Repeatability Evaluation Package (REP) Documentation for the Paper

"Learning Spatio-Temporal Aggregations for Large-Scale Capacity Expansion Problems"

Authors: Rahman Khorramfar, Aron Brenner, Saurabh Amin

1 Overview

The methodology presented in the paper consists of two parts. The first part is the autoencoder (AE) procedure explained in Section 4, and the second part is the capacity expansion model (i.e., GTEP) presented in Section 2. Therefore, the REP consists of two main folder: 1)'*AE-Module'* folder which contains all the data and codes used in the AE; 2) '*GTEP-Module'* folder which contains all the data and codes used in the GTEP. All the data used are from publicly available sources, hence our submission contains no proprietary data.
Each successful run involves at most two steps. In the first step the AE module is run to obtain spatio-temporal aggregations for the network size and number of representative days. The second step uses the output from the first step to run the GTEP for the given parameters. Both modules are implemented in Python with transferability and legibility in mind. The details of both steps is given in the Section 3 and 4 of this document.

The codes and data are all available in the GitHub repository at:
https://github.com/RahmanKhorramfar91/ICCPS-2023

2 System and Resource Requirements

All the codes are developed and tested in Python 3.8. The codes do not need installation as they are presented in the raw script format. The following packages are required for a successful run:

```
numpy, pandas, matplotlib, random, geopp, os, gurobipy, time, sys, pytorch,
torch_geometric, torch_sparse, torch_scatter, sklearn, scikit-learn-extra,
networkx, tqdm, datetime
```

Note that the systems must have Anaconda as wells as Gurobi solver installed. We run the AE module on a personal MacBook laptop, and run the GTEP module on the MIT Supercloud system (Intel Xeon Platinum 8260 processor with up to 48 cores and 192 GB of RAM) with multiple instances running in parallel. However, both modules are successfully tested on Windows machines. All the packages are generic Python packages, so we expect a smooth run on other machines with recent version of Windows, Linux, or Mac.

3 How to Run

This section explains the running procedure for both modules. The first module, which is AE, usually runs under 4 hours depending on the host machine and aggregation parameters. The run time of GTEP greatly varies based on the spatio-temporal aggregation level as well as problem parameters, but they are usually run under 10 hours.

3.1 Autoencoder

As mentioned, all code and data related to the autoencoder are available in the folder '*AE-Module'*. The only code that needs to be run can be found in `Autoencoder.ipynb`, and all data is given in the '*Data'* folder.

To train the spatial and temporal aggregation autoencoders, the Jupyter notebook `Autoencoder.ipynb` can be run cell by cell. The data can be uploaded and processed by running the cells under the section "*Constructing Datasets*." Specifically, the electricity load data is given in "`Data/Power Network Topology-full network (188 nodes)/bus_load_RM_2050.csv`", the NG data is given in "`Data/ng_daily_load2050_RM.csv`", the wind CF data is given in "`Data/wind-CF-188-nodes.csv`", and the solar CF data is given in "`Data/solar-CF-188-nodes.csv`".

To train the spatial aggregation autoencoder, one should run all cells in the section titled '*Spatial Aggregation Autoencoder*'. The parameters/variables to be tuned are as follows:

- Number of spatial clusters: the final spatial resolution (number of nodes) of the GTEP model. This tunes the graph pooling block in the autoencoder.
- Learning rate (*lr*): the learning rate for the Adam optimizer. This should be tuned to minimize the autoencoder validation loss.
- Epochs: the number of iterations of gradient descent during training. This should also be tuned to minimize the autoencoder validation loss.

The code then saves the learned spatial clusters to the file '`spatial_cluster.csv'`. This file has two columns:

- Node: the node label.
- Cluster: the assigned cluster of the corresponding node, or in other words, the node label to which it is absorbed in the spatially aggregated GTEP.

Similarly, to train the temporal aggregation autoencoder, one should run all cells before the Training subsection under '*Spatial Aggregation Autoencoder'* as well as all cells in the section titled '*Temporal Aggregation Autoencoder'*. The parameters/variables to be tuned are as follows:

- Days: the number of representative days to be used in the temporally aggregated GTEP. This is a hyperparameter for the K-medoids algorithm.

- Learning rate (*lr*): the learning rate for the Adam optimizer. This should be tuned to minimize the autoencoder validation loss.
- Epochs: the number of iterations of gradient descent during training. This should also be tuned to minimize the autoencoder validation loss.
- *alpha_G, alpha_W, alpha_S*: the objective weight coefficients corresponding to NG, wind CF, and solar CF reconstruction loss (relative to the electricity reconstruction loss).
- *max_iter*: the number of iterations for the K-medoids algorithm.

The code then saves the learned representative days to the file `temporal_cluster.csv`. This file has three columns:

