Senior Project Design Document

5/7/19

Wes Holman, Ant Macchia, Tyler Rambo, Kieran Walsh, Dan Russo, Raymond Pickett

Our group will be developing a ridesharing application, as it pertains to the needs of paratransit service. Users will be able to request a date, time, origin, and destination for scheduled trips, and our application will generate appropriate routes for a fleet of drivers to serve. To reduce net cost, the system should choose routes of minimal distance with efficient vehicle carrying capacities. This problem is equivalent to finding a cover of optimally constrained paths through a directed graph, and therefore brute-force search is combinatorial in nature and unsuitable for real-world inputs.

Paratransit services aid disenfranchised populations and provide vital resources to communities. Existent systems, such as New Jersey's Logisticare, typically require advanced notice for passengers, especially if mobility accommodations are needed. To simplify the problem domain and more closely mirror actual use cases, we have exclusively considered static algorithms, where all details of the set of ride requests are known ahead of time. Due to the wide availability of the internet, a web based interface will be used as it is a great way to increase user accessibility.

The framework of the web server is set up through Spring. The web interface will be used by both passengers and drivers. Passengers use it to make requests, while drivers will use it to view their routes. These requests and routes are stored by the database, as well as their password hashes for user authentication. Requests are stored and manipulated to generate the routes for the drivers by our Java application, which will run on a predetermined schedule.

The ridesharing problem is a widely studied problem, as per web search results for "ridesharing algorithm", and has been classified as NP-Hard (Bei, Xiaohui, and Shengyu Zhang). Most models employ graphs extensively to embed information about distance and time, and then

employ heuristic-based search methods to find valid Hamiltonian paths. In graphs where time is not modelled, the graph forms a completely connected n-clique, and finding a Hamiltonian path is trivial; most computational time is consequently spent searching over the sample space of Hamiltonian paths to find a local or global optimum. Some approaches represent the temporal dimension of ride requests by creating a disjoint series of graphs, where each graph contains a slice of time; linking together all the minimal Hamiltonian paths after computing them produces an entire route for a given driver.

Our first consideration, proposed by Chung-Min Chen, David Shallcross et al, represents the graph as a directed bipartite graph and attempts to find the maximum-weight matching of a specifically-chosen cardinality, through breadth-first search and a greedy merging heuristic. Nodes are connected via a directed edge if and only if the travel time between the two points does not violate a maximum wait time or makes a passenger miss their deadline. The authors propose repeatedly merging destinations and origins if they can be served in order, by greedily traversing the graph and adding these newly created composite tasks back into it. Any path through the graph is a valid solution, as it preserves directionality and time constraints, and by repeatedly reorienting edges and performing set operations after searching, we can find the maximum-weight matching of this representation.

The above approach is not without its drawbacks. Greedy linking of requests strongly favors local optimums, and often discards plausible solutions in the process. Gaps of time and distance are not limited; the algorithm may quickly run out of requests to join, as poor choices preclude better ones. Additionally, the time complexity of repeatedly finding paths while searching for the maximum weight-matching performs poorly on large inputs; each iteration

calculates up to n/2 paths via breadth-first search where n is the total number of nodes. While this algorithm is capable of calculating valid approximations to the graph, it is not sufficiently robust enough to work with all constraints of our problem domain.

Some academics have posited the merits of clustering spatial data before employing pathfinding algorithms(Qiang, Xiao, and Yao Shuang-Shuang). At first glance, this is a naive approach, wasting valuable computational cycles on a metric that may exhibit the same pitfalls as a greedy approach; pairing similar origins to one another does not guarantee their resultant destinations are nearby. By clustering over four dimensions, however, we can ensure to capture distinguishing features of a request's origin and destination, without loss of information. When further limiting a cluster's size to the capacity of a fleet vehicle, we can guarantee both that all passengers have an acceptable wait time and that the vehicle will never become overfilled. This eliminates a constraint posed by many studies, namely the issue of carrying capacity; search-based methodologies are forced to consider it upon every downward traversal of the search tree. Finding paths within generated clusters creates a built-in gravitation towards the global optimum, and is comparatively much more efficient to finding paths within a small cluster than a universal graph structure.

