STaRS: Simulating Taxi Ride Sharing at Scale

Masayo Ota, Huy Vo, Cláudio Silva, *Fellow, IEEE,* and Juliana Freire, *Member, IEEE*

Abstract—As urban populations grow, cities face many challenges related to transportation, resource consumption, and the environment. Ride sharing has been proposed as an effective approach to reduce traffic congestion, gasoline consumption, and pollution. However, despite great promise, researchers and policy makers lack adequate tools to assess the tradeoffs and benefits of various ride-sharing strategies. In this paper, we propose a real-time, data-driven simulation framework that supports the efficient analysis of taxi ride sharing. By modeling taxis and trips as distinct entities, our framework is able to simulate a rich set of realistic scenarios. At the same time, by providing a comprehensive set of parameters, we are able to study the taxi ride-sharing problem from different angles, considering different stakeholders' interests and constraints. To address the computational complexity of the model, we describe a new optimization algorithm that is linear in the number of trips and makes use of an efficient indexing scheme, which combined with parallelization, makes our approach scalable. We evaluate our framework through a study that uses data about 360 million trips taken by 13,000 taxis in New York City during 2011 and 2012. We describe the findings of the study which demonstrate that our framework can provide insights into strategies for implementing city-wide ride-sharing solutions. We also carry out a detailed performance analysis which shows the efficiency of our approach.

Index Terms—taxi ride sharing, simulation, shortest-path index, scalability, urban computing.

1 Introduction

Due to the steady growth in urban populations [1], cities now face huge challenges related to transportation, resource consumption, and pollution. Ride sharing has been proposed as a strategy to decrease road traffic and gasoline consumption [2], while at the same time serving the transportation needs of city dwellers. In large cities, there is substantial unused taxi capacity that can be filled by ride-sharing services. Consider, for example New York City (NYC): each day, taxi cabs make 500 thousand trips and serve 600 thousand passengers; this translates into an average occupancy rate of only 1.2 passengers per trip [3]. Private companies such as Uber, Lift, Via, Bandwagon and Cab With Me already provide ride-sharing services. However, they represent a small percentage of the market.

A wide deployment of ride sharing requires a better understanding of its tradeoffs. This is challenging since there are multiple stakeholders with different, and sometimes conflicting, interests. Governments want less traffic and pollution; taxi companies want to maximize their profits; and passengers would like to reach their destination quickly and cheaply. To design an effective policy, these interests need to be considered. Early approaches to this problem have been primarily devised on the basis of survey data [4] and analysis of psychological incentives [5], [6], [7]. Ride sharing has also been modeled as an optimization problem whose objective is to identify optimal ride-sharing schedules [8], [9], [10], [11], [12], [13]. However, these approaches focus on small-scale problems, such as sharing at airports, since large-scale optimization is often computationally infeasible.

- *M. Ota, C. Silva, and J. Freire are with Center for Urban Science and Progress, New York University, and Department of Computer Science and Engineering, New York University .*
  E-mail: {masayo.ota, csilva, juliana.freire}@nyu.edu
- *H. Vo is with Center for Urban Science and Progress, New York University, and Department of Computer Science, the City College of New York .*
  E-mail: hvo@cs.ccny.cuny.edu

The availability of large volumes of taxi trip data creates new opportunities to apply data-driven approaches to this problem. Santi et al. [14] proposed a graph-based model that computes optimal sharing strategies for trips and contains two key parameters: the maximum number of trips that can be shared and the maximum delay customers are willing to tolerate. While this allows the study of sharing benefits as a function of passenger inconvenience, the model has important limitations. Notably, it is intractable for scenarios that consider the sharing of three or more trips and it assumes that all trips are known in advance. Ma et al. [15], [16] introduced T-Share, a ride-sharing dispatch system that serves real-time requests issued by passengers and generates schedules that reduce the total travel distance. This system, however, was not designed to support simulations: it neither provides the necessary parameters to simulate different scenarios, nor does it scale to very large data sets.

We propose STaRS (**S**imulating **Ta**xi **R**ide **S**haring), a data-driven simulation framework that enables the analysis of a wide range of ride-sharing scenarios. Unlike [14], in our model, trips need not be known in advance: STaRS supports the simulation of real-time ride sharing which serves unplanned trips, and fits well the models using different vendors, such as Yellow cabs and Uber. By modeling taxis and trips as distinct entities, and by providing a rich set of variables, STaRS enables the study of a wide range of realistic scenarios that take into account the needs and constraints of multiple stakeholders. These include different customer preferences (e.g., maximum number of additional stops and wait time), and taxi-specific constraints often dictated by ride-sharing vendors, for example, the number of passengers on a per-taxi basis and maximum number of shared trips. This flexibility comes at a cost: assigning trips to taxis in real-time is computationally expensive. We describe a new optimization algorithm that is linear in the number of trips and makes use of an efficient indexing

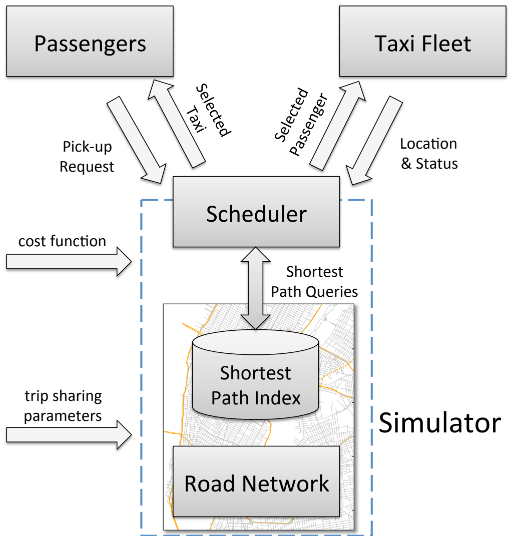


Fig. 1. Our taxi ride-sharing model consists of three main entities: taxis, passengers and a scheduler. Given a trip request issued in real-time, our model assigns this trip to a taxi in such a way to optimize a given cost function while at the same time respecting a set of pre-defined constraints.

Taxi Ride-Sharing Constraints	Description
C	capacity of taxi expressed as the number of passengers
n_{share}	maximum number of trips that can be shared (or -1 if this is set based on customer preference)
t_{delay} or d_{delay}	maximum time/distance the taxi is permitted to let its customers wait for pick-up
t_{extra} or d_{extra}	maximum additional time/distance incurred by sharing a ride
Vehicle States	**Description**
o	current occupancy, i.e., number of passengers on-board
id	taxi identification number
v	current speed
$s_0, \ldots, s_k$	list of stops the taxi has to make; s_0 is the lastest stop the taxi has made. The information stored in each stop is specified below.
$d_{odometer}$	current odometer reading
d_{driven}	distance from s_0
Stop Info	**Description**
p	location of the stop s_k (could be expressed as the intersection number in the road network)
$o_{\triangle}$	the number of passengers associated with this stop, where $o_{\triangle} > 0$: a pick-up $o_{\triangle} < 0$: a drop-off $o_{\triangle} = 0$: a waypoint (e.g., to look for riders)

TABLE 1
Taxi specifications: ride-sharing variables and vehicle state information associated to each taxi and stop event.

Passenger Ride-Sharing Constraints	Description
o_p	number of passengers
t_{pick}	time requested for pick-up
p_{pick}	pick-up location
p_{drop}	drop-off location
n_{share}	maximum number of trips to be shared
t_{delay}	maximum time the customer group is willing to wait for pick-up
t_{extra}	maximum time the customer group is willing to tolerate in addition to the actual trip time

TABLE 2
Customer group ride-sharing parameters.

scheme, which combined with parallelization, makes our approach scalable.

We evaluate the efficiency and effectiveness of STaRS using taxi data from NYC, which contains information about over 360 million trips taken by the NYC's 13,000 taxis in 2011 and 2012. The results demonstrate that our approach is efficient: one simulation using over 150 million trips can be run in under 10 minutes using a 1200-core cluster, allowing multiple scenarios to be studied in a timely manner. We show that the framework is effective and can provide insights into strategies for implementing city-wide ride-sharing solutions. We also experimentally compare our approach against [14] and argue that their model can underestimate the benefits of the taxi ride sharing.

2 Ride-Sharing Simulation Model

Given a trip request issued in real-time, our model assigns this trip to a taxi while optimizing a pre-defined cost function under a set of constraints.

2.1 Simulation Components

The main components of our simulation model are illustrated in Fig. 1. We describe them in detail below.

Taxi Fleet. The taxi fleet refers to the set of taxis that are involved in the simulation. In contrast to previous works, where taxis are considered as homogeneous objects, to support a multi-vendor environment (e.g., yellow and green cabs, black car services)[1] and different types of vehicles, we consider each taxi as a distinct object with its own specifications, which include: passenger capacity, maximum number of shared trips (often dictated by the vendors), maximum wait time for pick-up, and extra time for drop-off. In addition to these sharing constraints, each taxi also maintains information about its current speed, occupancy, and the list of stops it has to make to serve its scheduled riders. The list of variables associated with a taxi is given in Table 1.

A taxi in our model is a dynamic object and is always on the move. Even without any passenger on-board, the taxi still has a destination where it drives to, for example, to look for new passengers. To capture this, we allow each taxi to define favorite locations at certain times of the day that it would drive to when it is not occupied.

Passengers. We assume that passengers ride in groups of size greater than or equal to one. Each group is associated with a drop-off location and a set of ride-sharing constraints (e.g., how many other groups they are willing to share a ride with and how much additional time/distance they can tolerate). We assume this information is available when a customer initiates a taxi request, either by calling a control center, using a mobile app, or communicating this directly to a driver in case of street hailing. Table 2 lists the variables associated to passengers in our model.

