ordering one ride and a number of seats in one taxi or by limiting the maximum distance from taxi to passenger pickup (thus limiting the maximum waiting time for taxi).

B. Problem Categorization

The specified problem is considered to be NP-hard. We can transfer the problem to finding a minimum Hamiltonian path problem, by having only one taxi as one node and then having other nodes as either pickup or destination points with directed edges between them. It is proved in the resource [1].

II. Related works

A need for dealing with a ride sharing problem has been increasing over the past years. That is mainly due to the expansion of ride sharing services and increasing popularity of mobile devices. However, there are many variations of ride sharing problem which can different in smaller or larger aspects. One of the popular ride sharing problem types is Real-time ride sharing. The nature of this problem lies in the need to reflect new incoming requests for rides and flawlessly incorporate them into already running computation. Furthermore, in real-time ride sharing services it is more important to reflect a waiting time of passengers than in basic ride sharing problem (such as when finding a shared ride for commuting to work). A large scale real-time ride sharing problem is examined in more detail in [2]. Another type of ride sharing problem is the already mentioned work commuting. It differs from the usual taxi ride sharing in a way that the driver and passenger roles are not set by the input. It is up to the algorithm to decide who should be the driver and who should be a passenger in order to minimize costs. The work commuting problem is discussed in [3]. The article [4] is describing ride sharing matching algorithm using adjusted Dijkstra's algorithm. Their solutions seems to be fast enough for production use. On the other hand, their algorithm does not count with passengers per car or car switching, for example.

III. Problem solution

A. IV. Integer Programming Approach

At first, we have tried to tackle this problem with exact ILP algorithm and in this section, we will describe our approach.

First, from a list of n ride orders (pairs of pickup and destination) we need to create a complete graph, which will have $2 * n$ nodes (V) and parameter $c_{i,j} = x$, where a distance from i to j is x. A pickup point p will have a destination node $p + n$. This graph is then provided to ILP model. Then we will create a variable $r_{i,j} = 1$ if there will be a ride from node i to node j. Our goal is to minimize total number of driven distance so our objective function will be equations 1 with following constraints.

$$min : \sum_{i,j \in V} r_{i,j} * c_{i,j} \tag{1}$$

A constraint 5 cannot be written as constraint directly since it is nonlinear. It can be converted to linear constraint by using big M and additional variables $y1$, $y2$ and $y3$ and allowing only one of them to apply. The equations 7, 8, 9.

$$\forall j \in pickups : \sum_{i \in V} r_{i,j} \leq 1 \tag{2}$$

constraint 1: Max one route to pickup

$$\forall i \in pickups : \sum_{j \in V} r_{i,j} = 1 \tag{3}$$

constraint 2: One route from pickup

$$\forall i \in destinations : \sum_{j \in V} r_{i,j} = 1 \tag{4}$$

constraint 3: One route to destination

$$\forall i, j \in V : r_{i,j} + r_{j,i} \leq 1 \tag{5}$$

constraint 4: No route there and back

$$\forall i \in pickups : r_{i,i+n} + \sum_{j \in V} r_{i,j} * r_{j,i+n} + \sum_{j,k \in V} r_{i,j} * r_{j,k} * r_{k,i+n} = 1 \tag{6}$$

constraint 5: Direct route, or route with one or two stops

The described ILP model works well in finding shared rides of complex types such as $PickupA$, $PickupB$, $DestinationA$, $DestionationB$ or $PickupA$, $PickupB$, $DestinationB$, $DestionationA$. However, the performance of this ILP model turned out to be very poor a GLPK solver finds a solution in a reasonable time (under a minute) for problems with $n < 7$.

$$\forall i \in pickups : r_{i,i+n} * M * y_{1,i} \geq 1 \tag{7}$$

Fig. 1: Linear direct ride

$$\forall i \in pickups, \forall j \in : r_{i,j} + r_{j,i+n} * M * y_{2,i\ ,j} \geq 2 \tag{8}$$

Fig. 2: Linear ride with one stop

$$\forall i \in pickups, \forall j, k \in : r_{i,j} + r_{j,k} + r_{k,i+n} * M * y_{3,i\ ,j\ ,k} \geq 3 \tag{9}$$

