USING MATSIM TO SIMULATE AMOD

YOUSEF HINDY

February 15, 2017

CONTENTS

INTRODUCTION

This document outlines the correct usage procedures for simulating large-scale traffic situations using Multi-Agent Transportation Simulation (or MATSim). In particular, this guide will show you how to use ASL's routing algorithms in MATSim.

The simulation is flexible, however, and can provide other uses besides testing the algorithms and methods used in the CDC '16 paper. Many of the modules and features that are written in are easily configurable using the *config.xml* file that will be described later on in Section 2.2.

This guide is split into several parts. The first part outlines a quick overview on how to get the simulation up and running. It will describe the necessary files and formats that the optimizer takes in and give a brief description of how things are working under the hood.

Code Hierarchy

The following diagram depicts the high-level file hierarchy of the AMoD_taxi project.

```
AMoD_taxi
 |__doc
 |__JMatIO_071005
 |__MATLAB_utils
 |__matlabcontrol
 |__src
 |   |__main
 |       |__java
 |       |__resources
 |__test
```

Table 1: Hierarchy Descriptions

Folder	Purpose
doc	Java documentation
JMatIO_071005	Third party package for reading/writing MATLAB files
MATLAB_utils	MATLAB files for running optimization
matlabcontrol	Third party package for controlling MATLAB from Java
src	Source code & resources for MATSim extension
java	Actual java files for MATSim
resources	Necessary xml files and other data for simulation
test	Where the output of the simulations is written

QUICK START

This section will outline a top-level view of how to get a simulation up and running in MATSim. Details on the actual structure and layout of the code will be given later on. This part is therefore meant to be a cursory overview that enables a practitioner to have a simulation up and running.

Parameter	Function
taxisFile	Specifies the file that contains AMoD vehicle data.
pickupDuration	How long it takes for a car to pickup a passenger once it has arrived.
optimizer/type	Specifies the kind of optimizer we want (always AMoD).
reoptimizationTimeStep	How often do we call the optmizer (in seconds).
optimizerDelay	How long to wait after the optimizer has been called to use new routes.
rebalanceWeight	Rebalancing weight for the MATLAB optimizer.
optimizerDataFile	The file from which the MATLAB optimizer reads expected demand.
timeHorizon	How far ahead the optimizer plans for (in seconds).
tripThreshold	What percentage of trips do we give the optimizer.
endTime	The ending time of the simulation (in seconds).
vehicleDiversion	The ability to divert rebalancing vehicles to service customers
detailedStats	Detailed output of relevant taxi statistics
stationMap	The file containing the station to node map. See 2.2.2 for more details.
legacyRebalance	A flag for whether or not MATLAB should calculate rebalancing routes or only stations.
neighbourhoodSize	The number of vehicles to look at when trying to find a match for a customer.

Table 2: Taxi Config Section Parameters

System Requirements

In order to run the simulation, your computer must have the following:

1. Java JDK 1.8 or higher
2. MATLAB (2015b or later)
3. Python 3 or higher (optional, but very useful for generating necessary simulation data)

Necessary Data For Running A Simulation

XML Files For MATSim

There are four main files that MATSim uses to set-up and run a simulation.

NETWORK The first is the network file. Network files contain data for both nodes and links. Each node has an (x, y) coordinate that specifies its location on the map and a unique string identifier that the simulation uses to keep track of it. The coordinate system that is used should be Cartesian, as MATSim uses the Pythagorean theorem to calculate distances between points. Any spherical coordinate system will cause problems for the simulation. Links are specified by a start node, an end node, a capacity, a speed, a number of lanes, a length, and a unique identifier. The field "capperiod" specifies the time period over which the capacity is defined.

An example network file can be found at `src/main/resources/nynetwork.xml`.