Scheduler. For each pick-up request, the scheduler finds the most appropriate taxi based on pre-defined metrics. To do so, the scheduler must know all taxi locations along with their current states at all times (see Table 1). We describe the

1. http://www.nyc.gov/html/tlc/html/industry/current_licensees.shtml

Input Parameters	Description
m	number of taxis
C	default taxi capacity. Optionally, the capacity for each taxi may be customized by providing an additional array $\{c_0, c_1, ..., c_{m-1}\}$.
n_{share}	maximum number of trips to be shared by default. This number initially populates both of the n_{share} values of each taxi and passenger ride-sharing constraints (see Table 1, 2), unless specified otherwise. The motivation for having this parameter defined globally was to allow agencies/taxi vendors to define their fleet policy in addition to what described by the drivers and riders.
$t_{delay}(d_{delay})$, $t_{extra}(d_{extra})$	maximum wait and additional time(or distance) each customer could spend by default. Similar to n_{share}, the motivation for having these parameters defined globally is to provide users flexibility to customize the simulations.
$\mathbf{f}(r, c)$	cost (metric) function that given a taxi c and a pick-up request r, returns the cost of accommodating r with c. By default, this function evaluates the additional distance that c must drive to accommodate r.

TABLE 3
Input parameters for simulation.

details of the scheduling algorithm in Section 3.

Trip Data. The proposed simulation framework is data-driven. It leverages historical trip data to study the tradeoffs of different ride-sharing strategies. We assume the availability of trip data which include: taxi ID, pick-up and drop-off time, latitude and longitude for both pick-up and drop-off, travel distance, and number of passengers. We model trips and taxis as separate objects, and track the state of each taxi during the simulation. This is equivalent to assuming that the data set is a sample of trips from an unknown distribution and ignoring that each trip was originally associated with a particular taxi.

Road Network. The underlying road network of a city is represented as a directed graph $G(V, E)$. All taxis travel along this road network. Each directed edge $e \in E$ represents a road segment, and each node $v \in V$ represents the intersection of two or more roads. When a road allows traffic flow in both directions, there are two directed edges for that road. Given a segment e_i, t_i is the distance (or time) a vehicle must travel from one intersection point to another along e_i. Note that traffic conditions can be easily incorporated in the model by introducing weights on edges of the graph as explained in Section 3. We assume that the origin and destination of a trip correspond to nodes in this graph. If the trip begins or ends in the middle of a road segment, we approximate the location to the nearest intersection node.

2.2 Data-Driven Simulation

The simulation engine aims to derive the *best* ride-sharing scenario based on a set of input parameters (shown in Table 3) in a *data-driven fashion*, where pick-up requests are derived from historical data. It operates in an event-driven manner and updates its state when a pick-up request is issued. When a customer group requests a taxi, the scheduler receives the information and requests all taxis to report their status (i.e., position and sharing status). The scheduler then computes the additional cost for each taxi to accommodate this trip based on the cost function f, and selects the taxi with the minimal cost that satisfies all ride-sharing constraints. If no appropriate taxi is found, the request is denied. As we discuss in Section 3, the simulation engine allows different scheduling strategies.

Since taxi ride-sharing typically occurs in real-time, our approach needs to support online simulations, that is, simulations where trip requests are issued dynamically. Thus, the scheduler must evaluate the current conditions and respond to the customer immediately. If the request is accepted, the current state need to be updated as well. The need for high throughput and immediate responses separates us from previous work. For instance, [14] assumes that all the trips are known in advance which can substantially reduce computational requirements but does not lead to a realistic ride-sharing solution.

3 Simulation Algorithm

In the taxi ride-sharing problem, the goal is to minimize the total cost or maximize the total utility of sharing while meeting a set of constraints. Examples of costs include travel distance, CO_2 emissions, gasoline consumption, time-to-pick-up, idle time, or weighted combinations of these cost functions. We use the following formulation to design our optimization algorithm. Let $f(r_i, c_j)$ be the additional cost (e.g., distance, CO_2 emissions) a cab c_j incurs to share its current trips with a new trip r_i. Let n be the number of trips and m be the number of taxis. The minimum total travel cost $T(i)$ for the first i trips is:

$$T(i) = \begin{cases} T(i-1) + \min_{1 \leq j \leq m}\{f(r_i, c_j)\} \\ T(i-1), \text{ if there is no available cab.} \end{cases}$$

We can use a similar formulation to maximize the utility (e.g., revenue). Hence, to describe our algorithm, we use the additional travel distance for a cab c_j to accommodate a trip r_i as cost function $f(r_i, c_j)$ and try to minimize the total travel distance $T(n)$.

The algorithm considers all trips in chronological order. It attempts to mimic real-time dispatching by minimizing $T(n)$ in an online fashion. For each trip r_i, the state of a cab c_j is updated based on the time elapsed since the last trip r_{i-1} and the computed additional distance $f(r_i, c_j)$. Trip r_i is assigned to the cab with the minimum additional distance, and the total cost $T(i-1)$ is updated. Note that this is not a globally optimal solution: we do not consider all the possible combinations of trips to be shared. However, this matches a more realistic scenario in which we are not able to foresee the future trips or make changes to the past trips.

3.1 Preprocessing Phase

Before running the simulation, trips are sorted in chronological order (by pick-up time). We use a graph representation of the road network obtained from the Open Street Map data [17], which contains the longitude and latitude of each intersection as well as the distances between any two adjacent intersections. We apply Dijkstra's algorithm to compute shortest paths and distances between any two given intersections. In this step, we also make sure that the shortest paths are indexed in a cache-coherent layout to facilitate our in-simulation queries (see Section 3.4). The running time of the preprocessing phase is $O(k^3+n\log n)$,

Algorithm 1 Taxi ride-sharing simulation

```
1: Inputs: $R = \{r_1, r_2, \ldots\}$: a set of trips
2: Parameters: $m$: number of cabs, $C$: capacities of cabs, $n_{share}$, $d_{delay}$, $d_{extra}$, $f(r, c)$
3: $R \leftarrow \mathrm{Sort}(\mathrm{R})$
4: $\{c_1, c_2, \ldots, c_m\} \leftarrow \mathrm{InitializeCabs}(\mathrm{m}, \mathrm{C})$
5: for $i = 1$ to $|R|$ do
6:     $\mathrm{ElapsedTime} \leftarrow \mathrm{PickUpTime}(r_i) - \mathrm{PickUpTime}(r_{i-1})$
7:     $f^* \leftarrow \infty$
8:     for $j = 1$ to $m$ do
9:         $\mathrm{UpdateState}(c_j)$
10:        $f_{ij} \leftarrow f(r_i, c_j)$
11:        if $f^* > f_{ij}$ then
12:            $f^* \leftarrow f_{ij}$
13:            $c^* \leftarrow c_j$
14:        end if
15:    end for
16:    $\mathrm{Assign}(r_i, c^*)$
17:    $T(i) \leftarrow T(i-1) + f^*$
18: end for
```

Algorithm 2 Cost Function $f(r, c)$

```
1: Inputs: $r$: trip, $c$: cab
2: Parameters: $n_{share}$, $d_{delay}$, $d_{extra}$
3: $S \leftarrow$ list of stops of $c$ including its current location $\{s_0, s_1, \ldots, s_k\}$
4: $D_{drop} \leftarrow \mathrm{ShortestPathDistance}(s_k, r.p_{drop})$        ▷ $s_k$: the last stop in $S$
5: $idx_{pick}, D^* \leftarrow \mathrm{FindPickUpOrder}(r, c, S, D_{drop})$
6: if $idx_{pick} \leq k$ then
7:     $D_{pick} \leftarrow D^* - D_{drop}$
8:     $idx_{drop}, D^* \leftarrow \mathrm{FindDropOffOrder}(r, c, S, idx_{pick}, D^*, D_{pick})$
9: end if
10: return $D^*$
11: Output: $D^*$: an additional distance for cab $c$ to accommodate trip $r$
```

where k is the number of intersections and n is the number of trips.

Note that traffic conditions can be easily integrated in the simulation. One can replace the shortest path matrix with multiple matrices, one for each hour of the day. In each of these matrices, the distance between any two intersections is adjusted according to the traffic conditions. For instance, this can be achieved by scaling the weights of the edges in the road network by the ratio of the average speed on that edge at a given time and a base speed (for instance, speed limit). More generally, we can define traffic conditions as a weight matrix that changes over time, such that its entrywise product with the pre-defined shortest path matrix could be used in place of the latter to reflect traffic conditions into the calculation.

Finally, observe that by separating the notion of a cab from a trip, we gain extra flexibility that is not present in [14]. For example, we are able to study the relationship between the number of cabs and shareability. We can also consider scenarios in which different cabs have different capacities. Moreover, we can select the initial locations for each individual cab.

3.2 Simulation Phase

As shown in Algorithm 1, we consider each of the n trips in chronological order. For each pick-up request r_i, we go through all the cabs in the fleet and update their states based on the time elapsed from the last trip's pick-up time. For each cab c_j, we also compute additional cost $f(r_i, c_j)$ and assign r_i to the cab with the lowest additional cost.

