Digital Twin for Logistics / Transport Optimization Using Agent-Based Modeling and Data Analytics

**Introduction**

The objective of this project is to simulate the bus system in SUSTech by using a simulator introduced in [2] to solve the bus scheduling problem in our campus. A simulator is already built on our lab server and we also wrote a handbook about how to use and modify it to meet our demands. Also, necessary data are already collected from Teaching Affair Office of SUSTech.

The goal of our project is to solve the time arrangement of school bus, while the path of buses is constant. So, our main concentration would put on time arrangement and schedule. There are several definitions should be clarified. Therefore, the report would be in several parts:

Problem definition of RSP and BSP (Bus Sharing Problem), in which gives a definition on both problems and their common and in-commons. Modeling gives some useful graphic knowledge to transfer real world traffic route to graphs with vertices and edges. Then we state the algorithm part of Cargo in particular. Problem solving part gives several useful ways to solve this kind of problems, and some construction details are introduced. Then the simulator architecture part digs deep into our model to simulate and optimization part, gives a top-down view of digital-twin work. The data collection describes how we get the data used to do simulation, and make sure the simulation can measure reality.

Finally, in the summary part, we give a conclusion to the work till now, and also some future plans. The structure of this report is as follow.

**Recap**

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

In this project, we will use a digital-twin based simulation platform to simulate the bus system in our campus.

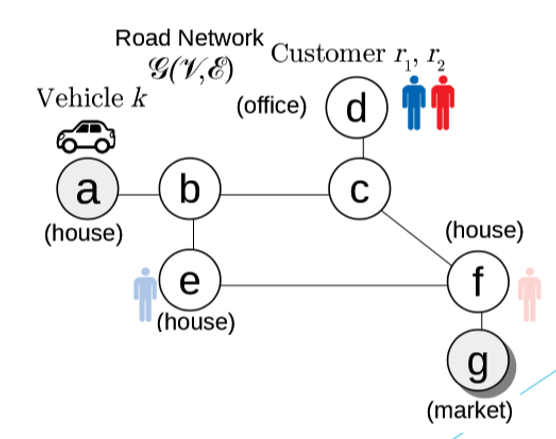


Figure 1

The road network is considered as a graph, and several existing greedy, heuristics and also metaheuristics algorithm will solve a RSP with specific objective function.

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.

The route of a vehicle can be represented by a schedule S [2], 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:

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 [2].

Algorithms for online RSP depend on *candidates filtering* and *customer insertion* [2].

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.

**New issue**

The problem we want to solve still have a slight difference from RSP. Since it is related to a bus system, the problem is a bus scheduling problem (BSP) SP is also an online problem requires online algorithm. Because nobody can know the exact time arrangement today.

The difference between BSP and RSP is that, the route of BSP is constant, which is set in the beginning. While the route of RSP should be calculated dynamically. This difference leads to the concentration difference. The RSP focus on arranging routes, while BSP focus on arranging time schedule.

Take our school bus as example. There are several different bus lines with different constant routes, and each line has an opposite one with exactly opposite direction. And the total number of bus is limited, the capacity of bus is limited, which means the arrangement is needed.

This causes 3 main ideas to arrange time schedule. At some time, should we send a bus to run? Would the bus be normal line or rush line? Which route should this bus be put on? These 3 questions are main questions should be solved by the algorithm.

Assume one bus has capacity , indicates this bus can carry at most students, and each station has some students need to aboard or get off, so the number of students on bus is dynamically changed. Each student has a time window, the early time means the earliest time one can arrive bus station, and the late time means the student should be in destination before that.

