Final Technical Report

DOE EERE Award Number: DE-OE0000885

Project Title:

A Probability-based Model for Cost-effective Integration of Renewables into the Electricity Grid

Name of Recipient

Virginia Polytechnic Institute and State University

Principal Investigator

Dr. Saifur Rahman, Professor and Director
Virginia Tech - Advanced Research Institute

September 2019

Acknowledgment:

This material is based upon work supported by the Department of Energy under Award Number DE-OE0000885.

Disclaimer:

EXECUTIVE SUMMARY

Funded by the U.S. Department of Energy during 2017-2019, the resource adequacy evaluation and production costing tool**,** called MATPLAN **(MAT**lab and probability-based **PLAN**ning), was developed. MATPLAN can take into account the variable nature of renewable energy sources (both solar PV and wind farms), rather than considering deterministic data as in the current practice. It combines both probabilistic and optimization techniques to allow the determination of optimal system expansion policy and perform effective load carrying capacity (ELCC) analysis of a system with high renewable penetration. This is in contrast to the current practice that treats these variable sources as negative loads. This tool expects to benefit system planners by enabling them to consider renewable energy sources as options for their expansion plans. System operators can use MATPLAN to calculate the capacity credit for each solar/wind farm using probabilistic forecasts, thus enabling quantification of the operating reserve requirements to cope with uncertainty and variability of solar/wind output committed in the market.

Aiming at implementing the functionality and flexibility to deal with various production costing problems, MATPLAN comprises the following key modules:

- **LOAD-CALC module** – is the load input module that requires system load during the planning horizon (e.g., 30 years) as the input. The major goal for the LOAD-CALC module is to generate the equivalent load duration curve (ELDC) of an existing power system, considering the forced outage of all generators, if any.
- **EXIST-GEN module –** consists of two sub-modules, EXIST-TH and EXIST-RE, that model the characteristics of existing conventional and renewable power plants. It requires the number of additions or retirements of each type generator and the year of such changes.
- **CANDI-GEN module –** consists of two sub-modules, CANDI-TH and CANDI-RE, that model the characteristics of candidate conventional and renewable power plants. Reserve margin requirements are also specified in this module through a JSON format file, CANDI-GEN.json.
- **CONFIG module** – is the expansion configuration module that generates additional constraints for the optimization problem in the OPTIMIZE module. It narrows down possible expansion configurations based on constraints inputted by a user. Required input data include, for example, minimum/maximum reserve requirements, minimum/maximum numbers of a particular expansion candidate that can be installed in a given year, and the acceptable LOLP in a given study period. The CONFIG module is used to generate all valid configurations over the planning horizon.
- **OPTIMIZE module** – is the optimization module that combines probabilistic and optimization techniques to determine the optimal system expansion policy based on inputs defined in other modules. The objective function has been formulated to take into account probabilistic model of generator availability, including renewable energy sources. This module also takes care of all economic calculation and cash flow analysis.
- **ELCC module** – is effective load carrying capability module responsible for calculating ELCC value of each power plant, including renewables, selected as expansion candidates. The output

of this model is the capacity credit of each new unit as the capacity added to a system, allowing the load to increase without compromising the generation adequacy.

These six modules when working together allows the user to consider renewable energy as options for expansion planning using a probability-based model. They can also handle different types of user-defined input data formats. The developed tool has been validated through comparison with another similar software package and further tested using realistic field data. The complete code repository has been released under the open-source license for public access.

The latest MATPLAN source code, together with its implementation and developer guides, have been made available on an online repository at https://github.com/wasp2019/MATPLAN.

Table of Content

List of Figures

List of Tables

1.0 Summary of Tasks Proposed and Accomplishments

This section summarizes project tasks based on the statement of project objective (SOPO) and the overall accomplishments, as well as the success toward meeting the project technical requirements.

1.1 Tasks Proposed

This project was divided into three phases, where Phase 1 (September 2017-September 2018) involved all the module development; Phase 2 (October 2018-March 2019) involved validation and software enhancement; and Phase 3 (April 2019-September 2019) involved case studies for functionality tests and result evaluation.

Figure 1-1 summarizes all tasks of this project, which is the excerpt from the SOPO as proposed by the Virginia Tech team.

PHASE 1: Develop the proposed expansion planning tool

Task 1: Develop the LOAD-CALC module
The key LOAD-CALC capability is to convert the hourly load input into an equivalent load duration curve (ELDC) seen by each possible generator for each study period. This can be derived by first, constructing the Load Duration Curve (LDC) based on the hourly forecasted load input data.

Task 2: Develop the EXIST-TH and CANDI-TH modules
Under this task, both EXIST-TH and CANDI-TH modules will be developed. Both modules will take user inputs about thermal and hydro units (e.g., capacity of existing power plants, fuel types, minimum and maximum capacity, FOR and maintenance requirements) and generates output files that summarize characteristics of these plants. While EXIST-TH focuses on existing power generating units, CANDI-TH focuses on candidates for power system expansion during the planning horizon. These modules also generate constraints for thermal and hydro units for use in the optimization module, such as: rating capacity of power plants that limit their generation output, a minimum generation requirement that indicates, if a plant is in operation, it has to operate at least a certain percentage of its installed capacity, etc.

Task 3: Develop the EXIST-RE and CANDI-RE modules
The EXIST-RE module will be developed takes user inputs about renewable energy units (e.g., the hourly output data, either historical or forecasted, FOR and maintenance requirements). They generate output files that summarize characteristics of existing renewable plants. While EXIST-RE focuses on existing renewable units, CANDI-RE focuses on renewable candidates for power system expansion during the planning horizon. These modules also generate constraints for renewable generating units for use in the optimization module, such as the limit of their generation output, depending on historical/forecasted data for each period.

Task 4: Develop the CONFIG module
Under this task, the CONFIG module will be developed to translate additional system-related inputs to additional constraints for the optimization module. These constrains are for example, the required reserve during each study period; the annual emission cap; and the reliability requirement of the system (LOLP). With these constraints, it allows the OPTIMIZE module to only consider allowable system configurations or possible expansion configurations based on constraint inputs form a user.

Task 5. Develop the OPTIMIZE module
Under this task, the OPTIMIZE module will be developed that combines probabilistic and optimization techniques to determine the optimal system expansion policy based on inputs defined in other modules. All economic

calculations are included, such as present-value calculations and escalation of fuel prices.

Task 6. Develop the ELCC module
The ELCC module is responsible to calculate ELCC of each existing and candidate power plants. This module will determine the ELCC of a power plant by measuring the contribution of an individual generator to system capacity with and without the generator of interest. This method can be explained as follows: first, take a note of the original system installed capacity (MW) and LOLP from the OPTIMIZE module as benchmark values. Second, remove the generator of interest and rerun the OPTIMIZE module with a new constraint to keep the original LOLP. Then, take a note of the revised system installed capacity (MW). The difference between the benchmark capacity and the revised system capacity is the ELCC of the generator in question. The module is rerun to obtain ELCC of all power plants.

PHASE 2: Validate the proposed tool
Task 7. Validate the proposed tool with a well-known expansion planning software, like WASP
Under this task, we will compare the resulting optimal solution of the proposed tool with that of a well-known expansion planning software, like WASP. This will allow us to verify the validity of the proposed tool for the expansion planning with traditional power plants, like thermal and hydro. However, the model's applicability to integrate renewables will need another step (task 8) for validation**.**

Task 8. Validate the proposed tool based on uncertainty parameters in the forecast
Under this task, the team will develop several possible solutions based on the confidence intervals of solar/wind forecasts while meeting all constraints. Solutions will be listed with confidence intervals explicitly stated to be used by ISOs/RTOs and marketers depending on their risk appetite.

