

## **PTRider: A Price-and-Time-Aware Ridesharing System**

Chen, Lu; Gao, Yunjun; Liu, Zixian; Xiao, Xiaokui; Jensen, Christian Søndergaard; Zhu, Yifan

*Published in:*  
Proceedings of the VLDB Endowment

*DOI (link to publication from Publisher):*  
[10.14778/3229863.3236229](https://doi.org/10.14778/3229863.3236229)

*Creative Commons License*  
CC BY-NC-ND 4.0

*Publication date:*  
2018

*Document Version*  
Publisher's PDF, also known as Version of record

[Link to publication from Aalborg University](#)

*Citation for published version (APA):*  
Chen, L., Gao, Y., Liu, Z., Xiao, X., Jensen, C. S., & Zhu, Y. (2018). PTRider: A Price-and-Time-Aware Ridesharing System. *Proceedings of the VLDB Endowment*, 11(12), 1938-1941.  
<https://doi.org/10.14778/3229863.3236229>

### **General rights**

Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain
- You may freely distribute the URL identifying the publication in the public portal -

### **Take down policy**

If you believe that this document breaches copyright please contact us at [vbn@aub.aau.dk](mailto:vbn@aub.aau.dk) providing details, and we will remove access to the work immediately and investigate your claim.

# PTRider: A Price-and-Time-Aware Ridesharing System

Lu Chen<sup>#,‡,1</sup>, Yunjun Gao<sup>#,2</sup>, Zixian Liu<sup>#,3</sup>, Xiaokui Xiao<sup>§,4</sup>, Christian S. Jensen<sup>‡,5</sup>, Yifan Zhu<sup>#,6</sup>

<sup>#</sup>College of Computer Science, Zhejiang University, Hangzhou, China

<sup>‡</sup>Department of Computer Science, Aalborg University, Denmark

<sup>§</sup>School of Computing, National University of Singapore, Singapore

{<sup>1</sup>luchen, <sup>2</sup>gaoyj, <sup>3</sup>zx\_liu, <sup>6</sup>xtf\_z}@zju.edu.cn    <sup>4</sup>xkxiao@nus.edu.sg    <sup>5</sup>csj@cs.aau.dk

## ABSTRACT

Ridesharing is popular among travellers because it can reduce their travel costs, and it also holds the potential to reduce travel time, congestion, air pollution, and overall fuel consumption. Existing ridesharing systems (e.g., lyft, uberPOOL) often offer each traveler only one choice that aims to minimize system-wide vehicle travel distance or time. In this demonstration, we present a price-and-time-aware ridesharing system, termed as PTRider, which provides more options. It considers both pick-up time and price, so that travellers are able to choose the vehicle matching their preferences best. To answer the ridesharing request in real time, PTRider builds indexes on the road network and vehicles separately, and utilizes corresponding efficient matching methods. A real-life dataset that contains 432,327 trips extracted from 17,000 Shanghai taxis for one day (May 29, 2009) is used to demonstrate that PTRider can return various options for every ridesharing request in real time.

## PVLDB Reference Format:

Lu Chen, Yunjun Gao, Zixian Liu, Xiaokui Xiao, Christian S. Jensen, and Yifan Zhu. PTRider: A Price-and-Time-Aware Ridesharing System. *PVLDB*, 11(12): 1938-1941, 2018.  
DOI: <https://doi.org/10.14778/3229863.3236229>

## 1. INTRODUCTION

Ridesharing refers to a transportation scenario where travellers with similar itineraries and time schedules share a vehicle for a trip and split the travel costs such as fuel, tolls, and parking fees. Many ridesharing systems (e.g., lyft [1], uberPOOL [2], etc.) already exist, due to the reduced travel costs, travel time, congestion, fuel consumption, and air pollution, for the travellers, for society, and for the environment.

Existing real-time ridesharing systems [6], [7], [8] find only one option per request that tries to minimize system-wide vehicle travel time or distance. Nonetheless, considering the case below. Assume that a couple has finished dinner at the seaside, which is far from the city center. They now want to travel home. Given that there are few nearby vehicles, getting a vehicle quickly is likely to cost extra because some vehicles must make a detour in order to pick them up. However, if they are willing to wait longer,

they may pay less since some vehicles will be nearby later on. If we offer more options with different pick-up times and prices, travellers can choose the options they prefer. Motivated by this, we develop a price-and-time-aware ridesharing system based on our previous work [5], termed as *PTRider*, which offers several options per ridesharing request. The options have different pick-up times and prices as well as they do not dominate each other (option  $r_i$  dominates option  $r_j$  iff the pick-up time and price of  $r_i$  are earlier and lower than those of  $r_j$  [3]).