Listing 1: An example network file

```
<?xml version="1.0" ?>
<!DOCTYPE network SYSTEM
    "http://www.matsim.org/files/dtd/network_v1.dtd">
<network>
  <nodes>
    <node id="1" x="0" y="0" />
    <node id="2" x="5" y="5" />
  </nodes>
  <links capperiod="01:00:00">
    <link id="1" from="1" to="2" length="25" freespeed="15"
        capacity="96" permlanes="1" />
  </links>
</network>
```

VEHICLES The second is the vehicles file. This xml file contains information about the fleet of autonomous taxis that we will have servicing customer requests. It is important to note that MATSim does not treat the vehicles as autonomous, but instead as simple taxis. This is a distinction that does not matter for the final simulation results, as the distinction between a self-driving and a normal automobile are not relevant to performance on this scale.

Each vehicle is accompanied by a unique identifier, a starting link, a t_0, and a t_1. The t_0 and t_1 denote the range of time in which the vehicle will be active. Both are measured in seconds. For example, if $t_0 = 0$ and $t_1 = 86400$, then the vehicle will be active $t \in [0, 86400]$, i.e. all day. An example of a vehicles file can be found at `src/main/resources/amod/amod_vehicles.xml`.

Listing 2: An example vehicle file

```
<?xml version="1.0" ?>
<!DOCTYPE vehicles SYSTEM
    "http://www.matsim.org/files/dtd/dvrp_vehicles_v1.dtd">
<vehicles>
   <vehicle id="amod_0" start_link="140" t_0="0" t_1="86400"/>
   <vehicle id="amod_1" start_link="210" t_0="0" t_1="86400"/>
</vehicles>
```

CONFIGURATION The third file is the configuration file. This file specifies several variables that tell MATSim how to run the simulation. The file is split up into different sections. The one most relevant to AMoD purposes is the "taxi" section. Table 2 on the preceding page contains the relevant parameters that can be configured.

More importantly, the configuration file also specifies to MATSim the location of the other three files in this section. The "controler" config group contains information about where to write the output of the simulation. The "qsim" config group specifies parameters for the engine that actually runs the simulation. The QSim is one of the several different kind of mobility simulations (mobsims) that MATSim offers. For our purposes, QSim works the best. It is recommended that you do not add the property of "endtime", as when it is omitted the simulation runs until all passengers have been serviced.

⇒ Quick note on the naming convention of configuration files: Take for example `01x05r_config.xml`. "01" stands for the day in March 2012 that is used as the optimizer data file. "05" stands for the day in March 2012 used for the plans file. "r" stands for the fact that the simulation only calculates rebalancing routes (for passenger and rebalancing routes it is "p", and for cars it is "c").

⇒ The output directory's name should always match the name of the simulation.

An example configuration file can be found at `src/main/resources/amod/amod_config.xml`.

PLANS The fourth is the plans file. This xml file contains the **actual** demand for the day. In a normal MATSim simulation, each customer can have multiple plans with multiple different legs, i.e. trips. In this case, however, each customer only has one plan that has one trip.

Each customer has a unique identifier and a set of plans. In the AMoD plans, a plan consists of two activities and one leg that connects them. Since the activity types are not relevant to the simulation, they are all set to "dummy". They can, however, be set to more descriptive titles like "home" or "work".

Each activity has a location (a link), a type, and an end-time. The leg that connects the two activities contains information about the actual trip the customer will have to make. It tells MATSim what time the customer departs and what means of transportation he or she will use. If we want a customer to use the AMoD service, we set the mode to "taxi". Likewise, for customers who will drive themselves, their mode should be set to "car".

An example plans file can be found at `src/main/resources/amod/amod_plans.xml`.

⇒ NB: The plans in `amod_plans.xml` were generated using real NYC taxi data, so each customer only has one leg.

Listing 3: An example plan

```
<person id="19">
      <plan selected="yes">
         <act end_time="18:00:00" link="703" type="dummy"/>
         <leg arr_time="18:01:00" dep_time="18:00:00" mode="taxi"
             trav_time="00:01:00">
            <route/>
         </leg>
         <act link="617" type="dummy"/>
      </plan>
</person>
```

Files for MATLAB

OPTIMIZER DATA FILE The optimizer data file contains trip data in the following format: the first column is the start-time of a trip (in seconds), the second column is the source of the trip as a node or station, and the third column is the sink of the trip. When the MATLAB optimizer is called, it reads data from this file and feeds that data into TIBalancedMCFlow or TIMulticommodityFlow.