PHASE 3: Run a case study based on a real-world data
Task 9. Run a case study
Once the tool is validated, it will be run using field data to showcase its applicability in a real-world environment. The team will have access to real-world data for our case studies from our industry partners, as indicated in the letters of support.

Figure 1-1. Proposed tasks according to the SOPO

1.2 Accomplishments

Accomplishments are summarized by Task, as shown in the table below.

Table 1-1 Summary of accomplishments

Tasks	Summary of accomplishments	Details in
Task 1: Develop the LOAD-CALC module	LOAD-CALC module was developed and tested individually with adopting LDC data.	Sections 3.0
Task 2: Develop the EXIST-TH and CANDI-TH modules	EXIST-TH and CANDI-TH sub-modules were developed and combined with EXIST-RE and CANDI-RE respectively.	Section 4.0 and 5.0
Task 3: Develop the EXIST-RE and CANDI-RE modules	EXIST-RE and CANDI-RE sub-modules were developed and combined with EXIST-TH and CANDI-TH respectively.	Section 4.0 and 5.0
Task 4: Develop the CONFIG module	CONFIG module was developed and verified individually with outputs of CONGEN functionality of WASP.	Sections 6.0
Task 5. Develop the OPTIMIZE module	OPTIMIZE module was developed.	Section 7.0

Task 6. Develop the ELCC module	ELCC module was developed.	Section 8.0
Task 7. Validate the proposed tool with a well-known expansion planning software, like WASP	Validation of the developed tool, MATPLAN, was performed by comparison with the WASP package.	Section 9.0
Task 8. Validate the proposed tool based on uncertainty parameters in the forecast	Validation of the developed tool, MATPLAN, considering uncertainty and probabilistic model of renewable energy sources, was performed.	Section 5.0 and 9.0
Task 9. Run a case study	Case studies were performed under different scenarios: (1) low penetration rate of renewable energy sources, (2) high penetration rate of renewable energy sources, (3) multi-location of renewable energy sources, and (4) different year-by-year cost structures of solar panels.	Section 9.0
Task 14: Project Management and Reporting (all Phases)	• Quarterly progress reports were submitted every quarter. • Peer review progresses were presented in June 2018 and June 2019. • Final report was submitted in September 2019 (This report).	This document

2.0 Introduction to MATPLAN

2.1 What is MATPLAN?

MATPLAN stands for **MAT**lab and probability-based **PLAN**ning tool that was developed for resource adequacy evaluation and production costing. MATPLAN was engineered to provide lightweight and flexible modules to deal with generation expansion planning (GEP) problems with consideration of renewable energy sources. MATPLAN offered: scalability, modularization, flexibility, interoperability, as well as parallel computing capability. Built on the widely used MATLAB, MATPLAN's source codes are available for public access.

2.2 MATPLAN Key Features

MATPLAN offers the following key features:

- **Flexibility** – MATPLAN can account for different types of thermal generator plants, as well as the variable nature of renewable energy sources (both solar PV and wind farms). It combines both probabilistic and optimization techniques to allow the determination of optimal system expansion policy and is in contrast to the current practice that treats these variable sources as negative loads. MATPLAN can also handle different input data formats, different cost structures of renewable energy sources, different time-division manners, and multiple renewable locations.
- **Modularization** – MATPLAN comprises six modules to reduce its computation complexity. These modules can work either coordinately or independently to provide useful intermediate results for expansion planning. This kind of modularization design allows users to reorganize and modify the decoupled modules for their own implementation purpose.
- **Scalability** – MATPLAN can handle up to 8760 time-division intervals, thousands of generation expansion planning configuration candidates and long-term planning time horizon (e.g., 15-30 years). If powered by its optional parallel computing capability, MATPLAN can handle even larger case studies with a large number of existing power generating units and candidates and longer-term planning horizon.
- **Source code available for public access** – MATPLAN's source codes – built on top of the widely used MATLAB – have been made available for public access.
- **Parallel Computing Capability** – MATPLAN provides an optional parallel computing version that allows the users to leverage the MATLAB Parallel Computing Toolbox ™ for higher computational efficiency and speed up the calculation process. By using this feature, MATPLAN can handle larger-scale study cases without compromise for the running time.
- **Support from the Advisory Committee** – MATPLAN was developed in consultation with an advisory committee from the beginning of the project. MATPLAN advisory committee comprised representatives from the following organizations: Policy and Economic Studies MISO, Dominion Virginia Power (DOM), Utility Variable-generation Integration Group, Operational Analytics NaturEner, and Renewable Energy Integration California ISO. See Figure 2-1.

Figure 2-1 Organizations represented by WASP advisory committee

2.3 Description of MATPLAN Modules

MATPLAN has been designed to perform power system expansion planning considering renewable energy sources using a probability-based model. Its software architecture is modular and is similar to that of a well-known generation expansion planning tool, *Wien Automation System Planning* (WASP). The MATPLAN architecture is depicted in Figure 2-2, comprising six modules - LOAD-CALC, EXIST-GEN (including EXIST-TH and EXIST-RE), CANDI-GEN (including CANDI-TH and CANDI-RE), CONFIG, OPTIMIZE and ELCC. These modules when working together allow the user an ability to consider renewable energy sources as options for expansion planning using a probability-based model. LOAD-CALC collects electrical load information at the system level and builds equivalent load duration curves. EXIST-GEN consists of two sub-modules, EXIST-TH and EXIST-RE, that model the characteristics of existing conventional and renewable power plants, respectively. CANDI-GEN consists of two sub-modules, CANDI-TH and CANDI-RE, that model the characteristics of candidate conventional and renewable power plants, respectively. CONFIG generates all candidate configurations (i.e., mix of power plants) with consideration of various constraints. OPTIMIZE generates the optimal solution or the most cost-effective generation expansion plan. ELCC calculates capacity credit of each scheduled conventional and renewable power plant. Their detailed functionalities are introduced individually in subsequent Chapters.

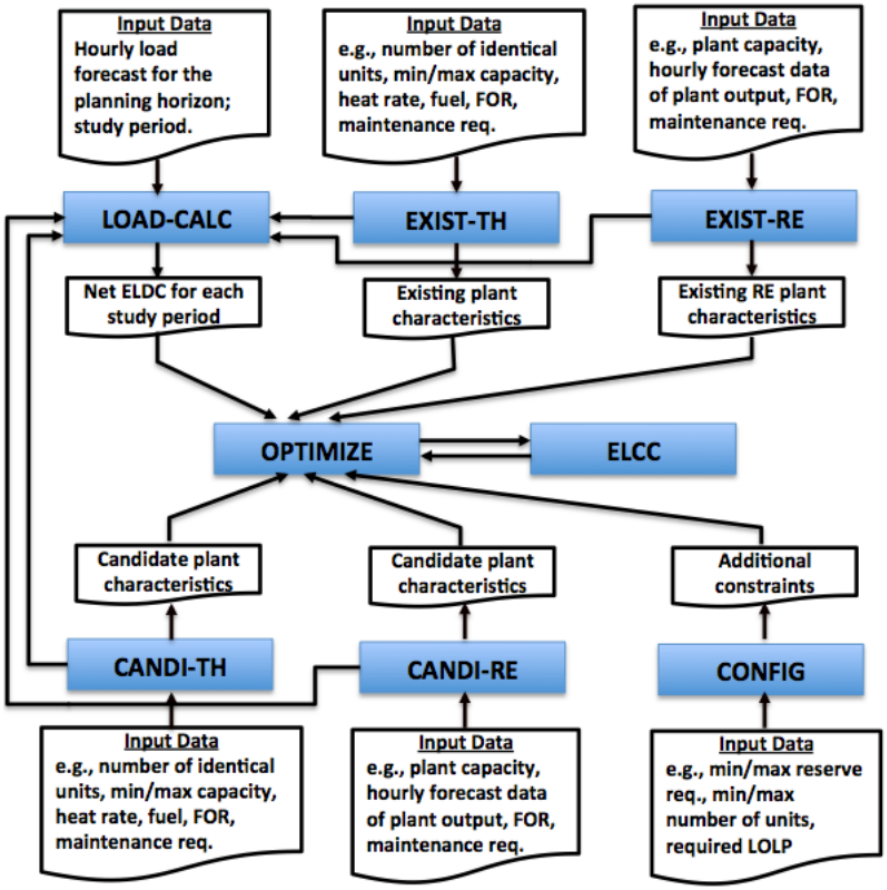


Figure 2-2 MATPLAN architecture with different modules

2.4 Summary Description of MATPLAN Computer Code

MATPLAN consists of various computer files that are responsible for different functionalities. All of these modules should run sequentially to obtain the optimal expansion planning configurations and ELCC values. The successful implementation of selected modules may depend on the outputs of other modules and user-defined input data specified in setting files. The overall relationship of MATPLAB computer code files is depicted in Figure 2-3.