Although one study [4] considers both pick-up time and price to return multiple options for each ridesharing request, it cannot be employed to solve our problem, due to the following two reasons. First, the problem definition is different. Cao et al. [4] assume that every vehicle has one pair of a start location and a destination, and serves *only one* group of riders during a trip, which limits the usability and scalability of the ridesharing system. In contrast, we assume that the destination of a vehicle is not limited and that it can accommodate *any number* of rider groups during a trip as long as it satisfies a capacity constraint. Second, Cao et al. [4] use Euclidean distance for pruning, which is inefficient. In order to answer the ridesharing request in real time, we build indexes on the road network and vehicles separately, and propose two efficient matching methods. To sum up, the key contributions of this demonstration are as follows:

- PTRider considers both pick-up time and price, and thus, it can return different options for each rider to choose.
- PTRider utilizes efficient indexing techniques for both the road network and the vehicles, and adopts two efficient matching approaches, which follow the single-side search paradigm and the dual-side search paradigm, respectively.
- We demonstrate PTRider using a real dataset that contains 432,327 trips extracted from Shanghai taxis, with high simulated ridesharing request and update workloads.

The rest of the demonstration is organized as follows. Section 2 defines the price-and-time-aware ridesharing. Section 3 presents a PTRider prototype. Section 4 provides demonstration details.

## 2. DEFINITIONS FOR PTRIDER

In this section, we give the definitions related to the price-and-time-aware ridesharing.

### 2.1 Road Network

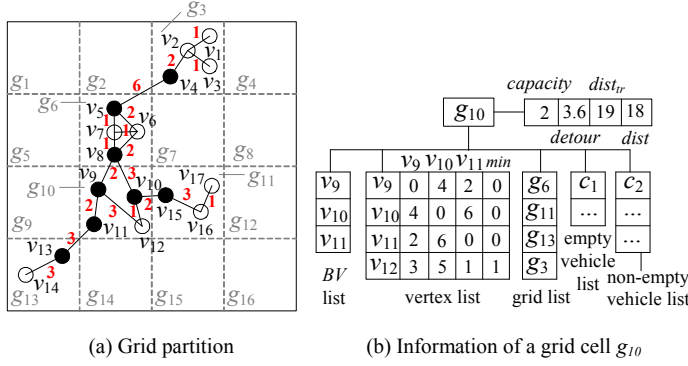
A road network  $G = \langle V, E, W \rangle$  consists of a vertex set  $V$  and an edge set  $E$ . Each vertex  $v \in V$  denotes the intersection of two roads. Each edge  $e = (u, v) \in E$  that connects two vertices  $u$  and  $v$  is associated with a weight  $W(e)$ , in which  $W(e)$  represents the travel cost between  $u$  and  $v$ . The travel cost could be either time or distance. When the speeds of vehicles are known, they can be

This work is licensed under the Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License. To view a copy of this license, visit <http://creativecommons.org/licenses/by-nc-nd/4.0/>. For any use beyond those covered by this license, obtain permission by emailing [info@vldb.org](mailto:info@vldb.org).

*Proceedings of the VLDB Endowment*, Vol. 11, No. 12

Copyright 2018 VLDB Endowment 2150-8097/18/8.

DOI: <https://doi.org/10.14778/3229863.3236229>



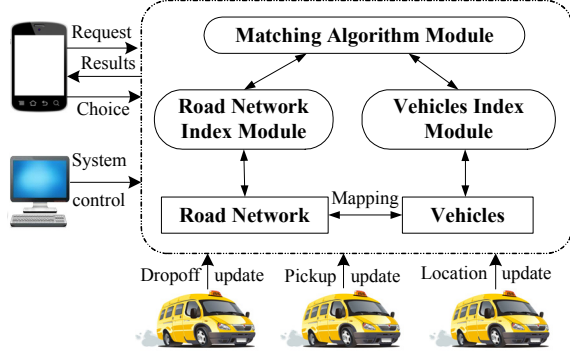


Fig. 2. Framework of PTRider

answers a ridesharing request follows three steps. (i) A rider requests a trip using his/her smart phone. He/She specifies the number of riders, and the start location and destination of the request. PTRider sets a global maximum waiting time and a global service constraint. (ii) Once the request is submitted, the matching method module finds all the qualified options, and sends them to the rider. Here, an option contains the pick-up time and price provided by a specific taxi. (iii) The rider chooses one that matches his/her preference best, and sends it back to the system. Then, the index modules are updated according to the rider's choice. Note that, the taxis update their locations periodically, and update their trip schedules when they pick up or drop off riders. Therefore, the index modules need to be updated.