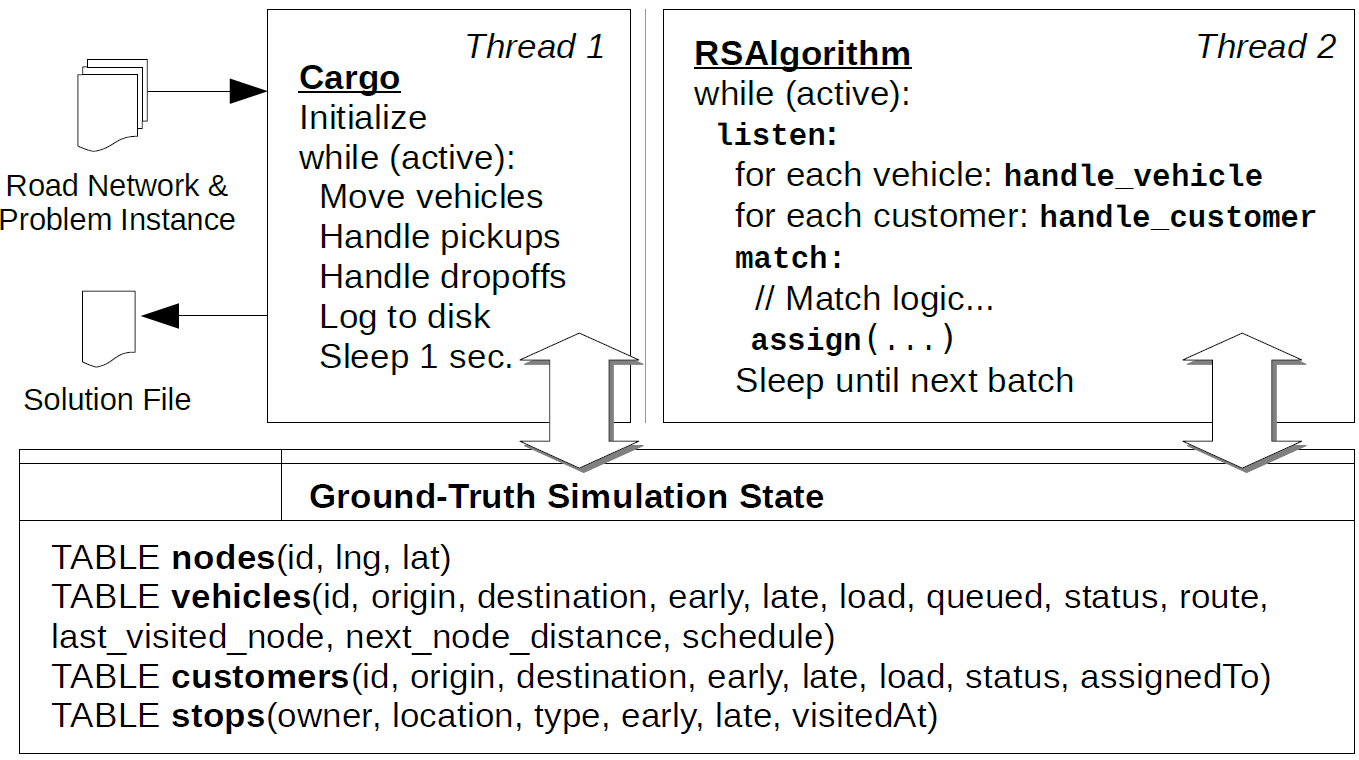
This is very similar to the RSP, we would talk about their common and in-commons more in the problem solving part.

**Cargo – Algorithm Thread**

## Motivation

The intention of Cargo is measuring the performance and quality of different RSP solving algorithms. This goal requires low coupling between simulator itself and the algorithm written by users. In this case, user can just use some predefine functions to access the problem instance and graph without knowing the implementation details of the simulator. Unfortunately, this system is quite new and authors haven’t provided any guideline or instruction yet. Thus, to implement our own algorithm on Cargo, we need to dig out the source code of Cargo, and write a handbook about how to use it.

## In-memory database



The structure of the simulator (VLDB, 2019)

Thread 1 and Thread 2 communicate with each other with an in-memory database constructed by sqlite3. At the beginning of the simulation, information of road network and problem instance are loaded into the in-memory database. Table nodes stores locations in the road network. Table vehicles and customer are initialized by problem instance and keep updating the information while running the algorithm. Table stops is just used to store information for statistics.