An example optimizer data file can be found at `src/main/resources/optimizerdata/1820SourcesSinksNodes.mat`. Note that there are usually two versions of every time period. The one that ends in "Nodes" has the trips stored as a pair of two nodes. The one that ends in "Stations" has the trips stored as a pair of two stations. Usually the nodes version is the one to be used, as runOptimization converts nodes to stations using the station to node map.

STATION-NODE MAP The station-node map file contains two MATLAB arrays. The first is `stationstonodes`, which is a $kx1$ array, where k is the number of stations. In the case of the New York City simulation, there are 100 stations. The entry `stationstonodes`(i) represents the node that is closest to the ith station.

The second array is `nodestostations`. This array is $nx1$, where n is the total number of nodes. The entry `nodestostations`(i) holds the station that is closest to the node i. In the case of the NYC simulation, there are 357 nodes.

An example station-node map can be found at `MATLAB_utils/station_node_map.mat`.

AVAILABLE MATRICES When the "cachedAeqFlag" is set to 1, the optimizer loads pre-built equality and inequality matrices from a database of matrices in the `MATRIX_DIRECTORY` path variable in runOptimization. The details of how the optimizer makes this work are discussed later, but for now it is important to understand that if you want to greatly improve the speed of a simulation, you need to have these matrices ready. They are built for m values from 100 to 5400 in increments of 100. The "available_As.mat" file contains an array of the matrices that have already been built.

Example matrices can be found on Phobos at `/media/Big_Data/matrices`.

⇒ A description of how matrix casting is used can be found in 4.1.2

ZHANG'S DATA This file contains lots of necessary information about the New York road system that Dr. Rick Zhang created in the `LoadRoadGraphLarge.m` file located in `MATLAB_utils`. It contains information about node locations, travel times, and most importantly the road graph. The RoadGraph variable contains information about which nodes can be reached from a particular node. This file is used heavily in generating data and running the simulation. It can be found at `MATLAB_utils/zhangNYdata.mat`.

Variable	Description
LinkLengths	An NxN matrix where LinkLengths(i,j) is the length of the link connecting node i and node j.
LinkSpeed	Similar to LinkLengths but for speeds.
LinkTime	Similar to LinkLengths but for travel time.
NumLanes	Similar to LinkLengths but for number of lanes.
RoadCap	Similar to LinkLengths but for road capacities.
NodesLocation	An Nx2 matrix containing the x-y coordinates of each node
polyx	An array containing x coordinates for the polygon bounding the network. Used for filtering out trips.
polyy	Similar to polyx but for y coordinates.
RefLocation	A reference location used in lla2flat for coordinate conversion.
RoadGraph	An Nx1 cell structure where the ith entry contains all the nodes that can be reached from node i.

Table 3: Necessary data fields for the Zhang data file.

DEMAND FILE In order to create the plans files and the optimizer data files, you will first need to format the demand data in such a way that is parse-able by the python and MATLAB files. The CSV file contains data in the following format: month, day, hour, minute, second, pickup x, pickup y, dropoff x, dropoff y, pickup latitude, pickup longitude, dropoff latitude, dropoff longitude, trip duration, trip length. The last six entries are not essential for the simulation, but it is important that they are there.

Running the NYC simulation

First, you need to do is open the project in Eclipse and open the file `RunAMoDExample.java`. Then, in the main method, change the `configFile` variable to point to the config file that you want to run. Also, you can configure the `otfvis` and `useAMoD` variables depending on whether you want to utilize visualization and the AMoD algorithms, respectively. Finally, press the green run button on the top menu bar. The output from MATSim is quite verbose, but you will know that the AMoD portion is working when you get the message: "Starting optimizer, this could take a while...".

