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.

**Problem Definition**

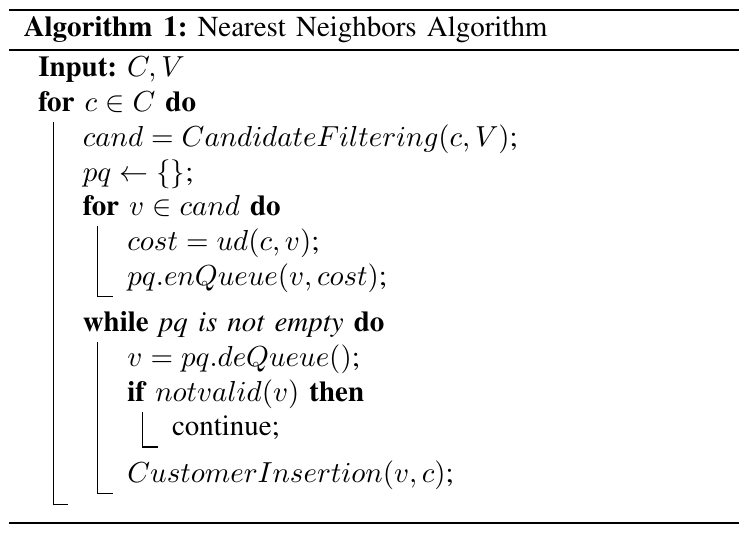
The goal is to minimize the total cost of all vehicles. The cost of a vehicle is the traveled route of a vehicle. If a customer cannot be picked up by a vehicle, then this customer will drive his/her own vehicle. In this case, the total cost of all vehicle may be larger. The formal definition of this problem is:



, where V is the set of all vehicles.

**Algorithm**

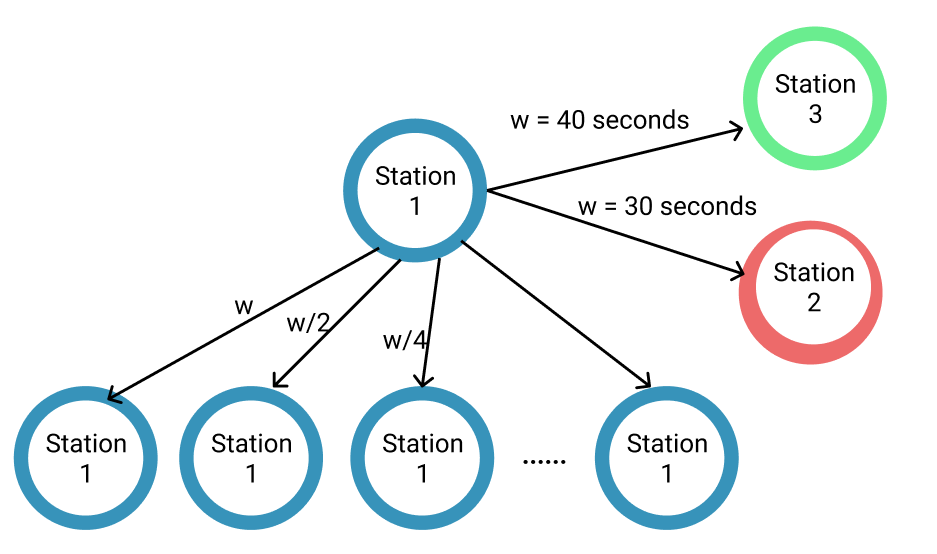
We use the Euclidean distance of a customer and a candidate vehicle as a heuristics function to find the solution. This base idea is to select a vehicle from all vehicles and assign it to a new request from a customer. First, we use candidate filtering to get a list of candidate vehicles. And then. We calculate the Euclidean distance from these candidate vehicles to the customer sending new request. Use the result as the priority of this vehicle and put them to a priority queue. After that, we can find the valid vehicle with smallest Euclidean distance to the customer and assign this vehicle to this customer using customer insertion. The pseudo-code is illustrated in the follow picture.



