# DTALite-s python version Users Guide

# Introduction

Among widely used traffic simulation tools, DTALite is a queue-based mesoscopic traffic simulator, documented in the paper by Zhou and Taylor. It is an open-source mesoscopic DTA (dynamic traffic assignment) simulation package designed to provide transportation planners, engineers, and researchers with a theoretically rigorous and computationally efficient traffic network modeling tool.

The transportation mobility simulation engine DTALite-S is an important extension based on DTALite, which integrates the agent-based dynamic traffic assignment and traffic simulation. This document mainly introduces the python version of DTALite-S.

# System Architecture

The software architecture of DTALite-S aims to integrate many rich modeling and visualization capabilities into an open-source traffic assignment model suitable for practical everyday use within the context of an entire large-scale metropolitan area network. Using a modularized design, the open-source suite of **simulation engine + visualization interface** can also serve future needs by enabling transportation researchers and software developers to continue to build upon and expand its range of capabilities. The **streamlined data flow** from static traffic assignment models can allow state DOTs and regional MPOs to rapidly apply the advanced STA/DTA methodology, and further examine the effectiveness of traffic mobility, reliability and safety improvement strategies, individually and in combination, for a large-scale regional network, a subarea or a corridor.



a

.

Network Data

node

.

csv

link

.

csv

b

.

OD Demand

demand

.

csv



c

.

Traffic Assignment Module



e

.

Space

-

Time Simulation

Point Queue

Spatial Queue

Link Flow

BPR Volume Delay

Function

Shortest Path Tree

Generation

Flow Assignment

Link Flow



Static Link Performance



Agent Trajectory



Network

d

.

NEXTA

:

Visualization

Interface



Dynamic Link Performance



Agent Trajectory



f

.

Simulation Output

Figure 1.1 Software System Architecture

The components and different modules in the system are listed as following:

**a. Network Data** includes two essential files, node.csv and link.csv for the network representation.

**b. OD Demand Database** includes the demand.csv that describes information such as agent type, o\_node\_id, d\_node\_id, departure time, which help users to represent the OD demand information.

**c. Traffic Assignment Module** includes the key steps of the assignment, including the BPR Volume Delay Function, Shortest Path Tree Generation, and Flow Assignment, which generates the link flow according to the UE principle.

**d. NEXTA: Visualization Interface Module** is able to visualize the network and the output of traffic assignment, including Static Link Performance and Agent Trajectory.

**e. Space-Time Simulation Module** utilizes the path flow output of Traffic Assignment Module to perform Space-Time Simulation, while the underlying traffic flow model in the Space-Time Simulation Module is Point Queue (PQ).

**f. Simulation Output Module** covers the output file of Space-Time Simulation Module, including Dynamic Link Performance and Agent Trajectory in terms of link\_performance.csv and agent.csv, which can be visualized in NeXTA.

# Detailed data structure descriptions

## 3.1 Input for network data

* The specific files for physical layer are *node.csv* and *link.csv*.
* Nodes in the physical network represent points of demand, including node\_id, zone\_id, and coordinates with an arbitrary coordinate system.
* A link is defined using upstream node and downstream node ids, with essential attributes such as length, free\_speed, lanes, capacity, link\_type, and coefficients of Volume Delay Function, typically required for static traffic assignment.

**File 1a: node.csv**

|  |  |  |
| --- | --- | --- |
| **Field Name** | **Description** | **Sample Value** |
| name | Optional for visualization only | Main street @ Highland Dr. |
| node\_id | Node identification number | 1001 |
| x\_coord | Longitude or horizontal coordinate in any arbitrary geographic coordinate system. | 100 |
| y\_coord | Latitude or vertical coordinate horizontal coordinate in any arbitrary geographic coordinate system | 200 |
| zone\_id | Indication of node’s physical location | 1 |

**File 1b: link.csv**

|  |  |  |
| --- | --- | --- |
| **Field Name** | **Description** | **Sample Values** |
| name | Optional for visualization purposes | Main Street |
| link\_id | Link identification number of the road | 1003 |
| from\_node\_id | Upstream node number of the link, must already be defined in *node.csv* | 1 |
| to\_node\_id | Downstream node number of the link, must already be defined in *node.csv* | 3 |
| link\_type\_name | Optional text label for visualization and data checking purposes | 1 |
| dir\_flag | Indication of directions of the link (=0, bi-direction; =1, single direction) | 1 |
| length | The length of the link (between end nodes), measured in units of miles or km. | 10 |
| free\_speed | Free-flow speed on defined link. Suggested Unit: mph or kmph | 60 |
| lanes | The number of lanes on the link | 1 |
| capacity | The number of vehicles per hour per lane | 4000 |
| link\_type | Index of link type name | 1 |
| VDF\_alpha1 | Coefficient used in the volume-delay function | 0.15 |
| VDF\_beta1 | Coefficient used in the volume-delay function | 4 |