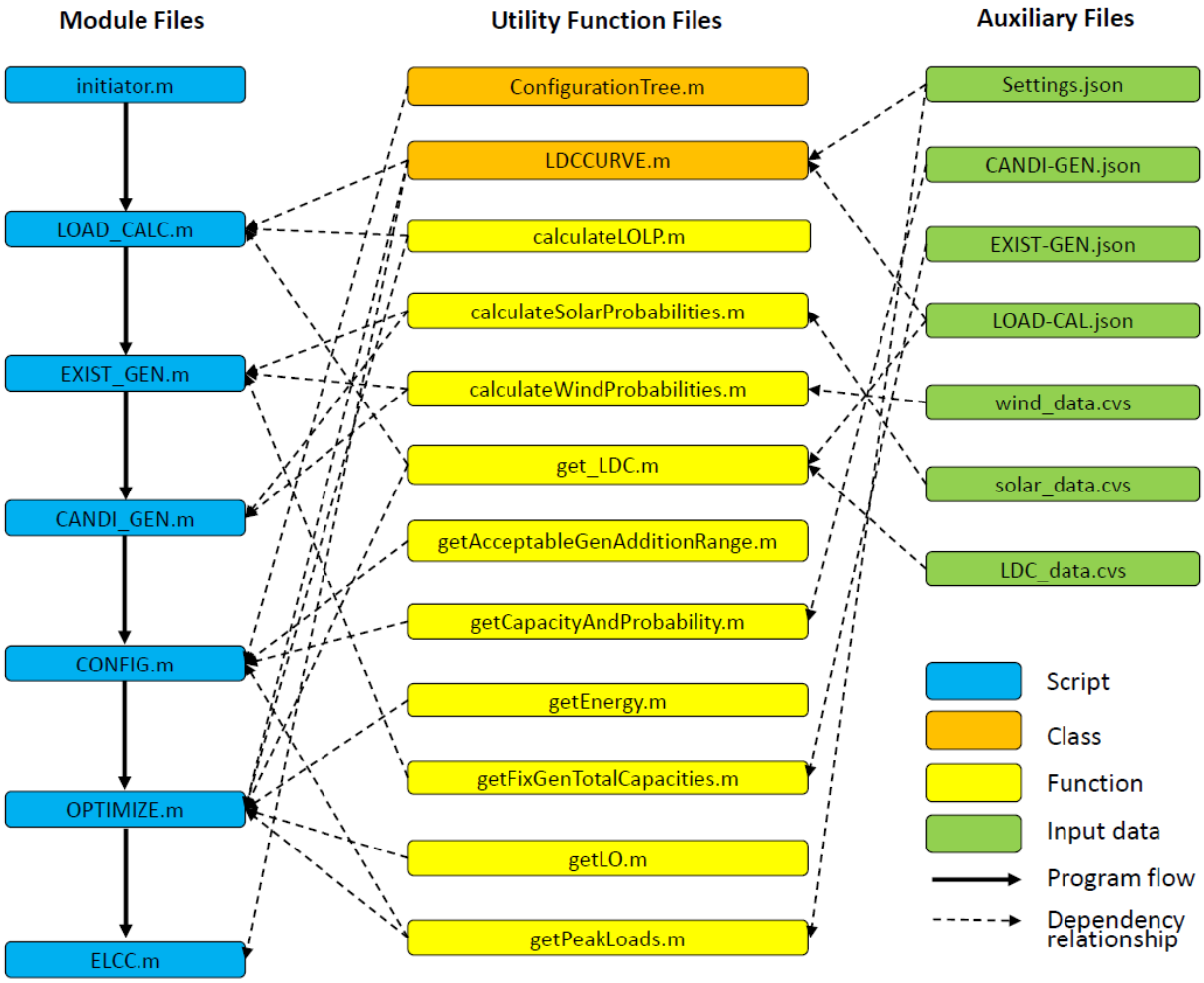


Figure 2-3 The overall relationship of MATPLAB computer code files

3.0 LOAD-CALC Module

The key LOAD-CALC capability is to convert the hourly load input into an equivalent load duration curve (ELDC) seen by each possible generator for each study period. This can be derived by first, constructing the Load Duration Curve (LDC) based on the hourly forecasted load input data.

3.1 Overview of LOAD-CALC Module

LOAD-CALC is the load input module that requires power demand during the planning horizon (e.g., 30 years) as the input. While WASP allows the maximum of 12 study periods in a year, the developed tool allows up to 8,760 study periods per year (hourly simulation for the whole year). This enables the study to be granular taking into account variations in renewable energy output. Forecasted load data from commercially or publically available software can be used as an input. The major goal for the LOAD-CALC module is to generate the equivalent load duration curve (ELDC) of an existing power system, considering the forced outage of all generators, if any. Three types of inputs are needed to determine ELDC: 1) original hourly load curve/data; 2) generator information such as thermal plant capacity and forced outage rate (FOR); 3) generation profiles of wind or solar plants and penetration rate. Similar to the LOADSY module in WASP, the LOAD-CALC module transforms the user inputs into the desired data format, and generates system load profiles for future usage. The LOAD-CALC module has been developed in MATLAB, called 'LOAD_CALC.m'.

3.2 Inputs of LOAD-CALC Module

The LOAD-CALC module accepts user inputs in a JSON format. The inputs of LOAD-CALC module have been designed to include:

- Planning start year
- Planning end year
- Peak load of each year
- One of the three types of data representing the system load duration curve

A user can specify the above information in the path of 'root/projects/project_1/user_input/LOAD-CALC.json'. An example of this file is demonstrated in Figure 3-1.

```
{
    "start": [2020],
    "end": [2035],
    "peak": [6000, 6333, 6725.65, 7109.01, 7496.45, 7897.51, 8304.23,
        8702.83, 9120.57, 9558.36, 10017.2, 10488, 10980.9, 11497, 12025.9, 12579.1],
    "ldc_data": {
        "0": {
                    "year": [2020],
                    "type": [0],
                    "path":"projects/project_1/user_input/data/dom_2016.csv"
          },
        "1": {
                    "year": [2025],
                    "type": [1],
                    "path":"projects/project_1/user_input/data/ldc_points.csv"
          },
        "2": {
                    "year": [2030],
                    "type": [2],
                    "path":"projects/project_1/user_input/data/ldc_coefficients.csv"
         }
}
```

Figure 3-1 Example of input to the LOAD-CALC module

In the example above:

- "*start*" and "*end*" specify the start and end years of the planning horizon.
  - In this example, the planning horizon is from 2020 to 2035.

- "peak" specifies the annual peak load during the planning horizon in MW.
  - In this example, the annual peak load in years 2020, 2021,…, 2035 are 6000MW, 6333MW,…, 12,579.1MW, respectively.

