Digital Twin for logistics/transport optimization using agent-based modeling and data analytics

**Introduction**

School bus scheduling in SUSTech is still adjusted manually and often faced with lots of problems. It’s quiet common that in a rainy day, a large quantity of students is waiting at the bus stations, while school buses are still too crowded to carry so many students. Another problem is that among after-school time, buses are resting somewhere and waiting to the start time of bus.

The major problem of school bus is that the time schedule doesn’t fit the dynamically changed requirements of passengers. Neither existing time schedule, or adjusting the schedule to another static version would solve the problem. Our way to solve such kind of question is to dynamically arrange peak lines by using an online algorithm, which adjust the schedule of school bus according to the needs of passengers.

We focus on solving the following two problems:

1. In rush hours, the school bus is too crowded and even some passengers cannot get into bus. However, as we have 7 buses in total, only 4 or 5 buses are in use among these hours. The first goal is to solve heavy traffic in course breaks, rainy days, and evening.
2. In early morning, especially weekends, bus is empty at most of time. If there isn’t passenger at all, buses shouldn’t be sent to a complete empty line.

Besides, a digital-twin [Digital Twin paper] based simulation platform would be used to preliminary the time arrangement. Send bus first, and then wait for result doesn’t realistic. So, a simulator to the physical service is required.

Let’s take a brief view of digital twin, and declare why we should use digital twin to do simulation. Digital twin is a concept of modeling and simulation, using a simulator to evaluate current performance of algorithms and do prediction to the future. In the current usage of digital twin, most cases this idea is used in industry, aviation, and Internet of Things. By using digital twin to do simulation, we can easily measure the quality of the algorithm, decide which algorithm to use for specific problem, and predict the performance of such algorithm.

Obviously, this can save a lot of time and money, and simplify the complex problem. But still, this approach has a lot of difficulties. The first of all is that the idea of digital twin hasn’t too much implementation, and we have very few existing materials to lookup, so it’s hard to implement a such platform includes data, algorithm, and simulator in one system.

However, our situation is rather simpler and the implementation shouldn’t be too complicated. Let’s take a brief glance at the current achievements.

**Problem Definition**

Ridesharing Problem (RSPs). [Ridesharing benchmark paper] Ridesharing is a kind of transportation mode in order to make the travelling easier. The basic idea is that any vehicles may be willing to pick up some customers and send them to their destinations if the driver's route is similar to the customers'. The total traffic may be reduced and fuels are saved if several people share a ride. We found that this framework can be used in many kinds of transportation problems, such as food delivery, express service and of course the traffic problem.

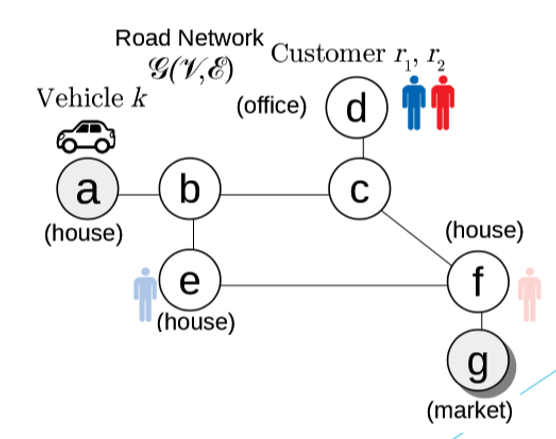


Figure 1

To model this scenario, we consider the road network as a graph, the vertices are some specified places while the edges are the direct path connecting two adjacent places. For simplicity, we consider that the customers and vehicles are all on the vertices. If the position of a customer or vehicle is on an edge in the real world, we can still use some techniques to guarantee the accuracy of the result [G-Tree paper].

The goal is to find the minimum cost set of vehicle routes that can serve all customers within some constraints. These constraints are time window, capacity and precedence. Time window is a pair of variables, [e, l], e is the earliest departure time and l is the latest arrival time. Both vehicles and customers have a time window. The specified vehicle cannot depart from the origin before e and the specified customer cannot be picked before e. The specified vehicle cannot arrive at its destination after l and the specified customer cannot accept to arrive at the destination after l. Capacity is the maximum number of customers on the vehicle at the same time. And finally, precedence is that the vehicle needs to pick up the customer at first and then it can send the customer to the customer’s destination.

