Pedestrian demand estimation - Manual (v0.0)

Flurin Hänseler & Nicholas Molyneaux
Transport and Mobility Laboratory, EPFL

July 27, 2015

Contents

1 Copyright and conditions of use

2 Goal

The purpose of the pedestrian demand estimation framework is to be able to predict the demand in train stations based on various data sources, all of which are relatively cheap to install in train stations. By using flow counts on specific links in the station coupled with information on the train timetables and detailed boarding and disembarking information, this framework allows the user to predict the demand throughout the station.

3 Framework structure

The framework is implemented in python, with various modules corresponding to parts in the publication cited previously. Some modules are case specific, whereas others are generic and can be re-used as is for different projects. A code documentation scheme is setup using Sphinx. With this it is possible to get an overview of all the functions and classes implemented in the framework. The following document contains the essential and important concepts and explanation about the modules to be able to understand and therefore use the code.
For confidentiality reasons, the data provided online is fictious. Generated randomly, this data set should be realistic but most certainly highly improbable, counter intuitive scenarios will most certainly take place.

main.py The *main* method of the code. This is the module to call to run the code. This module contains various information about simulation like the days or weights to give each data set. It also calls the class which loads the data, does the simulation and post-processing: *DemEstimator*.

DemEstimator.py This is the module which actually calls all the various parts of the simulation. From the data loading through to the post-processing of the results. The initial parameters passed from the *main* are transformed to be used nicely. Two variants are available, using one single day or combining multiple days. This module does not need modifying if the data format is respected precisely.

LausEstPar.py Information and parameters concerning the simulation with paths to the various data files.

LausNetw.py Specification of the network composing the train station.

The lists of vertices, edges, routes and more are given in this module. Of course this specification is case specific.

Assg.py Computes the assignment matrices, noted B and C in the publication. These matrices are only calculated if the precomputed folder is empty. This done as their computation takes a long time (up to 10-15 minutes per matrix). After the calculation the matrices are written to csv if they were not loaded. The sub-matrices are also calculated

DNL.py Actual computation of the link flow assignment matrices. The calculations are done in parallel over as many processors as possible.

LausSensData.py Loads the sensor data and changes the format to make is usable. Loads both the link flow counts from ASE and the tracking data from VisioSafe.

Prior.py Calculates the prior from the train timetable. The loading and unloading flows are calculated from the train induced events, i.e. trains arriving and leaving the station.

DemEst.py The place where all the parts are assembled and the optimization is performed. The routes demand is calculated before calling non linear least squares methods to perform the optimization.

Eval.py Process the raw results from the optimization and aggregates them into meaningful vectors and arrays. Two versions exist - Eval.py and EvalMD.py - this for both cases when the either single or multiday analysis are performed.

PoatProcessing.py Generates all the images and plots of the results. Amongst other things, generates the OD matrix and network load at each time interval.

4 Example data set: lausanne

The methodology is applied to the train in Lausanne, Switzerland. It is one of the critical hubs in the Swiss train network. For confidentially reasons some data cannot be shared, hence it has been modified so it can be published. The main data sources are the following, for detailed explanation of the data and the storage formats please see the files directly.

Link flow counts From ASE GmbH, usually denoted as ASE in the source code. These link flow counts are stored in csv format, with the number of pedestrians recorded in a given time interval in each link. For some obscure reason, the data is separated in two files denoted by *LS* and *add*.

Tracking data Recorded by VisioSafe SA, denoted as VS in the source code. The tracking data of each person in the station for the time period of interest. Stored in csv format with one line per entry (person).

Sales data Data provided by the shop sales, which returns similar information as ASE. Since there are very few entries for this data set and it is an average per day, this information is coded in the *LausSensData.py* module; unlike the other data sources which are loaded from csv.

Train timetable The model for the alighting and boarding passengers from the trains is used to build a prior and fixed flows on some links. The model is implemented in the module *LausTTTdata.py* which loads the parameters from csv files.

Train station model Modeled as a *networkx* object, the train station must contain the routes and nodes which exist. Python's dictionaries are extensively used to facilitate the implementation in other modules.

5 Documentation and running the framework

Some documentation using sphinx can be compiled by running the following commands

1. `/documentation/sphinx$ sphinx-apidoc -o source ../../src/`
2. `/documentation/sphinx$ make html`

where the first one generates the .rst files for each python module, the second compiles them into html format which can be viewed with any navigator. The main purpose of the sphinx documentation is to be able to overview all the methods and functions in the modules in a convenient manner. Remark: it is recommend to uncomment the line `if __name__ is "__main__":` located in the main, before compiling the sphinx documentation. This will avoid sphinx from running the whole code just to generate the documentation. The documentation can be viewed by opening the *docs.html* in the *documentation/sphinx/build/html* folder. To run the whole framework, the simple call to the main is required, which will output the results in the folder *output/*:

1. `/src$ python main-public.py`

6 Output

For each day, many plots and txt file are produced into the output folder. The figures are produced as static images and also tikz files for easy and good quality use in tex documents. The figures include OD matrix plots, the network loaded at each time interval and the comparison of the flow between the observation and predictions for each link.
Some txt files are also produced which contain the raw data to be used for other sorts of visualization if needed.