# Ridesharing with Passenger Transfers

Brian Coltin[1] and Manuela Veloso[1]

***Abstract*— Recently, ridesharing mobile applications, which dynamically match passengers to drivers, have begun to gain popularity. These services have the potential to fill empty seats in cars, reduce emissions and enable more efficient transportation. Ridesharing services become even more practical as robotic cars become available to do all the driving. In this work, we propose rideshare services which *transfer* passengers between multiple drivers. By planning for transfers, we can increase the availability and range of the rideshare service, and also reduce the total vehicular miles travelled by the network. We propose three heuristic algorithms to schedule rideshare routes with transfers. Each gives a tradeoff in terms of effectiveness and computational cost. We demonstrate these tradeoffs, both in simulation and on data from taxi passengers in San Francisco. We demonstrate scenarios where transferring passengers can provide a significant advantage.**

## I. Introduction

As climate change accelerates and gasoline prices rise, reducing reliance on personal automobiles has become critically important. Transportation by car seems especially wasteful when one realizes that the majority of car seats in a trip are often empty. To address this inefficiency, ridesharing services are beginning to catch on. In these, drivers input their destinations into their phones, and are matched with passengers headed the same way. Drivers pick up and drop off other passengers along the route to their own destination, filling empty seats to conserve fuel, reduce pollution and split costs. This process will become even more convenient as robotic cars become widespread.

In this work, we examine the possibility of reducing fuel use and emissions even further than traditional ridesharing by *transferring* passengers between vehicles. Our focus is on finding fast, sub-optimal algorithms to plan driver routes, incorporating passenger transfers to minimize fuel use and emissions. By allowing passengers to make use of multiple vehicles, passengers can travel further at less inconvenience to drivers (see Figure 1). This planning problem is especially challenging because it takes a problem that is already NP-hard, then increases the number of feasible plans exponentially by allowing transfers.

We assume that the passenger and driver destinations are known beforehand. We set aside passenger time windows, and instead assume that the requests are all flexible within a short window, for example a morning commute. Although

*This research was partially sponsored by the Office of Naval Research under grant number N00014-09-1-1031, and by the National Science Foundation under award number NSF IIS-1012733. The views and conclusions expressed are those of the authors only.

[1]B. Coltin (`bcoltin at cs.cmu.edu`) and M. Veloso (`veloso at cs.cmu.edu`) are with the Computer Science Department, Carnegie Mellon University, Pittsburgh, PA, USA

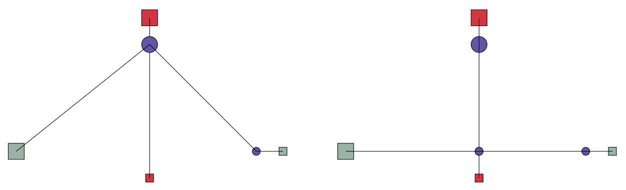

Fig. 1: Two vehicles deliver a passenger without transfers and with transfers. By transferring items, fuel is conserved. In the figure, larger shapes represent starting points, while smaller shapes represent transfer or ending points. Squares represent vehicles and circles represent passengers.

we look at ridesharing with robotic cars specifically, all the techniques in this paper can be directly to applied to other instances of the more general Pickup and Delivery Problem (PDP), including with mobile service robots [1], warehouse service robots [2], and other transportation problems.

In this paper, we present three heuristics to plan routes for ridesharing with transfers: a greedy heuristic, an auction algorithm, and an approach based on graph search. Each involves a trade-off between solution quality and computational cost. We test these three algorithms extensively in simulation of an urban area, and show that transferring passengers gives a significant reduction in total distance travelled. We also demonstrate the algorithms' effectiveness on a map of an actual city using the trips of taxi passengers.

## II. Related Work

Limited prior work has been done on ridesharing without transfers. One approach is to use auctions to assign passengers to vehicles [3]. Later work has focused on making auctions that are incentive compatible and encourage users to negotiate truthfully, however, currently this work is restricted to assigning a single passenger per driver [4]. In [5], Kamar and Horvitz assign passengers to vehicles with a set cover approximation algorithm, and consider mechanism design to construct a fair payment system. A third approach to dynamic ridesharing is presented in [6]: a combination genetic algorithm and insertion heuristic. This work considers the problem when passengers have time windows.