To update the state of a cab (i.e., location, occupancy and stops that it is visiting to pick up and drop off passengers), we need its speed, which is defined to be the speed of the trip whose pick-up or drop-off takes place next. This can be estimated using the trip duration (drop-off time − pick-up time) and distance. Once we have an estimation of the speed, the traveled distance and position of each cab can be interpolated from the time spent since the beginning of the trip. At the same time, we also update the cab occupancy and planned stops if there was any scheduled drop off or pick up along its traveled itinerary.

Algorithm 3 FindPickUpOrder

```
1: Inputs: $r$: trip, $c$: cab, $S = \{s_0, s_1, \ldots, s_k\}$: list of stops of $c$ including its current location, $D_{drop}$: the shortest distance between $s_k$ and $p_{drop}$
2: Fields: $c.C$: capacity of $c$, $r.o_p$: number of passengers of $r$
3: Parameters: $n_{share}$, $d_{delay}$, $d_{extra}$
4: $idx \leftarrow \max\{i | \mathrm{NumOfShares}(s_i) + 1 > n_{share} \vee$
5:                 $\mathrm{Occupancy}(s_i) + r.o_p > c.C\}$
6: $D^* \leftarrow \infty$
7: $idx_{pick} \leftarrow k + 1$
8: $prev \leftarrow s_{idx}$
9: for $j = idx + 1$ to $k$ do
10:    $D_1 \leftarrow \mathrm{ShortestPathDistance}(prev, p_{pick})$
11:    $D_2 \leftarrow \mathrm{ShortestPathDistance}(p_{pick}, s_j)$
12:    $D_3 \leftarrow \mathrm{ShortestPathDistance}(prev, s_j)$
13:    if constraints given by $d_{delay}$ and $d_{extra}$ are satisfied
14:                 for $r$ and trips containing $s \in S$ then
15:        $D \leftarrow D_1 + D_2 - D_3 + D_{drop}$
16:        if $D < D^*$ then
17:            $D^* \leftarrow D$
18:            $idx_{pick} \leftarrow j$
19:        end if
20:    else if constraint given by $d_{delay}$ is NOT satisfied for $r$ then
21:        break
22:    end if
23:    $prev \leftarrow s_j$
24: end for
25: return $idx_{pick}, D^*$
```

A straightforward way to compute the additional cost $f(r_i, c_j)$ is to explicitly find an optimal route for c_j that includes the pick-up and drop-off locations of r_i and to compare its cost with the cost of the current route for c_j. However, computing the optimal path is known as the Sequential Ordering Problem (SOP) which is a version of the Traveling Salesman Problem and is NP-hard [18]. Thus, to make the computation tractable, we use a heuristic to find a best route for c_j to accommodate r_i. We first find a position to insert the pick-up location p_{pick} of r_i into the list of stops S assuming that the order to visit those stops stays the same, and the drop-off location p_{drop} of r_i is added to the end of the route. After that, we adjust the order of p_{drop} so that we can find a route with the lower additional cost. Along with computing $f(r_i, c_j)$, we check if c_j has enough capacity for all the passengers of r_i and if the constraints given by n_{share}, d_{delay} and d_{extra} are satisfied. Otherwise, we set $f(r_i, c_j)=\infty$. After we assign r_i to the cab with the minimal additional cost, we update the stops S to reflect this assignment. The computation of $f(r_i, c_j)$ is described in Algorithms 2, 3 and 4; we discuss it in detail below.

Let $S=\{s_0, s_1, \ldots s_k\}$ be a list of scheduled stops for cab c_j, and let p_{pick} and p_{drop} be the pick-up and drop-off locations of r_i, respectively. If c_j is vacant, the additional cost $f(r_i, c_j)$ is simply the sum of shortest path distances between its current location s_0 and p_{pick}, and p_{pick} and p_{drop}.

If c_j has passengers, we need to find the best positions to insert stops p_{pick} and p_{drop} into S. The challenge is to find the order for those with the smallest additional distance and at the same time satisfy all the constraints, in particular, making sure that each trip is shared with at most n_{share}

Algorithm 4 FindDropOffOrder

```
1: Inputs: $r$: trip, $c$: cab, $S = \{s_0, s_1, \dots, s_k\}$: list of stops of $c$,
   $idx_{pick}$: index of $S$ where to insert $p_{pick}$, $D^*$: additional distance obtained
   from FindPickUpOrder, $D_{pick}$: additional distance to insert $p_{pick}$ at $idx_{pick}$
2: Parameters: $d_{delay}$, $d_{extra}$
3: $prev \leftarrow p_{pick}$
4: for $j = idx_{pick} + 1$ to $k$ do
5:     $D_4 \leftarrow \mathrm{ShortestPathDistance}(prev, p_{drop})$
6:     $D_5 \leftarrow \mathrm{ShortestPathDistance}(p_{drop}, s_j)$
7:     $D_6 \leftarrow \mathrm{ShortestPathDistance}(prev, \mathrm{s_j})$
8:     if constraints given by $d_{delay}$ and $d_{extra}$ are satisfied
9:                 for trips containing $s \in S$ then
10:        $D \leftarrow D_{pick} + D_4 + D_5 - D_6$
11:        if $D < D^*$ then
12:            $D^* \leftarrow D$
13:            $idx_{drop} \leftarrow j$
14:        end if
15:    end if
16:    $prev \leftarrow s_j$
17: end for
18: return $idx_{drop}$, $D^*$
```

other trips and the number of passengers in a cab does not exceed the capacity C at all stops. In addition, for each trip, we ensure that the delay distance (the distance from the location where this trip was assigned to its pick-up location) and extra distance (the length of the route from p_{pick} to p_{drop} including other stops minus the shortest path distance between p_{pick} and p_{drop}) does not exceed d_{delay} and d_{extra} respectively. To do so, we aggregate information about the maximum number of shared trips, maximum occupancy, the delay and distance among the stops and each stop maintains such information. We use this information to check if all the constraints would be satisfied for a trip request and the trips that are already being served.

To compute the additional distance to accommodate r_i, we assume temporarily that the drop-off will happen after the last stop s_k. Let D_{drop} be the length of a shortest path between s_k and p_{drop}. As mentioned above, we first find a position for p_{pick} (Algorithm 3). Suppose $l' \in \{0, \dots, k\}$ is such that each trip containing one of stops $s_{l'}, \dots, s_k$ can be shared with at least one more trip and the occupancy o of c_j does not exceed the capacity C of c_j at any stop of $s_{l'}, \dots, s_k$. For each $l \in \{l', \dots, k{-}1\}$, we try to insert p_{pick} between s_{l-1} and s_l. If the lengths of shortest paths between s_{l-1} and p_{pick}, p_{pick} and s_l, and s_{l-1} and s_l are D_1, D_2 and D_3 respectively, then an additional distance is defined to be $D{=}D_1{+}D_2{-}D_3{+}D_{drop}$. Then, we determine the position of p_{pick} that minimizes this additional distance D and at the same time satisfies the delay and extra distance constraints given by d_{delay} and d_{extra} respectively, for both r_i and all the trips in service. Let $f'(r_i, c_j)$ be the minimum of D over l and $s_p \in S$ be the first stop after p_{pick}.

Note that the algorithm performs pruning. It stops considering a cab once the delay constraint given by d_{delay} is no longer satisfied (Algorithm 3, lines 20, 21). It also prunes stops (the underlying search space of the approach) – it only considers stops that satisfy capacity and sharing constraints (Algorithm 3, line 4, 5).

Next, we search for a best position for p_{drop} (Algorithm 2, line 8). As shown in Algorithm 4, similar to the previous process, for each $s_l \in \{s_{p-1}{=}p_{pick}, s_p, s_{p+1}, \dots, s_{k-1}\}$, we query for the shortest path distances between s_l and p_{drop}, p_{drop} and s_{l+1}, and s_l and s_{l+1}. Let D_4, D_5, D_6 denote these distances respectively. Then, an additional cost for each new route is defined as $D'{=}f'(r_i, c_j){-}D_{drop}{+}D_4{+}D_5{-}D_6$. We select the route that minimizes the additional cost, i.e., $\min\{D', f'(r_i, c_j)\}$, and set $f(r_i, c_j)$ to be this quantity. If a route is such that the extra distance or delay for r_i and trips in service exceed d_{extra} or d_{delay}, then the route is discarded.

Time complexity. The complexity of the simulation phase of our algorithm is $O(nm)$, which is linear in the number of trips. This allows us to scale dynamic pick-up and delivery tasks to large data sets. However, a large number of shortest-path queries is needed for each trip and cab (Algorithm 2, lines 4, 5 and 8; Algorithm 3, lines 10-12; Algorithm 4, lines 5-7). Even though the algorithm is efficient, these queries become a bottleneck. Below, we propose two strategies that exploit parallelism and an efficient shortest-path indexing scheme to support large-scale simulations.

Discussion. Approximation algorithms for SOP have been studied extensively [18], [19]. While our heuristic may appear to have a performance that is suboptimal compared to these algorithms when a trajectory for a particular cab is considered, the tradeoff of using a heuristic approach is substantially reduced in our study due to the large number of cabs involved. More precisely, suppose $\mathcal{T}$ is a set of all possible trajectories. If we assume that the set of optimal trajectories $\mathcal{T}^* \subseteq \mathcal{T}$ associated with each cab is an i.i.d sample of size l, and $X_1, X_2, \dots, X_l$ are costs of these trajectories, then the probability that the minimum cost found by our heuristic is more than ϵ away from the optimal cost Θ is:

$$P(\min(X_i) - \Theta > \epsilon) = \left[P(X_i - \Theta > \epsilon)\right]^l = \left[\tfrac{M-\epsilon}{M}\right]^l \quad (1)$$

where M is the maximum possible cost and the second equality follows from the definition of uniform distribution. Note that Equation 1 assumes that X_i is uniformly distributed. However, a similar result can be proved for a more general class of continuous distributions with density bounded away from zero at Θ, at the cost of a more involved analysis and slightly worse constant. For instance, suppose l=5000, M=30 miles and ϵ=0.2 mile, the probability that the minimum cost derived by the heuristic would be more than 0.2 mile away from the optimal cost is $2.9857e{-}15$.

