e-HIGHWAY 2050

Modular Development Plan of the Pan-European Transmission System 2050

Contract number	308908	Instrument	Collaborative Project
Start date	1st of September 2012	Duration	40 months
WP 8	Enhanced Pan-European Transmission Planning Methodology		
D 8.4.b	Prototype for optimal modular plan to reach 2050 grid architectures		

Revision: [1.3 FINAL]

Due date of delivery: [January 2014]

		Date & Visa
Written by	Sergeï AGAPOFF, RTE	15/12/2014
Checked by	Leif WARLAND, SINTEF Patrick PANCIATICI, WP8 Leader Camille PACHE, RTE Jean MAEGHT, RTE	06/01/2015
Validated by	Brahim BETRAOUI, RTE Gérald SANCHIS, Coordinator	12/02/2015

Project co-funded by the European Commission within the Seventh Framework Programme		
Dissemination Level		
PU	Public	X
PP	Restricted to other programme participants (including the Commission Services)	
RE	Restricted to a group specified by the consortium (including the Commission Services)	
CO	Confidential, only for members of the consortium (including the Commission Services)	

Document information

General purpose

This document explains the tools and file formats developed for the task WP8.4 of e-Highway2050 project. Those explanations focus on the information structure and the actual running of the tools. The definition of the models can be found in deliverable D8.4.a "Enhanced methodology to define the optimal modular plan".

Change status

Revision	Date	Changes description	Authors
V0.0	24/11/2014	Initial version	S. AGAPOFF
V1.0	15/12/2014	First draft	S. AGAPOFF
V1.1	07/01/2015	Typographic corrections Candidate Selection I/O diagram modified Data file formats updated	S. AGAPOFF
V1.2	09/01/2015	Typographic corrections	S. AGAPOFF
V1.3	16/01/2015	Added “Parameters” section	S. AGAPOFF

EXECUTIVE SUMMARY

This document describes the prototypes proposed to perform the modules defined in the deliverable D8.4.a "Enhanced methodology to define the optimal modular plan".

Three modules were developed for the purpose of "Enhanced methodology to define the optimal modular plan": Snapshot Selection, Candidate Selection and Transmission Expansion Planning (TEP) optimization.

The three modules developed to achieve task 8.4's goal have been implemented in AMPL, Python and BASH. A folder structure has been proposed to handle the data and the different processes used by the modules. Finally a script has been written to easily run the whole methodology.

TABLE OF CONTENT

List of figures

List of tables

INTRODUCTION

This document describes the prototypes proposed to perform the methodology defined in the deliverable D8.4.a "Enhanced methodology to define the optimal modular plan".

Three modules were developed for the purpose of "Enhanced methodology to define the optimal modular plan": Snapshot Selection, Candidate Selection and Transmission Expansion Planning (TEP) optimization. We first present the general concepts and common structures and then explain the three modules and how they should be performed. In the last section, the formats of input files are presented.

1. Generalities

1.1. Platform, Languages and Data Format

The delivered scripts should be run on a Linux environment. CentOS release 6.5 was installed on the machine used for tests.

Three languages are used to implement the methodology: Python, AMPL and Bash.

Language	Version	Main use	Extension of runnable files
Python	2.7.8	Data Manipulation Clustering method	.py
AMPL	20131213	Optimizations	.run
BASH	4.1.2(1)-release	Control Python and AMPL when both are used for a single method	.sh

Several packages should be added to Python:
- Numpy
- Scipy
- Scikit-learn (scikit-learn.org)

All input and internal data is read in "space separated" text files. The intermediate output data (snapshots, candidates) is provided in that same format and the main outputs (TEP optimization) are provided in "semi-colon separated" csv files.

1.2. Scripts and Data structure

Since three modules were developed in this task we used a common folder structure for the data and tools files. Figure 1 describes how those folders are organized in the "Project" folder. In this file tree all folders, except "Project" and "Case", should be named as it is written in the figure.