To the best of our knowledge, the only researchers who have considered the problem of dynamic ridesharing with transfers are Herbawi and Weber. Their work finds routes for a single passenger while optimizing multiple objectives: cost, time, and the number of drivers in the route. They present an evolutionary algorithm to plan multi-hop routes with multiple objectives in [7]. We form a plan for multiple passengers with transfers, rather than a route for a single passenger.

The ridesharing problem is a variant of the related to the vehicle routing problem, in which a set of vehicles are dispatched to service tasks. The main property specific to ridesharing is that the vehicles have fixed destination locations, as opposed to being able to end at any location or required to retun to a hub. See [8] for a thorough overview of the vehicle routing problem. A second closely related field of research is robot task allocation. However, approaches to robot task allocation typically assume that tasks are independent [9]. In the ridesharing problem, this is not the case, since adjustments to the route of a vehicle for one passenger will affect the cost of picking up other passengers.

## III. The Ridesharing Problem

First, we define the rideshare problem formally. We begin by defining the rideshare problem without transfers, and then extend our definition to include transfers.

### A. Ridesharing without Transfers

A rideshare problem is defined by a a set of vehicles $V$, a set of passengers $P$, and a map the vehicles travel on with a shortest path function $sp$ and a distance function $d$.

Every vehicle $v$ and every passenger $p$ have starting locations $s_v, s_p \in M$ and ending locations $e_v, e_p \in M$. In addition, every vehicle $v \in V$ has a capacity $C_v$, the maximum number of passengers the vehicle can carry at one time, and a maximum total number of passengers $M_v$ that the driver is willing to pick up in a single trip. In this paper, we ignore time to focus solely on fuel conservation, and do not consider desired arrival times or the total time a trip takes a driver or passenger.

In the rideshare problem without transfers, the goal is to assign each vehicle $v$ to a route $r_v$, such that $r_v^1 = s_v$, and $r_v^{|r_v|} = e_v$ (the vehicle starts and ends at its starting and ending points). Furthermore, for every passenger $p$ there exists a vehicle $v$ such that for some $i$, $r_v^i = s_p$, and for some $j > i$ $r_v^j = e_p$. We seek to find the assignment which minimizes the total distance travelled, $\sum_{v \in V} \sum_{1 \le i < |r_v|} d(r_v^i, r_v^{i+1})$.

Additionally, we defined the total number of passengers transported by a vehicle $v$ as $tp_v$and constrain each vehicle such that $tp_v \le M_v$, meaning a single driver will pick up a limited number of passengers in one trip and will eventually arrive at his destination. Furthermore, at all times the number of passengers in the vehicle $v$ is bounded by the capacity $c_v$.

### B. Ridesharing with Transfers

Next, we extend this formulation to include transfers. We introduce the idea of a transfer point— a location where one driver drops off a passenger, and a different vehicle retrieves him. For the rideshare problem with transfers, the vehicle paths contain transfer points $t_{p,v_1,v_2}$ in addition to vehicle and passenger starting and ending points. At the transfer point $t_{p,v_1,v_2}$, vehicle $v_1$ transfers passenger $p$ to vehicle $v_2$.

The goal remains to deliver every passenger to their destination while minimizing distance travelled, but the passengers may be routed through one or more transfer points.

**Algorithm 1** `greedy`$(V, P)$: greedily form routes $r_v$ for vehicles $v \in V$ to delivery all passengers $p \in P$.

```
for $v \in V$ do
    $r_v \leftarrow \langle s_v, e_v \rangle$
end for
$A = \emptyset$
for $i$ from 1 to $|P|$ do
    $v, p \leftarrow \mathrm{argmin}_{p \notin A, v}\, d(\texttt{route\_insert}(r_v, \langle s_p, e_p \rangle))$
    $r_v \leftarrow \texttt{route\_insert}(r_v, \langle s_p, e_p \rangle)$
    $A \leftarrow A \cup \{p\}$
end for
```

We add a cost $c_T$ to transfers to represent the inconvenience to passengers and drivers. Our new objective is to minimize the sum of distance travelled and transfer cost,

$$\sum_{v \in V} \left( \sum_{1 \le i < |r_v|} d(r_v^i, r_v^{i+1}) + c_T \left|\{t_{p,v_1,v_2} \in r_v\}\right| \right).$$

One final complication remains: when the vehicles are allowed to transfer passengers, cyclical dependencies may form. Vehicle $v_1$ may need to hand off a passenger to $v_2$, who later on transfers a passenger to $v_3$. Then $v_3$ transfers a passenger to $v_1$ before $v_1$'s transfer to $v_2$. In this case, $v_1$ is waiting on $v_3$, and $v_3$ is waiting on $v_1$, so deadlock results.

