A Demonstration of SHAREK: An Efficient Matching Framework for Ride Sharing Systems

Louai Alarabi[1], Bin Cao[2], Liwei Zhao[2], Mohamed F. Mokbel[1], Anas Basalamah[3]

[1]University of Minnesota, Minneapolis, MN, USA
[2]Zhejiang University of Technology, Hangzhou, China
[3]KACST GIS Technology Innovation Center, Umm Al-Qura University, Saudi Arabia
[1]{louai,mokbel}@cs.umn.edu, [2]{bincao,201412044}@zjut.edu.cn, [3]abasalamah@gistic.org

ABSTRACT

Recently, many ride sharing systems have been commercially introduced (e.g., Uber, Flinc, and Lyft) forming a multi-billion dollars industry. The main idea is to match people requesting a certain ride to other people who are acting as drivers on their own spare time. The matching algorithm run by these services is very simple and ignores a wide sector of users who can be exploited to maximize the benefits of these services. In this demo, we demonstrate SHAREK; a driver-rider matching algorithm that can be embedded inside existing ride sharing services to enhance the quality of their matching. SHAREK has the potential to boost the performance and widen the user base and applicability of existing ride sharing services. This is mainly because within its matching technique, SHAREK takes into account user preferences in terms of maximum waiting time the rider is willing to have before being picked up as well as the maximum cost that the rider is willing to pay. Then, within its course of execution, SHAREK applies a set of smart filters that enable it to do the matching so efficiently without the need to many expensive shortest path computations.

Categories and Subject Descriptors

H.2.8 [**Database Applications**]: Spatial databases and GIS

Keywords

Ride Sharing, Dynamic Matching, Indexes, Road-network, Real-time systems

1. INTRODUCTION

Ride sharing services (e.g., Uber [12], Flinc [4], and Lyft [6]) have become very common forming a multi-billion dollars industry. The main idea of these services is to match a set of riders' requests to a set of drivers who have expressed their willingness to offer a ride for someone. All these ride sharing systems rely on a simple matching algorithm that matches each rider request to a close-by driver. Unfortunately, such simple matching has two main limitations [3]: (1) It assumes that the driver is either constantly driving to satisfy on-demand users' requests on the road or parking around a particular event place waiting for a ride call. The destination of the driver depends on the riders' destinations. Hence, the matching algorithm does not take into consideration the case that the driver is already heading to some predetermined destination. (2) The cost of the ride sharing service is set by the service provider based on the ride time and distance. Hence, the shared ride cost is not taken into consideration when matching riders with drivers. With such two limitations, the user base of ride sharing services is still limited as many users are not able to participate as drivers picking up riders on their way to work or back home. Meanwhile, recent research (e.g., T-Share [7] and Noah [5, 11]) has proposed matching techniques that either rely on the availability of enormous historical trajectories [7] or an expensive kinetic tree structure [5, 11]. Both assumptions render them impractical to sue for real-time scalable ride sharing services.

In this demo, we demonstrate SHAREK [2]; a driver-rider matching algorithm that can be embedded inside existing ride sharing services (e.g., Uber, Flinc, and Lyft). By avoiding the above two limitations of current matching algorithms, SHAREK has the potential to boost the performance and widen the user base and applicability of existing ride sharing services. SHAREK takes into consideration the direction that the driver is already heading to, which will immediately result in increasing the user base of ride sharing services. For example, using SHAREK as a matching service within Uber will allow someone who is returning from work to home to pick up a rider who is going into a similar direction or nearby destination. As of now, such user cannot use Uber, as the user does not want to wonder in the street. Yet, the user only wants little disturbance to her original route. As a result, SHAREK increases the user base participating as drivers for existing ride sharing services.

Meanwhile, SHAREK takes into consideration the cost of the ride when matching drivers with riders. A driver who will have less disturbance to her original route will be paid less, while a driver who will make a big divergence to her original route will be paid more. As a result, a rider will be matched with drivers who will be paid less as a matter of convenience. SHAREK takes that cost consideration in its own equation when it matches drivers with riders. Hence, users of a ride sharing service (e.g., Uber) that is equipped with SHAREK matching algorithm will pay less for the same ride that they will get from another ride sharing service that is not equipped with SHAREK. This will definitely affect the number of users using a certain ride sharing service.

The main idea of SHAREK is to match riders, requesting a ride sharing service, to a set of drivers who can provide the requested service, while taking into account: (a) the cost of the ride sharing

SIGSPATIAL'16 October 31 - November 03, 2016, Burlingame, CA, USA

ACM ISBN 978-1-4503-4589-7/16/10.