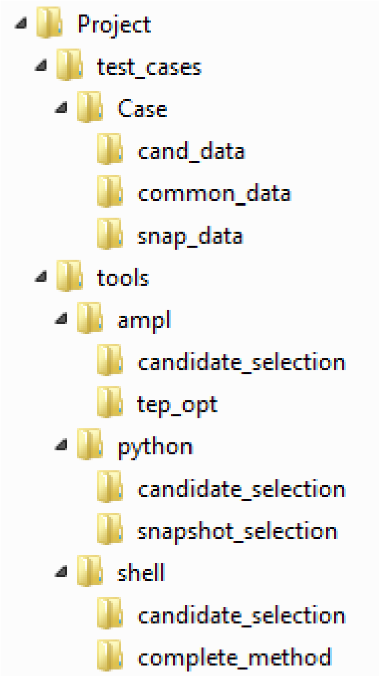

Figure 1 - General folder structure

In this entire document we consider that the test case is "Case" and that *the current directory is always the "Case" folder*.

As expected, all data files are found in the test_cases subfolder and all scripts in the tools subfolder. In each test case folder ("Case") the data folders and files should be initialized as follows:

- common_data
  - corr_char.txt *Characteristics of corridors in the existing grid*
  - corr_types.txt *Corridor Types*
  - inj_char.txt *Injections Characteristics*
  - inv_maximum.txt *Investment Maximum*
  - list_country.txt *List of the studied countries*
  - scenarios.txt *Scenarios information*
  - years.txt *Time-horizons information*
  - zones.txt *Zones characteristics*
- snap_data
  - input (/horizon/scenario/MCyear/week/)
    - snap_char.txt *Snapshots info (demand, production injections in each zones)*
    - snap_char_UD.txt *Updated snapshots info (after DCOPF on the zonal grid)*
    - zonal_price.txt *Zonal Prices for each snapshot (after DCOPF on the zonal grid)*

The purpose of the main data files will be explained in the following sections. Some files can be designated by the same name while being located in different folders (for example input and output files): when it happens it is clearly specified in the input/output diagrams of each process.

Most of those files are organized as tables of data: the specific formats are detailed in Section 5.

The tools are organized as follows:

- ampl
    - candidate_selection
        - candidate_analysis.run
        - candidate_management.run
        - common.dat
        - common.mod
    - tep_opt
        - inv_opex.dat
        - inv_opex.mod
        - inv_opex.run
- python
    - candidate_selection
        - generate_feasible_cand.py
        - merge_candidates.py
        - unique_potential_cand.py
    - snapshot_selection
        - accl_method.py
        - feature_construction.py
        - kmeans_method.py
        - kmedoids_method.py
        - method_analysis.py
        - snapshot_selection.py
        - solution_analysis.py
- shell
    - candidate_selection
        - candidate_selection.sh
    - complete_method
        - complete_method.sh

The **whole methodology** can be performed thanks to the "complete_method.sh" script. It assumes that the features have already been constructed (see Section 2.2) and performs the three modules consequently. The following command which specifies the feature (e.g. F1b) and the number of clusters (e.g. 20) to be used in the Snapshot Selection should be run:

```
$ ../../tools/shell/complete_method/complete_method.sh F1b 20
```

The next sections describe each module developed for this methodology.

2. Snapshot Selection

2.1. Objective

The goal of this module is to find a given number of representative snapshots among all the available grid simulations.

The main inputs to this process are the snap_char_UD and zonal_price files. Those files are time-series describing the system's behaviour at the zonal level (for each week of each Monte-Carlo year of each scenario in each time-horizon) and have been computed by running DCOPFs on the aggregated nodal injections from adequacy without grid:

- **snap_char_UD.txt** is an update of snap_char.txt (aggregation of the nodal injections from adequacy simulations): aggregated demand and generation injections are updated in each zone and for each hour of the week in order to comply with the initial zonal grid constraints.
- **zonal_price.txt** describes the Local Marginal Price of energy in each zone for each hour of the week.

The outputs from this module are two files (snapshots.txt and snap_char.txt) describing all the selected snapshots (their related horizon, scenario and weight) and their characteristics (injections in each zone).

Two steps are needed for this module:

- **feature_construction.py**: Compute the clustering features: the demand, generation and price values are combined to obtain the different features (see D8.4a Section 2.2)
- **snapshot_selection.py**: Cluster the snapshots based on a specific feature: use the chosen feature to assess the distance between snapshots and cluster them in the desired number of similarity groups.