Fig. 3: Linear ride with one stop

B. Greedy algorithm with request stream

Since the exact algorithm did not show much promise we have decided to focus our strength onto a more promising approach which also corresponds to a reality settings more. In a real world, an algorithm does not have information about future ride requests and must accommodate an incoming request with the information available till that moment. Apart from more reasonable results and possibility of production deployment, accommodating one request at given time is also a task of easier complexity. The algorithm must pair the incoming request with one taxi in such a way, that the total traveled distance of all taxis will be minimum possible. Once the ride request is paired with a taxi it cannot be changed later.

1) Design: Upon receiving a new ride request r, the algorithm selects a subset of all taxis, which are within some time $t_{pickupMax}$ from $r.pickup$ location. Each taxi has a plan of pickup and destination points it needs to visit. The algorithm tries to insert the locations $r.pickup$ and $r.destination$ into each selected taxi's plan in such a way that it does not break any imposed constraints. Each location has a timestamp *latestArrival* and inserting a new stops into taxi's plan must not make the taxi arriving late to any subsequent stops. Furthermore, the number of passengers in a taxi must not exceed its capacity. For each pair of insertions of $r.pickup$ and $r.destination$ into a taxi's plan a new travel distance is calculated by equation 10 and an insertion with minimum extra travel distance is selected. The taxi with minimal insertion is then paired with the incoming request and its route plan is changed accordingly.

$$additionalDistance = distance(A, P) + distance(P, B) - distance(A, B) \tag{10}$$

With taxi's capacity 4, there can be at most $4 * 2$ (pickup and destination) locations in a taxi's plan. Then, theoretically, there is at most 81 posibilities of $r.pickup$ and $r.destination$ insertions which the algorithm needs to try in order to find the additional travel distance for each taxi. On average, there is about 15 taxis selected within time $t_{pickupMax}$ of $r.pickup$, which gives us about 1215 route distance queries for each ride request. In reality, we shorter the insertion search by trying to insert $r.destination$ only after $r.pickup$or by halting parts of the search once a *latestArrival* condition is violated. Yet, there is many routing queries which were slowing down our algorithm. Therefore, we had to implement route query caching which we describe later in implementation section.

2) Implementation: Finding a route between two locations is essential to our problem. To not be limited by quota, price or network delay of online routing APIs, we have installed *Open Source Routing Machine Project* [5] on our local machine. It is a HTTP service designed to find shortest paths on map data from the *OpenStreetMap Project*. With this step we have managed to decrease a latency of 500ms per request on online routing APIs down to 6ms per request on local computer. With this latency an average request pairing could take about 5 seconds, which could be already usable for a production usage with small number of requests per second. Yet, it would unnecessarily overuse the CPU and our simulations would run slowly.

To improve a performance of our algorithm, we have implemented a duration and distance cache. When trying the pickup and destination detours, the algorithm needs to know only durations and distances. Our cache is implemented as a hashmap of hashmaps and location's coordinates are rounded to 3 decimal places. This rounding can give an inaccuracy of about 150m of air distance, which is still acceptable, at least for simulation purposes. The average difference in duration of rounded vs. not rounded coordinates route was about 16 seconds. The biggest problem comes with pickup points near impassable areas, such as forests or Vltava river, where a small location difference can cause route longer by several kilometres. However, these situations occur mostly out of Prague, where there are no bridges around. When using the hashmap, a coordinate is encoded into an integer which is used as a key to the map. A departure location's key is used to the first hashmap, which returns a hashmap that we use with arrival location's key. If a duration for our locations is not found in cache, we query our local routing service and cache the result. After our program finishes, the hashmap is serialized into a file and loaded when the program starts next time. However, sometimes it is necessary to get a detailed route with all stops or turns, so some route queries are always executed on the local routing service. It is about 1 route query per ride request.

Implementing the ride share matching algorithm is just one part of our application. To test and compare ride sharing with usual single rides, a simulation of taxi movements corresponding to reality as much as possible had to be implemented. Therefore, for each taxi instance we keep two attributes: a route plan and a stops plan.