- "*ldc_data*" specifies load duration curves (LDC) during the planning horizon.
  - In this example, three sets of "ldc_data" inputs are provided.
    - ldc_data(0) specifies the load of years 2020-2024 that are of input `"type": [0]` and `"path":"projects/project_1/user_input/data/dom_2016.csv"`
    - ldc_data(1) specifies the load of years 2025-2029 that are of input `"type": [1]` and `"path":"projects/project_1/user_input/data/ldc_points.csv"`
    - ldc_data(2) specifies the load of years 2030-2035 that are of input `"type": [2]` and `"path":"projects/project_1/user_input/data/ldc_coefficients.csv"`
  - Notice that, the "ldc_data" can be provided according to a "year" range based on different input types.
  - The "ldc_data" has three input types [0], [1] and [2], which can be explained as follows: `"type": [0]` - indicates that a user provides 'hourly load data' as the input. The input file must be in CSV file format, and its data have the dimension of 365*24. Each row contains the load data (in MW) of each day in a year (from Jan 1 to Dec 31), and each column contains the load data (MW) of each hour (from 00:00 to 23:00) in a day. These input data (i.e., hourly load) is sorted in the LOAD-CALC module to create the inverted load duration curve. In this case, the user is allowed to directly provide the hourly load profile as the input to the software without having to preprocess the data.

`"type": [1]` - indicates that a user provides 'points representation of load duration curve' as the input. This input is provided as a set of two-column data, specifying any number of points on the LDC with X-axis as load and Y-axis as duration (in a CSV file). See Figure 3-2(left). This is similar to the WASP standard "points" input for LDC, as shown in Figure 3-2(right).

Type [1] input example:

Load	Duration
0.4	1
0.4138	0.98
0.424	0.96
…	…
0.9964	0.001
1	0

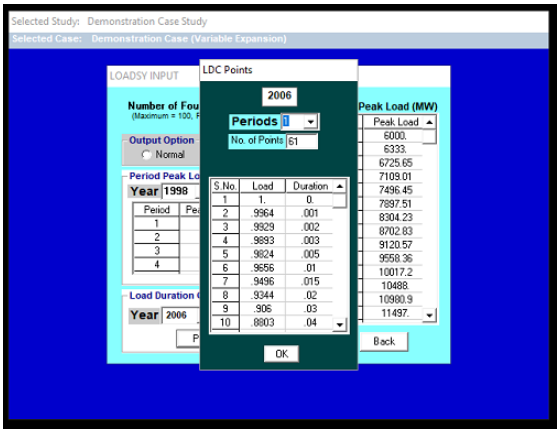


Figure 3-2 Type [1] input (left) and WASP standard "points" input for LDC (right)

`"type": [2] -` indicates that a user provides 'coefficient representation of load duration curve' as the input. This input is provided as a single row of data with 6 elements (A0, A1, A2, A3, A4 and A5), representing the 5th order polynomial fitting of the load duration curve (in a CSV file). The first column is the coefficient of the intercept and the last column is the coefficient of the 5th order component. See Figure 3-3(left). This is similar to the WASP standard "coefficients" input to represent LDC using the 5th order polynomial expression, as shown in Figure 3-3(right).

Type [2] input example:

A0	A1	A2	A3	A4	A5
1	-3.6	16.6	-36.8	36	-12.8

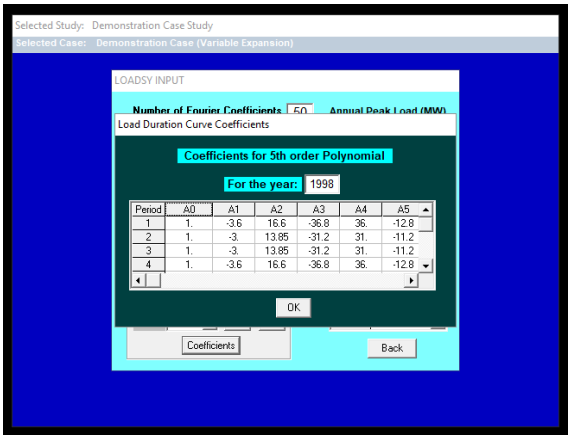


Figure 3-3 Type [2] input (left) and WASP standard "coefficients" input for LDC (right)

The input file format is required to be in the CSV format. These raw input files can be anywhere in the computer system as long as the paths are correctly given, however, it is a good practice to organize them in the project data folder:

```
projects/project_1/user_input/data/
```

3.3 Outputs of LOAD-CALC Module

The developed LOAD-CALC module has been tested with publically available system load data (from PJM metered load data (URL: http://www.pjm.com/markets-and-operations/ops-analysis/historical-load-data.aspx). Figure 3-4 illustrates the normalized inverted load duration curves using Types [0], [1] and [2] inputs.

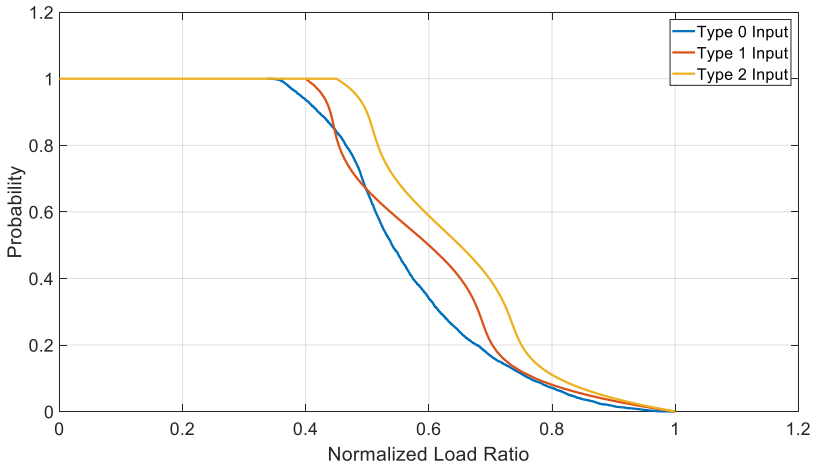


Figure 3-4 Normalized inverted LDC plots from three types of user inputs

By executing the 'LOAD-CALC.m' module, the inverted normalized LDC data are ready to use in the 'ldc_data' variable.

Outputs of the LOAD-CALC module:

In addition to the inverted normalized LDC, outputs of the LOAD-CALC module include:

- Planning period: an array of all years in the planning horizon, e.g., [2020, 2021, …, 2034, 2035].
- Annual peak load of each year in array
- LDC Year: the years when the type of LDC profile changes, e.g., [2020, 2025, 2030] in the example above.
- LDC data: the X-Y points depicting the normalized inverted load duration curves, one for each LDC year. This variable is a struct in MATLAB, according to the specified number of LDC years.

The variables above are used in other modules, and are saved in *structure* variable system_info and a '.mat' file under the path of *'root/projects/project_1/project_data/LOAD_CALC.mat'*. The variable names are 'study year', 'annual_load_peak', 'ldc_year' and 'ldc_data', respectively. The structure of LDC data information stored in the structure variable system_info is depicted in Figure 3-5.

