SUMO User Conference 2025

Conference paper

https://doi.org/10.52825/xxxx........ DOI placeholder (WILL BE FILLED IN BY TIB Open Publishing)

© Authors. This work is licensed under a [Creative Commons Attribution 4.0 International License](https://creativecommons.org/licenses/by/4.0/)

Published: (WILL BE FILLED IN BY TIB Open Publishing)

UTDF2GMNS: An Open-Source Python Implementation for Converting Synchro UTDF with Signalized Intersections to SUMO using GMNS Standard

Xiangyong Luo1[](https://orcid.org/0000-1111-2222-3333)https://orcid.org/0009-0003-1290-9983, Ross Wang1, Wan Li1, and Xuesong Simon Zhou2

1 Oak Ridge National Laboratory, US

2 Arizona State University, US

\*Correspondence: Xiangyong Luo, luox1@ornl.gov

**Abstract.** Modern transportation research relies on seamlessly integrating traffic signal data with robust network representation and simulation tools. We present UTDF2GMNS, an open-source Python-based tool that converts Synchro Universal Traffic Data Format (UTDF) files, including signalized intersections, into the widely adopted General Modeling Network Specification (GMNS). The resulting GMNS network is then ready for simulation in SUMO (Simulation of Urban Mobility). By automating the extraction of intersection control parameters and aligning them with GMNS conventions, UTDF2GMNS minimizes manual efforts and data loss. This streamlined workflow empowers researchers and practitioners to build accurate network models, test scenarios more efficiently, and maintain data consistency across platforms. Validation with multiple case studies confirms the tool’s reliability in modeling complex corridors with multiple signalized intersections. As a free and open-source solution, UTDF2GMNS promotes reproducibility and collaboration in the transportation community. Comprehensive documentation and tutorials are available on GitHub, facilitating broader adoption of standard practices in multimodal network modeling.

**Keywords**: UTDF, SUMO, GMNS, Network Simulation, Signal Control

Introduction

Literature

Source code and Datasets

*2.1. Source Code and Installation*

The mgwr source code (the examples in this paper were composed using mgwr version 2.0.1) is organized as a module of the Python spatial analysis library (PySAL) ([**https://pysal.org**](https://pysal.org/)) and is therefore available from a repository on the PySAL project GitHub page ([**https://github.com/pysal/mgwr**](https://github.com/pysal/mgwr)). Each PySAL module is complete with ‘docstrings’ (i.e., input and output documentation) for all available functions and code examples (i.e., Jupyter notebooks) that make it simple to replicate and extend the examples to new applications. In addition, ‘unit tests’ are provided that allow the source code to be continuously integrated while being developed. This ensures that new features and dependency updates do not unknowingly break existing features.

Currently, mgwr has four dependencies: numpy, scipy, libpysal, and spglm. The first two dependencies, numpy and scipy, are elementary within the Python scientific computing ecosystem and provide core data structures and data manipulation functions. The third dependency, libpysal, is central to PySAL and provides a repository of example datasets. Since libpysal is dependent upon pandas, then pandas is an indirect dependency for mgwr and is often useful for reading and managing data tables. The final dependency, spglm, provides a light-weight generalized linear model framework for calibrating each of the local parameter estimates within (M)GWR via iteratively weighted least squares. The most recent stable version of mgwr, along with these direct and indirect dependencies, may be installed from the Python packaging index (PyPI) using the pip package manager:

pip install mgwr

To obtain in-development features, it is also possible to install mgwr directly from the source code:

pip install [**https://github.com/pysal/mgwr/archive/master.zip**](https://github.com/pysal/mgwr/archive/master.zip)

Additional packages, namely matplotlib and geopandas, are used for presenting results from empirical demonstrations and can also be obtained via pip; however, they are not required for the core mgwr functions. Once all the necessary packages are installed, they can be imported for use in the following examples as such:

>>> import numpy as np

>>> import pandas as pd

>>> import libpysal as ps

>>> from mgwr.gwr import GWR, MGWR

>>> from mgwr.sel\_bw import Sel\_BW

>>> from mgwr.utils import compare\_surfaces, truncate\_colormap

>>> import geopandas as gp

>>> import matplotlib.pyplot as plt

>>> import matplotlib as mpl

*2.2. Datasets*

Two datasets are utilized throughout this paper to illustrate various (M)GWR functionality. First, is the well-known Georgia dataset that is described in [[**2**](https://www.mdpi.com/2220-9964/8/6/269#B2-ijgi-08-00269)] (2002) as well as subsequent publications [[**7**](https://www.mdpi.com/2220-9964/8/6/269#B7-ijgi-08-00269),[**11**](https://www.mdpi.com/2220-9964/8/6/269#B11-ijgi-08-00269)]. The second is a sample of Airbnb rental data from the Prenzlauer Berg neighborhood of Berlin from InsideAirbnb, which provides a more recent example with a relatively larger sample size.