Running a different simulation

If you are not using the NYC Zhang data then you may need to make some changes in the code of `runOptimization` to load the correct data for your simulation. There may also be some other bugs in the code, but I am working on trying to root those out.

Running the optimizer on its own

Occasionally it is necessary to run the MATLAB optimizer on its own for debugging purposes. To do so, open MATLAB and navigate to the folder that runOptimization.m is stored in. In the NYC simulation, this is in `MATLAB_utils`.

Listing 4: Syntax for running the Matlab optimizer on its own

```
runOptimization(startTime, timeHorizon, rebalancingWeight,
    percentOfTripsToKeep, optimizerDataFile).
```

Running the simulation without the optimizer

For debugging purposes, it is often useful to disable the optimizer and have the MATSim simulation run on one of the `optimizer_paths.mat` files so that a piece of MATSim can be debugged. In order to disable the MATLAB optimizer, comment out the following lines in `AMoDTaxiOptimizer.java`.

1. The first line in `initMatlab`, beginning with "`proxy`".
2. The lines from "`proxy.eval("cd ...`" to "`proxy.eval(commandToRun)`" in `callCPLEXOptimizer`.

Relevant Data You Can Expect From the Simulation

All of the results from your simulation will be stored in the output directory you specify in the configuration file. In the NYC simulation, the results are stored in `/test/output/NAME_OF_SIMULATION`.

There you will find several files that will be useful to analyzing the simulation. In the "ITERS" folder, you will find data for each iteration of the simulation that MATSim run. In most cases for AMoD, we are only concerned with the first iteration (it.0).

In the first iteration's folder you will find several useful files. The actual events of the simulation (pickups, dropoffs, cars entering links, cars leaving links, etc.) are in the "0.events.xml.gz" file. With a little bit of pythonic manipulation, you can extract data about almost any aspect of the simulation.

MATSim itself also produces some handy statistics for us. For example, it creates histograms depicting all of the departures and arrivals for each mode of transportation over the course of the simulation. The simulation also generates a text file that contains information about average travel times that can be used when plotting trip data.

GENERATING NECESSARY DATA

Demand File

To generate the demand file, perform the following steps:

1. Download taxi data from `http://www.nyc.gov/html/tlc/html/about/trip_record_data.shtml`.
2. Extract one of the days and relevant information using the "loadNYData.m" file located in `MATLAB_utils`.

MATSim Side

NETWORK FILE To generate the network file, run `nynetworkwriter.py`. It does not take any command line arguments. Some parameters within the file will need to be configured for your needs, like the zhangfile and the outfile names.

VEHICLES FILE To generate the vehicles file, run the `makevehicles.py` file. You can configure the t_0 and t_1 variables or change the distribution. By default, the file simply randomly assigns vehicles to a starting link. You will also need to supply how many links are in your network and how many vehicles you will want.

CONFIGURATION FILE To generate the configuration file for an AMoD simulation, you can reuse the `amod_config_template.xml` file and modify it to your needs. You will most likely only need to modify the network, vehicles, plans, optimizer data, and output directory fields.

For a car-only configuration, modify the `car_only_config.xml` file to suit your needs. You will most likely only need to configure the plans and network fields.

PLANS FILE To generate plans, run the `turnCSVIntoPlans.py` file with the demand file you want to turn into an xml as the first argument. You can configure the start time, end time, and output directory of the script.

MATLAB Side

OPTIMIZER DATA FILE To generate the optimizer data file, follow these steps:

1. Follow the steps outlined above for generating the demand file.
2. Use `PrepDayDataForOptimizer.m` to turn taxi data from a csv file into the desired optimizer data file. You may need to alter the filenames and paths in the file manually before doing so.

⇒ NB: The data should be in the format of the demand file. An example can be found at `/MATLAB_utils/res/March1Data.csv`. The format is laid out in section 2.2.2.