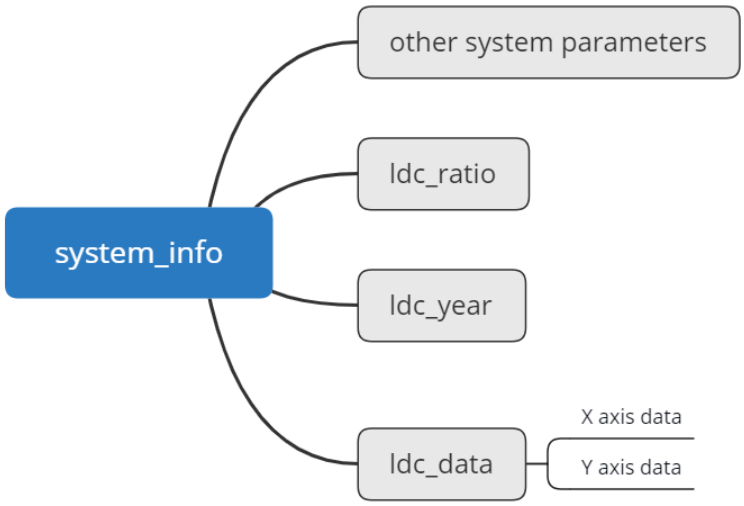


Figure 3-5 The structure of LDC data information

Designed verification for user input in case of invalid configuration:
User input validation is also developed for checking the dimension mismatch. For example, the code example below checks if the number of years in the planning horizon matches with the number of annual load peak provided. When a dimension mismatch is detected, the program is suspended and an error message is displayed.

```
if length(study_year) ~= length(annual_load_peak)
    error('Year number and annual load peak configuration do not match!')
end
```

4.0 EXIST-GEN Module

EXIST-GEN module is responsible for collecting user-defined input data and various types of parameters to characterize the existing power plants (both thermal plants and renewables).

4.1 Overview of EXIST-GEN Module

EXIST-GEN consists of two sub-modules, EXIST-TH and EXIST-RE, that model the characteristics of existing conventional and renewable power plants. It requires the number of additions or retirements of each type generator and the year of such changes. The characteristics of conventional generating units are kept in the EXIST-TH sub-module. In the EXIST-RE sub-module, on the other hand, a renewable energy unit is also defined with some parameters modified. For instance, a renewable energy unit usually has no heat rate and zero fuel cost, as well as a different operation and maintenance (O&M) cost structure.

4.2 Input of EXIST-TH Submodule

Inputs of the EXIST-TH:

The inputs of EXIST-TH module have been designed to include:

- Generator type code
- Unit number of such generator type
- Parameters used to describe such generator type
- Existing addition and retirement plan for the planning horizon

A user can specify these input parameters in a JSON format file in the path of *'root/projects/project_1/user_input/EXIST-GEN.json'*. An example of this file is demonstrated below in Figure 4-1. These six different types of generators are from the WASP demonstration case study.

```
[
  {"type": "thermal",
  "code": "FLG1",
  "unit_number": 4,
  "forced_outage_rate":10,
  "capacity": [270, 270, 270, 270, 270, 270, 270, 270],
  "para": [150, 270, 1, 3300, 2850, 10, 10, 56, 270, 600, 0, 4.06, 4.9, 1800,
2.5, 1.0],
  "existing_plan": {"number": [-1, -1], "year": [2003, 2014]}},

 {"type": "thermal",
  "code": "FLG2",
  "unit_number": 9,
  "forced_outage_rate":8.9,
  "capacity": [276, 276, 276, 276, 276, 276, 276, 276],
  "para": [150, 276, 2, 2900, 2550, 10, 8.9, 56, 280, 495, 0, 1.91, 2.0,
1800, 2.5, 1.0],
  "existing_plan": {"number": [-1, -1, -1], "year": [2006, 2009, 2014]}},

 {"type": "thermal",
  "code": "FCOA",
  "unit_number": 1,
```

```
  "forced_outage_rate":8.0,
  "capacity": [580, 580, 580, 580, 580, 580, 580, 580],
  "para": [400, 580, 3, 2800, 2300, 10, 8.0, 48, 600, 800, 0, 2.92, 5.0,
6000, 1.0, 2.0],
  "existing_plan": {"number": [1], "year": [1999]}},

 {"type": "thermal",
  "code": "FOIL",
  "unit_number": 7,
  "forced_outage_rate":7.3,
  "capacity": [145, 145, 145, 145, 145, 145, 145, 145],
  "para": [80, 145, 4, 2450, 2150, 10, 7.3, 42, 140, 0, 833, 4.57, 1.6,
10000, 1.0, 3.0],
  "existing_plan": {"number": [-1, -1], "year": [2012, 2013]}},

 {"type": "thermal",
  "code": "F-GT",
  "unit_number": 4,
  "forced_outage_rate":6.0,
  "capacity": [50, 50, 50, 50, 50, 50, 50, 50],
  "para": [50, 50, 5, 3300, 3300, 0, 6.0, 42, 50, 420, 0, 8.35, 1.6, 10000,
0.5, 0.5],
  "existing_plan": {"number": [-1], "year": [2009]}},

 {"type": "thermal",
  "code": "F-CC",
  "unit_number": 1,
  "forced_outage_rate":15,
  "capacity": [174, 174, 174, 174, 174, 174, 174, 174],
  "para": [87, 174, 6, 2048, 2048, 0, 15, 28, 180, 0, 1266, 2.1, 5.0, 11000,
0.0, 0.5],
  "existing_plan": {"number": [1, 1], "year": [2000, 2001]},
  }
]
```

Figure 4-1 Example of input to the EXIST-TH module

In the example above, the user specifies six types of generators. For each generator type, the following information is specified:

- "code": specifies the generator code, which can be any string. This example uses "FLG1", "FLG2", "FCOA", "FOIL", "F-GT" and "F-CC" to represent different types of generators.
- "unit number" specifies the total number of a specific generator type in the system;
- "para" specifies parameters describing the generator, as shown in Table 4-1. These parameters are always follow the same order of sequence in variable matrix for keeping the physical meaning.

Table 4-1 Parameters of thermal plants required for EXIST-TH

Plant's Names/Codes			
Fuels' Types/Names			
	Number of units		
Plant's features	Category	Variable	Example
	Electric-related	Minimal operating power of a unit (MW)	400
		Maximal generating power of a unit (MW)	580
	Cost-related	Fuel costs ($/million kcals)	800
		Fixed O&M costs (in $/kW Month)	2.92
		Variable O&M costs (in $/MWh)	5.00
	Reliability-related	Forced outage rate	8%
		Spinning reserve (% of unit capacity)	10
		Maintenance class size (MW)	600
		Scheduled maintenance days of a year	48
	Heat-related	Heat rate at minimal operating level (kcal/kWh)	2800
		Average incremental heat rate (kcal/kWh)	2300
		Heat value of the fuel used (kcal/kg)	6000
	Polution-related	Pollutant I (SO2) emission (% weight of fuel)	1
		Pollutant II (NOx) emission (% weight of fuel)	2

- "existing_plan" has two attributes: number and year, with a negative number representing a unit retirement (for existing thermal power plants) and positive number representing a power plant addition (for the planned addition of thermal power plants). In the example, two out of four 'FLG1' units are retired in years 2023 and 2024.

Tested the module with data from the WASP demo case study:

The generator information in the WASP demonstration case study are used for testing the developed EXIST-TH module. Inputs are provided in the '*EXIST-GEN.json*' file shown in Figure 4-1. By executing the '*EXIST_GEN.m*' module, all 26 generators' capacities and forced outage rates are read, as shown in Figures 4-2 and 4-3, respectively.