The route of a vehicle can be represented by a schedule S [Ridesharing benchmark paper], an ordered sequence, representing the route on which all the customers can be picked up and sent to their destination, consists of all the starting positions and the destinations of all customers. And its order represents the traveling order. For example, in the schedule

The vehicle will go to vertex s0 and pick up the customer 0 at first, and then it will pick up customer 1 at vertex s1. After picking these two customers, the vehicle will drop off customer 1 at e1 and then drop off customer 0 at e0.

The cost of schedule S based on the shortest route for vehicle k is [Ridesharing benchmark paper]:

where k0 is the position from where the vehicle depart, kd is the destination of the vehicle and cu,v is the shortest path cost from node u to node v.

**Modeling**

As mentioned before, RSP is a general transportation model. Since the bus system in our campus is quite simple, it is easy to use RSP to modeling the traffic in SUSTech.

To construct a road network and make a lot of shortest path queries on it, we can use G-Tree to represent the road network [G-Tree paper], which provides a fast way to query the shortest path.

The bus stops should be the vertices in the graph. Vertices should also include some crossroads. Some vertices should also be added to balance the distance between two adjacent vertices in the graph.

**Problem solving**

Algorithms for online RSP depend on *candidates filtering* and *customer insertion* [Ridesharing benchmark paper].

The basic idea of candidates filtering is using constraints to generate a list of valid candidate vehicles for a single customer to reduce the computation cost. The filter conditions are based on the constraints.

Customer insertion is a basic step to find the optimal schedule of a vehicle. For a vehicle k, its original schedule S can form a minimum-cost schedule, which is also specified as the augmented schedule S+, by inserting the origin and the destination of a new arriving customer r:

where P(S∪{r0, rd}) lists all permutations of the new schedule that obey the precedence constraints. This is specified as the exhaustive insertion. We can also use a simple insertion with less computation cost by maintain the order of the element is original schedule S and then insert r0 and rd into the schedule.

There are nine algorithms introduced in [Ridesharing benchmark paper]. We will choose the most suitable algorithms according to our problems. For now, we are going to use Kinetic Tree introduced in [Kinetic Tree Paper].

A simulator is also provided in [Ridesharing benchmark paper]. We will check the implementation and use it in our work.

**Simulator Architecture**

The simulator part isn’t so difficult to understand, we raised a demo architecture graph first, and take some deep look into it.

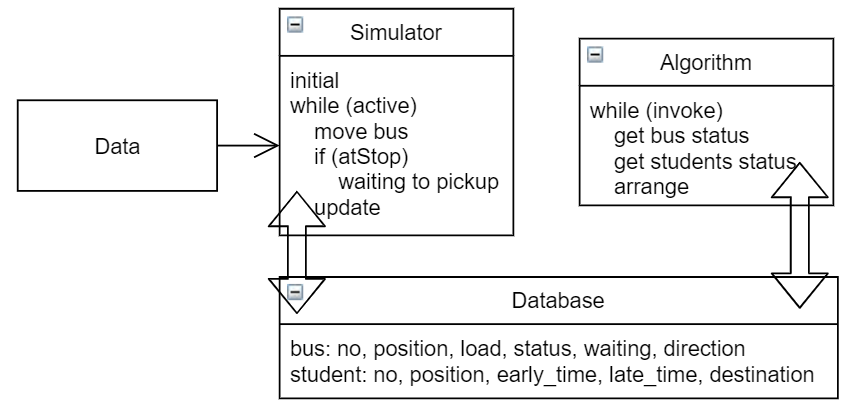


Figure 2

The whole architecture of the innovation project is shown in Figure 2, and clearly the simulator part communicates with algorithm through a local database, which stores the current information of buses and students, as we defined previously.

Simulator takes in data from local disk or remote file system, and then store them into local storage, so the simulation can be run without network. However, only part of the data can come into the database, that is the “current” status which can be accessed by the algorithm. Remember this architecture requires an online algorithm to do the work, so only part of data is public to the algorithm.

A synchronization between simulator and algorithm is also very important, we won’t rely everything on the database lock, but consider an access latency, which may lead to hierarchy problem between database and algorithm. For example, a bus is ready to start, its status becomes 1 means the bus was already left the station. However, a student come to the station at the same time, and coincidence read from database just before simulator write into new status, and the algorithm got 0. In this case, we assume the student can be into the bus, as the driver may stop to leave and wait the student to get into the bus in reality. This is an example to solve the hierarchy problem between threads.