The following figure shows the inputs and outputs for the features computation. The indexes i and v are used to describe the different features (see D8.4a Section 2.2)

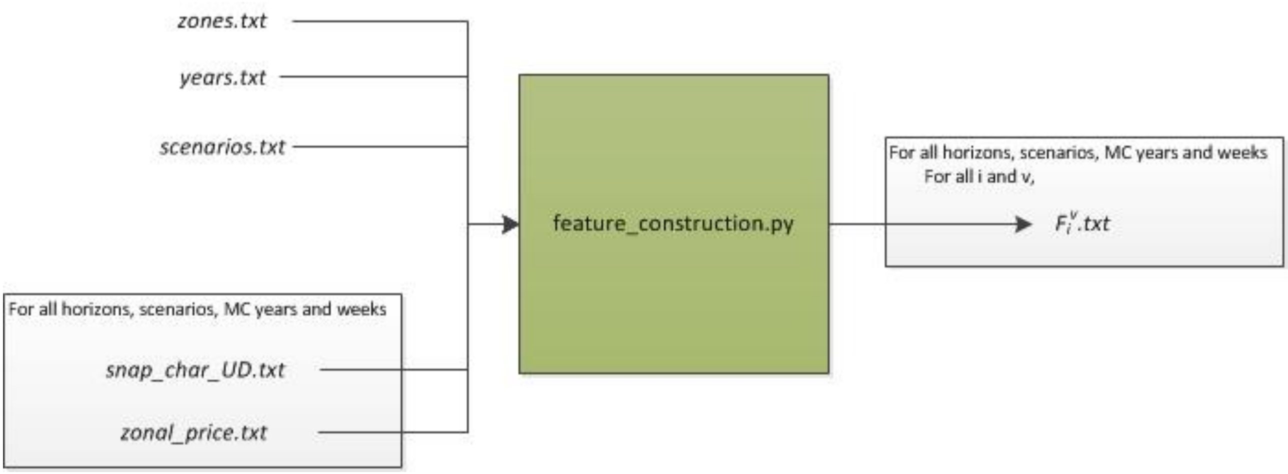

Figure 2 - Input/Output diagram for feature_construction.py

The following figure shows how the second step (the actual selection) is performed (in this diagram, F_i^v is the chosen feature).

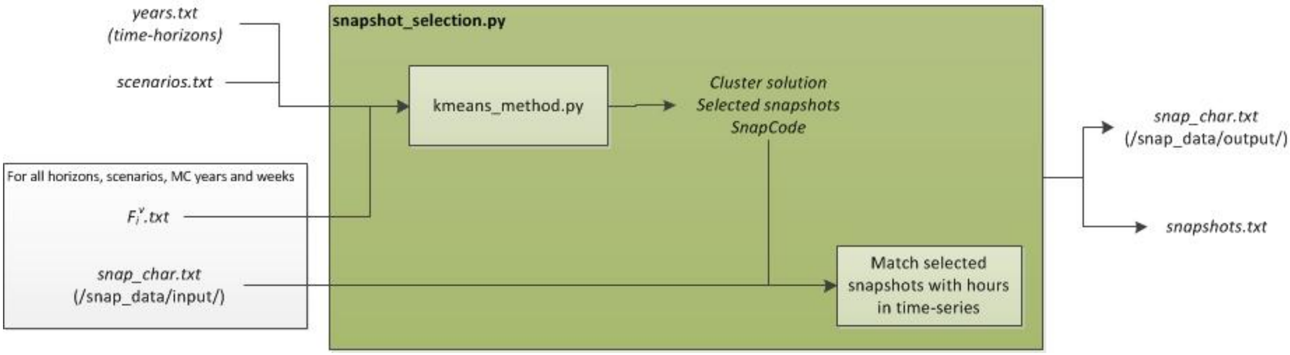

Figure 3 - Input/Output diagram for snapshot_selection.py

2.2. *Commands and description of outputs*

Run the feature construction script

```
$ python ../../tools/python/snapshot_selection/feature_construction.py
```

The outputs (one text file for each feature) are as follows:

- **F_i^v.txt**: for each hour of a given week, it gives the value of the feature for each item (a zone, a pair of zones or a statistical entity such as minimum, maximum etc.)