Several established clustering algorithms exist, with different paradigms, for processing Cartesian coordinates, with various amenable qualities. Density analysis and other statistical-based approaches seemed promising initially, but due to the most common distribution of urban and non-urban populations, we ultimately decided against these frameworks. Sparsely populated areas should be served alongside densely populated areas; the self-organizing grouping would create some hyper-efficient paths while leaving others sorely behind.

Hierarchical clustering, such as single-link clustering, was ruled out due to its tendency to form large groups of similar data and isolate present outliers. K-means was ultimately chosen, on account of its convergence, relatively fast computational speed, and the ability to indirectly influence the size of the resultant clusters. Pathfinding within generated clusters will be evaluated by a greedy algorithm or a branch-and-bound local search algorithm, depending on which perform better in regards to computational time and total path distance when we begin the testing phase of the project.

Functionality

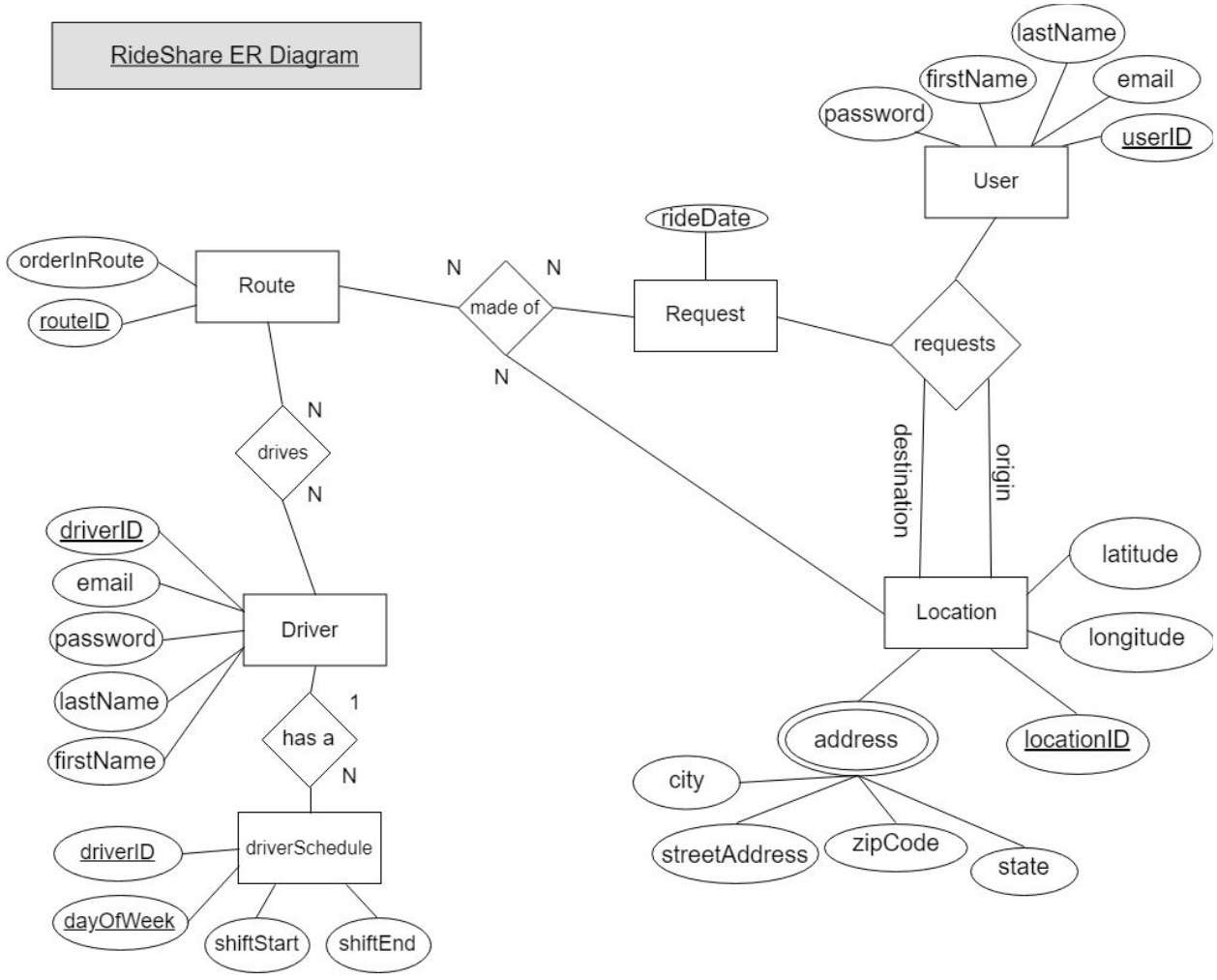
- Sign-Up Page
    - This page allows users to sign up for the service. It will request the necessary information from the user and create an account for them in the database.
- Login Page
    - This page allows users that have already signed up for the service to login into their account to access to the system. The system will access the database to see if the user exists allow them access if they do.
- Driver Page
    - This page will be where drivers are directed after they login to the system. This page will access the database to display current and previous routes assigned to the driver, as well as allow the driver to request another route to drive.
- Passenger Page
    - This page will be where passengers are directed after they login to the system. This page will access the database to display current and previous routes that the passenger belongs to, as well as allow the passenger to schedule another ride.
- Driver Schedule Page
    - This page allows drivers to view and update their current work schedule.