Cyclical dependencies are not permitted in a solution to the rideshare problem with transfers. To detect them, we form a directed *transfer graph* representing the routes of the vehicles and their dependencies induced by transfers. We construct the transfer graph by taking the union of the vehicles' directed paths $r_v$. Then, for each transfer point $t_{p,v_1,v_2}$ we add an edge from $t_{p,v_1,v_2}$ to $t_{p,v_2,v_1}$. If the transfer graph contains a cycle, then the vehicles will reach deadlock.

## IV. Ridesharing without Transfers

We present two algorithms which take a set of routes without transfers as input and output a schedule to deliver all passengers with transfers. Hence, we first introduce two algorithms to form schedules without transfers, a greedy approach and an auction approach. The auction approach is similar to [3] and [4]. An alternative algorithm, such as set cover [5], could be used instead.

### A. Greedy Passenger Insertion

In the greedy approach, we iterate through every passenger $p$ and vehicle $v$ pair, insert the start and end points $s_p$ and $e_p$ into $v$'s route at the points of lowest cost, and choose the assignment of lowest additional cost. This means we find the best insertion points for pickup up and dropping off passenger $p$ without rearranging the rest of $v$'s path. We repeat this process until no unassigned passengers remain (see Algorithm 1).

The procedure `route_insert`$(p_v, a)$ uses brute force search to find the optimal placement to insert the points in $a$ into the route $r_v$, while maintaining the points' ordering and $v$'s capacity and maximum passenger constraints. The function returns the resulting route, or an invalid route if no route is available. The distance function $d$, when applied to a path, returns the length of the path, or $\infty$ for an invalid path.

As is, the greedy algorithm's runtime is $O(|P|^2|V|M^2)$ where $M$ is the maximum number of riders in a single vehicle. However, calls to `route_insert` will not change across iterations of the for loop, except for vehicles which were assigned a passenger in the previous iteration of the loop. By caching these values, the algorithms complexity becomes $O(|P||V|M^2)$.

### *B. Auctioning Rides*

Next, we present an auction approach to solve the ridesharing problem without transfers. The auction consists of rounds, in which each passenger places a bid for the vehicle that can transport it at lowest additional cost (as determined by the `route_insert` procedure). For each vehicle, the passenger that can be transported at lowest cost is declared the winner, and is added to that vehicle's route. If multiple passengers bid on a single vehicle, the vehicle with the lowest cost wins. Rounds of the auction continue until all passengers are assigned to vehicles.

The worst-case runtime complexity of the auction is $O(|P|^2|V|M^2)$. However, since many passengers will be assigned to vehicles in the same round, in practice it often runs faster than the greedy algorithm. this algorithm also lends itself to a decentralized implementation.

## V. Ridesharing with Transfers

We present three heuristics for solving the ridesharing problem with transfers: a greedy approach, an auction approach, and a graph-based approach. The greedy and auction algorithms take as input a solution to the rideshare problem without transfers, while the graph-based approach constructs a solution from scratch.

However, before presenting these algorithms in detail, we look at the problem of how to select a transfer point for two vehicles to exchange items. Each of the algorithms for ridesharing with transfers must solve this subproblem.

### *A. Choosing an Exchange Point*

Given an edge on vehicle $v_a$'s route $\langle a_1, a_2\rangle$ and an edge on vehicle $v_b$'s route $\langle b_1, b_2\rangle$, our goal is to select an optimal meeting point $p$ such that $\sum_{v \in a_1,a_2,b_1,b_2} d(p, v)$ is minimized. In the case where the map is Euclidean, this is called the Weber problem and the point $p$ is called the geometric mean. We introduce a function `tpoint`$(a_1, a_2, b_1, b_2)$ which returns a proposed transfer point.