2.2.1. Georgia Dataset

The Georgia dataset consists of 159 counties in the state of Georgia ([**Figure 1**](https://www.mdpi.com/2220-9964/8/6/269#fig_body_display_ijgi-08-00269-f001)), and records socio-demographic characteristics from the 1990 US census. The county locations are abstracted as centroids so that inter-county distances can be computed within the (M)GWR routine, though it is convenient to visualize the model output using the county polygons, since they are the scale at which the observations are aggregated. A small subset of the available variables are selected here for an example modeling educational attainment. The covariates are described in [**Table 1**](https://www.mdpi.com/2220-9964/8/6/269#table_body_display_ijgi-08-00269-t001). Python code for loading and visualizing the Georgia dataset is as follows:

**Figure 1.** The 159 counties within the state of Georgia. Note: basemap and scalebar added using additional code.

**Table 1.** Georgia dataset.

#Load Georgia dataset and generate plot of Georgia counties ([**Figure 1**](https://www.mdpi.com/2220-9964/8/6/269#fig_body_display_ijgi-08-00269-f001))

>>> georgia = gp.read\_file(ps.examples.get\_path(‘G\_utm.shp’))

>>> fig, ax = plt.subplots(figsize = (10, 10))

>>> georgia.plot(ax=ax, \*\*{‘edgecolor’: ‘black’, ‘facecolor’: ‘white’})

>>> georgia.centroid.plot(ax = ax, c = ‘black’)

>>> plt.savefig(‘georgia\_shp’)

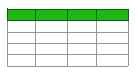
>>> plt.show()

2.2.2. Berlin Airbnb Dataset

The Berlin dataset consists of 2203 observations that are geolocated instances of Airbnb rental properties ([**Figure 2**](https://www.mdpi.com/2220-9964/8/6/269#fig_body_display_ijgi-08-00269-f002)) and their associated characteristics from 2017 in the Prenzlauer Berg neighborhood. Prenzlauer Berg is a gentrifying neighborhood known for its arts scene, shopping, and nightlife, and is therefore a popular tourist destination. A small subset of variables were selected for a rental price modeling example, which are described in [**Table 2**](https://www.mdpi.com/2220-9964/8/6/269#table_body_display_ijgi-08-00269-t002). Note that the logarithm of rental price is used here to correct the skewness of the variable. Since the data are not aggregated, the analysis and visualization of the results are carried out at the point-level.

**Figure 2.** 2203 rental properties in the Prenzlauer Berg neighborhood of Berlin. Note: basemap and scalebar added using additional code.

**Table 2.** Berlin dataset.



#Load Berlin dataset and generate plot of properties ([**Figure 2**](https://www.mdpi.com/2220-9964/8/6/269#fig_body_display_ijgi-08-00269-f002))

>>> prenz = gp.read\_file(ps.examples.get\_path(‘prenzlauer.zip’))

>>> prenz\_bound = gp.read\_file(ps.examples.get\_path(‘prenz\_bound.zip’))

>>> fig, ax = plt.subplots(figsize = (10, 10))

>>> prenz\_bound.plot(ax = ax, \*\*{‘edgecolor’: ‘black’, ‘facecolor’: ‘white’})

>>> prenz.plot(ax = ax, markersize = 10, \*\*{‘edgecolor’: ‘black’,

‘facecolor’: ‘black’})

>>> plt.savefig(‘prenz’)

>>> plt.show()

Methodology

Case study

Conclusion and Future work

**Data availability statement**

**Underlying and related material**

Github repository:

PyPI:

**Author contributions**

**Competing interests**

The authors declare that they have no competing interests.

**Funding**

National Science Foundation CNS-2034615

**Acknowledgement**

The research is partially supported by the National Science Foundation (NSF) grant CNS- 2034615. Any opinions, findings, conclusions, recommendations expressed in this paper are those of the authors and do not necessarily reflect the views of NSF.

**References**

1. J. K. Author, “Name of paper,” Abbrev. Title of Journal, vol.x, no.x, pp. xxx–xxx, Abbrev. month, year, doi: https.doi.org/……
2. J. K. Author, “Title of chapter in the book,” in Title of Their Published Book, xth ed. City of Publisher, (only U.S. State), Country: Abbrev. of Publisher, year, ch. x, sec. x, pp. xxx–xxx.
3. First Name Initial(s) Last Name. “Page Title.” Website Title. Web Address (date accessed)