Tech Stack

- Platform
    - Web
- Development Tools
    - Gradle
    - XAMPP

- RESTful APIs
    - Bing Maps API
- Backend
    - AWS EC2 Linux
    - MySql
- Programming Languages
    - Java
    - PHP
- Markup Languages
    - HTML
    - CSS

ER Diagram



RideShare ER Diagram Description

User - The User table contains notable information about the person using the ride share service.

- userID - an automatically assigned auto incremented ID to uniquely identify a row in the user table.
- firstName - first name of the user.
- lastName - last name of the user.
- email - unique email of the user to use as a point of contact.
- password - hashed and salted password for a user account.

Request - The Request table stores information about the request of the user.

- requestID - an automatically assigned auto incremented ID to uniquely identify a row in the ride table.
- userID - foreign key from the user table to track a user's request
- originID - a locationID that determines the where a User will be picked up.
- destinationID - a locationID that determines where a User will be dropped off.
- requestDate - date and time of when the User would like to be picked up.

Location - The Location table stores longitude and latitude as well as the address of a location

- locationID - an automatically assigned auto incremented ID to uniquely identify a row in the location table.
- latitude - coupled with the longitude to uniquely identify a location on a map.
- longitude - coupled with the latitude to uniquely identify a location on a map.
- address - a composite attribute that identifies the standard format of an address.

    - streetAddress - the number and/or street name of an address.
    - city
    - state
    - zipcode

Route - The route table stores information about a route. Each row in the table stores information about the driver, location, and sequential order of the location in the route.

- routeID - an automatically assigned auto incremented ID to uniquely identify a row in the route table.
- requestID - foreign key from the request table to reference request details
- driverID - foreign key from the driver table denoting the driver satisfying the request
- locationID - foreign key from the location table to reference the details about the location of the position in route.
- orderInRoute - a column needed to sequentially order a route

Driver - The driver table stores account information about a driver. Drivers are treated as employees.

- driverID - an automatically assigned auto incremented ID to uniquely identify a row in the driver table.
- firstName - First name of the driver.
- lastName - Last name of the driver.
- email - unique email of the driver to use as a point of contact.
- password - hashed and salted password for a user account

DriverSchedule- Each driver has a schedule to determine when the driver will be working during the week as well as the time frame.