They are stored in the following folders (for all h in horizons, s in scenarios, MCy in Monte-Carlo years and w in weeks):

snap_data/input/**h**/**s**/**MCy**/**w**/features/

Run the snapshot selection script by specifying:

- The feature (e.g. F1a)
- The number of clusters (e.g. 10)

```
$ python ../../tools/python/snapshot_selection/snapshot_selection.py F1a 10
```

The outputs are two files (in snap_data/output):

- **snapshots.txt**: list of the selected snapshots (representatives of the non-empty clusters) with their related time-horizon and scenario and their weight.
- **snap_char.txt**: system's behaviour (in adequacy simulations) for each selected snapshot (extracted from the initial snap_char file found in snap_data/input/)

Another output (**snapsel_raw.txt**) is provided by this script: it gives raw results of the Snapshot Selection and can be used to monitor the performances of the module. For each horizon and each scenario, the information it contains is:

- cluster_solution: assignment of the snapshots to clusters (designated by a number)
- selected_snapshots: list of the selected snapshots and the size of the related cluster

- distortion: total quantization error (sum over the clusters of intra-cluster distances to the mean)
- stable: boolean for the final state of the snapshot selection loop (True/False)
- snap_code: matching table between snapshots indexes and Monte-Carlo years, week and hour

Once the features have been constructed, there is no need to run the script feature_construction.py again if the data does not change.

All the available Monte-Carlo years and weeks are used for the selection.

3. Candidate Selection

3.1. Objective

The goal of this module is to find relevant candidates for the TEP optimization among all the available candidates. This module is performed through several processes in Python and AMPL controlled by a shell script (**candidate_selection.sh**):

- **generate_feasible_cand.py**: for each pair of zones, each corridor type is used to generate a new candidate
- **candidate_management.run**: successively assesses the profitability of the candidates, optimizes the expansion of the most profitable ones (relaxed TEP) and installs them in the grid
- **unique_potential_cand.py**: cleans the output of the candidate management (groups expansions corresponding to the same candidate and makes sure that profitable candidates appear only once)
- **candidate_analysis.run**: tests the installation of candidates that have been identified as "profitable" but not installed in the Candidate Management (DC power flow to assess the impact of such an installation).
- **merge_candidates.py**: the previous processes are performed for each scenario and time-horizon, this process merges all the obtained candidate pools and adds the complementary and substitute candidates identified in the Candidate Analysis.

For the Candidate Management, a dynamic description of the grid is needed: the file "corr_char.txt" is copied from the common_data folder into the cand_data/input/ folder and is named "init_corr_char.txt". This file is used as the initial grid and a "corr_char.txt" file in cand_data/input/ is used to successively install the optimal expansion of profitable candidates.

Figure 4 shows how the sub-processes are organized. In this figure we only show how the "candidates.txt" file is produced. The Candidate Selection also generates a list of combined candidates (to switch a corridor from the existing grid to another type of corridor): this list gives the information about which candidates (from the candidates.txt file) should be simultaneously optimized. This file is generated by the generate_candidates.py script and used by the other processes to find out if such combined candidates are profitable, installed and complementary or substitute. We do not represent this "combined_candidates.txt" file in the figure for the sake of simplicity.