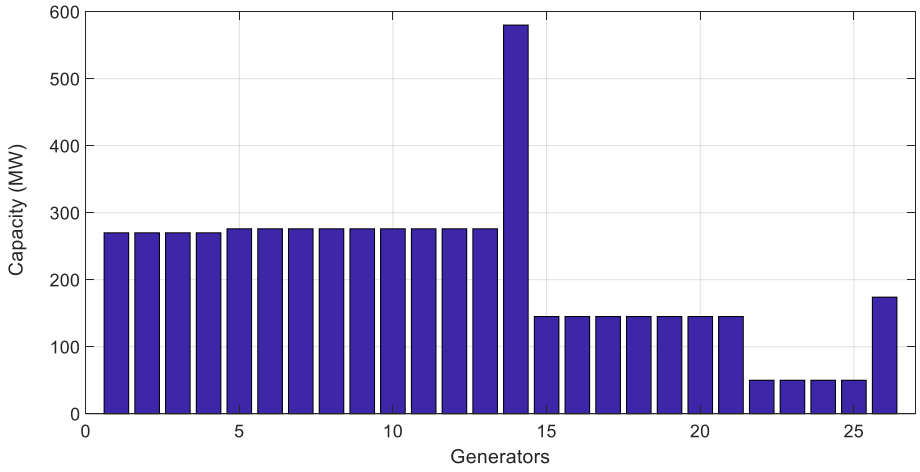


Figure 4-2 Capacity of all 26 generators in the example

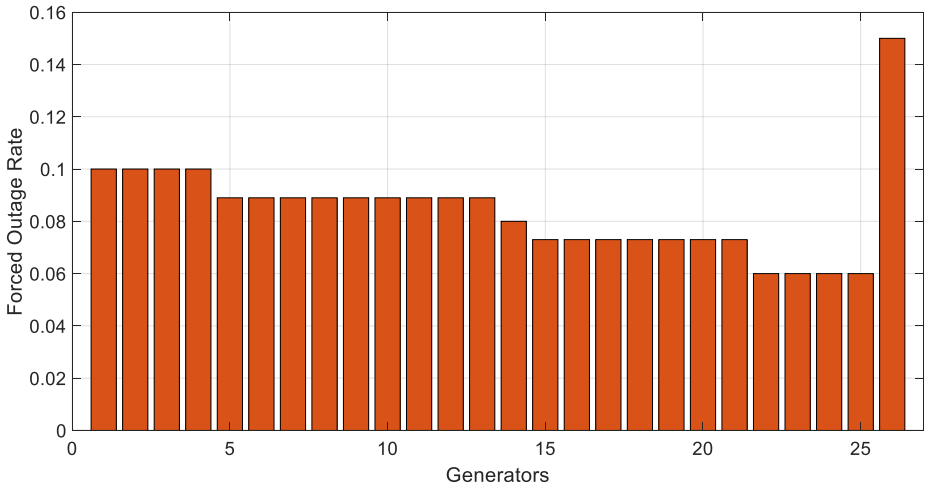


Figure 4-3 Forced outage rate of all 26 generators in the example

4.2 Input of EXIST-RE submodule

Inputs of the EXIST-RE submodule:

The inputs of the EXIST-RE module for a wind farm include:

- Generator type code
- Existing generating capacity
- Wind speeds (i.e., cut-in speed, rated wind speed and cut-off speed)
- State number (to be explained below)
- Feature (to be explained below)
- Existing addition and retirement plan for the planning horizon

The inputs of the EXIST-RE module for a solar farm include:

- Generator type code
- Existing generating capacity
- State number (to be explained below)
- Feature (to be explained below)
- Existing addition and retirement plan for the planning horizon

A user can specify this information in a JSON format file in the path of *'root/projects/project_1/user_input/EXIST-GEN.json'*, following the similar format of EXIST-TH submodule. An example is demonstrated in Figure 4-4, exemplify the case with two wind farms and one solar generator.

```
{
    "wind": {
        "0": {
        "code": "WIND1",
        "capacity": [500],
        "speed": [3.5, 12.5, 30],
        "state_number": [10],
        "feature":{
            "type": [0],
            "path":
"projects/project_1/user_input/data/wind_example1_5min.csv"
            }],
        "existing_plan": {
            "capacity": [100],
            "year": [2025]
            }
        },
        "1": {
        "code": "WIND2",
        "capacity": [250],
        "speed": [3.5, 12.5, 30],
        "state_number": [10],
        "feature":{
            "type": [1],
            "ksigma": [1.87393780943606 7.38164769483493]
            }],
        "existing_plan": {
            "number": [50],
            "year": [2030]
            }
        }
    },
    "solar": {
        "0": {
        "code": "SOLAR1",
        "capacity": [500],
        "state_number": [10],
        "feature":{
            "type": [0],
            "path":
"projects/project_1/user_input/data/solar_example1_5min.csv"
            }],
        "existing_plan": {
```

```
            "number": [1],
            "year": [2025]
            }
        }
    }
}
```

Figure 4-4 Example of input to the EXIST-RE module

In the example above, the user specifies two types of wind power plants "0" and "1"; and one type of solar power plant "0". This can be explained as follows:

For wind farms:

- "*code*": specifies a generator code, which can be any string. This example uses "WIND1" and "WIND2".

- "*capacity*": specifies the capacity of each generator (in MW). In this example, the capacity of "WIND1" is 500 MW and that of "WIND2" is 250MW. Note that: compared with thermal power plants, renewable energy units (i.e., solar panels and wind turbines) have much smaller generating capacity. Hence, instead of treading them as individual units like in the case of thermal power plants, these renewable sources are treated at the aggregated level (i.e., wind farm and solar farm). Therefore, renewable power plants are specified using the site capacity (MW) instead of the number of generating units as in the thermal plants configuration.

- "*speed*": specifies the cut-in speed, rated wind speed and cut-off speed (m/s). In the above example, these speeds of "WIND1" are 3.5 m/s, 12.5 m/s and 30 m/s, respectively.

- "*state_number*": specifies the number of intervals according to the wind/solar farm capacity. The larger state number can be used to represent larger wind/solar farm capacity. The larger the state number, the more accurate it is in the follow-on analysis, but with more time-consuming computation.

- "*feature*": specifies the input used to describe the renewable energy probabilistic distribution feature. Two input types are acceptable to describe output characteristics of wind/solar power plants. They are explained in detail below.

  `"type": [0] -` means the solar/wind output is provided as time-series fixed-interval data (e.g. hourly data or 5-min interval data). The path to the data input file is required. The input file must be in the CSV file format.
  - For wind, the data have the dimension of N*2; where N is the data size depending on the interval data used. For example, N=8,760 if hourly data are provided; N=105,120 if 5-minute interval data are used. The first column is the wind speed and the second column is the corresponding power generation.
  - For solar, the data only need to have a single column for the power generation. This data set has the dimension of N*1.

  `"type": [1] -` means the wind output is provided as a probability function.
  - For wind, the Ksi (ξ) and Sigma (σ) parameters representing the Weibull distribution of the wind speed data are required.

- Type [1] cannot be used for solar generation as no widely accepted probability distribution exists.

- "existing_plan": specifies the capacity of generation and the year for installation. For example, this example specifies "WIND1", a 500MW wind farm, has additional 100 MW wind generation capacity installed in the year of 2025 according to some existing plan.

4.3 Output of EXIST-TH and EXIST-RE Submodules

Outputs of the EXIST-TH and EXIST-RE submodules are combined and unified as outputs of the EXIST_GEN module, which include:

- Generation capacity: an array containing the capacity of all generators in the system.
- Availability rate: an array containing the availability rates of all generators (including both thermal and renewable generators) in the system.
- Generation configuration information: generator characteristics and various cost parameters

The variables above are used in other modules (e.g., OPTIMIZE, ELCC), and are saved in a '.mat' file under the path of *'root/projects/project_1/project_data/EXIST-GEN.mat'* for the EXIST-GEN module. The output information is also stored in the structure variable *system_info* as *existing_gen_info* and *existing_gen_conf*, which are depicted in Figure 4-5. The variable names are 'capacities' and 'availability_rate'. Availability rates are actually "one minus outage rate" for thermal generators, and availability probability for renewable energy sources. The thermal generator characterized by the outage rate, in this way, is equivalent to two-states representation similar to multi-states representation renewable energy sources. The detailed analysis of multi-states representation and generator probabilistic model is introduced in Chapter 5.0 CANDI-GEN Module.