- driverID - A foreign key from the driver table used as the primary key.
- dayOfWeek - an enumerated type with valid values [Sunday - Saturday]
- startTime - Starting time of the driver's shift
- endTime - End time of the driver's shift

Mid-Assessment

By the mid-assessment, the development team will implement an efficient path-finding algorithm, implement a well-scored clustering algorithm, improve communication with the database as well as its overall integrity, develop metrics for evaluating performance, and work with non-homogeneous data.
Kieran Walsh will be fostering data communication and building communication tools with the database.
Ant Macchia will be ensuring good team coding practices, developing performance metrics, and initializing the frontend framework.
Dan Russo will be creating and improving upon a pathfinding algorithm.
Raymond Pickett will be helping merge code bases and implementing the clustering algorithm.
Tyler Rambo will be managing the database, helping with PHP functionality, and fulfilling his duties as scrum master.
Wes Holman will be communicating with the faculty sponsor, implementing the clustering algorithm, uniting disparate code bases, and performing build and release maintenance on the Git repository.

Works Cited

Chung-Min Chen, David Shallcross, Yung-Chien Shih, Yen-Ching Wu, Sheng-Po Kuo, Yuan-Ying Hsi. "Smart Ride Share with Flexible Route Matching," 2018, https://ieeexplore.ieee.org/document/5746090

Bei, Xiaohui, and Shengyu Zhang. "Algorithms for Trip-Vehicle Assignment in Ride-Sharing." Association for the Advancement of Artificial Intelligence, 2018, www.ntu.edu.sg/home/xhbei/papers/ridesharing.pdf.

Qiang, Xiao, and Yao Shuang-Shuang. "Clustering Algorithm for Urban Taxi Carpooling Vehicle Based on Data Field Energy." Advances in Decision Sciences, Hindawi, 9 Aug. 2018, www.hindawi.com/journals/jat/2018/3853012/.

Masoud, Neda, and R. Jayakrishnan. "A Real-Time Algorithm to Solve the Peer-to-Peer Ride-Matching Problem in a Flexible Ridesharing System." Institute of Transportation Studies , July 2015, pdfs.semanticscholar.org/1bc0/584ab57a215598fa6b9445c9965b0a263854.pdf.

Schreieck, Maximilian, et al. "A Matching Algorithm for Dynamic Ridesharing." ScienceDirect, Transportation Research Procedia , 2016, www.excell-mobility.de/wp-content/uploads/2017/01/A-Matching-Algorithm-for-Dynamic-Ridesharing.pdf .

Li, Yiming, et al. "A Dynamic Programming Algorithm for the Ridersharing Problem Restricted with Unique Destination and Zero Detour on Trees." Journal of Applied Mathematics and Physics, 2017, file.scirp.org/pdf/JAMP_2017091316502450.pdf.

Ghoseiri, Keivan. "DYNAMIC RIDESHARE OPTIMIZED MATCHING PROBLEM." Digital Repository at the University of Maryland, Cogitatio, 1 Jan. 2012, drum.lib.umd.edu/handle/1903/13023.

Coltin, Brian, and Manuela Veloso. "Ridesharing with Passenger Transfers." Carnegie Mellon University School of Computer Science, www.cs.cmu.edu/~mmv/papers/14iros-ColtinVeloso.pdf .

Masoud, Neda, and R. Jayakrishnan. "A Decomposition Algorithm to Solve the Multi-Hop Peer-to-Peer Ride-Matching Problem." May 2017, arxiv.org/pdf/1704.06838.pdf.

Xia, Jizhe, et al. “A New Model for a Carpool Matching Service.” PLoS One, 30 June 2015, www.ncbi.nlm.nih.gov/pmc/articles/PMC4488330/.

Mallus, Matteo, et al. “Dynamic Carpooling in Urban Areas: Design and Experimentation with a Multi-Objective Route Matching Algorithm.” Sustainability — Open Access Journal , 10 Feb. 2017, www.mdpi.com/2071-1050/9/2/254/pdf.

Cici, Blerim, et al. SORS: “A Scalable Online Ridesharing System.” 2016, laoutaris.info/wp-content/uploads/2016/10/iwcts_2016.pdf.

Alonso-Mora, Javier, et al. “On-Demand High-Capacity Ride-Sharing via Dynamic Trip-Vehicle Assignment.” PNAS, National Academy of Sciences, 17 Jan. 2017, www.pnas.org/content/114/3/462.