Noah: a dynamic ridesharing system

Conference Paper · June 2013

DOI: 10.1145/2463676.2463695

5 authors, including:

Yan Huang
University of North Texas
102 PUBLICATIONS **4,190** CITATIONS
SEE PROFILE

Zhi Liu
University of North Texas
4 PUBLICATIONS **57** CITATIONS
SEE PROFILE

Ruoming Jin
Kent State University
143 PUBLICATIONS **3,815** CITATIONS
SEE PROFILE

Noah: A Dynamic Ridesharing System

Charles Tian, Yan Huang, Zhi Liu
University of North Texas
charlestian@my.unt.edu
huangyan@unt.edu
zhiliu@my.unt.edu

Favyen Bastani
MIT
fbastani@mit.edu

Ruoming Jin
Kent State University
jinruoming@gmail.com

ABSTRACT

This demo presents Noah: a dynamic ridesharing system. Noah supports large scale real-time ridesharing with service guarantee on road networks. Taxis and trip requests are dynamically matched. Different from traditional systems, a taxi can have more than one customer on board given that all waiting time and service time constraints of trips are satisfied. Noah's real-time response relies on three main components: (1) a fast shortest path algorithm with caching on road networks; (2) fast dynamic matching algorithms to schedule ridesharing on the fly; (3) a spatial indexing method for fast retrieving moving taxis. Users will be able to submit requests from a smartphone, choose specific parameters such as number of taxis in the system, service constraints, and matching algorithms, to explore the internal functionalities and implementations of Noah. The system analyzer will show the system performance including average waiting time, average detour percentage, average response time, and average level of sharing. Taxis, routes, and requests will be animated and visualized through Google Maps API. The demo is based on trips of 17,000 Shanghai taxis for one day (May 29, 2009); the dataset contains 432,327 trips. Each trip includes the starting and destination coordinates and the start time. An iPhone application is implemented to allow users to submit a trip request to the Noah system during the demonstration.

Categories and Subject Descriptors

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

General Terms

Design, Management, Performance, Algorithms

Keywords

Dynamic Ridesharing, Mobile Indexing, Kinetic Tree, Road Networks

1. INTRODUCTION

Despite our struggle with energy, pollution, and congestion, many private and public vehicles continue to travel with empty seats. The mean occupancy rate of personal vehicle trips in the United States is only 1.6 persons per vehicle mile [1]. In 1999, if 4% of drivers would rideshare it would have offset the increase in congestion in the 68 urban areas completely that year [2]. Several large cities begin to encourage taxi sharing.

Real-time ridesharing [3, 4], enabled by low cost geo-locating devices, smartphones, wireless networks, and social networks, is a service that dynamically arranges ad-hoc shared rides. In a real-time ridesharing with service guarantees on road networks problem (hereafter referred to simply as ridesharing), a set of servers travel over a road network, cruising when not committed to any service and delivering passengers otherwise. Requests for rides are received over time, each consisting of two points, a *source* and a *destination*. Each request also specifies two constraints, a *waiting time* defining the latest time to be picked up and a *service constraint* defining the acceptable extra detour time from the shortest possible trip duration. When a new request is received, it is evaluated immediately by all servers. In order to be assigned the request, a server must satisfy all constraints, both those of the new request and those of requests already assigned to the server. *The goal is to schedule requests in real-time and minimize the servers' traveling times to complete all of the committed service while meeting service quality guarantee.*

The traditional dial-a-ride problem [5] aims at designing vehicle routes and schedules for small to middle sized trip and vehicle sets, e.g. a few vehicles serving tens of requests. Large scale private car sharing and real-time on-demand taxi or cab sharing are becoming increasingly popular. Increasing numbers of users use mobile devices or the Internet to request and participate in these ride-sharing services. Tickengo [3], founded in 2011, is an open ride system where over 50,000 people participate in ridesharing. Other companies include Avego, PickupPal, Zimride, and Zebigo. In an urban city like Shanghai, there are approximately 120,000 road intersections, 40,000 taxis, and more than 400,000 taxi trips per day (these numbers are derived from our experimental dataset). Slight change of weather such as light rain will send the city into a gridlock. With the mounting energy, pollution, and congestion problems in the urban and metropolitan areas that are growing at tremendous rates and already host more than half of the entire human population, trading a small amount of privacy and convenience for energy and cost savings using ridesharing is promising and maybe inevitable.