## Data Structure

A G-Tree, introduced in [1], is used to accelerate the shortest path query. And grid, introduced in [2], is used to accelerate vehicle query by its location. Before doing the simulation, user should generate g-tree file with a predefine g-tree builder in Cargo and road network instance file. The main idea of g-tree is storing some frequent used shortest path query answers, and use assembly-based method to construct other shortest path by these pre-stored shortest paths in a fast way.

## Algorithm Framework

User only need to implement two virtual functions in C++. The listen function will get active vehicles and waiting customers. User can just call the pre-defined function. handle\_vehicle and handle\_customer need to be implemented by the users. These two functions are the main part of the algorithm. And finally, assign function will update the state automatically. Some helper functions defined in function.cc are also useful for developing new algorithm. All the instructions are written into Cargo handbook.

**Problem Solving**

The optimization goal isn’t measured by only one factor. We should optimize according to different goals. First of all, the total time taken by the bus.

Assume the bus runs according to the route, stop at some station and wait, pick up students and drop off some. The total time in this station would be , this constant can be set by algorithm. Then remember that there still time during path, this time denoted as , which determined by graph itself, and cannot be adjusted.

So, how about the total time? Assume the number of students on board is at time i, and the number of stations is c, so the total time is . Remember the prefix means the time of drives should also be taken into consideration.

However, beside this one, there are several other optimization goals. Such as the lower late rate, the lower number of buses used, and so on.

**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.

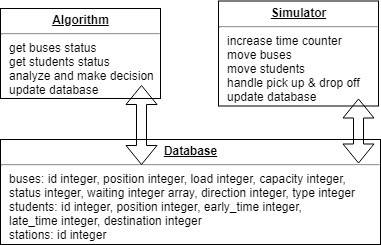


Figure 2

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**

As stated in the last report, the data needed to build up our school bus simulation system is divided into two subparts: Basic data and Extended data. The basic data includes everything we need to build our system and run the algorithm. The extended data describes ideas that could be feasible in a city bus system range. In the second stage, we mainly focused on retrieving the first type of data. Additionally, we produce random data and pseudo-simulation data to test our system environment and estimate the result.

According to the API, our school currently has no more than 30 buses, each with a capacity of 31 people. As for basic data, we have already collected all the bus lines, bus stops and the bus schedules. These data have been fully transformed into formatted input files for our system. The three input files include a `.edges` file - which specifies the number of nodes and edges in the network as well as a list of linking edges, a `.rnet` file – which records additional information (longitude and latitude) of nodes, and an `.instance` file – which specifies the number of school buses, passenger requests in total and the terminal, time windows for each bus or passenger.

In the current stage, the network contains the normal line, peak line and an iPark line of the school bus system. There are generally 14 bus stops inside the campus and together they form a 19-edge connection network. The edge weight of the network is temporarily represented as the travel time along this edge. Values on two directions have been evaluated and an average is assigned as the weight of the edge. All the precise locations – the longitudes and latitudes of the bus stops have been retrieved from Google Map marking system. Using an old API with access to the school bus database, all the information above can also be found. There is also a real-time location interface which we can make use of.

For the time window estimation algorithm, we currently use the following logic. we assume all students are initially distributed uniformly at four dormitory areas and eventually they will all get back to their dormitories. Divided by class periods, there are seven time-windows to select. Every time, there are 30% of the requests from classrooms to classrooms, 30% from classrooms to dorms, 30% from dorms to classrooms, 10% random travel. With this logic, the basic result can be generated as a nice estimation.

In the next stage, we will try to retrieve the latest bus location data to help with edge weight estimation. We have also successfully retrieved the course schedule for this semester from Teaching Office Affair lately. The course schedule will further help us simulate the time windows of the passengers. The data we might still want to make use of is the number of waiting people and waiting time. This requires us to apply for the latest bus data from the bus manager directly.

**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 no 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 we discovered some differences between RSP and BSP. We may take more time than plan to redesign the algorithm for BSP. But this isn’t a large work still. So, the future plan is to redesign RSP model to satisfy the BSP model.

**References**

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

[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.

[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.

[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.

[5] E. J. Hastings, J. Mesit, R. K. Guha: Optimization of Large-Scale, Real-Time Simulations by Spatial Hashing