Scram Applied to Car Sharing

Anonymous

Abstract

Carsharing programs are springing up in many major cities. These programs provide an alternative to private vehicle ownership. While they have seen growth there are still obstacles to wide scale adoption. Autonomous vehicles would be able to improve user access to vehicles and remove one of the main challenges these programs face. While the ability to easily move cars to meet demand would be significant for carsharing programs it could lead to worse system performance if the assignment of cars to users is not done in an intelligent way. In this paper we consider car sharing with autonomous vehicles as an assignment problem and examine different methods for matching cars to users to meet different criteria. We show how applying a recent algorithm for minimizing the maximal edge in a perfect matching can result in a more efficient carsharing system. Our results also highlight some of the problems with greedy or decentralized approaches.

Introduction

Autonomous vehicle technology is becoming a reality. Driverless cars have the potential to make travel safer, increase fuel economy, and reduce pollution (need citation). However the potential of this technology does not have to be limited to a new way to move people from point A to point B. Autonomous vehicles have the potential to revolutionize modern transportation systems (cite some AIM related work). One such system is carsharing.

Carsharing is a transportation service where users can access a personal vehicle when desired without the need to own a private vehicle. Users reserve vehicles which they can then use as a private vehicle. Service begins at designated car parking spots and must end at one of these spots. Current systems such as Car2Go or ZipCar have car parking spots scattered throughout the city to give users flexibility in trip origin and destinations. One problem with current systems is that vehicles may be moved from high demand to low demand areas. Then the vehicle is likely to remain unused for a long time before being used again. Additionally, users may not desire to walk to an access point or walk from

a dropoff location to their final destination. Both of these problems have hindered the growth of carsharing programs.

Autonomous vehicles offer a promising solution to these problems. An autonomous vehicle can meet a user at a requested position and drop them off at the user's desired destination. When not in use, the vehicle can move to other users or strategically relocate to an area with high demand (Fagnant and Kockelman 2014). Driverless cars can remain in service at all times (except for service or refueling) so the carsharing fleet remains close to full size throughout the day. And automated communication between cars can allow for coordination to improve system performance. This paper considers coordinating the assignment of vehicles to users to improve the system performance.

For an autonomous carsharing system to be used it needs to provide users service in a reasonable amount of time after they request a ride. It should also not give some users fast service at the expense of long waits for others. Miles traveled without passengers in the vehicle should be minimized as they are not directly serving trips but still contribute to pollution and add fuel costs. And the system must be reliable so we want all trips to be served.

While the experiments in this paper use a model of carsharing with autonomous vehicles, they may extend to a fleet of taxi cabs or ridesharing systems such as Uber or Lyft.

We seek to improve the performance of a fleet of shared autonomous vehicles through improved matching of vehicles to passengers requesting rides.

Car Sharing Model

For experiments we use the agent-based car sharing model proposed by Fagnant and Kockelman (Fagnant and Kockelman 2014). This model represents a 10 mile by 10 mile city as a grid of 0.25 mile by 0.25 mile grid cells. Vehicles can can move north, south, east, or west to adjacent cells. Motion within the cells is not modeled. Diagonal motion is not permitted and cars do not interfere with each other's path. Time is discretized into five minute intervals for a total of 288 time steps per day. Each run of the simulation corresponds to one 24 hour day.

Users requesting trips are generated in each grid cell according to a rate that decreases the farther a cell is from the city center. This rate is the mean for a Poisson process from which a number of trips is drawn for the corresponding cell.

The distance and start time for each trip is drawn from a distribution based on U.S. National Highway Traffic Safety Administration trip distance data. While all trips for the day are generated at the start of the run, requests for rides are not made until the start time of the trip.

At each time step available vehicles are matched to users requesting trips. If a user cannot be served they are added to a wait list and can request a ride again at the next time step. If a user cannot be served after 30 minutes of wait time the trip is considered unserved. Vehicles that could not serve trips because they were low on fuel go to the nearest fueling station (assumed to be within the same grid cell) and are out of service for two time steps (10 minutes). Then vehicles move to the passengers. Vehicles already with passengers proceed to their trip destination. Each vehicle can move a fixed number of grid cells depending on the current maximum speed.