STATION-NODE MAP The station to node map has to be made using the station locations of `tripDataNY100Stations.mat` located in the `MATLAB_utils` folder. In order to generate the stations, you must use the `findClosestNode.m` file and a few loops to generate the proper arrays. For example, to create the station to node map, you would go through each station's location and find the closest node from `NodesLocation`.

AVAILABLE MATRICES There is a file called `makeMatrices.m` that will run through and create the equality and inequality matrices by calling `TIBalancedMCFlow` several times with `cachedAeqflag` set to 0. In order to speed up this process, it is useful to comment out the sections of `TIBalancedMCFlow` that do not pertain to making the matrices, i.e. the building of the cost function, the assembling of the matrices, and the actual calling of the cplex optimizer.

⇒ N.B. If you are running a non-NYC simulation, you will have to make new matrices, as they encode the structure of the road graph and the capacity constraints for a particular scenario.

ZHANG'S DATA You will not need to create a new file like this one if you are running a New York City simulation, but if you want to make a new city, you will have to create a file analagous to `LoadRoadGraphLarge.m` that will designate all of the same properties. Unfortunately this has to be done by hand, but once it is created it can be used for many different simulations on a particular road network.

SIMULATION MECHANICS

Modes

There are three main modes in which you can run the simulation:

1. Calculating only rebalancing routes.
2. Calculating passenger and rebalancing routes.
3. Regular cars only.

You can switch between the first two settings in the `runOptimization.m` file using the boolean flag `onlyRebalance` (set to 1 for option 1, 0 for option 2). To have only regular cars running in the simulation, set the `useAMoD` variable in `RunAMoDExample.java` to false.

⇒ NB: When doing cars only, in the plans file, all customers' modes must be set to "car".

When the optimizer is called, it writes to a file called `optimizerpaths.mat` located in `/src/main/resources`. MATSim then reads and parses this file to store the paths in maps.

Rebalancing Routes Only

When the optimizer is told only to calculate rebalancing routes, MATSim relays information about vehicle location through the `vehicledata.mat` file, which contains the number of vehicles and an array called `vehdistribution`. This array is $kx1$, where k is the total number of stations. vehdistribution(i) contains the number of vehicles in that station.

The rebalancing sources and sinks are then calculated using a steady-state distribution. In equilibrium, each station should have a certain amount of vehicles. By default, the distribution should be uniform, so the expected number of vehicles at each station should be $\frac{N}{k}$, where N is the total number of vehicles and k is the number of stations.

There are two version of the rebalancing: congestion-aware and legacy. When working with legacy, make the `legacyRebalance` parameter of the config "true".

⇒ NB: Some stations in the simulation do not correspond to any nodes and are therefore "dead" stations. The optimizer takes this into account and divides by the number of functioning stations.

The rebalancing sources and sinks are then calculated according to the difference between the steady-state distribution and the actual distribution. For example, if a station is meant to have 5 vehicles but only has 2, it will become a rebalancing sink of intensity 3. Passenger paths in MATSim are calculated using Dijkstra.

The resulting sources and sinks are then passed to `TIMulticommodityFlow` with `milpflag` set to 0, and `congrelaxflag` and `sourcerelaxflag` both set to 1. The output is then passed to `TIRebPathDecomposition`, which gives back the paths in the cell format the MATSim optimizer can parse.

Passenger & Rebalancing Routes

The procedure is a little bit different for calculating passenger and rebalancing routes together. In this case, the optimizer is not given any information about vehicle distribution and instead is only given a start time (t), a time horizon (Δt), and an optimizer data file (along with a few other configuration parameters).

Sources and sinks are found by going through the optimizer data file and only keeping the trips that have a starting time $t_i \in \{t, \Delta t\}$. In addition, only a certain percentage of the trips, determined by `tripThreshold`, are kept. The sources and sinks are then consolidated so that repeat trips are represented as a single trip with a non-unit flow. The arrays `Sources`, `Sinks`, and `FlowsIn` are then shortened down to the nearest hundred so that `TIBalancedMCFlow` can used the cached matrices mentioned in 2.2.2. The output of `TIBalancedMCFlow` is then decomposed into passenger paths using `TIPaxPathDecomposition` and `TISamplePaxPaths`.