**Make the Bus Halt**

In the previous version, the bus cannot halt but keep moving. We found that if the bus need to pickup a customer in a bus station after it just left the station for a while, it needs to arrive the next station at first, then it can come back to the previous station to pick up that customer. It is a waste of time to do that. Actually, if the time cost between these two stations is w. this event just like the bus halt at the first station for 2w. Thus, if the bus needs to halt at the first station, the halt time should be the factor of 2w.

To make the bus be able to halt for any time it wants (this is decided by the algorithms), we came up with a tricky way which is adding a path starting from a station and ending to the same station, which is actually a ring. The time cost of this path can vary, so that the bus can halt for any time it wants. However, dynamically changing edge is not allowed in the simulator. To solve this problem, we added multiple ring with different time cost. Then the question becomes how many paths do we need to achieve this goal. The answer is log2w. First, we need to find the minimum weight w from all neighbor edges of a vertices. Then we add rings with weight w/20, w/21, w/22…, 1. In this case, the bus can halt for any time it wants by adding different rings to its 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) 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.

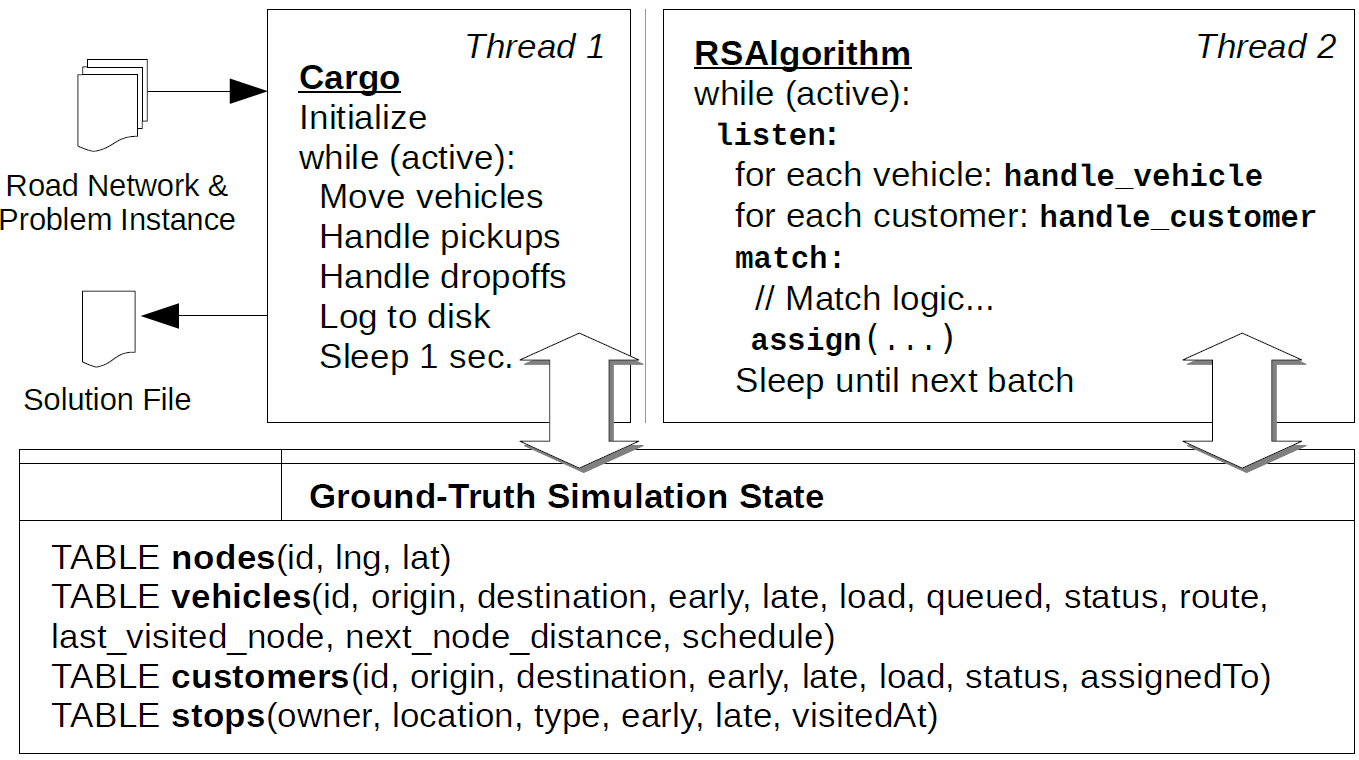
This is very similar to the RSP, we would talk about their common and in-commons more in the problem-solving part.