Although the idea of the optimal meeting point is quite simple, it is difficult to compute. In fact, it has been shown that no closed form solution exists. However, numerous algorithms have been developed to find the optimal meeting point with gradient descent and other techniques, including a near-optimal solution for general maps [10].

In this work, our focus is not on finding the individual transfer points, but on finding the overall schedule of when the vehicles should transfer. Hence, we use a fast heuristic rather than the computationally expensive near-optimal methods. We simply consider each of the three possible pairings of the points $a_1, a_2, b_1,$ and $b_2$, and find the intersection points of the shortest paths between each set of paired points. We choose the point which creates the lowest edge cost as a proposed transfer point. On a general map, none of the shortest paths may intersect, in which case we do not place a transfer point between these line segments. Although we choose a non-optimal approach for selecting a transfer point, any other approach from the literature could be substituted to find a better solution at a cost of additional computation.

For less general maps, in particular a world composed of city blocks where the distance function is Manhattan distance, the optimal meeting point between two edges can be found. For each edge we take the smallest rectangle, aligned to the city blocks, which includes the starting and ending point of the edge. If the rectangles overlap, any point in the overlap region is an optimal meeting point. If not, we choose one of the closest points equidistant to both rectangles.

In addition to the `tpoint` function, we introduce a helper routine, `split_route`$(r, v_1, v_2)$, which adds a transfer of the passenger $r$ between vehicle $v_1$ and vehicle $v_2$. The passenger is assumed to already be part of $v_1$'s route. After the function is called, $v_1$ will either pick up or drop off the passenger as it had before, but a transfer will be inserted with $v_2$ which will complete the other action originally in $v_2$'s route. We search through every feasible pair of edges in the routes of $v_1$ and $v_2$ (without reordering the routes) and find the transfer point of lowest cost which obeys the capacity constraints and does not induce a cycle. The `split_route` function effectively splits a segment of a passenger's route between two vehicles.

We also introduce a budgetary heuristic in the `split_route` function. If the start and ending point of the passenger $p$ in $v_1$ are both a greater distance than a budget $b_v$ from the edge being considered from $v_2$ for a transfer point insertion, then that potential transfer point is rejected. With this heuristic, transfer points between vehicles that do not cross near each other are not considered. This helps alleviate the computational load of iterating through $O(|V|^2)$ potential transfer points.

### *B. Greedy Route Splitting*

The first approach we propose greedily adds transfer points to an existing solution without transfer points. We form a queue $q$ containing passengers and the vehicles that are transporting them. For each passenger and vehicle in the queue, we iterate through the other vehicles and check if a different vehicle could take the passenger part of the way at a lower cost, using the `split_route` function. If such a transfer exists, we find the other vehicle which would lower the cost the most, and add the transfer. We then add both halves of the split route back into the queue $q$, where we may later add additional transfers to subdivide the route

**Algorithm 2** `greedy_transfer`$(V, P, \{r_v : v \in V\})$: greedily adjust routes $r_v$ for vehicles $v \in V$ to delivery all passengers $p \in P$, with transfers.

```
$q \leftarrow$ nil
for $p \in P$ do
  for $v \in \{v : p$ transported in $r_v\}$ do
    enqueue$(q, (p, v))$
  end for
end for
while $q$ not empty do
  $(p, v_1) \leftarrow$ dequeue$(q)$
  $best_c \leftarrow \infty, best_v \leftarrow \emptyset$
  for $v_2 \in V, v_2 \neq v_1$ do
    $c \leftarrow$ split_route_cost$(p, v_1, v_2)$
    if $c < best_c$ then
      $best_c \leftarrow c, best_v \leftarrow v_2$
    end if
  end for
  if $best_c \neq \infty$ then
    split_route$(p, v_1, best_v)$
    enqueue$(q, (p, v_1))$, enqueue$(q, (p, v_2))$
  end if
end while
```