3.3 Exploring Parallelism

Although our simulation algorithm achieves linear scaling with the number of trips, running a simulation at a large scale, e.g., with one year of data, can be prohibitively expensive. Using one CPU core, our algorithm takes almost 15 minutes to complete a simulation with 11,500 taxis for a single day. To address this problem, we leverage two forms of parallelism.

Intra-Request Parallelism. Since each simulation step depends on the results of the previous step, it is not possible to achieve parallelism at this level, i.e., having each thread execute one step. However, computing the sharing cost of each taxi with respect to a pick-up request can be done in parallel since this evaluation for each taxi is independent from each other (Algorithm 1, lines 9-14). In our implementation, we use a thread pool model to distribute work, i.e., a set of taxis across multiple machine cores. To minimize inter-thread communication, each worker thread processes multiple taxis at time.

Workers still need to synchronize with each other at the end of each pick-up request (Algorithm 1, line 16-17) to assign the best solution for the trip request. Generally, a lock has to be used to avoid race conditions. However, using locks on hundreds of millions of iterations would be a bottleneck itself. Thus, to work around this issue, we make use of atomic operations that are available on modern x86 architectures to construct our work queue in a lock-free manner [20]. Using 8 threads, our simulator was able to finish a one-day run in just under four minutes, with the lock-free queue giving us a 20% boost in performance.

Inter-Partition Parallelism. The intra-request strategy parallelizes tasks based on the number of taxis, which is relatively small. Thus, it is not able to leverage larger systems with thousands of cores. In order to utilize larger resources, we need to parallelize tasks based on the number of pick-up requests. As stated above, running simulations in parallel on separate pick-up requests is not possible due to the dependency of steps. However, we can take advantage of an inherent characteristic of the taxi data: virtually no trips are shared during the early morning hours – 4AM, which coincides with the AM shift change of NYC taxis [3]. Thus, we can safely divide our data into independent simulations of one-day in size without sacrificing correctness. This means that a simulation using one year of data can be divided into 365 sub-tasks, each of which can be run in parallel. A final reduction step is needed for reporting the result of the entire simulation. We note that this phenomenon is not specific to NYC: periods of low taxi activity that occur naturally due to human diurnal cycle can be used to parallelize the simulation.

While traditional frameworks such as MPI can be used to spawn the processes for the above sub-tasks, they offer little aid in performing analysis of the results (such as generating plots of shared trips). We have extended our system to support the MapReduce framework and allowing the integration of analysis tasks using MapReduce jobs. In our setup, each mapper is a simulator program that can process a set of pick-up requests independently. Data filtering may be applied at this stage to limit data requests based on constraints, e.g., a spatio-temporal condition. Most analysis tasks happen in the reduce phase of our framework. Depending on the analysis, users can specify an appropriate output for the map phase. The results reported in Section 4 were obtained using the MapReduce implementation. The pseudocode for one simulation experiment is given in Algorithm 5.

3.4 Cache-Coherent Shortest Path Index

Our simulation algorithm uses shortest path queries extensively (Algorithm 2, 3, 4). This is where our computation spends the most time. In particular, each computation of $f(r_i, c_j)$ makes a series of shortest path queries to all stops of c_j to exhaustively find the minimal solution. Since the complexity of our algorithm is $O(nm)$, this could result in a very large number of queries. For example, performing a simulation of 11500 taxis with $n_{share}{=}4$ on one day worth of taxi trip records (∼300k requests) would require over **3 billion** shortest path queries. Thus, it is of utmost importance that we build an efficient shortest path indexing scheme to support such queries.

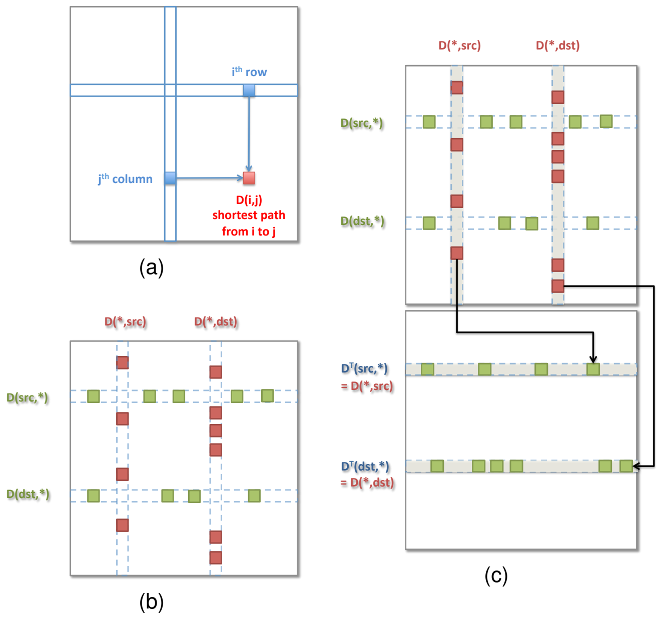


Fig. 2. Cache layout for precomputed shortest paths: (a) a distance matrix as a result of running Dijkstra's algorithm on all intersection pairs; (b) memory access patterns from computing $f(r, c)$ for a pick-up request going from *src* to *dst* – red indicates a high chance of cache misses; (c) storing the transpose of the distance matrix to avoid cache misses in backward lookups.

Our initial approach was to precompute and cache the shortest distances (and their predicates) for all possible intersection pairs in NYC. The storage size of this matrix is fairly small (about 500MB for roughly 10,000 intersections), and would fit completely on commodity PCs. Therefore, our shortest path queries are now reduced to just memory accesses. The data structure for this caching scheme is depicted in Fig. 2a.

Nevertheless, our experiments still show under utilization of CPUs when running large experiments. Inspecting further, we noticed that there was a large number of L2/L3 cache misses for these queries. In fact, over 50% of the memory accesses resulted in a cache miss (more than 1.5 billion misses out of 3 billion accesses). This can be explained by the memory access pattern which is described in Fig. 2b. Each time we compute the cost to accommodate a request, we have to issue shortest path queries originating from as well as going to its pick-up location *src* and drop-off location *dst*. In fact, over 95% of the queries involve *src* and *dst*. Among these queries, forward lookups, i.e., finding shortest paths originating "from" an intersection, would present a cache-coherent memory access patterns (depicted in green). Ideally, each CPU would only need to cache 2 rows of data in order to have all queries resided in cache. However, backward lookups, i.e., finding shortest paths going "to" an intersection, is likely to incur cache misses most (if not all) of the time. This is because elements inside a column could be tens or hundreds of megabytes apart from each other. In this case, the entire shortest path computation would need be in cache to have all requests served without any penalty.

We propose a simple, yet efficient, layout to increase the cache coherence of shortest path lookups. By transposing the shortest path matrix, backward lookups become forward

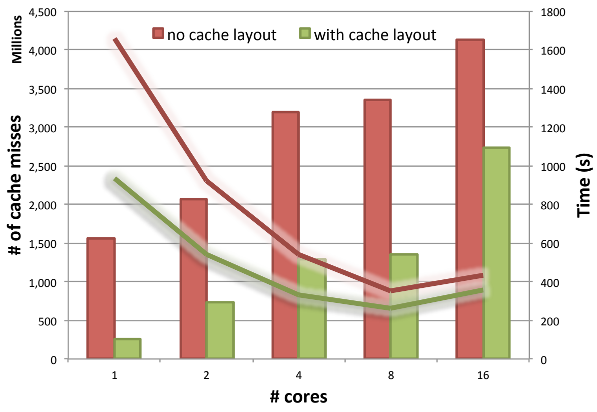


Fig. 3. Shortest path query performance with (green) and without (red) cache-coherent layout. The cache-coherent layout reduced the number of cache misses substantially (3x-6x less); thus, improve the overall simulation performance. The experiments were run on an entire day worth of taxi trip records (~300k requests), using n_{share}=4. The speedup of using multiple cores was not demonstrated after 4 cores, especially at 16 cores, due to the saturation of memory bandwidth that overshadowed the computation cost.

lookups and vice versa. Therefore, we elected to store an additional transposed matrix in our shortest path cache to convert all backward lookups to forward lookups. As illustrated in Fig. 2c, each CPU only needs to keep 4 rows of data in its L2/L3 cache to serve all shortest path queries related to a trip request. This comes at a cost of doubling the shortest path data structure; however, it significantly improves the simulation performance. Fig. 3 shows the shortest path query performance with (green) and without (red) cache-coherent layout. The cache-coherent layout was able to reduce the number of cache misses up to 6 times on a single core, thus, resulting in a 2x speedup of the overall performance. The experiments were run on an entire day worth of taxi trip records (~300k requests), using n_{share}=4. Increasing the number of cores above 4 brings no added benefit. The speed-up from using multiple cores was not observed for more than 4 cores. Using 16 cores, the memory bandwidth is saturated, overshadowing the computation cost. Since using 4 cores yields the best performance ratio, we used this setup for our experiments.