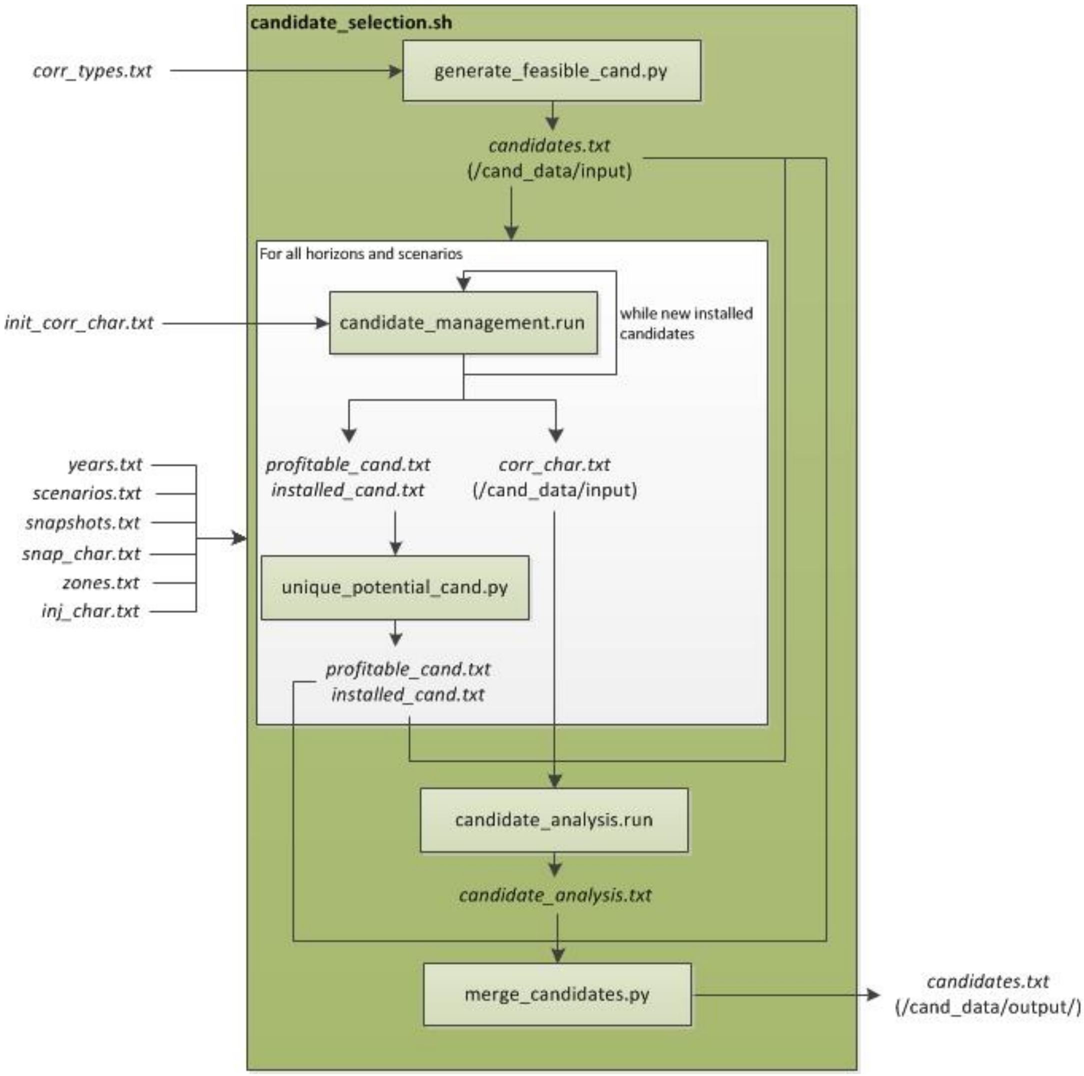

Figure 4 - Input/Output diagram for candidate_selection.sh

3.2. Command and description of outputs

Run the candidate selection script

```
$ ../../tools/shell/candidate_selection/candidate_selection.sh
```

The intermediate files created by the different sub-processes are stored in cand_data/input/. The actual outputs of the method (in cand_data/output/) are:

- **candidates.txt**: list of selected candidates with their characteristics (end zones, type of corridor, capacity, reactance per unit of length and maximal number of increments).
- **combined_candidates.txt**: list of pairs of candidates to be optimized simultaneously (to switch a corridor from the existing grid to another type of corridor)

4. Transmission Expansion Planning (TEP) optimization

4.1. Objective

The goal of this module is to optimize the expansion of the previously selected candidates by minimizing the total investment cost and the operational consequences (deviation from adequacy simulation on the selected snapshots). One script is needed for this module:

- **inv_opex.run**: it optimizes the expansion of the candidates over the different scenarios and time-horizons, with a common development for the first time-horizons.

The following figure shows the inputs and outputs of this module.