## 3.2 Indexing of PTRider

### 3.2.1 Indexing the Road Network

We partition the road network using a grid, as illustrated in Fig. 1(a). If an edge  $e = (u, v)$  belongs to more than one grid cell, we call  $u$  and  $v$  **border vertices**. In order to estimate the shortest path distance more accurately, we maintain a matrix that stores the lower bound distance for every grid cell pair. Also, the lower bound distance is associated with a vertex pair  $(x_i, y_j)$ , where  $x_i$  and  $y_j$  are border vertices in the grid cells  $g_i$  and  $g_j$  respectively, i.e.,  $(x_j, v_1, \dots, v_k, y_j)$  is the shortest path connecting  $g_i$  and  $g_j$ .

As depicted in Fig. 1(b), each grid cell maintains five lists: (i) a **border vertex list** of border vertices belonging to the grid cell; (ii) a **vertex list** of vertices belonging to the grid cell, as each vertex  $v$  is associated with its shortest path distances to the border vertices ( $BV$ ) of the grid cell ( $\{\langle u, \text{dist}(v, u) \rangle \mid u \in BV\}$ ) and is also associated with  $\min\{\text{dist}(v, u) \mid u \in BV\}$  (denoted as  $v.\text{min}$ ); (iii) a **grid cell list** of other grid cells sorted in ascending order of the travel time from those grid cells to the grid cell (i.e., sorted in ascending order of the lower bound distances when speed is constant); (iv) an **empty vehicle list** of the vehicles with empty ridesharing requests in  $g_i$ ; and (v) a **non-empty vehicle list** of vehicles  $c_j$  having a non-empty set of ridesharing requests, i.e., vehicles  $c_j$  are currently located in  $g_i$  or are scheduled to enter  $g_i$ .

Base on the grid index, several pruning lemmas are developed by estimating price and pick-up time. For more technical details, please refer to our full research paper [5].

### 3.2.2 Indexing Vehicles

Each vehicle  $c$  is represented by using (i) the unique identifier  $c.ID$  of the vehicle; (ii) the current location  $c.l$  of the vehicle; (iii) the set  $c.S$  of unfinished ridesharing requests assigned to it, sorted

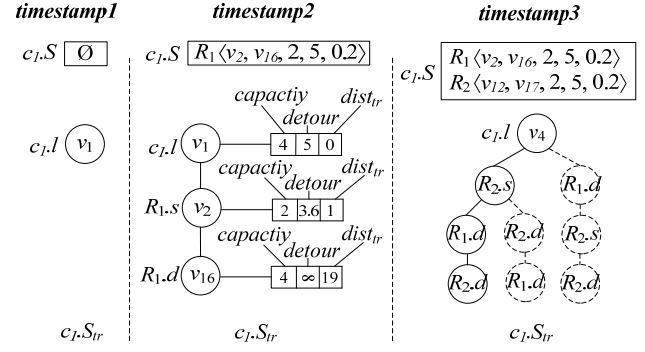


Fig. 3. Kinetic tree examples on vehicle trip schedules

in ascending order of their timestamps; and (iv) the set  $c.S_{tr}$  of all valid vehicle trip schedules.