We consider two schemes for determining the number of cars that are needed to provide adequate service. As in (Fagnant and Kockelman 2014), the number of cars can be determined by running the simulation for 20 runs with each run initialized with zero cars. The simulation cycle proceeds normally except after a trip has waited for 10 minutes a new car is generated at the start location of the trip. The number of cars generated is an estimate of the required fleet size. The average number of cars generated across these 20 runs is the fleet size. A final run repeats this procedure until the correct fleet size has been generated. Alternatively, since we wish to compare the sensitivity of different methods to variations in the fleet size, we can specify a fixed number of vehicles. In this case the above procedure can be repeated until the requested fleet size is reached and then no more cars are generated.

This model approximates the effects of a car sharing system in a 10 mile by 10 mile urban area. Car speed has small variations due to time of day and distance from the city center but is otherwise fairly constant. The simulation only models vehicles and passengers using the car sharing service. The presence of other modes of transportation is reflected in the trip generation rates and speed variations but these are only approximations. Nonetheless the model has been used in other car sharing studies (Fagnant, Kockelman, and Bansal 2015) to date and is realistic enough for the purpose of this work.

Matching Algorithms

Using the described carsharing model we want to determine how the assignment of vehicles to users affects the performance of the system. An easy way to handle the assignment problem would be to let users request the nearest vehicle to them on a first come first served basis. Unfortunately such simple approaches are not likely to be optimal in terms of the amount of waiting that users have or the number of trips that can be served. In addition, we would like to have a fair system in which some users are not waiting for long periods of time so that other users do not have to wait as long.
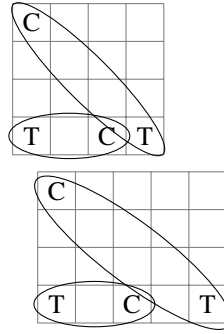

Figure 1: These small examples show the shortcomings of the various matching algorithms discussed in this paper. In the first example

Decentralized Greedy Matching Without a centralized trip-car assignment system trips would need to request the nearest vehicle, similar to how users of Uber or Lyft look for nearby vehicles. This decentralized method can be implemented as a greedy matching of trips to vehicles. In a random order, trips are assigned to the closest vehicle to them. If one is not available they are added to a wait list and will make requests at the next time step. We assume that passengers can request a vehicle from anywhere in the city. In practice this might be constrained to a local area to prevent cars from having to cross the entire city to reach a passenger. This would result in some passengers not receiving service due to no cars in the area. Figure **??** illustrates how this approach can result in inefficiencies.

Centralized Greedy Matching To improve on the decentralized greedy approach we consider a centralized greedy approach. Each trip looks for a car within a one cell radius of itself. After all trips have made this search the radius is increased and trips search again. The advantage of this approach is that a vehicle will not be assigned to a passenger if it could have served another passenger in less time. The disadvantage is that there are no guarantees that the assignment will be optimal. Figure **??** gives a scenario where this approach fails to produce the optimal solution.

This radius can either increase up to searching the entire city or terminate at a fixed limit (e.g. the distance a vehicle can travel in one time step).

Hungarian Algorithm A natural improvement is the Hungarian algorithm (Kuhn 1955) which finds a minimum cost perfect matching in a bipartite graph. The Hungarian algorithm can be implemented to run in time $O(n^3)$ which is fast enough to compute our car to passenger assignment in a few seconds. The input to the algorithm is a set of n cars and n trips and a distance metric to be minimized. Since we have an input of n cars and m trips we can create $|n - m|$ dummy passengers if $n > m$ or dummy cars if $m > n$. Setting the distance to a dummy car/passenger to zero results in the Hungarian algorithm returning the minimized distance for unequal m and n (MacAlpine, Price, and Stone 2015).