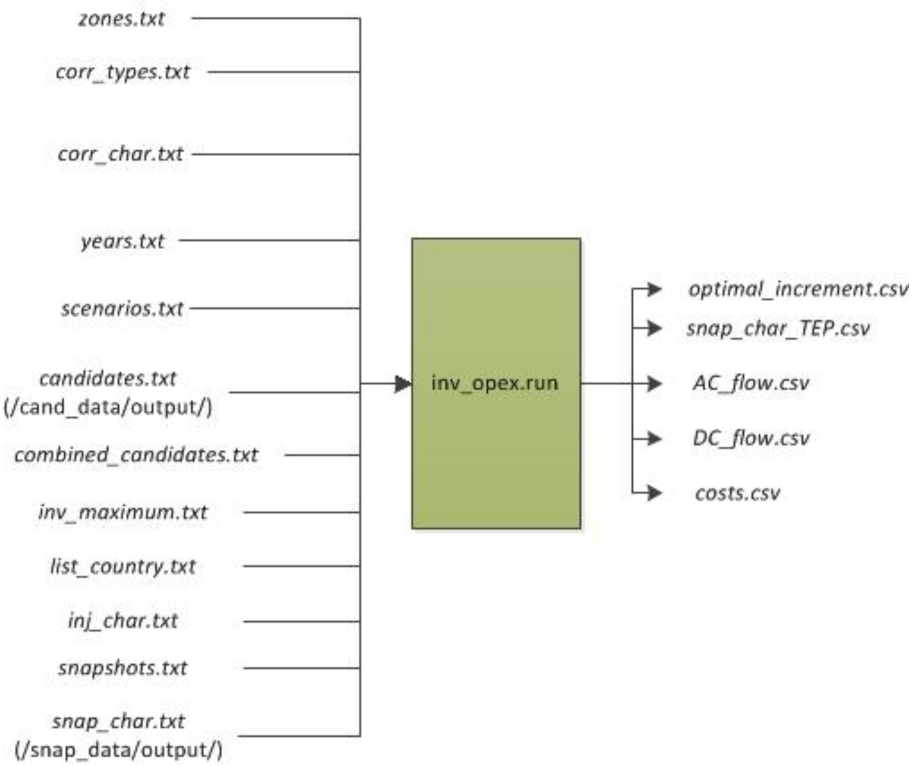

Figure 5 - Input/Output diagram for inv_opex.run

4.2. Command and description of outputs

Run the TEP optimization script

```
$ ampl ../../tools/ampl/tep_opt/inv_opex.run
```

Output files are stored in the current directory ("Case") and are named as follows:

- **optimal_increment.csv**: expansion solution for each candidate in each scenario and time-horizon
- **snap_char_TEP.csv**: update of the injections in all the considered snapshots
- **AC_flow.csv**: values of the AC flows in each snapshot

- **DC_flow.csv**: values of the AC flows in each snapshot
- **costs.csv**: detailed costs of the expansion plan (Investment, OPEX, OM)

5. Input data files

In this section we give the precise format of the input files. We suggest writing the columns' names in the header of the file (beginning with the hash character "#"): it is not needed for the computation but helps reading and understanding the files. The following line is an example of header for file "corr_char.txt":

```
#CORRIDORS CorrZoneA CorrZoneB CorrType InitCap InitX PstNum
```

Usually the first column of such files is the index of the entity it describes. Most columns' names have been chosen so that they are self-explanatory, if not (and especially for columns involving a literal value) we explain what kind of data is expected.

corr_char.txt

Item	CORRIDORS	CorrZoneA	CorrZoneB	CorrType	InitCap	InitX	PstNum
Type	Integer	Integer	Integer	Integer	Float	Float	Integer
Unit	/	/	/	/	MW	p.u.	/

- PstNum is the number of the PST linked to the corridor

corr_types.txt

Item	CORRTYPES	Tech	OM	Inv	Cap	...
Type	Integer	Literal	Float	Float	Float	
Unit	/	/	€/(km.year)	€/km	MW	

Item	...	X_u	LifeTime	MaxLength	NegType	SwitchTo
Type		Float	Integer	Float	Integer	Integer
Unit		p.u./km	years	km	/	/

- Tech indicates if the corridor type refers to AC or DC technology
- NegType is the type of which this specific time is the "negative" counterpart (-1 if the type is a regular corridor type) see D8.4.a Section 3.3 for more details
- SwitchTo is the type to which corridors of the considered type can be switched (-1 if switch is not allowed for this type) see D8.4.a Section 3.3 for more details

inj_char.txt

Item	INJ	Sign	Ctrl
Type	Literal	Integer	Binary
Unit	/	/	/

- INJ is the name of the injection type

- Sign is +1 (production) or -1 (load)
- Ctrl (0/1) indicates the controllability of the injection (this is only used for feature construction and not operation of the system)

inv_maximum.txt

Item	SCENARIOS	HORIZONS	InvMax
Type	Integer	Integer	Float
Unit	/	/	€

- SCENARIOS is the number designating the scenario
- HORIZONS is the year designating the time-horizon
- InvMax is the maximum investment allowed in each time horizon and scenario

list_country.txt

Item	COUNTRY
Type	Literal
Unit	/

- COUNTRY indicates the name of the country or area to be considered

scenarios.txt

Item	SCENARIOS	Ws
Type	Integer	Float
Unit	/	/

- SCENARIOS is the number designating the scenario
- Ws is the weight of the scenario

snap_char.txt (and snap_char_UD.txt)