3.5 Complexity Analysis

The *serial complexity* of our simulation is $O(nm)$, asymptotically. A tighter bound is:

$$O(n \cdot (mC_f + C_a)) \tag{2}$$

where C_f and C_a are the complexity of our cost function (Algorithm 2) and taxi assignment (Algorithm 1, line 16), respectively. From Algorithm 3 and 4, both C_f and C_a are $O(|S|)$, where $|S|$ is the maximum number of stops each taxi maintains. Since each trip cannot result in more than two stops, $O(|S|)$ is equivalent to $O(2 \cdot n_{share})$, or simply $O(n_{share})$. This means that the complexity of our algorithm is indeed $O(n_{share}mn)$. However, given that n_{share} is usually much smaller than m and n (e.g., 4 or 5 vs. roughly ~10k and ~500k, respectively), we can consider n_{share} as just a constant.

For the *parallel complexity analysis*, we show that our implementation is *cost optimal*, i.e., its asymptotic running time multiplied by the number of parallel processors involved in the computation is comparable to the running time of the best serial implementation [21]. Since the inter-partition parallelism is a direct share-nothing computation and is expected to achieve cost optimality through the map phase of MapReduce, our focus is on the intra-request parallelism. For a thread pool of size p, the sharing cost computation of m taxis is evenly distributed to p threads, resulting in the time complexity:

$$O(n \cdot (\tfrac{m}{p}C_f + C_s + C_a)) \tag{3}$$

where C_s is the synchronization time of all threads at the end of each request. In our case, C_s=$O(p)$ since we only need to compare solutions of p threads to select the optimal one. By definition, our framework can achieve cost optimality iff:

$$\begin{aligned} p \cdot (n \cdot (\tfrac{m}{p}C_f + C_s + C_a)) &= O(nm) \\ p \cdot (n \cdot (\frac{m}{p}C_f + p + C_a)) &= O(nm) \\ n \cdot (mC_f + p^2 + p \cdot C_a) &= O(nm) \end{aligned} \tag{4}$$

In order to satisfy Equation 4, the following must be true: C_f=$O(1)$, p^2=$O(m)$ (or $p \leq \sqrt{m}$), and $C_a \leq \frac{m}{p}$. In our case, where p is always set to at most 8 (based on our experiment on shortest path query performance in Section 3.4) and n_{share} is a constant, all of the above conditions are always true. Thus, the parallelism in our framework is cost optimal.

The above constraints also define the class of algorithms that can be plugged into our framework without loss of scalability. In particular, the algorithm that finds a route for each cab (C_f) should have a time complexity that is independent of m and n while selecting which cab to service a given trip (C_a) must be done in linear time. These guarantee the asymptotic complexity of the system to be $O(nm)$. However, if a more complex algorithm is desired, it can still be integrated into our framework, possibly with an additional cost. For example, employing a trip selection algorithm that runs in $O(m)$ time or a taxi selection algorithm that runs in $O(m^2)$ time would result in an overall complexity of $O(nm^2)$. Nevertheless, the inter-partition parallelism would always be in place regardless of the scheduling algorithms chosen.

4 Experimental Evaluation

In this section, we demonstrate the scalability of our approach using the NYC taxi trip data set described below. To run our experiments, we used the open source Apache Hadoop software library on a 1200-core cluster. An example MapReduce implementation of a simulation is given in Algorithm 5. We also describe findings of our study which indicate that our approach is effective and derives information that may be useful to policy makers, the taxi industry and riders.

4.1 Data

For our study, we used the NYC taxi trip data set of 2011 and 2012 which was provided to us by Taxi & Limousine

Algorithm 5 Large-scale simulation using MapReduce (Varying the degree of sharing)

```
1:  Parameters P: $m$: number of cabs, $C$: capacities of cabs, $n_{share}$, $d_{delay}$, $d_{extra}$
2:  Input: $L$: list of filenames each of which contains one day worth of trip data
3:  procedure MAP(TripDataFile)
4:      for $n_{share} = 0$ to $4$ do
5:          Result $\leftarrow$ Algorithm 1(TripDataFile, P)
6:          Emit($n_{share}$, (DateOfTrip, Result))
7:      end for
8:  end procedure
9:
10: procedure REDUCE(Key, Values)
11:     Initialize Output
12:     Sort(Values)                                    ▷ Sort by DateOfTrip
13:     for (DateOfTrip, Result) in Values do
14:         for $i = 0$ to Length(Result) do
15:             Output[$i$].Append(Result[$i$])
16:         end for
17:     end for
18:     Emit(Key, Output)
19: end procedure
```

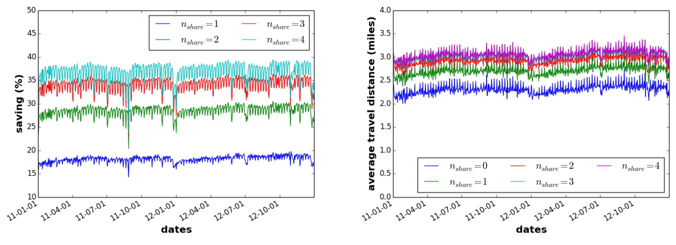
Commission (TLC) through a FOIL request.[2] This represents a superset of the data set used in [14], where only 2011 trips were considered. The data contains information about 360 million trips taken by the 13,237 taxis in NYC. Each trip is represented by a vector with the following fields: taxi ID (medallion ID), pick-up and drop-off times, latitude and longitude for both pick-up and drop-off, travel distance in miles, and number of passengers. All IDs are anonymized. We considered all trips that occurred within Manhattan, and between Manhattan and the two international airports: John F. Kennedy and LaGuardia. These trips constitute the majority of all taxi trips in NYC [22]. We also eliminated data that appeared to be erroneous, including trips without passengers, with the average speed less than 3 mph, and that start or end at invalid locations. The 260 million trips that remained after selection and cleaning were used in our experiments.

4.2 Taxi Ride Sharing in NYC: A Case Study

Recall that taxi ride sharing involves three major entities: the city, taxi companies (drivers) and passengers. Each of them aims to minimize different costs: the city is interested in reducing pollution and traffic, the taxi companies want minimize the operating costs while maximizing revenues, and passengers want to get their destinations as fast and cheaply as possible. Thus, a solution to the taxi ride-sharing problem has to take into account tradeoffs between these competing objectives.

A natural metric that can be used to analyze the effects of ride sharing is the total distance traveled by all the cabs, which correlates with both the traffic volume and emissions. A decrease in the total traveled distance can also serve as an incentive for the taxi industry to engage in ride sharing, since such a decrease is likely to lead to a proportional decrease in the cost of running the business. However, ride sharing also imposes additional costs to the passengers in the form of extra travel distance and additional stops along the way. Therefore, for taxi ride sharing to be a practical solution for reducing emissions and traffic, one also needs to control the burden placed on customers. To account for this,

2. Taxi trip data is now open and can be downloaded from TLC at: http://www.nyc.gov/html/tlc/html/about/trip_record_data.shtml.



(a) (b)

Fig. 4. (a) The percentage of saved total travel distance through ride sharing for n_{share}=$1, 2, 3, 4$ (b) The average travel distance for n_{share}=$0, 1, 2, 3, 4$.

we introduced two parameters in our simulation model: d_{extra}, the maximum extra distance for each trip, and n_{share}, the maximum number of trips that each trip can be shared with. In addition, we use another parameter, d_{delay}, which is a bound on the maximum distance a taxi is allowed to travel to pick up a customer. Note that a constraint on d_{delay} is naturally present in the scenario where no sharing occurs as well. Any constraints on d_{extra} and d_{delay} are equivalent to constraints on t_{extra} and t_{delay} which are expressed in terms of time. For our simulations, we express the constraints in terms of distance. We omit traffic conditions in the experiments because we did not have access to the appropriate traffic data.

The parameters supported by STaRS are described in Table 3. By definition, each trip can be shared with at most n_{share} other trips. Note that the bound on n_{share} controls the maximum number of extra stops, which is at most $2n_{share}$. This parameter is also used in [14], where k=n_{share}+1 is used. However, the optimization proposed in [14] is NP-hard for k>2; thus, k=2 was chosen for the solution to be computationally feasible. Our approach scales for larger values of n_{share} as well, which allows us to study the effects of this parameter on the total savings and costs of the proposed ride-sharing solution.

To keep the waiting and service times within a reasonable interval, we set d_{delay}=1 mile and d_{extra}=2 miles (at most 5 minutes of waiting time and 10 minutes of extra service time if the average taxi speed is 12 mph). Note that our approach scales to include both of these parameters in the study. Finally, for simplicity, we set each taxi's capacity C=4. Our simulation model can handle individual constraints by letting each customer set their own n_{share}, d_{delay}, and d_{extra}. This makes it possible to support more complex scenarios where riders have different ride-sharing preferences.

