Northwestern University Freight Rail Infrastructure and Energy Decarbonization (NUFRIEND) Package

User Guide

March, 2024

Version 2.0

Table of Contents

1. Introduction

The NUFRIEND Framework and Dashboard is a comprehensive industry-oriented tool to optimize and simulate the deployment of new energy technologies across U.S. freight rail networks. Scenario-specific simulation and optimization modules provide estimates for carbon reductions, capital investments, costs of carbon reductions, and operational impacts for any given deployment profile. Recent updates to the underlying framework and code package allow for the consideration of a greater range of scenarios and modeling objectives, including modeling hybrid diesel-battery locomotive deployment and time-dependent technology rollout optimization strategies.

1.1. Purpose

The purpose of this user guide is to introduce users to the code package underlying the NUFRIEND Framework and how it may be used to run a range of scenarios of interest to users. A demonstration video has been published for additional reference.[i]

1.2. Information

Users can refer to the NUFRIEND Framework - NUTC webpage for more updated information on the overall project. Any questions regarding the dashboard can be directed to Max Ng and Adrian Hernandez.[ii, iii]

1.3. Data Flows

In contrast to the internal data flow adopted for the NUFRIEND Dashboard, all data used in the NUFRIEND Framework code package must be on the user's local machine. This means that in addition to users specifying all the required scenario parameters, they must either (a) accept the provided data files, or (b) substitute the data files with data of their own.[1] Importantly, most of the provided data files contain state of the art parameter estimates on energy, economic, and operational values.[2] In order to illustrate the code package's functionality while keeping with our confidentiality agreements, we have substituted the freight flow data used in our analysis with a ramdomly generated dataset. This randomly generated dataset allows users to run the simulation tool and understand the required file format for any self-provided freight flow data.

Data outputs are stored in the objects that are returned by the program, which can be used to graphically display the results or print/write the results to files for further user analysis.

2. Getting Started

The NUFRIEND Framework code package has been made publicly available in a GitHub repository.[iv] Users can access the code and necessary data by cloning the repository or by directly downloading all of the files onto their local machine.

2.1. Code Package Structure

The code package contains several kinds of files and subdirectories. The entire program is written in python, with each of the ".py" files containing module code. The two primary files are "run.py," which contains the method for running a provided static deployment scenario and "run_mp.py," which contains the method

[1] If users wish to provide their own data files, their formats and names must match the formats and names of the files they are replacing.

[2] The sources and methods for computing these values are discussed in greater detail in the Final Technical Report available on the NUTC Webpage.

for running a provided dynamic/rollout deployment scenario. Both of these methods apply all the other code modules and functions on a specified scenario input file that must be created and appropriately filled out by the user. Users will also see two directories "input" and "output". The "input" directory contains several subdirectories which each contain relevant input data files. Though users can replace/edit any files in these subdirectories, we note file structure and names should remain the same as those provided in the package.[3]

The "scenario" subdirectory contains the most relevant files for specifying a simulation scenario. The file "input_legend.csv" contains information on allowed input values for scenario specification. The accompanying "input_template.csv" file is a blank file users can copy to fill out and specify any desired scenario in. Several test scenarios have been provided for users to experiment with, these files begin with "test" and are each followed by a number. To ensure scenarios are running as they should be, we have included pregenerated verification files for each test scenario provided (i.e., "test1_check.csv" can be used to load the scenario to check the locally run scenario produced from "test1.csv"). Greater detail on this process is provided in Section 3.4. Finally, the suffix "_mp" on all the code and input files stands for "multi-period" and is used to separate the static optimization framework files from the rollout optimization framework files.

The "output" directory contains several subdirectories for storing the scenario results when caching is desired. This allows future scenarios to be loaded instead of re-run. The files found in these subdirectories are meant to allow users to check whether the program is functioning as expected on their machine and are not essential to running the actual program.

3. Running Scenarios with the Code Package

This package was developed using Python 3.10 and requires all the packages (and any dependencies) listed in the "util.py" file. Because the foundation of the NUFRIEND Framework rests on network optimization, a Gurobi for Python with a valid Gurobi License is required to run most scenarios.[v,vi]

3.1. Specifying a Scenario

Users will need to create and directly enter their scenario specification parameters into a ".csv" file within the "input>scenario" subdirectory. This file must follow the exact structure provided in the "input_template.csv" file (or "input_template_mp.csv" file for rollout optimization) and satisfy the input value restrictions detailed in "input_legend.csv" (or "input_legend_mp.csv" file for rollout optimization). If users wish to provide their own freight flow data, this provided file must match the format of the "flow_template.csv" file (or "flow_template_mp.csv" file for rollout optimization) in the "input>flow" subdirectory. The name of the desired flow data file must be included as the value for the "flow_data_filename" row within scenario specification file.