**Algorithm 3** `auction_transfer`$(V, P, \{r_v : v \in V\})$: adjust routes $r_v$ for vehicles $v \in V$ to delivery all passengers $p \in P$, by bidding on transfers.

```
for $v \in V$ do
  $bid_v \leftarrow \infty$
end for
for $p \in P$ do
  $v_1, v_2 \leftarrow \text{argmin}_{v_1 \neq v_2}$ split_route_cost$(p, v_1, v_2)$
  $c \leftarrow$ split_route_cost$(p, v_1, v_2)$
  if $c \neq \infty$ and $bid_{v_2} > c$ then
    $bid_{v_2} \leftarrow c$, $bidp_{v_2} \leftarrow p$, $bidv_{v_2} \leftarrow v_1)$
  end if
end for
$assigned \leftarrow$ false
for $v \in V$ do
  if $bid_v \neq \infty$ then
    split_route$(bidp_{v_2}, bidv_{v_2}, v_2)$
    $assigned \leftarrow$ true
  end if
end for
if $assigned$ then
  repeat auction
end if
```

further. The full details of the greedy approach are presented in Algorithm 2.

The effectiveness of this algorithm, as with many greedy algorithms, is dependent on the order in which we iterate over the passenger and vehicle pairs. We only consider transfers between pairs of vehicles at a time, ignoring better routes which could be obtained by transferring between multiple vehicles. However, routes which transfer to multiple vehicles may be obtained when the greedy algorithm is applied recursively to partial routes of passengers on individual vehicles.

## C. Auctioning Partial Rides

Our second approach is an auction in which passengers bid for vehicles. In one round of the auction, each passenger finds the single transfer point which could be added to obtain the greatest cost decrease, by applying the `split_route` function. Each passenger places a bid for each vehicle. Then, each vehicle accepts the passenger with the lowest bid which will decrease the total cost the most. If no assignments were made the auction ends; otherwise we repeat the auction for another round (see Algorithm 3).

Like the greedy approach, the auction algorithm has the shortcoming that it only considers a single transfer at a time. However, the auction approach is less greedy in the sense that the assignment does not depend on the ordering the passengers or vehicles are examined in.

## D. Graph Search

Our final algorithm for ridesharing with transfers differs from the first two in that it does not begin with a solution for ridesharing without transfers. Instead, it plans for transfers from the beginning. However, since it finds routes in an exponentially larger space which includes transfers, this algorithm is significantly slower to execute than the previous approaches.

The graph search algorithm is greedy in the sense that we iterate through each passenger, and plan the best path for that particular passenger. To do so, we construct a directed multi-graph containing all the vehicles' current routes and potential transfer points. Each vehicle's initial route $r_v$ is set to contain only $s_v$ and $e_v$. The edges on the graph represent segments of vehicles' paths or transfers that the passenger could take on his route. The weights of the edges represent the additional cost to the vehicles of using that means of transportation for the passenger. Then, the shortest path on the graph gives the route of least cost for the passenger.

We construct a graph when searching for a route for passenger $r$ which contains the following nodes:

- $\forall v, \forall x \in r_v$ $x$, every point already visited by a vehicle;
- $s_p, e_p$, the starting and ending points of $p$; and
- $\forall v_1, v_2, 1 \leq i < |r_{v_1}|, 1 \leq j < |r_{v_2}|$ $t_{p,v_1,v_2}$ (for each pair of edges on two vehicles' paths) we add the transfer point `tpoint`$(r_{v_1}^i, r_{v_1}^{i+1}, r_{v_2}^j, r_{v_2}^{j+1})$. Transfer points are not added if they would cause a cycle or if reaching them would exceed one of the two vehicles' budgets.

These nodes are connected by directed edges representing possible paths for the passenger, with the edge weights equalling the change in the distance travelled if these routes are taken. We add the edges, $\forall v, \forall 1 \leq i < |r_v|$:

- from $r_v^i$ to $r_v^{i+1}$ with weight 0, representing paths the vehicles already plan to travel (at no additional cost);
- from $s_p$ to $r_v^i$ with weight $d(r_v^i, s_p) + d(s_p, r_v^{i+1}) -$

$d(r_v^i, r_v^{i+1})$, representing the passenger being picked up on an edge of the original path;

- from $e_p$ to $r_v^{i+1}$ with weight $d(r_v^i, e_p) + d(e_p, r_v^{i+1}) - d(r_v^i, r_v^{i+1})$, representing the passenger being dropped off on an edge of the original path; and
- from $s_p$ to $e_p$ with weight $d(r_v^i, s_p) + d(s_p, e_p) + d(e_p, r_v^{i+1}) - d(r_v^i, r_v^{i+1})$, representing the passenger being both picked up and dropped off on the same edge.