To calculate rebalancing routes, the sources and sinks are cleaned up as follows: if a node has an inflow of intensity 6 and an outflow of intensity 3, it becomes a rebalancing source of intensity 3. This greatly streamlines the process of determining the routes. Afterwards, the flows are "flattened" so that the sources and sinks both have the same dimensions. They are then passed to `TIDecomposeFracRebSol` along with the output of `TIBalancedMCFlow` from index $N^2M + 1$ to index $N^2(M + 1)$, where N is the number of nodes and M is the number of original flows, i.e. the size of `Sources` and `Sinks`. `TIRebPathDecomposition` gives the final rebalancing paths.

Car

The final mode is the car-only mode. In these scenarios, MATSim does all of the calculation of routes for agents in the simulation. Instead of using taxis, people take their own cars on the road, departing at the end of their first activity. This is the standard way that MATSim simulates traffic situations. Each agent is assigned a score based on the performance of their route, and in subsequent iterations they try to change their plans in order to maximize their score. This means that unlike the AMoD simulation, it makes sense to allow MATSim to run the car-only version for several iterations.

Under the Hood in MATSim

DVRP and Taxi

The AMoD simulation builds heavily upon two extensions that were written by Michal Maciejewski of Poznan University of Technology: DVRP and taxi.

The Dynamic Vehicle Routing Problem (DVRP) extension allows for agents in the MATSim simulation to alter their plans during the middle of the simulation, which cannot be accomplished in the basic MATSim build. The general idea is that each agent has a schedule of tasks that he or she completes over the course of the day. The beauty of DVRP is that one can implement logic for the agents to choose their next action. Optimizers can be written for the extension that handle different kinds of events and activities too.

The taxi extension builds heavily upon the DVRP extension by implementing a series of task, request, schedule, and optimizer classes that are tailored to the specific problem of simulating a fleet of taxis in a large-scale urban environment. Naturally this extension fit very well into the framework of the AMoD problem, so with a few modifications we are able to simulate both the servicing behavior of a taxi and the rebalancing behavior of an autonomous vehicle.

Class Hierarchy

The following diagram depicts the file hierarchy of the java files necessary for running an AMoD simulation.

```
java
├── AMoDDispatchFinder.java
├── AMoDNodeSchedulingProblem.java
├── AMoDPerformance.java
├── AMoDQSimProvider.java
├── AMoDRebalanceTask.java
├── AMoDSchedulingProblem.java
├── AMoDStationSchedulingProblem.java
├── AMoDTaxiOptimizer.java
├── AMoDTaxiOptimizerParams.java
├── AMoDTaxiScheduler.java
├── MatlabConnector.java
├── Pair.java
└── RunAMoDExample.java
```

The files are well-documented, but it is worth going in depth into the workings of the `AMoDTaxiOptimizer`.

AMoDTaxiOptimizer Mechanics

OPTIMIZER DELAY The optimizerDelay variable in the configuration file specifies how long MATSim should wait before using the new routes that the MATLAB optimizer produced. This feature emulates the real-life time that it would take for the optimizer to run, for in MATSim the simulation stops when the MATLAB optimizer is called. To have no delay, set this to 0.

PATH STORAGE The output of the MATLAB optimizer is a cell array where each entry contains a kx2 matrix. The left column is the actual path, given as a list of nodes. The right column is the flow through each of those nodes. For the purposes of the simulation, the second column is irrelevant.

The decomposePassPathsStations and decomposeRebPathsStations methods in the AMoDTaxiOptimizer class contain the functionality to turn these cluttered cell arrays into clean MATSim Paths. A MATSim Path contains a list of links, a list of link travel times, a total travel time, and a total travel cost. The Path object should not be confused with the VrpPathWithTravelData object, which contains more information about when the trip starts and ends.