3.2. Running a Scenario

A scenario can be run in a few simple ways, depending on whether the scenario is to be run on the static optimization framework or on the rollout optimization framework. The relevant code blocks are displayed below, each using one of the provided test scenario filenames as inputs.

Users may wish to cache the scenario results produced by the static optimization framework. If a scenario was previously run and cached, it can be loaded as shown in Code Block 2.

[3] Should a file be accidentally altered or replaced, we recommend users redownload the package (or specific data files) from the GitHub repository.

Code Block 1 - Code for running static optimization framework on scenario specified in 'test1.csv'. No prior scenario is loaded and the scenario output is cached.

```
from run import *
G, f = run_scenario_file(scenario_code='test1', plot=True,
load_scenario=False, cache_scenario=True)
```

Code Block 2 - Code for loading cached test results for static optimization framework on scenario specified in 'test1.csv'.

```
from run import *
G, f = run_scenario_file(scenario_code='test1_check', plot=True,
load_scenario=True, cache_scenario=False)
```

Code Block 3 - Code for running rollout optimization framework on scenario specified in 'test1_mp.csv'.

```
from run_mp import *
G, f = run_mp_scenario_file(scenario_code='test1_mp', plot=True)
```

3.3. Accessing Scenario Ouputs and Results

The two scenario running functions presented in Section 3.2 return two objects: (1) a NetworkX DiGraph object, `G` and (2) a list of Plotly Figure objects, `f`.[4]

3.3.1. Printing Scenario Results

Scenario results are stored across multiple Python dict objects within `G` (some of which contain other dict objects within them). More specifically, these are stored in three different locations within `G`, which can be accessed as shown in Code Block 4Code Block 6. Examples some specific data users can access is shown in Code Block 7. Using standard Python methods, users can write the results from specific dictionaries to any desired output files (e.g., json, csv, or txt files) for further analysis.

Code Block 4 - Method for accessing graph-level scenario results.

```
G.graph
# G.graph.keys()  # provides the dict keys for reference
```

Code Block 5 - Method for accessing node-level scenario results (for the 11th node in the list of nodes). Line 3 shows how a user would access the keys of the Python dict object that is returned.

```
node_list = list(G.nodes())
G.nodes[node_list[10]]
# G.nodes[node_list[10]].keys()  # provides the dict keys for reference
```

[4] NetworkX Graph objects are useful for storing large networks with attribute information at the node, edge, and graph levels. The Python package documentation is available at https://networkx.org/documentation/stable/.

Code Block 6 - Method for accessing edge-level scenario results (for the 5th edge in the list of edges). Line 3 shows how a user would access the keys of the Python dict object that is returned.

```
edge_list = list(G.edges())
G.edges[edge_list[4]]
# G.edges[edge_list[4]].keys()  # provides the dict keys for reference
```

Code Block 7 - Examples of valid dictionary keys for accessing results stored in G.

```
print(G.graph['operations']['deployment_perc']['TOTAL'])
print(G.nodes[node_list[10]]['facility'])
print(G.edges[edge_list[4]]['battery_avg_kwh']['TOTAL'])
```

3.3.2. Plotting Scenario Results

Scenario visualizations are stored in the various plots contained in `f` (the returned list of generated Plotly Figure objects). Each of the plots are interactive and contain additional scenario information that can be seen by hovering one's mouse over different plot components. The plots can be displayed by indexing a specific element of the list `f`, as shown in Code Block 8. The static framework list of plots contains 6 plots, while the dynamic rollout framework list of plots contain 2 plots. The 6 plots generated for the static framework scenario are (in order as stored in `f`): network map, scenario summary table, emissions plot, levelized cost of operation plot, deployment percentage pie chart, and cost of avoided emissions table. For the dynamic rollout framework results, the 2 plots produced are: dynamic network animation and composite rollout network plot.

Code Block 8 - Example of how to display a particular plot from list of returned Plotly Figures.

```
f[0]  # plots the network map for static framework results
```

3.4. Testing Code Package

Prior to specifying their own scenarios, users are encouraged to utilize the provided test scenario files in the "input>scenario" subdirectory to test the framework's installation on their local machine and to familiarize themselves with the framework's inputs and outputs. Samples to verify the framework is running properly have been provided for the static optimization framework. Users can follow the example in Code Block 9 to verify the functionality of the program on their local machine.