A route plan is used for quick calculation of current taxi's position and is implemented for the simulation only. It is a simple list, where location at index i and location at index $i+1$ are t_{delta} seconds apart. t_{delta} is a constant, we are using a 20 seconds interval. Then, when a new ride request arrives at time t, our simulator calculates the right index in each taxi's route plan list using the equation 11 and gives us the taxi's position at the time t. This search is very fast, therefore, we can easily execute it for each request and each taxi without slowing down the simulation.

$$getPositionAtTime(time) = routePoints[\lfloor (t_{timeFromStart} - t_{routePointsStart})/t_{delta} \rfloor] \tag{11}$$

A stops plan is a list of all stops that a taxi needs to visit. That is either a pickup or destination of some ride. It is used by our ride matching algorithm for finding the best place to insert a ride request's pickup and destination as described earlier.

When the algorithm finds the best taxi to server a request, it inserts the request's pickup and destination into the taxi's stops plan. Then, taxi's new route plan is created with the help of OSRM routing service.

While we tried to make our simulation as realistic as possible, we know about two factors which are a bit unrealistic. First one are traffic delays. In a real world, the delays on roads vary and make ride sharing more difficult, because several small delays can accumulate and result in a bigger delay for a later ride. Second unrealistic factor is the behaviour of a taxi after dropping of a passenger. In real world, especially if the drop off point is away from the city down-town, a taxi driver immediately turns and returns to the city. Unfortunately, we did not find a way how to simulate this behaviour and therefore, in our simulation the taxis wait on their last positions until they are assigned to a new ride.

maximum duration difference:	175s
$\bar{a}$ duration difference:	16s
% of differences over 60s:	0.04
maximum distance difference:	4237m
$\bar{a}$ distance difference:	188m
% of differences over 1km:	0.02

TABLE I: Route caching differences from reality

	single rides	ride sharing	improvements
ride requests:	627	627	-
served (not served) rides:	596 (31)	610 (17)	2% (82%)
shared rides:	-	312	-
paid distance/total travelled distance:	0.746	0.88	18%
total earnings:	124 812 CZK	129 065 CZK	4.6%
earning/km	20.9 CZK	24.7 CZK	18%
$\bar{a}$ ride length:	7.496 km	9.113 km	-21.5%
$\bar{a}$ pickup time	9 minutes	6.5 minutes	27%

TABLE II: Route caching error rate

IV. Experiments

A. Benchmark Settings

For testing and evaluation of our algorithm we use data of real taxi rides in Prague from Liftago. Liftago is Prague based company developing an application which connects passengers seeking a ride with available taxis. Our testing data consists of all rides in Prague for the past 3 months and for each ride we know an ordering time and pickup and destination locations. Since passengers count per order is not available (Liftago does not collect this information), we had to randomly assign a passengers count to each ride. We have decided to assign 1,2,3 and 4 passengers to 50%, 30%, 11% and 9% of rides respectively. Our simulations were ran on a laptop with 8GB of RAM and i7-4500U CPU.

We have selected a time period from 17:00 on Friday, 29th of April until 02:00 on Saturday, 30th of April on which all our simulations were ran. We have chosen this period since usually Friday nights are the most busy times and the data are the most recent. In this period there have been 627 rides completed in Prague which we run our simulation with.

B. Results

1) Route caching: The route caching is very important performance boost in our algorithm so we have decided to devote some time into measuring its performance. We have compared a difference between route duration and distance with rounded coordinates and with route without rounded coordinates. We have ran the test on routes between pickup and destination of orders within our time period. The results are displayed in table I. While running 627 real route queries on local routing service takes 3318ms, querying the cache still takes less than 1ms in total. Given the results and quite low error rate we think that this caching optimization is suitable for finding the best detours even in production use. However, we would suggest to implement one last reliable not cached route check (ideally with traffic information) before changing some taxi's route.

2) Ridesharing simulations: As we described earlier, the simulator has some factors which are different from reality. To fairly compare ride sharing with single rides we also implemented an algorithm which matches a ride request with a nearest free taxi. In this section we would like to present our findings about how ride sharing differs to usual single rides. Our simulations were ran with 100 taxis distributed over Prague.