Furthermore, for each transfer point newly added as a node $ex = \texttt{tpoint}(r_{v_1}^i, r_{v_1}^{i+1}, r_{v_2}^j, r_{v_2}^{j+1})$, we add the edges:

- from $s_p$ to $ex$ with weight $d(r_{v_1}^i, s_p) + d(s_p, ex) + d(r_{v_2}^j, ex) - d(r_{v_1}^i, r_{v_1}^{i+1}) - d(r_{v_2}^j, r_{v_2}^{j+1}) + c_T + h_T$;
- from $r_{v_1}^i$ to $ex$ with weight $d(r_{v_1}^i, ex) + d(r_{v_2}^j, ex) - d(r_{v_1}^i, r_{v_1}^{i+1}) - d(r_{v_2}^j, r_{v_2}^{j+1}) + c_T + h_T$;
- from $ex$ to $e_p$ with weight $d(ex, e_p) + d(e_p, r_{v_2}^{j+1}) + d(ex, r_{v_1}^{i+1})$;
- from $ex$ to $r_{v_2}^{j+1}$ with weight $d(ex, r_{v_2}^{j+1}) + d(ex, r_{v_1}^{i+1})$; and, finally,
- for every other additional transfer point $ex_2 = \texttt{tpoint}(r_{v_2}^j, r_{v_2}^{j+1}, r_{v_3}^k, r_{v_3}^{k+1})$, an additional edge from $ex$ to $ex_2$ with weight $d(ex, ex_2) + d(ex, r_{v_1}^{i+1}) + d(r_{v_3}^k, ex_2) - d(r_{v_3}^k, r_{v_3}^{k+1})$. These edges allow a vehicle to receive an item and transfer it to another vehicle along the same edge in the original path, although it may not find the best transfer locations as if we were to construct additional transfer points recursively.

An edge is not added to the graph if that edge would cause a vehicle to exceed its capacity or maximum passenger constraint.

Once the graph is constructed for passenger $p$, we use the Bellman-Ford algorithm to find the shortest path on the graph from $s_p$ to $e_p$, the passenger's route. Dijkstra's algorithm cannot be used since the edge lengths may be negative. From the shortest path, we construct the route taken by the passenger, and modify the routes $r_v$ of the appropriate vehicles to accommodate the passengers. Then, we repeat the algorithm, finding a route for the next passenger.

The resulting path found for a passenger may induce a cyclical dependency by passing through *multiple* transfer points. If this is the case, we remove the transfer points that caused cycles from the graph and try again.

This approach is significantly slower than the greedy and auction approaches, since the number of exchange points and the size of the graph increases quadratically with the number of edges in the vehicles' paths. However, there is substantial room for speed-ups if the graph is constructed incrementally, or if heuristics are added to reduce the number of considered transfer points and the edges between them.

## VI. Selected Experimental Results

To verify the effectiveness of the three algorithms, we compare them in simulation and using actual passenger routes on a city map. We demonstrate a reduction in fuel use by planning for transfers.

For each experiment, we set the number of vehicles $|V|$, the number of passengers $|P|$, a vehicular capacity $C_v$, the maximum number of riders per vehicle $M_v$, and the budget heuristic $B_v$. The starting and ending points for vehicles and passengers are selected randomly.

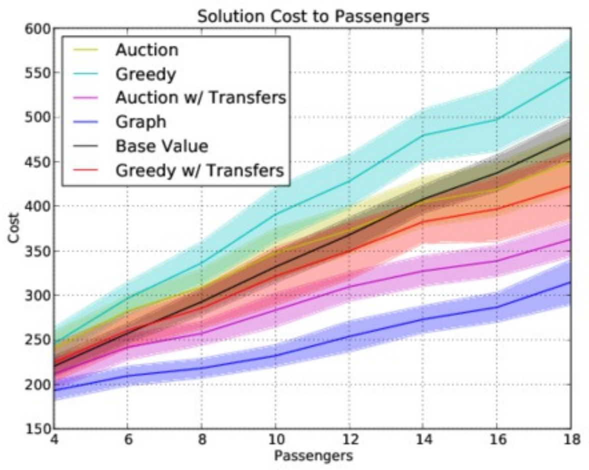

Fig. 2: The solution cost found for each method in the simulated urban area with $|V| = 20$, $C_v = 5$, $M_v = 7$, $B_v = 6$, and $c_T = 0$.