**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.

**Why we Use Simulator?**

This question is raised these days. That why we not just design an algorithm to arrange bus time schedule, but use a simulator to do those not easy part? This part would answer to this question.

First of all, the bus line timetable setting problem is not just one-shot problem. Which means the student behaviors may change, the school bus number would change, and even the bus line would change. Then we need something to measure the performance of algorithm. If simply run the algorithm, this would be a waste of time and money.

Secondly, the simulator can be open be expand. Take our future plan as example, if we want to make this bus time schedule dynamically generate, instead of static. How should we do with a single algorithm? Even online algorithm, it cannot know the result of the command, so, a simulator would solve this question.

Thirdly, if we step another epoch from the second one, if we want to combine human control and algorithm control together, and we need to know how the interrupt of human control would affect the result. This is no other better idea but a simulator.

The last one, this simulator can be used to train a heuristic algorithm to adjust its parameters. Use realistic environment is slow and expensive, but with simulator, the tuning can be done in a much lower cost.

**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.

As for the problem raised before, that the problem definition isn’t pure RSP, but more likely to be BLTSP. The big problem is that we cannot generate a static time schedule according to the data result we got, so, we must find some solutions to make the route static. We take some considerations.

The first, and the simplest, is to use direct-edged graph. The solution is simple: replace the undirected graph with directed graph. So, we tried to replace the edge weight of reverse direction to be a large enough number, which is 99999 in my case. However, the running result isn’t good. All the buses won’t leave the initial point, and this is caused by the limitation of Cargo system: the directed graph would lead to large backward cost, so the average cost would be smaller that with penalty.

The second problem is that the waiting time is hard to realize. If we don’t modify the source code of Cargo, then the waiting time is not so easy to measure. But we find some tricky solution. If we add self-circle to nodes, the waiting time is then very simple to measure.

Third and the last problem, is to use current result to generate solution, which is a static bus route. To do this, we can find position of buses according to time, instead of generate time schedule directly. Which means, use the time of stations instead of buses, to find whether there is buses at each station at specific time. We then can find a time schedule with waiting time according to this trick.

**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.

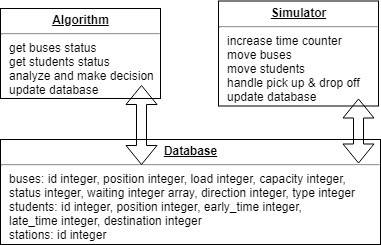


Figure 2

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**

In Stage 3 of the project, there are few things accomplished in data collection. Firstly, considering the short research period before final exam, more data on bus stops have not yet been collected, as complicated communications are involved. For the current stage, it seems best to fully utilize the data we have now first.

In general, one of our primary attempts in this stage is adding the function of bus waiting time optimization. The approach of adding self-loops for each bus stop has been covered in the previous parts. To implement this function, the .edge and .rnet files need to be modified accordingly. First, the waiting time of the bus has been excluded from its initial neighbor edge weights. Then, edges starting from and ending at a node itself are set up. For each node, there are several self-loops, with different weights from w to w/2, w/4, ..., 5. These weights specify the waiting time of the bus at a specific stop. w represents the minimum neighbor edge’s weight, and 5 is the least waiting time set via experience. In the .rnet file, longitudes and latitudes for each node are added accordingly.

Another big step taken is that we now have 17\*7 new .instance files to make use of. In the last stage, we have already retrieved the course data from TAO. In the current stage, course data decoders have been implemented and the time, space, student count information of courses have been extracted out. Since we only need one day range data, the course schedule for the whole semester provide us with 17\*7 real course data based .instance files. A .instance file for a whole semester can also be generated with ease. The new data has been put into test and some results are retrieved.

**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.

The final result of this project is to find a better time schedule than current one. Without changing other limitations, such as bus route, and bus line number. If we want to step further, we may be able to generate dynamical time schedule, and arrange bus sent and route, as well.

**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