DOI: http://dx.doi.org/10.1145/2996913.2996983

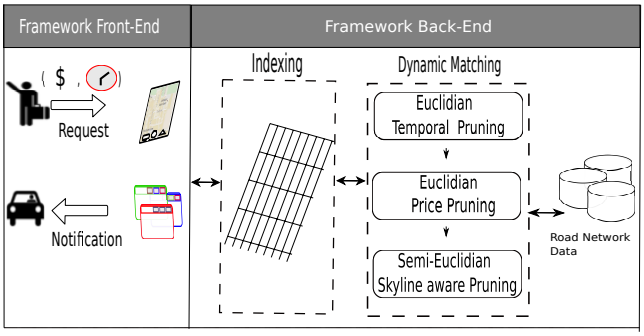

Figure 1: System Overview

service, and (b) the convenience of the service for both rider and driver. SHAREK achieves its efficiency and scalability through applying three consecutive phases, namely, *Euclidian Temporal Pruning*, *Euclidian Cost Pruning*, and *Semi-Euclidean Skyline-aware Pruning*, with an explicit goal to avoid the expensive shortest path computations as much as possible. We will demonstrate SHAREK through a full fledged system prototype where conference audience are allowed to submit ride sharing requests through either a nicely designed web-interface or mobile application. Large synthetic data sets will be generated by Minnesota Traffic Generator [8], based on Brinkhoff model [10] to simulate a large set of riders and drivers in San Francisco, USA.

2. SYSTEM OVERVIEW

Figure 1 presents the system overview of SHAREK. The *Front-End* of the system includes a web-interface and mobile app, allowing users to interact directly with SHAREK. Ride service providers (i.e., drivers) login to SHAREK to indicate their origin and destination locations. On the other side, riders submit their requests by disclosing four pieces of information: (1) current location, (2) final destination, (3) maximum waiting time before being picked up by a driver, and (4) maximum price that the rider is willing to pay for ride sharing service. SHAREK *Back-End* includes two main components: indexing and dynamic matching. As SHAREK users constantly move in the space, SHAREK maintains a *Grid index* to serve as an infrastructure that is capable of supporting location-related queries while updating workloads generated by large numbers of traveling users (Section 3). The *Dynamic Matching* component starts by defining a cost model for each user request (Section 4). Then, it goes through three phases (Section 5), namely, *Euclidian Temporal Pruning*, *Euclidian Cost Pruning*, and *Semi-Euclidean Skyline-aware Pruning*, that employ the cost model to prune as many drivers as possible without calculating actual road network shortest path distances. Finally, the rider receives a set of drivers from SHAREK that can offer the requested ride within the waiting time and price constraints. Once the rider selects the corresponding driver, the driver is notified immediately to accept or decline the ride service.

3. INDEXING

Indexing is one of the core modules in SHAREK, as given in Figure 1. In order to sustain the recent locations and facilitate the searching of candidate drivers, the underlying spatial index needs to be well selected. Tree-based moving object indexing methods would encounter frequent update operations that cause too much overhead to the index structure. Hence, a simple, yet effective, grid-based spatial index [13] is adopted by SHAREK to maintain such dynamic locations.

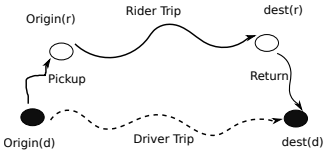

Figure 2: The illustration for different costs

Once a driver registers with the system, SHAREK is allowed to track the driver location to assess its suitability for any ride sharing service. Once the driver reaches its destination, the driver is unregistered from SHAREK. During this life-cycle, the grid index is updated whenever the driver location moves between the index grid cell. SHAREK also manages other static location information in the same index structure, such as users' origins and destinations locations .

4. COST MODEL

Figure 2 gives an illustrative example for the price cost model of SHAREK. The origins and destinations of driver d and rider r are plotted as black and white circles, respectively. The dotted line represents the original driver d trip. The solid line represents the detour that the driver d will encounter to provide a ride sharing service to r. Basically, d has to travel from its origin location $orig(d)$ to the rider origin location $orig(r)$. Then, d has to go through the rider trip till $dest(r)$ to drop off r. Finally, d will need to go to its destination $dest(d)$.