Varying the Degree of Sharing. We studied the effects of parameter n_{share}. The pseudocode for this experiment is given in Algorithm 5. The simulations assumed 9,500 cabs were active on Sundays and 11,500 cabs were active on the other days of the week. The simulation results for n_{share}=$1, 2, 3, 4$ are presented in Fig. 4 for each day between Jan 1st, 2011 and Dec 31st, 2012. Fig. 4a shows the ratio of the total travel distance saved by ride sharing to the total travel distance of original trips (i.e., the sum of the shortest distances between pick-up and drop-off locations of the serviced trips). Fig. 4b shows the average travel distance for each n_{share}=$1, 2, 3, 4$ as well as n_{share}=0 (no ride sharing). Fig. 5a shows the

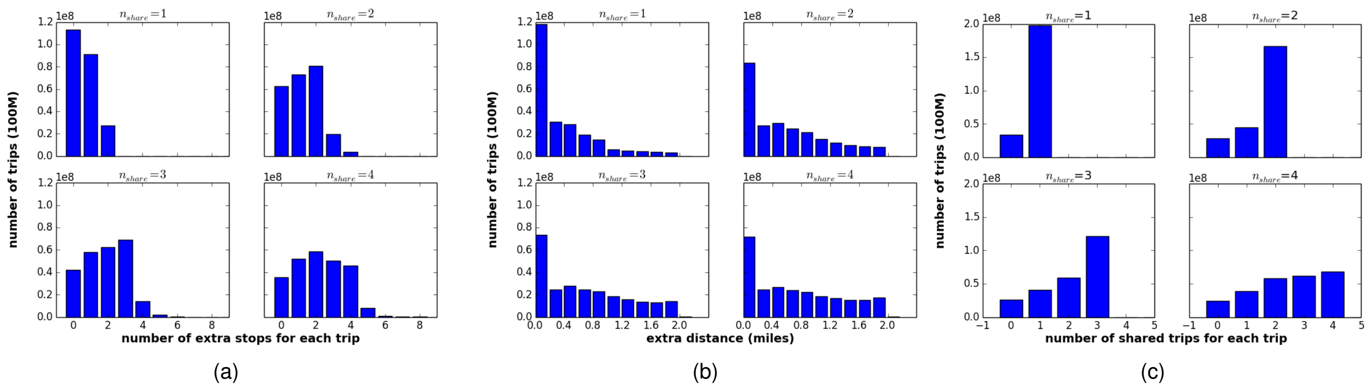


Fig. 5. (a) The distribution of number of extra stops for n_{share}=1, 2, 3, 4. Note that the number of additional stops for a trip is bounded by $2n_{share}$. (b) The distribution of extra distances for n_{share}=1, 2, 3, 4. Note that 0 mile extra distance for a trip means that between the pick-up and drop-off locations of this trip a cab traveled without picking up or dropping off other trips. (c) The distribution of number of shared trips for n_{share}=1, 2, 3, 4

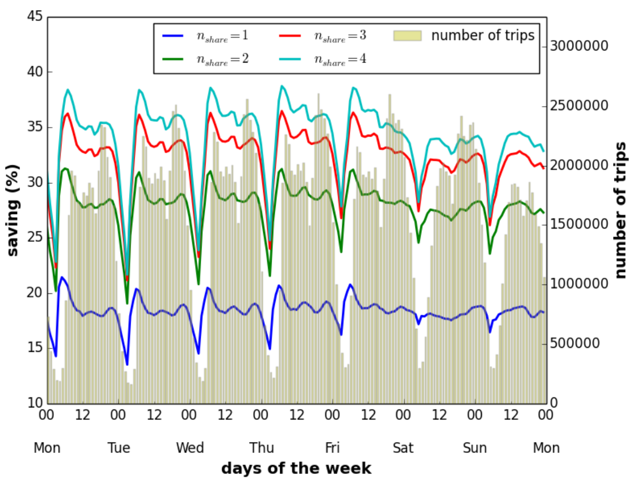


Fig. 6. The average saving by hour for $n_{share} = 1, 2, 3, 4$ and the total number of trips.

	Saved Distance (%)			
n_{share}	1	2	3	4
Airport Trips $C = 4$	33	42	44	44
Airport Trips $C = 8$	35	46	50	54
Trips within Manhattan	17	27	33	36
All Trips	18	29	34	37
	Avg Extra Distance (mi.)			
Airport Trips $C = 4$	0.21	0.35	0.42	0.44
Airport Trips $C = 8$	0.22	0.38	0.48	0.54
Trips within Manhattan	0.35	0.57	0.68	0.72
All Trips	0.35	0.57	0.68	0.72

TABLE 4
Comparison of city-wide ride sharing with city-airport ride sharing. Recall that C is the capacity of each cab. See Section 2 for details.

distribution of the number of additional stops for different values of n_{share} (1 to 4). For example, for n_{share}=1, 120×10^6 trips have no additional stops, and the maximum number of additional stops is 2. But when n_{share} increases to 2, fewer than 80×10^6 trips have 0 additional stops, and now, there are trips with up to 4 additional stops. Note that without ride sharing, the number of extra stops for each trip is 0. Fig. 5b is similar to Fig. 5a, but shows the distribution of the extra travel distance for different sharing levels. We have also analyzed the distribution of the number of trips each particular trip was shared with. Fig. 5c suggests that as we increase n_{share} shareability increases.

In addition to the simulations by day (4am to 4am on the next day), we also conducted simulations by hour on the same data set and results are summarized in Fig. 6. Further variations between weekend and weekday, as well as, other temporal features, can be captured in our simulation by replacing a single shortest path matrix with a different one for each hour of the day. A similar approach can also be used to capture traffic conditions when data is available.

Naturally, our results quantify a tradeoff between the savings in the total distance through ride sharing and burden incurred by customers. As we increase n_{share}, the savings in the total distance increase. On the other hand, this also leads to an increase in the travel distance and number of extra stops for each trip. However, our results also suggest that contrary to the findings in [14], n_{share}=2 or even 3 may be an optimal bound on the maximum number of trips to be shared, offering a better tradeoff between the savings and costs. In particular, for n_{share}=2 the total saving is 28.6% on average with the average extra distance of 0.57 miles, while for n_{share}=1 the saving is 18.2% with the average extra distance of 0.35 miles.

Airport vs. City Trips. Trips between Manhattan and the airports are easier to share due to the fact that their origins or destinations are often co-located, so these trips can provide a benchmark for our ride-sharing solution. We have conducted a separate simulation that included only trips between Manhattan and John F. Kennedy International Airport and trips between Manhattan and LaGuardia Airport. The results, summarized in Table 4, suggest that airport ride sharing is very effective and can serve as the first step for a city-wide implementation. Furthermore, sharing trips within Manhattan leads to savings that, albeit lower (see Figure 4b), are comparable to the airport results, and thus in-city sharing should also be considered.

Varying the Number of Cabs. To further demonstrate the flexibility of our framework and scalability of our algorithm, we have studied the effects of another parameter, the number of cabs, on the ride-sharing simulation. Our results show that the savings, average extra distance, av-

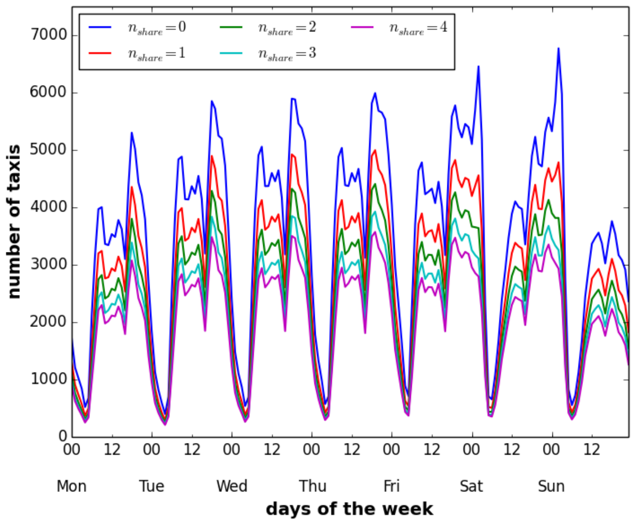


Fig. 7. The average of the minimum numbers of cabs required to achieve 95% of service rate for $n_{share} = 0, 1, 2, 3, 4$.

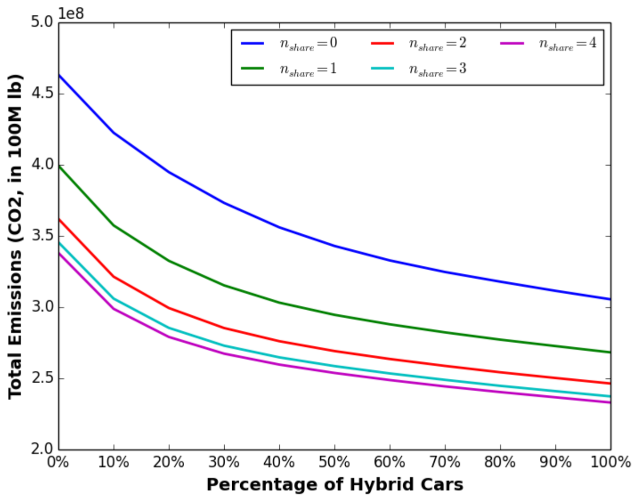


Fig. 8. Total CO_2 emissions for $n_{share} = 0, 1, 2, 3, 4$.

erage number of extra stops, and the distribution of the number of shared trips are not affected by the number of cabs used in simulation. However, as the number of taxis increases, the number of accommodated trips increases as well. We have also observed that without ride sharing, we can serve at most 95% of the trips in the data set due to the nature of our simulation. This motivated us to study the following question: what is the minimum number of taxis that is required to achieve a good service rate with taxi ride sharing?

To answer this question, we used binary search to find the minimum number of cabs that can achieve 95% service rate for $n_{share} = 0, 1, 2, 3, 4$ for every hour on each day in 2011 and 2012. We would like to emphasize that in general this binary search procedure is computationally intensive since it requires that an individual simulation be run for each parameter setting on each iteration. However, due to scalability properties of our framework, we can efficiently handle such parameter sweeps. The running time of this binary search on 2 years of data was roughly 1 hour 10 minutes.