However, providing ridesharing service at the urban scale is a non-trivial problem. The core is to devise a real-time matching algorithm that can quickly determine the best taxi to satisfy incoming service requests. Traditional solutions to the related dial-a-ride problems using branch-and-bound or mixed integer programing approaches are not designed to deal with these enormous modern situations. Furthermore, most previous solutions focus on scenarios

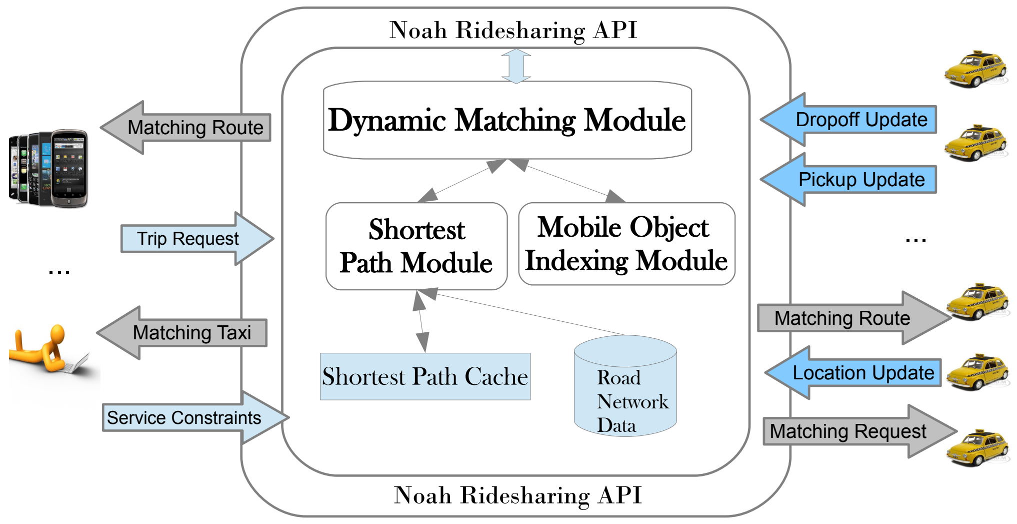


Figure 1: Architecture of the Noah System

where requests are known ahead of time and servers originate and finish at known depots. The dynamic and en route nature renders many of these algorithms either inapplicable or inefficient.

In this demo, we show Noah performing large scale real-time ridesharing. Noah implements several dynamic ridesharing and shortest path matching algorithms. Different from offline or schedule-ahead system, Noah accepts requests in real-time and assigns the request to a taxi which is either roaming or having a pre-engaged route. The assignment guarantees the services constraints of the request in terms of waiting time and detour percentage.

2. NOAH ARCHITECTURE

Figure 1 depicts the architecture of the Noah system. The Noah ridesharing core components include a Dynamic Matching Module, a Shortest Path Module, and a Mobile Indexing Module. A user requests a trip using her smartphone. The request specifies her origin and destination locations. A system user can also modify the service constraints that include the maximal time a user will wait before being picked up and the maximal detour that she can tolerate based on the shortest path without ridesharing. Once the request is submitted, the dynamic matching module will try to find the set of taxis that can serve the request and suggest the taxi that provides the shortest route to the user along with the associated route. The selected taxi will also be notified with the request and the new route incorporating the new trip. Taxis update their locations periodically. They also update the Noah system when they pick-up or drop-off a customer. Taxis' locations are indexed by the Mobile Indexing Module which helps the Dynamic Matching Module in filtering the taxis that are clearly not matching the incoming request. The Dynamic Matching Module implements various matching algorithms that are scalable when there are a large number of taxis and requests. A ridesharing matching algorithm typically involves many pairs of shortest path queries on a road network. The shortest path algorithm on a road network can be quickly become the system bottleneck if not designed and implemented efficiently. The Shortest Path Module implemented in Noah uses a caching scheme to avoid repeated calculation of the same pairs of shortest paths.

2.1 Dynamic Matching Module

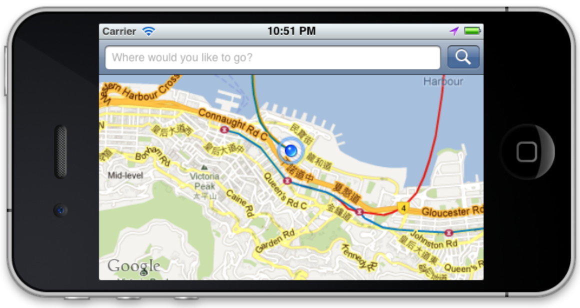
The main challenge in ridesharing is to determine how to handle trip requests as they flow into the system in real-time. From a server's point of view, for any new request, each server may have already selected (and be executing) a trip schedule for its existing customers. Given this, how can we quickly determine whether it can accommodate a new request? Note that in order to respond to such a request, one may have to reshuffle the predefined schedule and the reshuffled one has to be a valid schedule. Furthermore, there might be tens to even hundreds of servers in the region surrounding the pickup point of a new request.