Code Block 9 - Example of how to use the provided cached scenario results to test for proper program functionality.

```
from run import *
# run the scenario locally
G, f = run_scenario_file(scenario_code='test1', plot=True,
load_scenario=False, cache_scenario=False)

# load the provided cached file corresponding to this scenario
G_check, f_check = run_scenario_file(scenario_code='test1_check',
plot=True, load_scenario=True, cache_scenario=False)

# check if the deployment percent of the locally generated instance
#  matches the deployment percent of the results loaded from the
#  corresponding cached file provided with the repository
print(G.graph['operations']['deployment_perc']['TOTAL'])
print(G_check.graph['operations']['deployment_perc']['TOTAL'])

# can also plot each of the figure objects to check if networks match
# f[0]
# f_check[0]
```

3.5. Altering Scenario Data

Users can also edit, alter, or completely substitute the provided data files that are used by the underlying optimization and scenario evaluation models. As discussed in Section 2.1, the data files containing the required input parameter values are stored in the subdirectories contained in the "input" directory. Any edits made to these files must conform to the exact file structure as the original (any files that are substiritued must match the exact file format, have the same data types, and contain the same name as the original file it is replacing). A summary of the "input" subdirectories and the important files they contain users can edit is provided below:

- "commodity": subdirectory for commodity-specific data
  - "commodity_energy_ratios.csv": file containing commodity-specific energy intensity ratios
- "facility": subdirectory for facility-specific data
  - "facility_info_test1_mp.csv": file containing facility-specific data for use in the dynamic rollout facility location and flow selection optimization model. This is only relevant for dynamic rollout scenarios. The filename must be included in the "facility_info_filename.csv" field of the scenario input filename for a specific scenario.
- "flow": subdirectory for freight flow data. Conatains files with freight flow data broken down by railroad, commodity, time window, and forecasted year (if applicable).
- "general": subdirectory containing general parameter data.
  - "constants.csv": contains information on several scientific constants—should not be changed.
  - "fuel_tech_efficiency_factor.csv": contains information on fuel technology efficiency factors, relative to diesel locomotives.
  - "hybrid_energy_intensities.csv": contains information from hybrid simulation results on resulting diesel and battery energy intensities for a particular assumed train configuration.
  - "SPLC_station_master.csv": contains a legend to map geographic coordinates to their neares SPLC railyard identifier. Can be used to process and tie carload waybill data to GIS rail networks.
- "LCA": subdirectory contains life-cycle analysis parameter estimates for different energy technologies. All files are for different energy technologies and different time periods. File edits/replacements must preserve the data structure completeness of the previous file.

- “matrices”: subdirectory contains matrices generated by the optimization program used to facilitate future optimization runs. This directory, and the files within it are not critical inputs.
- “networks”: subdirectory contains the rail network data files. These files are stored as “.pkl” files to preserve the data structures.
    - Users that wish to provide their own network data should do so by loading a GIS shapefile of their choice into a NetworkX Graph object with all the necessary data to match the structure of the provided sample netwokrs. The file name should follow the convention of “<railroad name>” followed by “_geo_graph_simplified.pkl”.
- “TEA”: subdirectory contains techno-economic analysis parameter estimates for different energy technologies. All files are for different energy technologies and different time periods. File edits/replacements must preserve the data structure completeness of the previous file.

Template Revision History

Date	Version	Description	Author
December 2022	1.0	Initial Version	Northwestern University Transporation Center and Argonne National Laboratory
March 2024	2.0	Updated Version including hybrid diesel-battery locomotive technologies and a rollout optimization framework	Northwestern University Transporation Center and Argonne National Laboratory

This dashboard has been developed by Northwestern University Transportation Center (NUTC: Hani S. Mahmassani, Pablo Durango-Cohen, Adrian Hernandez, Max Ng) and Argonne National Laboratory (ANL: Amgad Elgowainy, Michael Wang, Joann Zhou, Nazib Siddique) as part of the LOwering CO2: Models to Optimize Train Infrastructure, Vehicles, and Energy Storage (LOCOMOTIVES) project funded by the Advanced Research Projects Agency - Energy (ARPA-E) under the United States Department of Energy (USDOE).

---

[i] URL: https://drive.google.com/file/d/1GxRHQ3dJDNbQLOKKMjsPROvc0wGu8Ph9/view?usp=share_link
[ii] URL: https://www.transportation.northwestern.edu/research/featured-reports/locomotives.html
[iii] max.ng@u.northwestern.edu (Max Ng), AdrianHernandez2025@u.northwestern.edu (Adrian Hernandez)
[iv] URL: https://github.com/NUTransport/NUFRIEND
[v] Insturctions for setting up Gurobi for Python can found at https://support.gurobi.com/hc/en-us/articles/360044290292-How-do-I-install-Gurobi-for-Python.
[vi] A Gurobi License can be obtained at https://www.gurobi.com/downloads/