According to this setting, the price for the ride sharing service offered from driver d to rider r, $Price(d, r)$, has two components: (1) The cost of the rider trip from its origin to destination. This is intuitive as at least the rider needs to pay the cost of its own route. (2) The cost of the detour that the driver d will encounter to pickup and drop off r, then to return to its own destination. This part of the price cost will play a major role in matching drivers to riders, as drivers with less detour will be favored over drivers with longer detours. Based on the cost model, SHAREK can pinpoint those drivers that can make it to the rider within its cost and temporal constraints. Formally, the price can be represented by the following equation:

$$Price(d, r) = RiderTrip(r) + Detour(d, r)$$

$Detour(d, r)$ can be calculated as the difference between the new route of the driver d (the solid line in Figure 2) and its original route (the dotted line in Figure 2, taking the rider trip into consideration as it is not necessary has a same route to the driver trip. This can be formally stated as:

$$\begin{aligned} Detour(d, r) = Pickup(d, r) + RiderTrip(r) \\ +Return(d, r) - DriverTrip(d) \end{aligned}$$

From the above two equations, we get the equation used in SHAREK to calculate the cost of any ride sharing service from driver d to rider r, as

$$\begin{aligned} Price(d, r) = Pickup(d, r) + 2 * RiderTrip(r) \\ +Return(d, r) - DriverTrip(d) \end{aligned} \quad (1)$$

It is important to note here that the price cost of any trip between two end points is proportional to the shortest path road network distance between the two end points. For example $Pickup(d, r)$ is proportional to the shortest path network distance between $orig(d)$ and $orig(r)$.

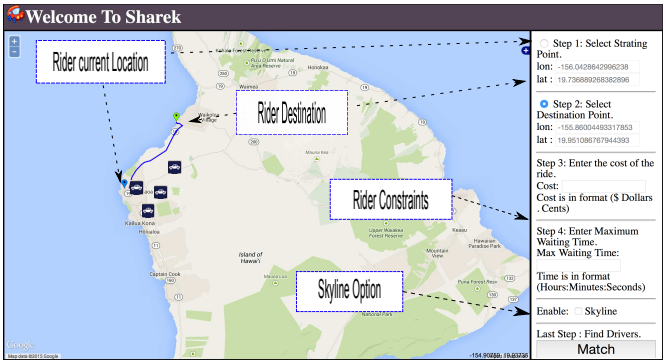

Figure 3: SHAREK Front-end Interface

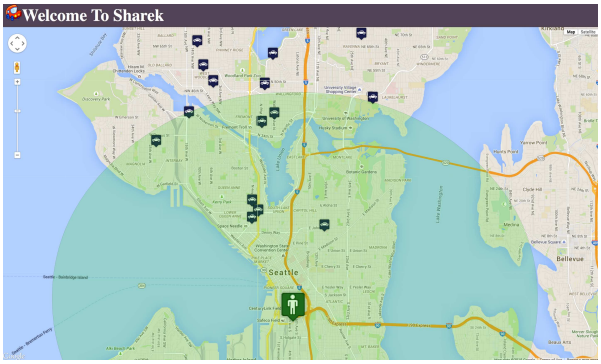

Figure 4: Phase I: Euclidean Temporal Pruning

5. DYNAMIC MATCHING

SHAREK aims to match riders with drivers that satisfy the rider's constraints: (1) Maximum waiting time that the rider is willing to wait before being picked up and (2) Maximum price the rider affords to pay for the ride sharing service. One trivial way to realize the above matching criteria is to calculate the actual price cost and waiting time for all registered drivers in the system. Then, for those drivers that satisfy rider constraints, we run a two-dimensional skyline algorithm to report drivers that are not dominated by each other [1]. Such trivial way is prohibitively expensive as it encounters large numbers of road network shortest path computations and it is not practical given the online environment of ride sharing requests.

The challenge of implementing a ride sharing framework is to determine how to match drivers with riders efficiently with minimal shortest path computations. SHAREK successfully avoids such prohibitive computations through the three following consecutive phases:

Phase I: Euclidian Temporal Pruning. In this phase, SHAREK runs a range query that exploits the grid index to retrieve the drivers within a circular range centered at the rider location with a radius set to the Euclidean distance corresponding to the maximum waiting time constraint of the rider. With this, SHAREK significantly narrows down the search space for matching by ignoring those drivers that are outside the range query, since there is no way for them to satisfy the rider temporal constraint. The output of this phase is a list of candidate drivers that can pickup the rider within the maximum waiting time constraint.