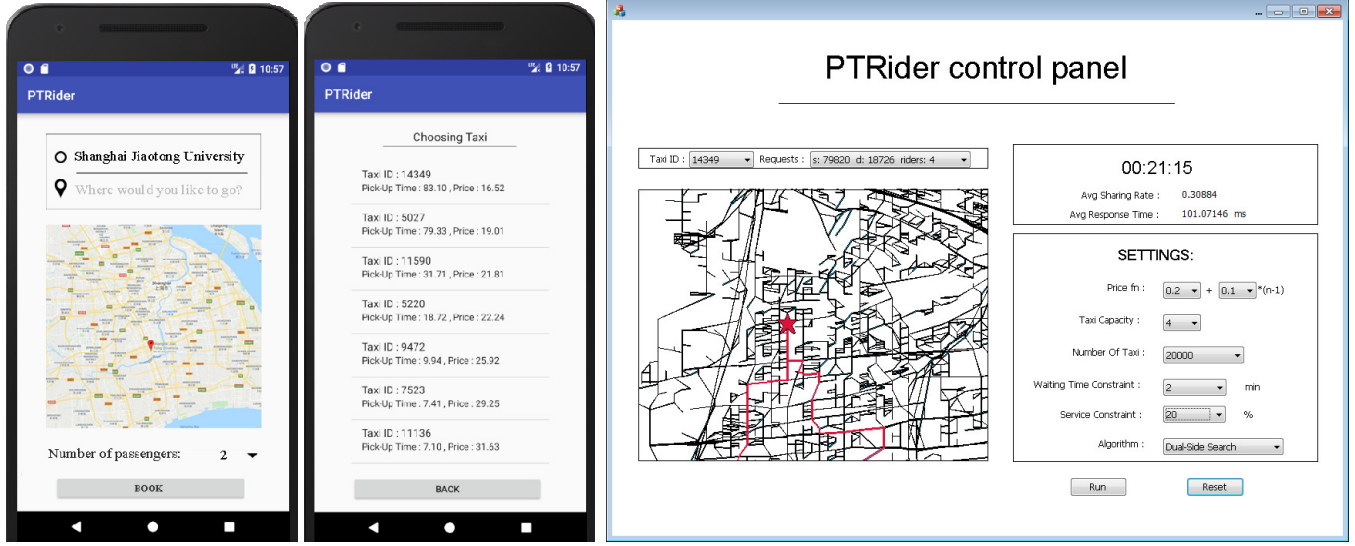
The set  $c.S_{tr}$  of all valid vehicle trip schedules can be managed by a kinetic tree [7], as illustrated in Fig. 3, where every branch denotes a valid trip schedule. In addition, we add three variables to each node  $o_x$  in the kinetic tree: (i) a current vehicle's *capacity*, (ii) the minimal *detour* distance allowed, and (iii) the trip distance  $\text{dist}_{tr}$  from  $c$ 's current location to  $o_x$ . The *detour* can be computed easily from the waiting time and service constraints. If an edge  $\langle o_x, o_y \rangle$  of vehicle  $c$ 's kinetic tree intersects with, or is contained in a grid cell  $g_i$  (i.e., the shortest path between  $o_x$  and  $o_y$  intersects with or is included in  $g_i$ ), then  $\langle o_x, o_y \rangle$  belongs to the non-empty vehicle list of  $g_i$ . The kinetic tree is updated per time unit.

## 3.3 Matching Algorithms

A naive method can be extended directly from the kinetic tree algorithm. We evaluate every vehicle to find all possible pairs of pick-up time and price that cannot dominate each other when inserting the request  $R$  into its kinetic tree. However, it can be improved in two ways: (i) we can filter unqualified vehicles in advance instead of verifying all vehicles; and (ii) the number of the shortest path distance computations ([7] calculates all the distances before verification) can be reduced when inserting the request into the kinetic tree. To boost efficiency, PTRider utilizes two algorithms proposed in [5], as described below.

**Single-Side Search Algorithm.** For each ridesharing request  $R = \langle s, d, n, w, \epsilon \rangle$ , single-side search algorithm starts from the grid cell  $g_i$  where  $s$  locates, and then, it searches the grids in order of their distances to  $s$ . During the search, empty and non-empty vehicles are processed separately. For empty vehicles that cannot be pruned,  $R$  can be easily inserted into its trip schedule set. For every non-empty vehicle that cannot be pruned, the inserting is similar as that in [7], by estimating the lower and upper bounds of the shortest path distance to avoid unnecessary computations.

**Dual-Side Search Algorithm.** Single-side search algorithm filters unqualified vehicles using pruning lemmas when searching from the start location. Similarly, we can also filter unqualified vehicles using pruning heuristics when searching from the destination. Take the following case as an example. One existing trip schedule is near the start location of a ridesharing request, but it is far from the destination of the ridesharing request. In this case, the dual-side search paradigm can avoid more unnecessary verifications of vehicles. Thereafter, similar as single-side search algorithm, dual-side search algorithm verifies each vehicle that cannot be pruned in order to find qualified results.