Fig. 7 shows the average of the minimum number of taxis by hour for each day of the week. The weekly pattern shown in Fig. 7 for $n_{share} = 0$ (no sharing) matches the statistics presented in [3], which provides evidence for the validity of our simulation approach.[3] For $n_{share} > 0$, we see that larger values of n_{share} lead to a fewer number of cabs required to achieve 95% service rate and decreasing the number of cabs in combination with ride sharing might be a potential solution to decrease emissions and traffic.

Different Cost Function. We also illustrate the flexibility of our framework by optimizing a different cost function: the total CO_2 emission. Note that if taxis are all of the same type, that is, each taxi emits the same amount of CO_2, optimizing total CO_2 emission is the same as optimizing the total distance. However, since hybrid cars were introduced, taxi companies have replaced some of conventional gasoline cars with hybrid ones to reduce emissions and lower fuel cost. Thus, for this simulation, we assume that there are two different types of cars, namely hybrid and conventional gasoline cars and they emit 0.57 lb. CO_2 and 0.87 lb. CO_2 per mile respectively [23]. In our experiment, we simulated ride sharing for $n_{share} = 0, 1, 2, 3, 4$ and at the same time varied the percentage of hybrid cars in the fleet. Fig. 8 presents the simulations results. We observe that if a trip shares a cab with at most 1, 2, 3 and 4 trips without hybrid cars the amount of total CO_2 is approximately equivalent to having 20%, 40%, 50% and 60% of hybrid cars on the road. In addition, the results indicate that by sharing taxis we could still reduce gasoline emissions further once the taxi fleet consists entirely of hybrid cars.

3. Note that we do not exactly match the results from [3] since we do not explicitly include the transition between hours in our simulation.

4.3 Comparing Different Optimization Procedures

To assess the effectiveness of our optimization procedure, we compared it against three distinct approaches: selecting a taxi that is the closest to the pick-up location of a trip request; randomly selecting a taxi; and a combination of our optimization procedure and the random approach that samples a pre-defined number of taxis (4 in our experiments) and assigns a trip to the one with the minimal additional cost. All of these approaches must comply with the same constraints as our optimization procedure. The results presented in Fig. 9 show that our optimization procedure achieves a significant lower objective value compared to other natural baseline approaches. Furthermore, the effectiveness of our optimization procedure increases compared to other baselines as we increase n_{share}. Note that the worst case computational complexity of all of these algorithms is the same since at each iteration the status of all taxis has to be updated.

4.4 Comparison against the Shareability Network

We compare the simulation results obtained by our approach against the results reported for the Shareability Network (SN) in [14], which, to the best of our knowledge, is the only prior work that supports large-scale ride-sharing simulation. For direct comparison, we ran simulations with

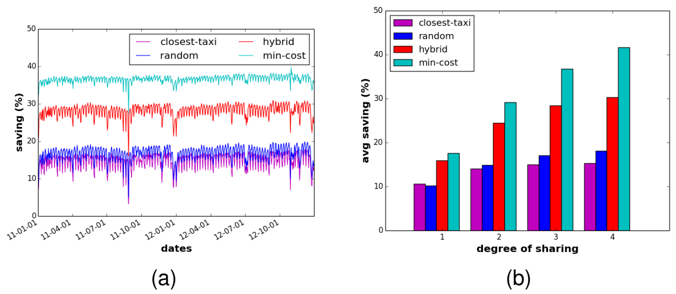


Fig. 9. (a) The comparison of the percentage of saved total travel distance through ride sharing for n_{share}=3 for different optimization procedures: `min-cost` (ours), `closest-taxi`, `random` and `hybrid`. (b) The comparison of the average percentage of saved total travel distance through ride sharing for n_{share}=1, 2, 3, 4 for different optimization procedures.

	Our Framework		Shareability Network [14]	
Parameter Settings	Saved Trips	Saved Time	Saved Trips	Saved Time
n_{share}=1, d_{extra}=0.6 mi.	47%	18%	16%	8%
n_{share}=1, d_{extra}=1 mi.	46%	16%	27%	13%
n_{share}=2, d_{extra}=0.6 mi.	60%	31%	16%	8%
n_{share}=2, d_{extra}=1 mi.	61%	29%	27%	13%

TABLE 5
Comparison of our framework and Shareability Network approach [14]. Note that n_{share}=1, 2 are equivalent to k=2, 3 and d_{extra}=0.6, 1 mi. correspond to Δ=3, 5mins in [14]. Also, for direct comparison we took the results of [14] for δ=0 (δ: time window) since our framework is a real-time model.

the same road network and data as in SN. The results are presented in Table 5. In columns 2 and 4, we report the ratio of the resulting number of trips through sharing to the original number of trips in the data set. Columns 3 and 5 contain the ratio of the simulated travel time to the original travel time. Observe that in our simulations, most of the trips are shared: 47% of saved trips with n_{share}=1, translating into 94% of trips being shared. On the other hand, the percentage of shared trips using SN is much lower; this is because with δ=0 (where δ is the time window; see Section 5), the nodes in the SN are connected only if the start time of the two trips is very close and it leads to a much smaller network. This also results in significantly smaller savings in travel time.

4.5 Performance and Scalability

We conclude this section by reporting the execution time of ride-sharing simulation for different data sizes using a 1200-core Hadoop cluster. Table 6 summarizes the results. We observe that the average time it takes to process one day of data decreases as the size of data increases since we can use the parallelism more effectively. This shows that our framework scales to large data sets and enables us to study taxi ride sharing by exploring many different scenarios in timely manner.

5 Related Work

Ride sharing has been framed as an instance of the "dynamic pickup and delivery" problem [8], [9], and it has

Data Size	Execution Time	Avg Execution Time per Day
1 day	2 mins, 18 sec	—
1 week	3 mins, 44 sec	32 sec
1 month	5 mins, 56 sec	12 sec
3 months	6 mins, 39 sec	4.4 sec
0.5 year	7 mins, 21 sec	2.4 sec
1 year	9 mins, 55 sec	1.6 sec
2 years	18 mins, 17 sec	1.5 sec
3 years	26 mins, 40 sec	1.5 sec

TABLE 6
Execution time of the simulation on a 1200-core Hadoop cluster.

also been addressed via linear programming [10], [11], [12]. These optimization-based approaches are applicable to small-scale problems, for instance, sharing within airports, since large-scale optimization is often computationally infeasible. Heuristic-based solutions have been proposed for real-time dispatching of taxis [25], [26], but they also have limited scalability.

More recently, data-driven approaches have emerged. The work that is most closely related to ours is the simulation model proposed by Santi et al. [14]. They used a graph-based approach based on the notion of "shareability network", where nodes correspond to taxi trips, and two nodes are connected if those trips can be shared. Their model aims to maximize the number of shared trips or minimize the total time taken to accommodate all the trips. The structure of the shareability network depends crucially on two parameters: the maximum number of shared trips k per service and the maximum delay Δ that a customer can tolerate in a shared taxi service trip. These parameters control computational complexity of the problem. This solution is tractable only for k=2: the problem becomes NP-hard for larger values of k. Similarly, larger values of Δ translate into larger networks, requiring longer computation time. These restrictions limit the scenarios that one can explore with this approach. Moreover, their model can derive solutions that are not feasible in real life, since it does not explicitly take into account taxi positions and their capacity – it only examines whether it is beneficial to share a set of trips. For example, suppose that we decide that two particular trips t_1 and t_2 must be shared. Then, there must be a cab, say c, that will serve these two trips. Suppose also that in the original data set, c was serving trip t_1 and some other trip t_3 in this order. If t_1 and t_2 are assigned to c, it may not be possible for c to serve t_3 any more (or the cost may be too high). It is also unclear in this scenario what happens to the cab that was serving t_2 in the original data set.

Another limitation of this approach is that it assumes that trips are known in advance. While this assumption matches well car pooling scenarios where time and location of each trip are fixed in advance, it is not suitable for taxi ride sharing, since trip requests arrive in real time. To address this issue, Santi et al. proposed a refinement of their model that prunes the shareability network to allow trips that start within a time window δ (e.g., five minutes) from each other to be shared. However, the model is real time only when δ=0, in which case our experiments suggest that this model tends to underestimate the benefits of the taxi ride sharing.

Ma et al. [15], [16] proposed a real-time dispatch system

	Our Framework	**Shareability Network** [14]	**T-Share** [15], [16]	**Approach in** [24]
Optimization	Minimize total travel distance/time	Maximize # of shared trips or minimize total travel time	Minimize total travel distance/time	Minimize total travel distance/time
Assumptions and limitations		All trips known in advance, cabs' locations and capacities are not part of the model		Infinite cab capacity
Model	Real-time simulation	Offline simulation	Real-time dispatch	Real-time scheduling
Parametrized	✓ $n_{share} \geq 0$ (see Table 3 for other parameters)	Limited $k{=}n_{share}{+}1 \leq 2$ and max additional travel time Δ	✗	✓ Scheduling capacity ($\approx n_{share}$), waiting time and service constraints, and # of servers
Scalability				
as dispatch system	✓	✗	✓	✓
as study tool	✓	Limited	Limited	Limited

TABLE 7
Comparison of data-driven approaches. Recall that n_{share} is the maximum number of trips shared by a trip.

for taxi ride sharing. While related, our goal is different: we aim to support the simulation of a wide range of ride-sharing scenarios, and designed a model that can be parameterized accordingly. Ma et al. attain fast response times by splitting a region into grid cells such that the distance between any two locations can be computed "heuristically" as the distance between the cells containing them. This allows their system to keep shortest path computations at a minimum, but at the cost of reduced accuracy. Moreover, the results are dependent on the selected grid size. Like Ma et al., our model demands fast response times for queries that match trips to cabs. However, our system always uses the "exact" shortest paths for optimizing ride-sharing schedules. We were able to achieve this with good performance and scalability using a cache-coherent indexing scheme (Section 3.4). While our focus is on simulation, the experimental results indicate that our approach is promising for the dispatching scenario as well.