Phase II: Euclidean Cost Punning. In this phase, SHAREK employs a conservative Euclidean distance computation to prune more drivers from the candidate drivers list received from Phase I, without computing any road network shortest path. This idea is to substitute the price cost from Equation 1 with its corresponding Euclidean_price cost as in Equation 2. If a certain driver cannot satisfy Euclidean_price, then there is no need to calculate the actual price cost, i.e., shortest path network cost. The output of this phase is a list of candidate drivers that can pickup the rider within the maximum waiting time and Euclidean_price cost.

$$EuclideanPrice(d,r) = EuclideanPickup(d,r) + 2 * RiderTrip(r) + EuclideanReturn(d,r) - DriverTrip(d) \quad (2)$$

Phase III: Semi-Euclidean Skyline-aware Pruning. The input to this phase is a list of candidate drivers produced from Phase II. SHAREK adopts an incremental road network nearest-neighbor algorithm (iKNN) [9] that retrieves the drivers one by one from the candidate list, based on their actual road network shortest path. At this stage, all drivers in the candidate list should realize the price cost model in Equation 1. A straight forward solution is to report all candidate drivers to the rider, after pruning out those who do not pass the price cost model. Yet, it is not practical to bother users to select a driver from such a long list. Therefore, SHAREK decided to shift the burden from the rider by pruning drivers that dominate others in term of maximum waiting time and maximum price. Thus, the skyline computations are injected inside this phase, which helps in pruning even more drivers without any shortest path computations. As a matter of fact, instead of considering the skyline computations as an overhead, SHAREK actually considers it as a blessing, where SHAREK inherits the skyline behavior inside its technique [2]. Finally, the output of this phase is a drivers list, where all drivers satisfy the rider constraints and they do not dominate each other in both dimensions, i.e., maximum waiting time and price.

6. DEMONSTRATION SCENARIO

This section shows four different scenarios that show the usability, efficiency, and internal processing of SHAREK. All these scenarios are fully interactive where conference attendees can interact with SHAREK either through seeing it running live (Scenario 1), submitting a ride request (Scenario 2), visualizing the system internal operations (Scenario 3), or understanding its output results in terms of its skyline pruning techniques (Scenario 4).

6.1 Scenario 1: Simulation Environment

SHAREK will be continuously running throughout the conference demo session by simulating dynamic matching of 1,000 rider requests to 10,000 drivers in San Francisco, USA. The drivers and riders routes are generated by Minnesota road network traffic generator [8] based on Brinkhoff model [10]. Each trip includes the origin, destination, and set of locations associated with time-stamp. Conference audience can visualize riders requests coming and how SHAREK is handling them internally.

6.2 Scenario 2: Requesting a Ride

Conference audience can interact with SHAREK system by submitting ride sharing requests through either a nicely designed web interface or mobile app, as shown in Figure 3. SHAREK obtains user's current location immediately from the smart phone. Then, the user specify her final destination along with the ride constraints. The final set of drivers is reported to the user on a side panel, where the user can select her preferred driver based on time and price preferences.