### A. Simulated Urban Area

The first set of experiments was performed in a simplified simulated urban area, a regular 20 by 20 grid of city blocks. Manhattan distance was used, and starting points and destinations were chosen randomly from street intersections.

Figure 2 shows the costs of the solutions found by each algorithm, when passenger starting and ending points are at least 10 blocks apart, and intended vehicle routes are between 5 and 7 blocks. Instances where passengers travel further than vehicles (for example, long-distance hitchhiking) particularly benefit from transfers. The shaded regions denote the standard deviation across the fifty trials, and the "base cost" is what the cost in fuel would be if all of the passengers and drivers drove themselves in their own vehicles directly to their destinations. In this domain, the greedy algorithm without transfers performs worse than the base cost, and the auction algorithm finds solutions of approximately the same cost as the base.

However, with transfers, we can outperform both the algorithms without transfers and the case when everyone drives themselves, reducing fuel usage. The greedy transfer algorithm, the auction transfer algorithm, and the graph-based algorithm each offer a successive improvement. With 18 passengers and the graph-based algorithm, transfers reduce the distance travelled by nearly 30% compared to the auction algorithm without transfers, and by 34% compared to the base cost when each passenger drives themself.

Although the graph method finds the best solutions, its effectiveness comes at a computational cost. For 20 vehicles and 18 passengers, the greedy algorithm took an average of 0.7 s to run, the auction algorithm took 3.3 s, and the graph algorithm took 484.7 s. There is significant room for further optimization in each algorithm's implementation, particularly

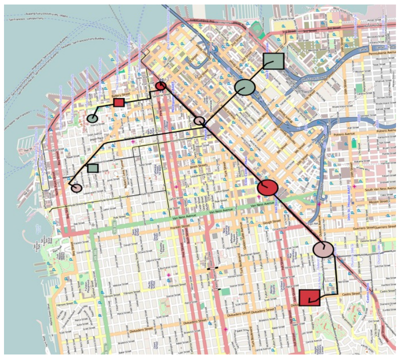
Fig. 3: An example solution with transfers for a problem in San Francisco. Two passengers are transferred to other vehicles.

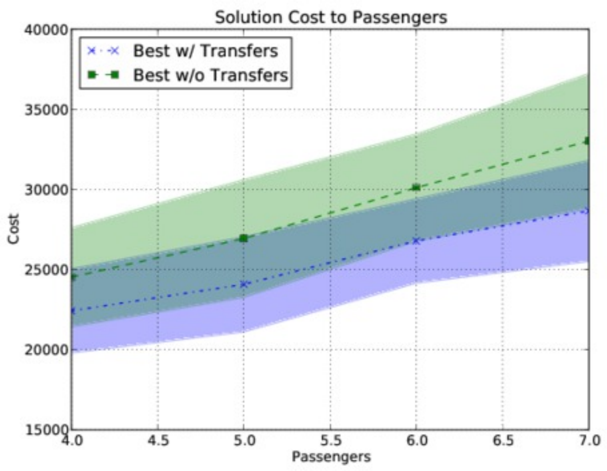

Fig. 4: The solution cost found for each method in San Francisco with $|V| = 18$, $C_v = 4$, $M_v = 7$, $B_v = 1.5$ km, and $c_T = 0$.

by constructing the graph used to find the route for each passenger incrementally based on the previous graphs.

We also ran experiments with the same setup as before, except without a maximum distance restriction for the vehicles. In this case, for 20 vehicles and 18 passengers, auctions without transfers improved on the base cost by an average of $23.8\%$, while the graph algorithm improved on the base cost by $35.0\%$. Each passenger made an average of $1.55$ transfers.