Minimal Makespan Matching The Hungarian algorithm produces a minimum cost perfect matching but the optimal solution does not consider fairness of the system. In our scenario this equates to the possibility that some trips have a much longer wait time even though the total wait across all trips is minimized. To improve on this we use a scalable collision-avoiding role assignment with minimal-makespan (SCRAM) algorithm to find an assignment that minimizes the longest distance that any vehicle must travel to a passenger. Specifically we implement the Minimum Maximal Distance + Minimum Sum Distance^2 (MMD+MSD^2) proposed in (MacAlpine, Price, and Stone 2015). Given a bipartite graph, $G = (S, T, E)$ this algorithm first finds the minimum maximal edge in a perfect matching from S to T. Then edges in E with length larger than this edge are removed from the edge set. Finally, the Hungarian algorithm is ran on this reduced set of edges to find the minimum cost matching.

Experimental Setup

To compare different approaches we run two sets of experiments. The first set of experiments compares the performance of the above algorithms over 100 consecutive days of fleet use. Since each the events of each day affect the next, multiple days are needed to get an idea of system performance. In these experiments the fleet size is set to 1000 vehicles which was experimentally found to be able to serve all users with any method of assignment. It is important that all trips be served in order to make meaningful comparisons. This is because an assignment strategy could reduce average wait time by simply not serving passengers that would have to wait a long time. We run 50 trials of 100 days each for each method of assignment.

The second set of experiments compares the ability of different methods to serve trips with different fleet size. Starting at a fleet size of 500 vehicles, fleet size is increased in increments of 100 and each method is ran for 100 days with each fleet size.

Results

This section presents and discusses the experimental results. Statistical significance is computed using a t-test for all results.

Unoccupied Miles Traveled

Passenger Wait Time

Sensitivity of Methods to Fleet Size

Conclusion and Future Work

We have presented a comparison of different methods for assigning vehicles to users for a carsharing system using autonomous vehicles. Our experiments show that employing an optimal perfect matching algorithm can improve performance of a system in terms of wasted travel and customer service.

The current approach is short sighted in that it only considers vehicles available at the time of assignment. Any of
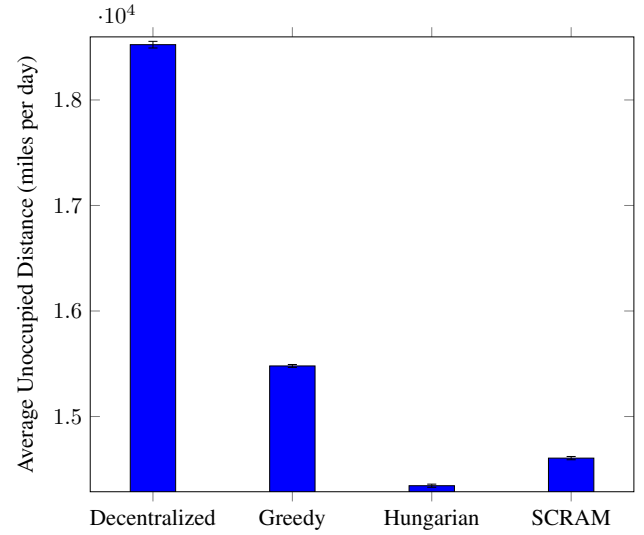

Figure 2: This bar graph shows how different assignment methods result in different amounts of unoccupied travel. Error bars are for a 95% confidence level.
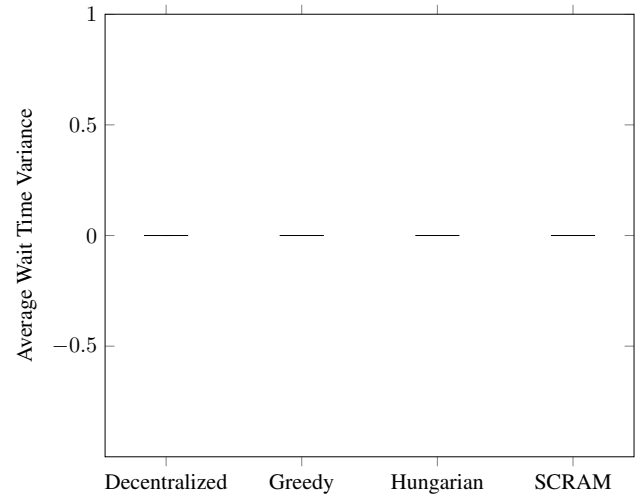