Noah implements several dynamic matching algorithms including a branch-and-bound algorithm, a mixed integer programing algorithm, and a kinetic tree algorithm. The demo will allow users to select among those algorithms and observe the system performance.

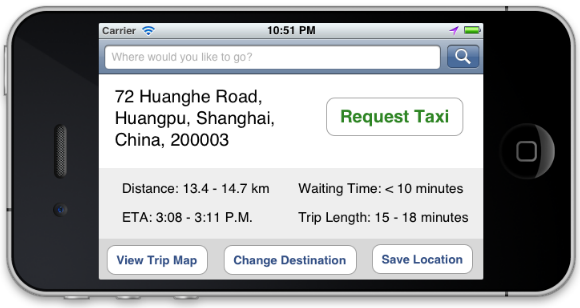
The branch-and-bound algorithm systematically enumerates all candidate schedules and organizes the candidates into a schedule tree. It estimates and maintains a lower bound of each partially constructed schedule and stops building candidate schedules that have lower bounds greater than the best solution found so far. The algorithm first expands the partial candidate with the lowest lower bound (best first search). The key to a branch-and-bound algorithm is to find an effective lower bound. The bound that Noah uses is the sum of the minimum-cost-edge incident to the nodes that are not yet scheduled.

Mixed integer programing is a popular alternative. Noah divides the location indexes of pickup and dropoff locations N into subsets: ((1) dropoff locations of those already picked up but not dropped off; let the size of this set be k; (2) pickup locations of trips not started yet; let the size of this set be n; and (3) dropoff locations of trips not started yet; the size of this set is also n. The problem can be defined on a complete directed graph $G = (N, A)$ where $N = D' \cup P \cup D \cup \{0\}$, $D' = \{1, 2, \ldots, k\}$, $P = \{k+1, k+2, \ldots, k+n\}$, $D = \{k+n+1, k+n+2, \ldots, k+2n\}$. Here Noah assigns an integer to each point in N while node 0 represents the current position of the server. For a pickup i in P, its matching dropoff in D is $i+n$. Constraints are coded such that schedule structure is enforced. That is each node is visited exactly once and the schedule starts from node. The waiting and service requirement are also programmed as constraints. Traditional solvers are used by Noah to find the solution.