## 3.2 Input for demand data

* The specific files for demand layer is *demand.csv*

**File 2: demand.csv**

|  |  |  |
| --- | --- | --- |
| **Field Name** | **Description** | **Sample Values** |
| agent\_id | Agent identification number | 1 |
| agent\_type | Index of agent type name | v |
| o\_node\_id | Origin node id, must already defined in *node.csv* | 1 |
| d\_node\_id | Destination node id, must already defined in *node.csv* | 2 |
| o\_zone\_id | Origin zone number of the link, must already defined in *node.csv* | 1 |
| d\_zone\_id | Destination zone number of the link, must already defined in *node.csv* | 2 |
| departure\_time\_in\_min | Departure time from origin node id(min) | 420 |
| PCE | Passenger Car Equivalent (PCE) of the agent | 1 |

# Classes and Functions overview

Here we list the classes and functions used in the DTALite-s

Table 1: Classes Overview

|  |  |
| --- | --- |
| **Class** | **Description** |
| class Node | Class for node |
| class Link | Class for link |
| class Agent | Class for agent |
| class Network | Class for generating shortest path |

Table 1: Functions Overview

|  |  |
| --- | --- |
| Function | **Description** |
| g\_A2R\_simu\_interval() | Convert absolute time to relative time |
| g\_ReadInputData() | Read the input files |
| g\_TrafficAssignment() | Traffic assignment |
| g\_TrafficSimulation() | Space time simulation |
| g\_OutputFiles() | Generate simulation output files |

# Space-time network construction for a point queue model with constant capacity

Consider a physical transportation network with a finite set of nodes and a finite set of links where nodes and directed link . A space-time network , where is the set of space-time vertexes, is the set of space-time link under planning time horizon . Each arc indicates a directed space-time path from node departing at time to node arriving at time . Thanks to the discretized space-time network structure, it is easy to incorporate passengers’ travel request time windows and vehicles’ operating hour requirement, and more importantly, simplified queueing models (SQM) which can distinguish vehicles’ speed in free-flow and congested conditions, under the assumption of constant bottleneck discharge capacities. Some researchers provide a systematic comparison among modes of point queues, spatial queues, and an extended version with time-dependent capacity and spillback along the backward wave. Along this line, one can extend the method from Lawson et al. to calculate the spatial and temporal extents of queue and the actual waiting time spent upstream of a bottleneck. The schematic trajectories of vehicle from node to node in space-time network are shown in Fig. 2.



## 6 Simulation flowchart based on simple data structure

Illustrated in Algorithm 2, we need to perform two loops of time and agents across different links to check the available road and vehicle carrying capacity. As we follow a point queue-based system, without complicated data structure, we only need to be concerned about very few key variables namely arrival time and departure time of vehicle  on link: *, ,* as well as cumulative arrival/ departure counts of vehicles on link at time , ) and ).

**Algorithm 2.** Simulation process in DTALite-S using a simplified queue based model

|  |
| --- |
| **Step 1: Initialization: prepare input data with nodes,links and agents**  **Step 2: Traffic assignment(MSA,** **method of successive averages)**  **For** (*i*=0;*i*< number\_of\_assgnment\_iterations; i++) # loop for each assignment iteration  **For** (=0; <; ++) # loop for each link in the network  Update the link cost according to the BPR function  Find the shortest path for the corresponding agents  Update the link volume |
| **Step 3: Perform simulation** |
| **For** (=0;<; ++) # loop for each simulation time  **If** (agent v is ready to depature)  Activate agent and put it in the entrance queue on the first link in the path |
| **For** (=0; <; ++) # loop for each link in the network  Pop the agents from the entrance queue and push them back into the exit queue on the link, and update the agents’ departure time  **For** (=0; <; ++) #loop for each node in the network  **For** each incoming link:  #check if the current link has sufficient capacity and there are agents in exit queue  **While**(link outcapacity>0 and exit queue is not empty)  # check if the agents’ departure time on this link is later than the current time  **If(**agent’ departure time< the current time**)**  **Break**  # ready to move next link  **If(**current link is the end of the path**)**  the agent complete the simulation  **Else**  Pop the agents from the exit queue on current link and push them back into the entrance queue on the next link  update the Calculate link capacity |
| **Step 4: Output data for statistics collection** |

In step 3, Illustrated in Fig. 4, the waiting time and queue length can be derived from the grant traffic state variables of ) and ). Overall, it could be memory consuming to store a full matrix of variables *,* , one can use dynamically allocated vectors to store and update the link sequences along their paths.



**Fig.4** Queue length and travel time various in simulation process, where WT is the waiting time