Item	SNAPSHOTS	ZONES	INJ	InjRef	InjMin	InjMax	MCp	MCn
Type	Integer	Integer	Literal	Float	Float	Float	Float	Float
Unit	/	/	/	MW	MW	MW	€/MW	€/MW

- INJ is the name of the considered injection
- InjRef is the reference zonal injection from adequacy simulations
- MCp/n are variation costs of injections, up and down respectively

years.txt

Item	YEARS	CommonDev
Type	Integer	Binary
Unit	/	/

- YEARS is the year to be considered (e.g. 2020) [starting year has to be included]
- CommonDev indicates if the TEP should be common for the considered year

zonal_prices.txt

Item	SNAPSHOTS	1	... i ...	n
Type	Integer	Float	Float	Float
Unit	/	€	€	€

- $i \in \{1, \ldots n\}$ is the number designating zone i.
- In column i the values indicate the Local Marginal Price of energy in zone i for each snapshot

zones.txt

Item	ZONES	ZoneSub	CoordX	CoordY	ZoneCountry	Swing
Type	Integer	Integer	Float	Float	Literal	Binary
Unit	/	/	km	km	/	/

- ZoneSub is the substation number (nodal grid) corresponding to the zone number (zonal grid); it is unused in the considered tools (see other WP8 tasks for explanation)
- ZoneCountry indicates in which country/area the zone is located

6. Parameters

For the different modules, parameters are needed and can be modified in the different files used for the processes.

The available parameters are defined in Table I:

- The first unit-less parameters (from $\boldsymbol{\tau}$ to $\boldsymbol{\sigma}_-$) should be given a value in $[0;1]$
- L_{ref} should be strictly positive (if $\boldsymbol{\sigma}_+ = \boldsymbol{\sigma}_- = 0$ the Length Reference is not used and any value greater than 0 can be given)
- $MaxInc$ is an integer
- $SwitchLen$ can be 0 if needed.

Table I - Parameters

Parameter	*Description*	*Unit*	*File(s) to modify*
$\boldsymbol{\tau}$	Discount rate	Ø	ampl/tep_opt/inv_opex.run ampl/candidate_selection/common.mod
$\boldsymbol{StableAssign_{threshold}}$	Cluster assignment is considered stable when the similarity between two successive clustering solutions is higher than this threshold	Ø	python/snapshot_selection/kmeans_method.py
$\boldsymbol{FlowVar_{threshold}}$	In the Candidate Analysis, complementary and substitute candidates are only considered if they change the flow of more than 50% in at least one of the identified candidates	Ø	python/candidate_selection/merge_candidates.py
$\boldsymbol{\sigma}_+$	Positive penalization (Architecture focus)	Ø	ampl/tep_opt/inv_opex.mod
$\boldsymbol{\sigma}_-$	Negative penalization (Architecture focus)	Ø	ampl/tep_opt/inv_opex.mod
$\boldsymbol{L_{ref}}$	Length Reference (Architecture focus)	km	ampl/tep_opt/inv_opex.mod
$\boldsymbol{S_n}$	Nominal Power (fixed in other parts of the WP8 methodology)	MW	ampl/candidate_selection/common.mod ampl/tep_opt/inv_opex.run
$\boldsymbol{MaxInc}$	Maximal Number of units by candidate	Ø	python/candidate_selection/generate_feasible_cand.py
$\boldsymbol{SwitchLen}$	Length above which a switch is automatically proposed	km	python/candidate_selection/generate_feasible_cand.py

7. Conclusion

The three modules developed to achieve task 8.4's goal have been implemented in AMPL, Python and BASH. A folder structure has been proposed to handle the data and the different processes used by the modules.

We presented the data and scripts needed for those processes. Instructions were also given to run the whole methodology as well as each module individually.

The presented modules are ready for integration and can be automatically run. Parallelization has not been achieved: it can be done on a high level by scheduling independent parts of the modules to run simultaneously (for example the Candidate Selection is performed independently for each time-horizon and each scenario, it could be done in parallel with a common run to merge the different outputs).