The two approaches above both reschedule the unfinished pick-



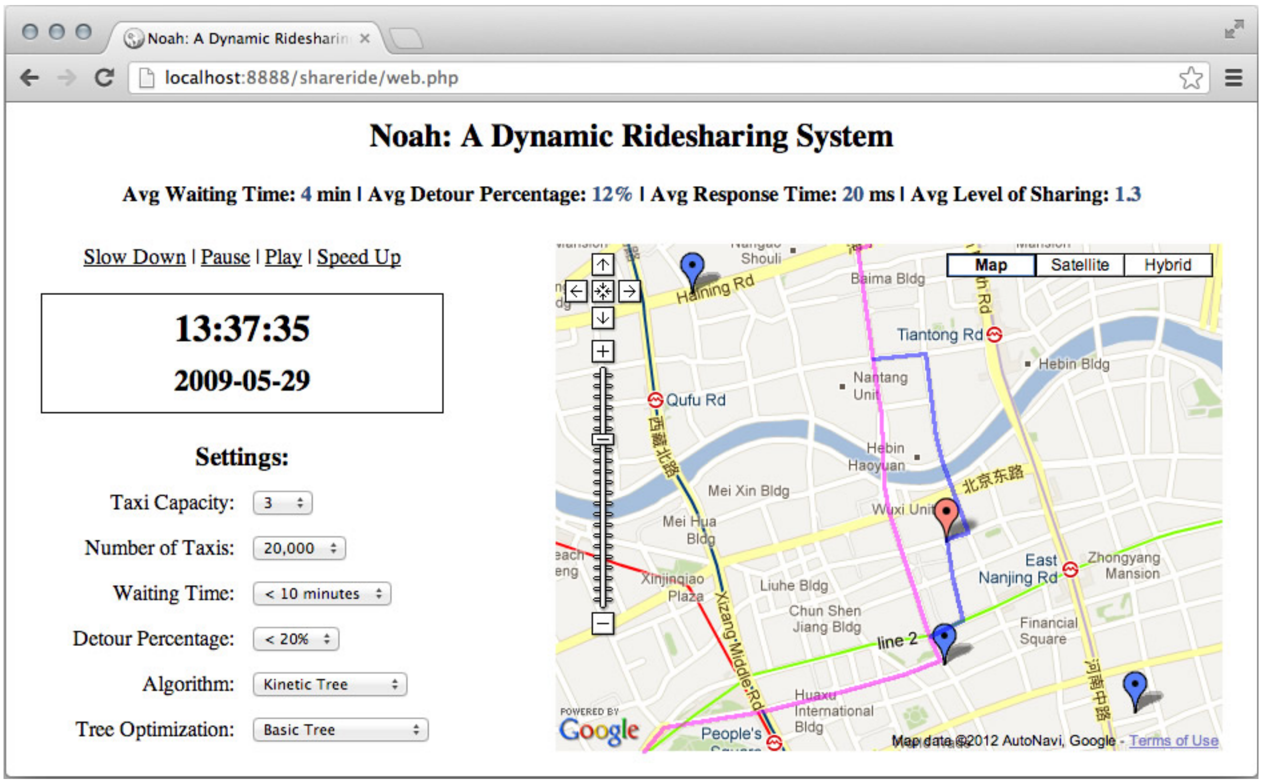
(a) Noah Client Interface 1



72 Huanghe Road, Huangpu, Shanghai, China, 200003

Distance: 13.4 - 14.7 km | Waiting Time: < 10 minutes
ETA: 3:08 - 3:11 P.M. | Trip Length: 15 - 18 minutes

(b) Noah Client Interface 2



Noah: A Dynamic Ridesharing System

Avg Waiting Time: 4 min | Avg Detour Percentage: 12% | Avg Response Time: 20 ms | Avg Level of Sharing: 1.3

13:37:35
2009-05-29

Settings:

Taxi Capacity: 3
Number of Taxis: 20,000
Waiting Time: < 10 minutes
Detour Percentage: < 20%
Algorithm: Kinetic Tree
Tree Optimization: Basic Tree

(c) Noah Web-based Visualization and System Analyzer

Figure 2: Noah Interfaces

ups and dropoffs with the new request from scratch without reusing the computations performed before. The structure of the two algorithms make it difficult to adapt to the dynamic nature of the problem. Noah implements a kinetic tree structure that can maintain and update the calculation performed up-to-now and use these calculations effectively when a new request is issued.

The main problem with the basic tree algorithm is the exponential explosion of the size of the tree when there are multiple pickup or dropoff locations close to each other. For example, if we have 8 pickups occur in spatial proximity around similar time. e.g airport terminals, any permutation of the pickups may result in a valid schedule. So there are 8!= 40,320 possibilities already without considering the dropoff points. Noah implements an approximation approach with bound to reduce the search space.

2.2 Shortest Path Module

Computing shortest path on road networks has been widely studied (see [6] for an extensive review). A variety of techniques [6], such as A^*, Arc-flag (directing the search towards the goal), highway hierarchies (building shortcuts to reduce search space, transit node routing (using a small set of vertices to relay the shortest path computation), and utilizing spatial data structures to aggressively compress the distance matrix, have been developed. Recently, Abraham *et al.* [7] discovered that several of the fastest distance computation algorithms need the underlying graphs to have small *highway dimension*. Furthermore, they demonstrate the method with the best time bounds is actually a labeling algorithm [7].

Noah implements a fast and practical algorithm to heuristically construct the distance labeling on large road networks, where each vertex records a set of intermediate vertices (and their distance to them) for the shortest path computation [8].

However for large scale ridesharing, the shortest path algorithm is called very frequently. We observe the repeated calling follows a pattern that preserves locality. Noah implements a simple Least Rare Used caching scheme that substantially reduce the response time. Demo attendees will be able to explore the difference of the performance with and without caching.

2.3 Mobile Object Indexing Module

Many moving object indexing methods have been proposed that includes RUM-tree, TRP-tree, Bx-tree, Bdual-tree, and STRIPES. Indexing can substantially decrease the searching of the candidate taxis. However, a trade off needs to be made between maintaining a complex and search-efficient index and relying on a search-approximate but easy to maintain index. In our dataset, around 1,7000 taxis update their locations every 20 to 60 seconds. Noah opts to use a simple grid-based spatial index. The index is updated when a vehicle moves across boundaries of the index bounding box. For each request, it identifies the vehicles possibly within w of the request, asks the vehicle's actual location, and then tests if these vehicles can accommodate the request.