(a) Request input interface (b) Result display interface (c) The website interface

Fig. 4. Demonstration of PTRider

## 4. DEMONSTRATION

PTRider is demonstrated by using a real data set that contains 432,327 trips extracted from 17,000 Shanghai taxis for one day. The ridesharing requests are generated by using trips in the real dataset, or submitted by the users. The vehicles are initialized uniformly in the road network. They follow a specified route when customers are on board or, otherwise, follow the current road segment, choosing a random segment to follow at intersections. A constant speed  $D$  (48km/hr) is assumed. The underlying system of PTRider runs on an Intel Core i7 3.6GHz PC of a 16GB memory. PTRider can be visited through a smartphone interface to book a taxi in real time, or via a website interface to set the parameters and view the statistics.

### 4.1 Smartphone Interface

The smartphone interface allows the users to book a taxi that matches their requests best, i.e., sending a ridesharing request and finding their preferred taxis. First, as depicted in Fig. 4(a), the rider inputs a start location (suggested as the current location), a destination, and the number of riders in the textbox, and clicks the book button to send the ridesharing request to the system. Second, the system returns all the possible options back to the rider, including pick-up time and price, as illustrated in Fig. 4(b). Third, the rider selects his/her preferred option, and clicks the chosen option to send his/her choice to the system.

### 4.2 Website Interface

The website interface (i) shows the trip schedules on a map, (ii) provides the statistics of the trip information, and (iii) allows the system administrator to set the constraints.

The left-side of Fig. 4(c) shows the trip schedules for any taxi. The system administrator first selects a taxi ID, and then, the unfinished ridesharing requests assigned to the selected taxi are depicted in a drop-down box next to the taxi ID. In addition, all possible trip schedules of the selected taxi are shown as the red lines on the map, where the red star denotes the current position of taxi. Each red line denotes a possible trip schedule for the selected taxi, i.e., each branch in the corresponding kinetic tree.

The top-right of Fig. 4(c) illustrates the statistics of PTRider, including the current time, the average response time, and the average sharing rate. As observed, the system is efficient (low average response time) and effective (high average sharing rate).

In the bottom-right of Fig. 4(c), the system administrator is able to input the parameters including the taxi capacity, the number of taxis, the maximal waiting time, the service constraint, and the price calculator function. Although some parameters can be set by the rider himself/herself (e.g., the maximal waiting time and the service constraint) or by the taxi driver (e.g., the taxi capacity), PTRider adopts a global setting for simplification. In addition, the particular matching algorithm (i.e., single-side search algorithm or dual-side search algorithm) used in PTRider can be specified.

## 5. ACKNOWLEDGEMENTS

This work was supported in part by the 973 Program Grant No. 2015CB352502, NSFC Grants No. 61522208, and U1609217, MOE2015-T2-2-069, SUG from NUS, DiCyPS project, and Obel Family Foundation. Yunjun Gao is a corresponding author.

## 6. REFERENCES

- [1] [online] lyft. <https://www.lyft.com>
- [2] [online] uberPOOL. <https://www.uber.com>
- [3] S. Borzsonyi, D. Kossmann, and L. Stocker. The skyline operator. In *ICDE*, 2001, 421–430.
- [4] B. Cao, L. Alarabi, M. F. Mokbel, and A. Basalamah. SHAREK: A scalable dynamic ride sharing system. In *MDM*, 2015, 4–13.
- [5] L. Chen, Q. Zhong, X. Xiao, Y. Gao, P. Jin, and C.S. Jensen. Price-and-Time-Aware Dynamic Ridesharing. In *ICDE*, 2018.
- [6] S. Ma, Y. Zheng, and O. Wolfson. T-share: A large-scale dynamic taxi ridesharing service. In *ICDE*, 2013, 410–421.
- [7] Y. Huang, F. Bastani, and R. Jin. Noah: A dynamic ridesharing system. In *SIGMOD*, 2013, 985–988.
- [8] R. S. Thangaraj, K. Mukherjee, G. Raravi, and A. Metewar. Xhare-a-Ride: A search optimized dynamic ride sharing system with approximation guarantee. In *ICDE*, 2017, 1117–1128.