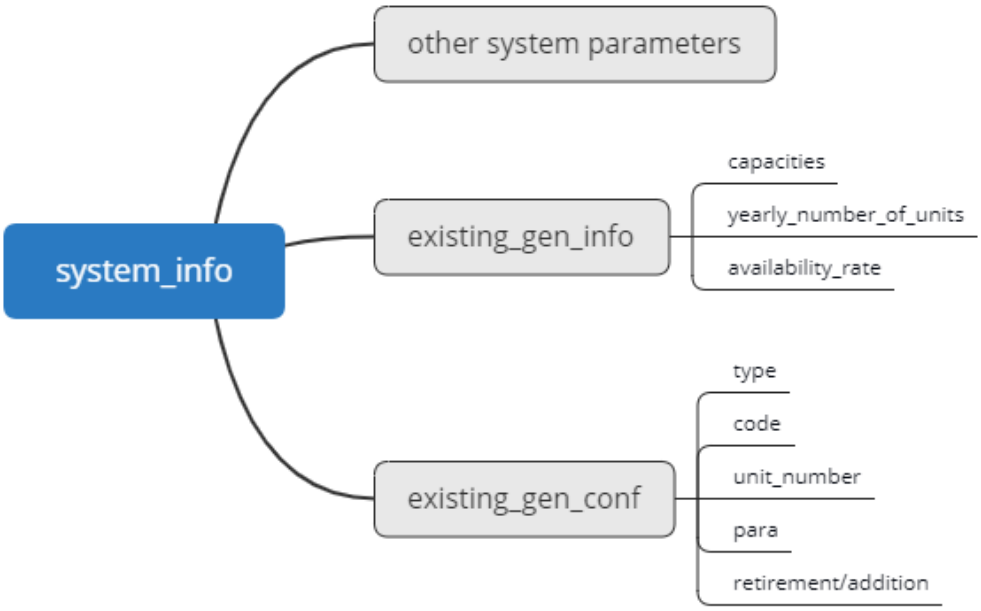


Figure 4-5 The output structure of existing generator information

5.0 CANDI-GEN Module

CANDI-GEN module is responsible for collecting user-defined input data and various types of parameters to characterize the candidate power plants (both thermal plants and renewables).

5.1 Overview of CANDI-GEN Module

Similar to the EXIST-GEN module, CANDI-GEN consists of two sub-modules, CANDI-TH and CANDI-RE, that model the characteristics of candidate conventional and renewable power plants, respectively. Reserve margin requirements are also specified in this module through a JSON format file, *CANDI-GEN.json*. It is noteworthy that each wind/solar farm is treated as one single giant unit because their power production variation is fully correlated. The consideration of renewable energy sources as normal candidate generators with probabilistic modeling instead of treating the renewable generator output as negative loads are also highlights of this project.

5.2 Multi-state Representation for Renewable Power Plants

In order to help understanding how the renewable energy sources are modeled, we introduce a commonly used multi-state representation method that can take into account the variable nature of renewable energy sources. And in this way, these renewable energy sources can be treated as normal generation candidates in a power system expansion planning. This is unlike considering these variable resources as negative loads as being used in the current practice. Previously, when calculating the loss of load probability (LOLP) and expected energy not supplied (EENS) of a system with renewable energy, many methods treat renewable energy generation as negative loads. However, the negative load approach requires a time-series annual system load profile, as well as an annual renewable energy generation profile. In case of the absence of such data, the analysis cannot be conducted. Therefore, a probabilistic method is developed to represent renewable energy generation in the system.

Based on the wind power generation input file, the developed tool derives the relationship between wind power generation and wind speed - as shown in Figure 5-1. In the figure, $P_{\max}$ represents for the maximum generation of the wind farm (in MW); and $V_{\min}$ is the minimum wind speed (m/s) to generate $P_{\max}$.

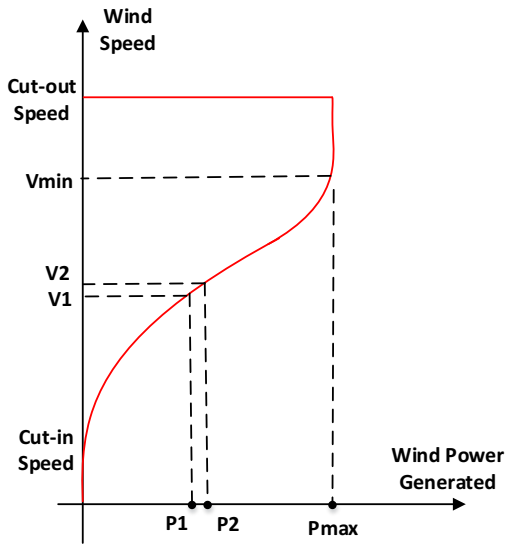


Figure 5-1 Inverted V-P Chart

Considering the Weibull distribution, the probability of wind power generation between P1 and P2 can be calculated as:

$$\Pr(p \in (P_1, P_2)) = \Pr(V \in (V_1, V_2)) = F_w(V_2) - F_w(V_1) = e^{-(\frac{V_1}{\sigma})^{\xi}} - e^{-(\frac{V_2}{\sigma})^{\xi}} \quad (1)$$

$F_w(x)$ is the Cumulative Distribution Function (CDF) of the Weibull distribution; ξ is the shape parameter; and σ is the scale parameter.

$$F_w(x) = 1 - e^{-(\frac{x}{\sigma})^{\xi}} \quad (2)$$

Assuming a mapping function $V = g(p)$ maps the power generation in the range of $[0, P_{\max}]$ to the corresponding wind speed.

Then, (1) can be written as:

$$\Pr(p \in (P_1, P_2)) = e^{-(\frac{g(P_1)}{\sigma})^{\xi}} - e^{-(\frac{g(P_2)}{\sigma})^{\xi}} \quad (3)$$

Assume $P_1 = P_2 - \Delta P$ and ΔP is a small step, the probability that the wind generation is around P_1 can be calculated using the following equation:

$$\Pr(p \approx P_1) = e^{-(\frac{g(P_1)}{\sigma})^{\xi}} - e^{-(\frac{g(P_1+\Delta P)}{\sigma})^{\xi}} \quad (4)$$

For fossil-fuel generators, to calculate Equivalent Load Duration Curve (ELDC), the following formula is used:

$$f^i(x) = p_i f^{i-1}(x) + q_i f^{i-1}(x - c_i) \quad (5)$$

For wind power plants, the power range $[0, P_{\max}]$ can be divided into N equal steps, each step size is ΔP. This relationship can be expressed in (6). Typically, N can be set at 10 or 20, depending on the size of the renewable power plant.

$$P_{\max} = N \cdot \Delta P \tag{6}$$

For k as specified, there is:

$$\Pr(p \approx k \cdot \Delta P) = e^{-(\frac{g(k \cdot \Delta P)}{\sigma})^{\xi}} - e^{-(\frac{g(k \cdot \Delta P + \Delta P)}{\sigma})^{\xi}} \quad k \in [1, N-1] \tag{7}$$

Otherwise, when $k = 0$ and $k = N$, there are:

$$\Pr(p = 0) = \Pr(V < V_{cut-in} \text{ or } V > V_{cut-out})) = F_w(V_{cut-in}) + 1 - F_w(V_{cut-out}) = e^{-(\frac{V_{cut-out}}{\sigma})^{\xi}} + 1 - e^{-(\frac{V_{cut-in}}{\sigma})^{\xi}} \tag{8}$$

$$\Pr(p = P_{\max}) = \Pr(V \in (V_{\min}, V_{cut-out})) = F_w(V_{cut-out}) - F_w(V_{\min}) = e^{-(\frac{V_{\min}}{\sigma})^{\xi}} - e^{-(\frac{V_{cut-out}}{\sigma})^{\xi}} \tag{9}$$

As a result, to calculate the ELDC considering the variation of wind generation, the following equation can be used:

$$\begin{aligned} f^i(x) &= \sum_{k=0}^{N} \Pr(p \approx k \cdot \Delta p) \cdot f^{i-1}(x - (P_{\max} - k \cdot \Delta p)) \\ &= \Pr(p = 0) \cdot f^{i-1}(x - P_{\max}) + \Pr(p = P_{\max}) \cdot f^{i-1}(x) \\ &+ \sum_{k=1}^{N-1} (e^{-(\frac{g(k \cdot \Delta P)}{\sigma})^{\xi}} - e^{-(\frac{g(k \cdot \Delta P + \Delta P)}{\sigma})^{\xi}}) \cdot f^{i-1}(x - (N-k) \cdot \Delta p)) \end{aligned} \tag{10}$$

In summary, variables in the following table are needed to represent wind power generation in the developed tool, which are provided by the user in the '*EXIST-GEN.json*' and '*CANDI-GEN.json*' files.