It is also important to list our simulation enviroment settings. For calculating earnings we consider the price 28 CZK per kilometer. The maximum detour for a ride can be at most 90% of a direct route distnace. A pickup or destination stop is considered to take 2 minutes. Last but not least, we do not change the ride's price for passenger while ride sharing. We charge the passenger for direct route distance without any detours as we think it will be more transparent and fair. The benefit for passengers should come in overall decrease of tariff's price.

By looking at the data in table II, we can make several conclusions. We can see that ride sharing is performing better in most of the metrics we were tracking. By looking at the ratio of paid distance/total travelled distance we can say that ride sharing makes the taxis more efficient by 18%. This essentially increases the profit of taxi drivers per kilometre, from 20.9 CZK to 24.7 CZK.

It was expected that the average length of ride will be greater than without ride sharing, and we consider the increase of 21.5% to still be an acceptable detour for most of the passengers.

The reduction of pickup time by 27% was not expected and we think that in some cases it can compensate for the longer ride. We explain this reduction by making the taxis more flexible and thus available.

From a performance point of view, our simulation runs for 627 rides about 8 seconds. Given this result we think that our algorithm is ready to be used in production in Prague.

C. Discussion

Since the earnings of taxis per kilometre increased, we think it would be fair to split these savings between the driver and passenger to motivate passengers to use ride sharing. After running several simulations, we found that the average earnings of 22.8 CZK per kilometre (middle between 20.9 and 24.7) can be achieved with tariff 25.8 CZK per kilometre. As a result, a passenger on average ride of 210 CZK would save 18 CZK and driver's income would increase by 18 CZK if a ride sharing was introduced in a way we described.

We also tried to increase the maximum allowed detour from 90% of direct route distance to 200%. Such an increase brings efficiency improvement of 22% but we think that detours of this length would not be acceptable for most passengers.

We were also wondering how well is our simulator close to a reality. Since we do not have any real data for ride sharing we tried to evaluate our single ride simulation. The metric we found interesting to compare and thus verify correctness is the ratio between paid and total travelled distance. From surveys Liftago did with taxi drivers, the average is about 0.6 - 60% of their travelled distance is with passengers and 40% is without. Even though our simulation shows a ration of 0.74, we explain this by not simulating the movement of taxi after dropping off last passengers and waiting for next ride. In reality, these taxis usually return to downtown immediately and thus worsen their effectiveness ratio.

V. Conclusion

Based on our findings, we think that ride sharing would increase taxi transportation efficiency in Prague by 18%. While that is not insignificant improvement, we think that this is the upper limit - our best estimation. Real ride sharing brings other problems such as accumulated traffic delays which we did not consider in our simulation. Furthermore, it means a certain discomforts to passengers and drivers. We are not sure whether a decrease in price of 9% would be attractive to passengers, especially on short rides.

References

[1] Douglas O. Santos, Eduardo C. Xavier, *Dynamic Taxi and Ridesharing: A Framework and Heuristics for the Optimization Problem,* Web. 13 Mar. 2016. http://www.ijcai.org/Proceedings/13/Papers/424.pdf

[2] Yan Huang, Ruoming Jin, Favyen Bastani, Xiaoyang Sean Wang, *Large Scale Real-time Ridesharing with Service Guarantee on Road Networks,* Web. 27 Mar. 2016. http://arxiv.org/pdf/1302.6666.pdf

[3] Niels Agatz, Alan Erera, Martin Savelsbergh, Xing Wang, *The Value of Optimization in Dynamic Ride-Sharing: a Simulation Study in Metro Atlanta,* Web. 27 Mar. 2016. http://www2.isye.gatech.edu/ alerera/pubs/aesw10.pdf

[4] Roberts Geisberger, Dennis Luxen, Sabine Neubauer, Peter Sanders, Lars Volker, *Fast Detour Computation for Ride Sharing*.

[5] Luxen, Dennis and Vetter, Christian, *Real-time routing with OpenStreetMap data,* Web. 26 Apr. 2016. https://github.com/Project-OSRM/osrm-backend