Huang et al. [24] proposed scheduling algorithms to dynamically match trip requests to vehicles with the minimum cost while trip waiting and service time constraints are satisfied. They showed that the kinetic tree algorithms outperform commonly used approaches, such as branch-and-bound and mixed-integer programming. As discussed in Section 3, such algorithms can be integrated into our framework.

6 Conclusion

In this paper, we presented STaRS, a new framework that is both scalable and flexible to support the simulation of a rich set of realistic taxi ride-sharing scenarios. The scalability properties of the framework make it possible to run large-scale studies that explore a wide range of what-if scenarios through parameter sweeps. We have shown that this model attains a good balance between simplicity and expressiveness. Another important contribution of this work is the novel shortest path indexing scheme where we make use of cache-coherent layout to speed up shortest path queries substantially. The implementation of our simulation model is fully integrated with Hadoop's MapReduce, thus, enabling a variety of batch analysis tasks on taxi ride sharing. We applied the model to NYC taxi data and presented a case study that illustrates the capabilities and effectiveness of our system and design decisions.

There are several avenues we plan to pursue in future work. Our current shortest path indexing technique maintains a full distance matrix in memory. Though this could be mapped on disk, the storage size ($O(|V|^2)$) will not scale well for a large road network. We would like to experiment with a tiled caching strategy where we only keep the distance matrix for the most popular intersections and perform full shortest path computation for less popular nodes. In addition, we would like to implement a load balancer for the shortest path queries where the shortest path database could be located on a separate machine/cluster. This would allow us to make better use of the computing resources when having multiple simulator instances.

Acknowledgments

The authors thank the New York City TLC for providing the data used in this paper. We also thank Paolo Santi and Giovanni Resta for making their data available and sharing valuable insights with us. This work was supported in part by a Google Faculty Award, an IBM Faculty Award, an CUNY IRG Award, the Moore-Sloan Data Science Environment at NYU, the NYU School of Engineering, the NYU Center for Urban Science and Progress, AT&T, DARPA, NSF awards CNS-1229185, CNS-1405927, CNS-1544753, and CCF-1533564.

References

[1] T. W. Bank, "Urban Development," http://data.worldbank.org/topic/urban-development, 2014, accessed on February 2nd, 2014.
[2] V. Handke and H. Jonuschat, *Flexible Ridesharing*. Springer Berlin Heidelberg, 2013.
[3] M. Bloomberg and D. Yassky, "Taxicab factbook," 2014.
[4] C. Morency, "The ambivalence of ridesharing," *Transportation*, vol. 34, no. 2, pp. 239–253, 2007.
[5] M. Ben-Akiva and T. J. Atherton, "Methodology for short-range travel demand predictions: Analysis of carpooling incentives," *JTEP*, vol. 11, no. 3, pp. 224–261, 1977.
[6] R. Teal, "Carpooling: Who, how and why," *Transportation Research Part A: General*, vol. 21, no. 3, pp. 203–214, 1987.
[7] K. Dueker, B. Bair, and I. Levin, "Ride sharing: psychological factors," *Transportation Engineering Journal*, vol. 103, no. 6, pp. 685–692, 1977.
[8] J. Yang, P. Jaillet, and H. Mahmassani, "Real-time multivehicle truckload pickup and delivery problems," *Transportation Science*, vol. 38, no. 2, pp. 135–148, 2004.
[9] G. Berbeglia, J.-F. Cordeau, and G. Laporte, "Dynamic pickup and delivery problems," *EJOR*, vol. 202, no. 1, pp. 8 – 15, 2010.
[10] A. Marin, "Airport management: taxi planning," *Annals of Operations Research*, vol. 143, no. 1, pp. 191–202, 2006.
[11] G. Keith, A. Richards, and S. Sharma, "Optimization of taxiway routing and runway scheduling," in *AIAA Guidance, Navigation and Control Conference and Exhibit*, 2008.

[12] P. Shaw, "Using constraint programming and local search methods to solve vehicle routing problems," in *Principles and Practice of Constraint Programming CP98*, ser. Lecture Notes in Computer Science, M. Maher and J.-F. Puget, Eds. Springer Berlin Heidelberg, 1998, vol. 1520, pp. 417–431.

[13] B. Coltin and M. Veloso, "Scheduling for transfers in pickup and delivery problems with very large neighborhood search," in *AAAI*, 2014, pp. 2250–2256.

[14] P. Santi, G. Resta, M. Szell, S. Sobolevsky, S. Strogatz, and C. Ratti, "Quantifying the benefits of vehicle pooling with shareability networks," *PNAS*, vol. 111, no. 37, pp. 13 290–13 294, 2014.

[15] S. Ma, Y. Zheng, and O. Wolfson, "T-share: A large-scale dynamic taxi ridesharing service," in *ICDE*. IEEE, 2013.

[16] S. Ma, O. Wolfson, and Y. Zheng, "Real-time city-scale taxi ridesharing," *IEEE Transactions on Knowledge Discovery and Data Engineering*, vol. 27, pp. 1782–1795, 2015.

[17] "Open street map," http://openstreetmap.org/.

[18] M. Mojana, R. Montemanni, G. Di Caro, and L. M. Gambardella, "An algorithm combining linear programming and an ant system for the sequential ordering problem." *ATAI*, pp. 80 – 85, 2011.

[19] N. Ascheuer, M. Jnger, and G. Reinelt, "A branch & cut algorithm for the asymmetric traveling salesman problem with precedence constraints." *Computational Optimization and Applications*, vol. 17, no. 1, pp. 61–84, 2000.

[20] D. Dice, D. Hendler, and I. Mirsky, "Lightweight contention management for efficient compare and swap operations," *CoRR*, vol. abs/1305.5800, 2013.

[21] D. Sima, T. Fountain, and P. Kacsuk, *Advanced Computer Architectures*. Addison-Wesley, 1997.

[22] H. Farber, "Is tomorrow another day? the labor supply of new york city cabdrivers," *Journal of Political Economy*, vol. 113, no. 1, pp. 46–82, 2005.

[23] U.S. Department of Energy, "Emissions from Hybrid and Plug-In Electric Vehicles," http://www.afdc.energy.gov/vehicles/electric_emissions.php, 2012, accessed on November 26th, 2014.

[24] Y. Huang, F. Bastani, R. Jin, and X. S. Wang, "Large scale real-time ridesharing with service guarantee on road networks," *PVLDB*, vol. 7, no. 14, pp. 2017–2028, 2014.

[25] W. M. Herbawi and M. Weber, "A genetic and insertion heuristic algorithm for solving the dynamic ridematching problem with time windows," in *GECCO '12*, 2012, pp. 385–392.

[26] A. Kleiner, B. Nebel, and V. A. Ziparo, "A mechanism for dynamic ride sharing based on parallel auctions," in *IJCAI*, 2011, pp. 266–272.

Masayo Ota is a PhD candidate in Computer Science and Engineering at New York University, where she works under the supervision of Professor Juliana Freire. Her research interests include big-data analysis, management and integration, with application to urban data. Prior to starting the doctoral program at New York University, she received a B.S. in Computer Science from the University of Toronto, Canada, and a B.A. in Economics from Rikkyo University, Japan.

Huy Vo is an Assistant Professor of Computer Science at the City College of New York (CCNY) and a member of the doctoral faculty at The Graduate Center, City University of New York (CUNY). He is also a faculty member at the Center for Urban Science and Progress, New York University. His current research focuses on high performance systems for interactive visualization and analysis of big data sets, specifically in urban applications. He has over ten years of research experience in large-scale data analysis and visualization. He has co-authored 39 technical papers and 3 patents, and contributed to several widely-used open-source systems.

Cláudio Silva is a professor of computer science and engineering and data science at New York University. His research interests include visualization, data science, graphics and geometric computing, in particular the analysis of urban and sports data. He has held positions in academia and industry, including at AT&T, IBM, Lawrence Livermore National Labs, Sandia National Labs, and the University of Utah. He has published over 220 journal and conference papers, is an inventor of 12 US patents. He is an IEEE Fellow and was the recipient of the 2014 IEEE Visualization Technical Achievement Award. Cláudio's research is currently being funded by the NSF, DOE, NASA, AT&T, IBM, MLB Advanced Media, the Moore Foundation, and the Sloan Foundation.

Juliana Freire is a Professor of Computer Science and Data Science at New York University. Her recent research has focused on big-data analysis and visualization, large-scale information integration, provenance management, and computational reproducibility. She has held positions in industry and academia, including at Bell Labs and the University of Utah. Prof. Freire is an active member of the database and Web research communities, with over 170 technical papers, many open-source systems, and 11 U.S. patents. She is an ACM Fellow and a recipient of an NSF CAREER, two IBM Faculty awards, and a Google Faculty Research award. Her research grants are from the National Science Foundation, DARPA, Department of Energy, National Institutes of Health, Sloan Foundation, Gordon and Betty Moore Foundation, W. M. Keck Foundation, Google, Amazon, the University of Utah, New York University, Microsoft Research, Yahoo! and IBM.