Figure 3: This bar graph shows how different assignment methods result in different amounts of variance within travel time. Error bars are for a 95% confidence level.
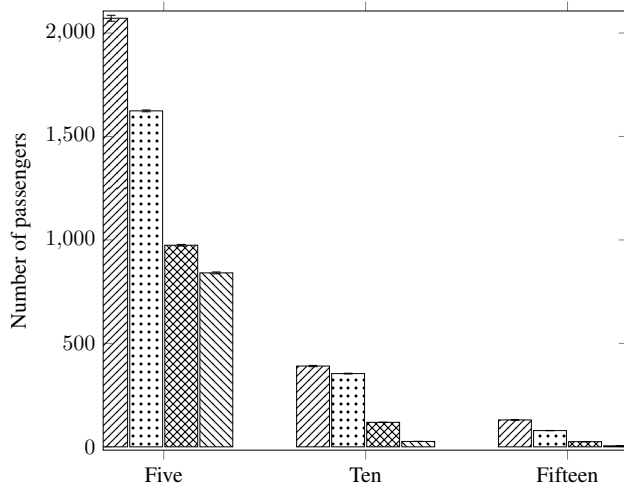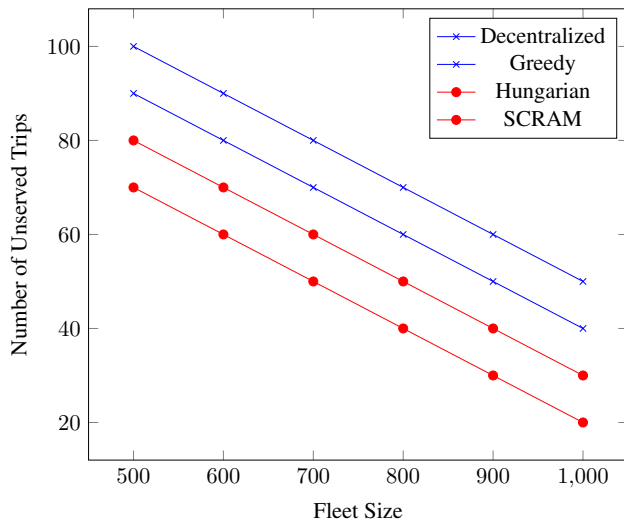
Figure 4: This bar graph shows how different algorithms can match more of the trips sooner. Error bars are for a 95% confidence level.

the assignment methods could be adapted to consider vehicles that will soon be available. This could be done by representing the vehicles distance to a passenger as the distance from the passenger to the vehicle destination plus the vehicles remaining distance to travel. In situations where the fleet size is limited this could allow the system to directly tell users if it will be possible to serve them and how long they have to wait. Another potential direction is ridesharing which may reduce to a vehicle routing problem.

References

Fagnant, D., and Kockelman, K. 2014. The travel and environmental implications of shared autonomous vehicles, using agent-based model scenarios. *Transportation Research Board Part C* 40:1–13.

Fagnant, D.; Kockelman, K.; and Bansal, P. 2015. Operations of a shared autonomous vehicle fleet for the austin, texas market. *Transportation Research Board*.

Kuhn, H. W. 1955. The hungarian method for the assignment problem. *Naval research logistics quarterly* 2(1-2):83–97.

MacAlpine, P.; Price, E.; and Stone, P. 2015. Scram: Scalable collision-avoiding role assignment with minimal-makespan for formational positioning. In *AAAI Conference on Artificial Intelligence (AAAI)*.