Finally, we conducted the same less constrained experiment, but expanded to $100$ passengers and $80$ vehicles on a $30$ by $30$ block grid. Auctions without transfers gave a $40.26\%$ average improvement over the base cost, while auctions with transfers gave a $46.52\%$ improvement. The auction with transfers took an average of $58$ s to compute. The graph algorithm does not yet scale to such large instances.

### B. San Francisco

We also ran tests using real world taxicab data on an actual city map. We obtained a map of San Francisco from OpenStreetMap [11] and found shortest paths on this map using pgRouting [12] (see Figure 3 for an example solution). We sampled passenger and vehicle starting points from real-world taxi cab data [13], limited to the downtown area. Vehicles are constrained to trips between $0.22$ and $0.56$ km, and passengers are constrained to trips greater than $1.33$ km. Selected results are shown in Figure 4.

Routing on the map of San Francisco is significantly more costly than on the Euclidean plane. With 7 passengers and 18 vehicles, the greedy algorithm with transfers took over 80 seconds and the auction algorithm with transfers took nearly 200. The graph algorithm was not run due to its' cost. Faster path planning algorithms or approximate distance estimates will need to be used to scale up. However, there were average savings of approximately $15\%$ from using transfers.

## VII. Conclusion

We proposed transferring passengers to create more efficient ridesharing services and reduce emissions. We presented three heuristics to plan for transfers: a greedy approach, an auction, and an approach based on graph search. Each algorithm trades off computation time with effectiveness. We demonstrated that transferring passengers reduces the distance travelled by nearly $30\%$ for certain problems. Much future work remains, including further optimizing the graph-based algorithm, planning for transfers in *dynamic* settings, and considering time and convenience.

## References

[1] B. Coltin, M. M. Veloso, and R. Ventura, "Dynamic user task scheduling for mobile robots." in *Automated Action Planning for Autonomous Mobile Robots*, 2011.
[2] P. R. Wurman, R. D'Andrea, and M. Mountz, "Coordinating hundreds of cooperative, autonomous vehicles in warehouses," *AI Magazine*, vol. 29, no. 1, p. 9, 2008.
[3] S. Abdel-Naby, S. Fante, and P. Giorgini, "Auctions negotiation for mobile rideshare service," in *Int. Conf. on Pervasive Computing and Applications*. IEEE, 2007, pp. 225–230.
[4] A. Kleiner, B. Nebel, and V. Ziparo, "A mechanism for dynamic ride sharing based on parallel auctions," in *Int. Joint Conf. on AI (IJCAI)*, 2011, pp. 266–272.
[5] E. Kamar and E. Horvitz, "Collaboration and shared plans in the open world: Studies of ridesharing," in *Int. Joint Conf. on AI (IJCAI)*, 2009, pp. 187–194.
[6] W. Herbawi and M. Weber, "A genetic and insertion heuristic algorithm for solving the dynamic ridematching problem with time windows," in *Proc. of the Int. Conf. on Genetic and Evolutionary Computation*. ACM, 2012, pp. 385–392.
[7] ——, "Evolutionary multiobjective route planning in dynamic multi-hop ridesharing," in *Evolutionary Computation in Combinatorial Optimization*, ser. Lecture Notes in Computer Science, P. Merz and J.-K. Hao, Eds. Springer Berlin / Heidelberg, 2011, vol. 6622, pp. 84–95.
[8] P. Toth and D. Vigo, *The vehicle routing problem*. Soc. for Industrial Mathematics, 2002, vol. 9.
[9] B. Gerkey and M. Matarić, "A formal analysis and taxonomy of task allocation in multi-robot systems," *The International Journal of Robotics Research*, vol. 23, no. 9, pp. 939–954, 2004.
[10] P. Zebrowski, Y. Litus, and R. Vaughan, "Energy efficient robot rendezvous," in *Computer and Robot Vision, 2007. CRV'07. Fourth Canadian Conference on*. IEEE, 2007, pp. 139–148.
[11] OpenStreetMap, "OpenStreetMap," 2012, http://openstreetmap.org.
[12] The pgRouting Project, "pgRouting Project," 2012, http://pgrouting.org.
[13] M. Piorkowski, N. Sarafijanovoc-Djukic, and M. Grossglauser, "A Parsimonious Model of Mobile Partitioned Networks with Clustering," in *Int. Conf. on Communication Systems and Networks*, January 2009.