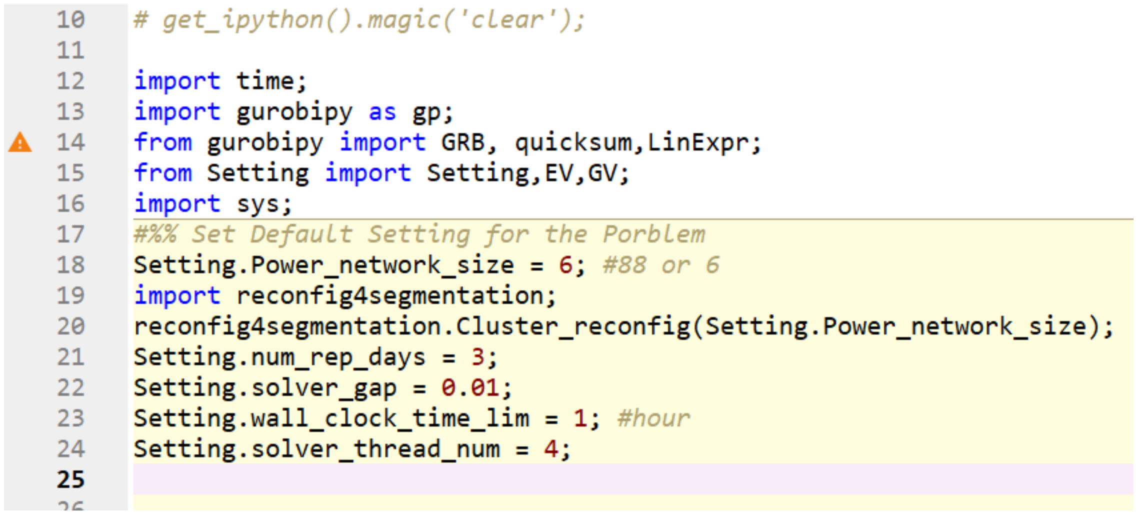
- Day of Year: number corresponding to the day of the year (e.g., 0 corresponds to January 1)
- Date
- Weight: the relative weight of the representative day in the temporally aggregated GTEP. This is given by the number of members of the cluster as learned by the K-medoids algorithm.

3.2 GTEP

Once the Autoencoder codes are run, the output is stored in '*joint_CF_with_extreme_days'* folder insider the 'GTEP-Module' folder. The 'GTEP-Module' folder contains several folders, Python scripts, and CSV file from which we only explain the files that are directly related to running the GTEP model. The GTEP model runs from `UB.py` script. The code can both be run from the Anaconda's command window or an IDE such as Spyder. The scrips require 5 key parameter as follows:

- Network size: the number of nodes in the power system. The parameters should set to one 6, 10, 15, or 20. Default value is 6.
- Number of representative days: the number of planning days considered in the model. The parameters can take any integer number between 2 and 30. Default value is 3.
- Solver Gap: the MIP gap for Gurobi to terminate. The parameters is any number between 0 and 1. Default value is 0.01.
- Solver Time Limit in terms of hours: the solution time limit for the solver. The parameter can take any integer number. Default value is 1.
- Solver Thread: the number of CPU core threads dedicated to the solver. The parameters takes integer numbers and can vary between 1 and the number of thread on the machine. Default is 4.

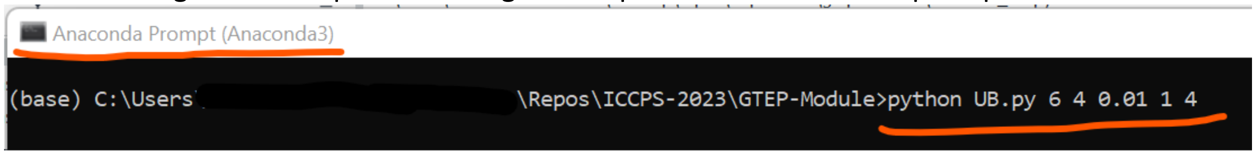
The following shows these parameters in the `UB.py` opened in Spyder:



```
# get_ipython().magic('clear');

import time;
import gurobipy as gp;
from gurobipy import GRB, quicksum,LinExpr;
from Setting import Setting,EV,GV;
import sys;
#%% Set Default Setting for the Porblem
Setting.Power_network_size = 6; #88 or 6
import reconfig4segmentation;
reconfig4segmentation.Cluster_reconfig(Setting.Power_network_size);
Setting.num_rep_days = 3;
Setting.solver_gap = 0.01;
Setting.wall_clock_time_lim = 1; #hour
Setting.solver_thread_num = 4;
```

The following is an example of running the script from the anaconda prompt:



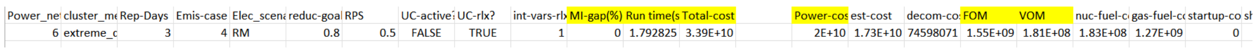
```
(base) C:\Users[redacted]\Repos\ICCPS-2023\GTEP-Module>python UB.py 6 4 0.01 1 4
```

The parameters after `UB.py` should be separated by a comma and are in the order they were explained above.

The results of each run is stored in the output file `JPoNG_Results.csv`. The code reports a detailed output in 87 columns and two rows. The first row is the header for each column and the other column is the value for each column. The initial columns show the parameters of the instance. Column K and L show MIP gap and run time of the problem in seconds. Other columns provide details of the solution and are named mnemonically for readability. The following columns used in the paper:

- Column N (Total-cost): total cost of the GTEP model including the total cost of power and NG systems
- Column O (*Power-cost*): total cost of the power system
- Column P (*est-cost*): power plant establishment cost
- Column R (*FOM*): fixed operating and maintenance cost for the power system
- Column S (*VOM*): variable operating and maintenance cost for the power system
- Column AH (*NG-cost*): total cost of the NG system

The following figure shows part of the solution output for an instance:



Power_ne	cluster_me	Rep-Days	Emis-case	Elec_scena	reduc-goal	RPS	UC-active?	UC-rlx?	int-vars-rlx	MI-gap(%)	Run time(s	Total-cost		Power-cos	est-cost	decom-co	FOM	VOM	nuc-fuel-c	gas-fuel-c	startup-co	sh
6	extreme_c	3	4	RM	0.8	0.5	FALSE	TRUE	1	0	1.792825	3.39E+10		2E+10	1.73E+10	74598071	1.55E+09	1.81E+08	1.83E+08	1.27E+09	0	

All computational experiments reported in the paper can be repeated by running the GTEP model with different parameters. The submitted code can also be run for many other instances with certain configurations.