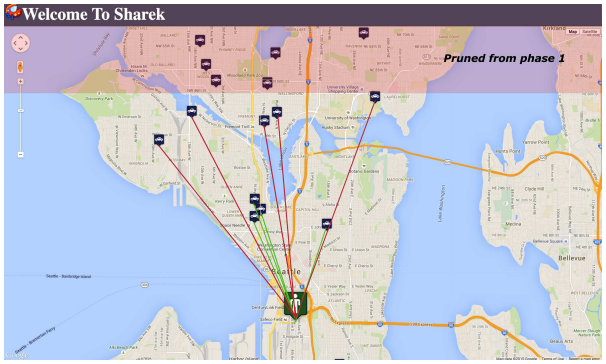

Figure 5: Phase II: Euclidean Cost Pruning

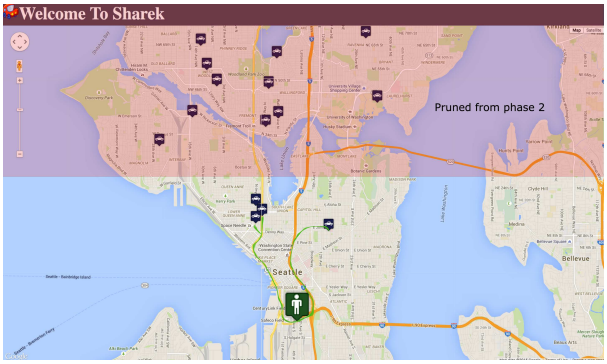

Figure 6: Phase III: Semi-Euclidean Skyline-aware Pruning

6.3 Scenario 3: System Internals

Figures 4, 5, and 6 illustrate SHAREK system internals, which help conference attendees to distinguish the high performance of the framework and visualize the processing of each phase sequentially. Once a request is received, SHAREK processes it by employing three consecutive phases. The first phase (*Euclidean Temporal Pruning*) is shown in Figure 4, where it takes advantage of user maximum waiting time constraint to prune a set of drivers without computing any shortest path operation. Figure 5 illustrates the second phase (*Euclidean Cost Pruning*), where SHAREK prunes drivers from the previous phase based on Euclidean cost price model (Equation 2). In the Third phase, *Semi-Euclidean Skyline-aware Pruning*, SHAREK starts computing the shortest paths in a very conservative way to prune drivers and report a candidate driver list that satisfies not only the temporal and cost constraints, but also represents a set of skyline results. Figure 6 shows how computing the shortest path in conservative way done using an incremental nearest-neighbor query (iKNN).

6.4 Scenario 4: Understanding Skyline Drivers

Figure 7 shows the final result of SHAREK dynamic matching algorithm, in which only three drivers reported that did not dominate by others in term of maximum waiting time and maximum price; so-called (Skyline drivers). SHAREK employs a *skyline technique*, which enables SHAREK to prune more drivers that are dominated by others. Conference attendees can encounter the effectiveness of this optimization method by disabling and enabling the skyline option. It is important to note that by default skyline option is enabled in SHAREK, but for a demonstration purposes we allow conference audience to experience both options to understand the effectiveness of reporting only skyline drivers compared to retrieving all candidate drivers.

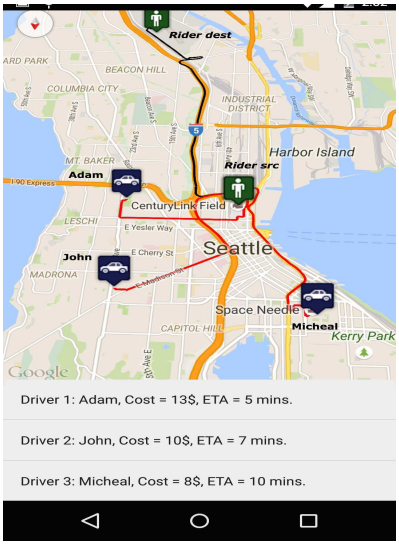

Figure 7: Understanding Skyline Drivers

7. REFERENCES

[1] S. Börzsönyi, D. Kossmann, and K. Stocker. The skyline operator. In *ICDE*, pages 421–430, 2001.

[2] B. Cao, L. Alarabi, M. F. Mokbel, and A. Basalamah. SHAREK: A Scalable Dynamic Ride Sharing System. In *MDM*, pages 4–13, Pittsburgh, PA, jun 2015.

[3] L. Chen, A. Mislove, and C. Wilson. Peeking beneath the hood of uber. In *Proceedings of the 2015 ACM Internet Measurement Conference, IMC 2015, Tokyo, Japan, October 28-30, 2015*, pages 495–508, 2015.

[4] Carpooling-the flinc carpooling service: flinc. https://flinc.org/.

[5] Y. Huang, F. Bastani, R. Jin, and X. S. Wang. Large scale real-time ridesharing with service guarantee on road networks. In *VLDB*, volume 7, pages 2017–2028, 2014.

[6] Lyft: On-demand ridesharing. http://www.lyft.me.

[7] S. Ma, Y. Zheng, and O. Wolfson. T-share: A large-scale dynamic taxi ridesharing service. In *ICDE*, pages 410–421, 2013.

[8] M. F. Mokbel, L. Alarabi, J. Bao, A. Eldawy, A. Magdy, M. Sarwat, E. Waytas, and S. Yackel. Mntg: an extensible web-based traffic generator. In *SSTD*, pages 38–55. Springer, 2013.

[9] D. Papadias, J. Zhang, N. Mamoulis, and Y. Tao. Query processing in spatial network databases. In *VLDB*, pages 802–813, 2003.

[10] Thomas brinkhoff: Network-based generator of moving objects. http://iapg.jade-hs.de/personen/brinkhoff/generator/.

[11] C. Tian, Y. Huang, Z. Liu, F. Bastani, and R. Jin. Noah: A dynamic ridesharing system. In *SIGMOD*, pages 985–988, 2013.

[12] Uber: On-demand ridesharing. https://www.uber.com.

[13] X. Yu, K. Q. Pu, and N. Koudas. Monitoring k-nearest neighbor queries over moving objects. In *ICDE*, pages 631–642, 2005.