For station-wise routing, the routes are stored differently depending on whether they are passenger paths or rebalancing paths. For passenger paths, there is a map that connects an integer representing a station to another map. The interior map connects a destination station to a Set of Paths. This makes it easy to find a route between two particular stations. Rebalancing storage is much simpler. Each station is mapped to a set of possible rebalancing routes out of that particular station.

NOTIFY BEFORE SIM STEP The method notifyMobsimBeforeSimStep is called before every time-step of the MATSim simulation. In short, what this method does is determines whether or not re-optimization is required, i.e. has reoptimizationTimeStep seconds passed since the last optimization, and then manipulates the situation for the best outcome. If so, the function calls the cplex optimizer in MATLAB and prints data about the current state of the simulation.

If it is not time to re-optimize, the optimizer will check whether or not it should start using the new routes from the previous optimization. Finally, it schedules all of the trips using the scheduleUnplannedRequests method.

CALLING THE CPLEX OPTIMIZER The AMoDTaxiOptimizer calls the MATLAB optimizer using a proxy established using the MatlabConnector class. In callCPLEXOptimizer, the proxy is used to call the runOptimization script in MATLAB_utils.

It does so by first calculating the vehicle distribution of the simulation at that time step and writing it to a file called vehicledata.mat. Afterwards, it calls runOptimization and reads the result from the optimizerpaths.mat file. Finally, it calculates the number of vehicles required and decomposes the paths so that they can be used by the scheduler later on.

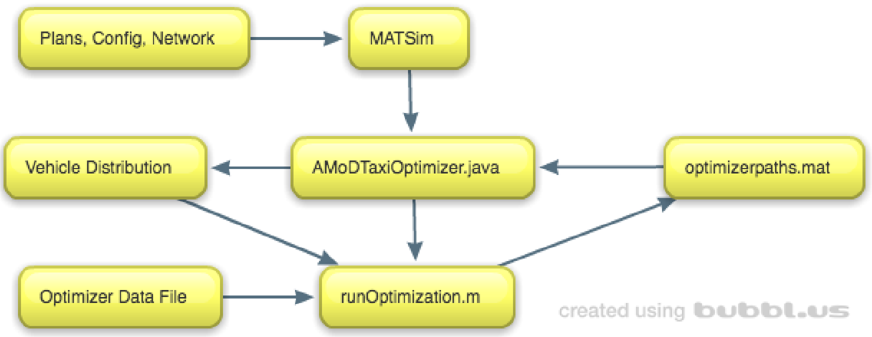

Figure 1: Optimization Work Flow

VISUALIZING RESULTS

OTFVis

The open source On the Fly Visualizer (OTFVis) is useful for visualizing the results of MATSim simulations. In order to make use of this extension, perform the following steps.

1. Download MATSim 0.8.0 from `http://matsim.org/downloads` and the OTFVis 0.8.0 extension from `https://github.com/matsim-org/matsim/releases/tag/matsim-0.8.0`. Place the OTFVis directory inside the `matsim-0.8.0` directory.
2. To create the visualization file, run the following command:

Listing 5: Command to generate visualization file

```
java -cp matsim-0.8.0.jar:otfvis-0.8.0/otfvis-0.8.0.jar
    org.matsim.contrib.otfvis.OTFVis -convert path-to-events
    path-to-network path-to-output.mvi snapshot-period
```

3. To actually visualize the results, run the following command:

Listing 6: Command to open and view visualization file

```
java -cp matsim-0.8.0.jar:otfvis-0.8.0/otfvis-0.8.0.jar
    org.matsim.contrib.otfvis.OTFVis path-to-mvi
```

ACKNOWLEDGMENTS

I would like to thank Prof. Marco Pavone and Federico Rossi for allowing me to work on this project with them. I have learned a lot this summer and your guidance has been invaluable.

I would also like to thank the MATSim developers, especially Michal Maciejewski, for being so helpful in answering my inquiries. Without you none of this would have been possible.