3. DEMONSTRATION SCENARIO

Noah will simulate ridesharing of 1,000 - 20,000 taxis serving 432,327 real trips in Shanghai one day (May 29, 2009). Each trip includes the starting and destination coordinates and the start time. In addition, Noah allows users to interact with the system by requesting taxis in real time from their current location. The demo attendee will be able to choose from a list of preset locations, and request a taxi from that location. (Because our taxi request and map data is for the city of Shanghai, we cannot allow users to request taxis from their current location.) From there, the demo attendee will receive real time information regarding the taxi that they have requested, such as the taxi's current location, the planned route, and the estimated time the taxi will arrive. Noah also implements a web interface that visually represents taxis and taxi routes using the Google Maps API. The web interface allows users to set constraints that will be followed by the simulation, and provides users with statistics regarding taxi trip information.

3.1 Smartphone Application

Users will be using their iOS devices to request taxis based on their location. We provide the user with a map view application, in which they can search for locations (addresses, restaurants, etc) and set them as requested destinations. The iOS device communicates with the Noah server, providing the user's pickup location and intended destination. Once the request has been made, it will be placed in the pickup queue, and a taxi allocated to pick up the passenger. The iOS application will also allow the user to view relevant information regarding their taxi.

Figure 2(a) shows the user interface for the smartphone application aforementioned. The application utilizes built-in location services to locate the user's current location. From there, they can search for specific addresses and locations using the text field at top of the screen, and request taxis to take them to set destinations.

Figure 2(b) shows the smartphone application interface the user will be presented with when requesting a taxi to take them to their determined location. The user will be provided with some information about their trip, as well as the functionality to save the destination and view the trip map.

3.2 Web Interface

The web interface gives the demo attendee an general view as to the locations and routes of all taxis within an area. The web interface consists of primarily a map, on which taxi routes and locations are plotted and updated in real time. Clicking on a specific taxi will provide the user with more detailed information regarding that particular taxi, such as its current destination, whether it is occupied or not, the number of passengers, etc. Additionally, the user can also zoom in and out on the map, depending on whether they would like a more general overview of many taxis, or a specific view of a select few.

Figure 2(c) displays the Noah web interface given to the attendees. Users will be able to adjust the simulator constraints by selecting from drop down menus, as well as pause, resume, slow down, and speed up the simulation. Information regarding key statistics such as average waiting time, detour percentage, response time, and level of sharing is given at the top of the page.

The interactive map allows users to observe the location of taxis (blue markers) within a certain frame of view. In Figure 2(c) the route of one of the taxis is highlighted in purple. We see the route adjust and a detour being made when a new request is received by Noah servers and allocated to that specific taxi (the red marker). The detour meets the constraints set by the user, and its path is highlighted in blue.

4. REFERENCES

[1] K. Ghoseiri, A. Haghani, and M. Hamedi, "Real-time rideshare matching problem," *Final Report of UMD-2009-05, U.S. Department of Transportation*, 2011.

[2] J. F. Dillenburg, O. Wolfson, and P. C. Nelson, "The intelligent travel assistant," in *The IEEE 5th International Conference on Intelligent Transportation Systems*, 2002, pp. 691–696.

[3] TICKENGO, "Tickengo," http://tickengo.com.

[4] G. Gidofalvi, T. B. Pedersen, T. Risch, and E. Zeitler, "Highly scalable trip grouping for large-scale collective transportation systems," in *Proceedings of the 11th international conference on Extending database technology: Advances in database technology*, ser. EDBT '08, 2008, pp. 678–689.

[5] M. Lipmann, X. Lu, W. d. Paepe, R. Sitters, and L. Stougie, "On-line dial-a-ride problems under a restricted information model," in *Proceedings of the 10th Annual European Symposium on Algorithms*, 2002, pp. 674–685.

[6] D. Delling, P. Sanders, D. Schultes, and D. Wagner, "Algorithmics of large and complex networks," D. Lerner, J.and Wagner and K. Zweig, Eds., 2009, ch. Engineering Route Planning Algorithms.

[7] I. Abraham, A. Fiat, A. V. Goldberg, and R. F. Werneck, "Highway dimension, shortest paths, and provably efficient algorithms," in *SODA '10*, 2010.

[8] I. Abraham, D. Delling, A. V. Goldberg, and R. F. Werneck, "A hub-based labeling algorithm for shortest paths in road networks," in *Proceedings of the 10th international conference on Experimental algorithms*, 2011.