The output of simulator is given by 4 indicators, that is the total time cost from taking bus, the number of bus sent by the algorithm, the longest cost time for a student to reach one’s destination, and percentage of students that arrive their destination later than their time window’s late time. All the indicators should be optimized to the smaller value.

**Data Collection**

The data needed to build up our school bus simulation system is divided into two subparts: Basic data and Extended data.

For basic data, there are some necessities to form the graph specified in the Ridesharing system model, including bus stops information (count, positions, names), bus lines information (count, number of peak Lines, property, stops, time schedule…) and standard information of school buses (count, capacity, departure and arrival time of each bus stop with the given schedule). The data specified above can all be retrieved by observations or talking to the managers responsible.

There are also necessary yet hard-to-collect data for supporting the simulator. For example, to help with finding the shortest paths, we have to retrieve the Euclidean distances between bus stops. To do this, we consider measuring the distance with an RS map or the official map SUSTech provides. We also need the distances of the road segments between bus stops. Measuring it with an RS map or by computing from the bus travelling speed and time may both be feasible solutions. The next thing specified is a hard task. We need to somehow get the data about the latest time a passenger can wait until at the waiting stop, and the earliest time he/she wishes to reach the destination at. Unfortunately, this kind of data is easy related to some privacy issues and we might not be able to get a collection of them from the Teaching Affairs Office. One choice to get the data is to spread out questionnaires to passengers. In addition, we can retrieve the official class schedules and research schedules from “Teaching Affairs Office” to help estimate the traffic flow of bus stops near teaching and research buildings. We can also figure out the time data roughly by counting the number of passengers getting on the bus at the bus stop. With these alternatives, we might be able to estimate a general situation of the school bus system environment, and with the data above, retrieving bus loads and passenger requests will also not be a problem.

In the future, we will improve the functionalities of our system and apply online algorithms to it, so more mapping data and the benchmark of action latency will be needed. We also consider to extend our school bus system to a city range, which will definitely need more data about the traffic flow status and the region features in the city.

**Summary**

The innovation project of “digital twin based school bus time schedule arrangement” would be divide into 3 parts: data collection for benchmark, algorithm to arrange time schedule dynamically, and the digital twin based simulator. Another part is the design of database, but this part needs the information from all 3 parts before, so we can only choose to do the database design together.

Task distribution is like this: data collection by Jingran Shen, algorithm design by Weihao Wang, simulator design by Zhao Li. We would share our data, code, and API document through github, and arrange regular meeting to communicate about process in every weekend. The task distribution is highly uncoupled, so everyone can concentrate on one’s work, and the merge step can be done very easy.

Time arrangement for us is basically as follow. Data collection is the first step, but we still can use randomly generated data for demo version to test algorithm and simulation. Time for collecting data would not longer than the half of our development period, which is 5 weeks. In the same time, the design of algorithm and simulator can be done synchronously, as we explained our works are highly uncoupled.

As the whole architecture is basically decide, we can write precise algorithm and simulator according to the current architecture, the further work is rather clear. For the next step, we may step out of the campus, and do the simulation on traffic condition of the whole Shenzhen city, but it’s more than an innovation project, so let’s finish our digital twin based bus time schedule arrangement first!

**References**

[1] Saddik, A. El (April 2018). "Digital Twins: The Convergence of Multimedia Technologies". *IEEE MultiMedia*. **25** (2): 87–92. [Digital Twin paper]

[2] J. Pan, G. Li and J. Hu, "Ridesharing", *Proceedings of the VLDB Endowment*, vol. 12, no. 10, pp. 1085-1098, 2019. Available: 10.14778/3339490.3339493. [Ridesharing benchmark paper]

[3] Y. Huang, F. Bastani, R. Jin and X. Wang, "Large scale real-time ridesharing with service guarantee on road networks", *Proceedings of the VLDB Endowment*, vol. 7, no. 14, pp. 2017-2028, 2014. Available: 10.14778/2733085.2733106. [Kinetic Tree Paper]

[4] R. Zhong, G. Li, K. Tan, L. Zhou and Z. Gong, "G-Tree: An Efficient and Scalable Index for Spatial Search on Road Networks", *IEEE Transactions on Knowledge and Data Engineering*, vol. 27, no. 8, pp. 2175-2189, 2015. Available: 10.1109/tkde.2015.2399306. [G-Tree paper]