Table 5-1 Information needed to model wind generation, specified as the inputs in the '*CANDI-GEN.json*' files

Variable Name	***Description***
$P_{\max}$	*Wind farm capacity (MW)*
$V_{cut-in}, V_{\min}, V_{cut-out}$	*Wind speeds (m/s)*
N	*State number*
-	`"type": [0]` *feature: the wind speed-power profile, specified as a path to a CSV file (e.g., in the above example, the path is* `"projects/project_1/user_input/data/wind_example1_5min.csv"`*).* *This wind speed-power profile is used to derive the mapping function (i.e.,* $V = g(p)$*) representing the relationship between the wind speed and wind power output profile.*
ξ, σ	`"type": [1]` *feature: shape and scale parameters for Weibull distribution, respectively*

For solar power plants, the power generation probability is represented by the statistical frequency as shown in (11). Since the volume of solar generation data is large enough (at least 8760 data points for annual hourly data, and even more data points if the data set is in higher granularity, e.g., in 1-min intervals), it is reasonable to use occurrence frequency to represent probability.

$$\Pr(p \approx k \cdot \Delta P) = \frac{Number\ of\ data\ points\ with\ power\ between\ (k \cdot \Delta P)\ and\ (k \cdot \Delta P + \Delta P)}{Number\ of\ all\ data\ points} \quad k \in [0, N-1] \quad (11)$$

This model was tested and compared with the negative load approach in the case studies.

5.3 Input of CANDI-TH Submodule

The inputs of CANDI-TH submodule have been designed to include:

- Generator type code
- Unit number of such generator type
- Parameters used to describe such generator type
- Existing addition and retirement plan for the planning horizon

A user can specify these input parameters in a JSON format file in the path of *'root/projects/project_1/user_input/CANDI-GEN.json'*. An example of this file is demonstrated below in Figure 5-2.

```
{
    "reserve_margins":{
        "margins":[[-50,50],[20,40]],
        "year":[1998,2004]
    },
    "generators": [
        {
        "type": "thermal",
        "code": "V-CC",
        "depreciable_capital_cost": [318, 477, 11.92, 25, 3],
        "forced_outage_rate":10,
        "capacity": [600, 600, 600, 600, 600, 600, 600, 600],
        "para":[300, 600, 6, 1950, 1950, 0, 10, 28, 600, 0, 1200, 2.1, 4.0,
11000, 0.0, 0.5],
        "plans":{
            "number": [[0,2],[1,2], [2,2]],
            "year": [2001, 2003, 2004]
            }
        },
        {
        "code": "VLG1",
        "type": "thermal",
        "depreciable_capital_cost": [594, 891, 19.2, 25, 5],
        "forced_outage_rate":10,
        "capacity": [280, 280, 280, 280, 280, 280, 280, 280],
        "para":[150, 280, 1, 3100, 2700, 10, 10, 56, 280, 710, 0, 2.7, 6.0,
1800, 2.5, 1.0],
        "plans":{
```

```
            "number": [[0,2], [1,2], [2,2], [3,2]],
            "year": [2002, 2005, 2013, 2016]
            }
        },
        {
        "code": "VLG2",
        "type": "thermal",
        "depreciable_capital_cost": [544, 817, 19.2, 25, 5],
        "forced_outage_rate":10,
        "capacity": [280, 280, 280, 280, 280, 280, 280, 280],
        "para":[150, 280, 2, 3000, 2600, 10, 10, 56, 280, 1100, 0, 2.7, 6.0,
1800, 2.5, 1.0],
        "plans":{
            "number": [[0,2], [1,2], [2,2], [3,2], [4,2], [5,2]],
            "year": [2007, 2010, 2011, 2013, 2014, 2017]
            }
        },
        {
        "code": "VCOA",
        "type": "thermal",
        "depreciable_capital_cost": [495, 743, 19.2, 25, 5],
        "forced_outage_rate":8,
        "capacity": [580, 580, 580, 580, 580, 580, 580, 580],
        "para":[400, 580, 3, 2600, 2200, 10, 8, 48, 600, 0, 800, 2.92, 5.0,
6000, 1.0, 2.0],
        "plans":{
            "number": [[0,2], [1,2], [2,2], [3,2], [4,2], [5,2], [6,2],
[7,2], [8,2]],
            "year": [2002, 2006, 2007,2009,2010,2012, 2013, 2014, 2016]
            }
        },
        {
        "code": "NUCL",
        "type": "thermal",
        "depreciable_capital_cost": [730, 1703, 26.0, 30, 7],
        "forced_outage_rate":10,
        "capacity": [600, 600, 600, 600, 600, 600, 600, 600],
        "para":[300, 600, 0, 2600, 2340, 7, 10.0, 42, 600, 0, 194, 2.5, 0.5,
0, 0, 0],
        "plans":{
            "number": [[0,1], [0,2], [1,2], [2,2]],
            "year": [2006, 2011, 2015, 2017]
            }
        }
}
```

Figure 5-2 Example of input to the CANDI-TH submodule

In the example above, the user specifies five types of generators. For each generator type, the following information is specified:

- "code": specifies the generator code, which can be any string. This example uses "VCC", "VLG1", "VLG2", "VCOA", "NUCL" to represent different types of generators.

- "type" specifies the type of candidate generators, thermal or renewable;

- “depreciable capital cost” specifies the construction cost and related information, such as plant life, depreciation rate and construction years.
- “forced outage rate” specifies the forced outage rate.
- “capacity” specifies the generator unit capacity in different time periods.
- “para” specifies parameters describing the generator, as shown in Table 5-2. These parameters are always following the same order of sequence in variable matrix for keeping the physical meaning.

Table 5-2 Parameters of thermal plants required for CANDI-TH

Plant's Names/Codes			
Fuels' Types/Names			
	Number of units		
Plant's features	Category	Variable	Example
	Electric-related	Minimal operating power of a unit (MW)	200
		Maximal generating power of a unit (MW)	300
	Cost-related	Fuel costs ($/million kcals)	600
		Fixed O&M costs (in $/kW Month)	2.42
		Variable O&M costs (in $/MWh)	4.00
	Reliability-related	Forced outage rate	12%
		Spinning reserve (% of unit capacity)	10
		Maintenance class size (MW)	300
		Scheduled maintenance days of a year	53
	Heat-related	Heat rate at minimal operating level (kcal/kWh)	2300
		Average incremental heat rate (kcal/kWh)	2100
		Heat value of the fuel used (kcal/kg)	5800
	Polution-related	Pollutant I (SO2) emission (% weight of fuel)	1
		Pollutant II (NOx) emission (% weight of fuel)	2

- “plans” has two attributes: number and year, which are used to constraint the minimal and maximal number of generator units allowed to be installed in the specified year. Some years that are not indicated follows the previous ones.

5.4 Input of CANDI-RE Submodule

The inputs of the CANDI-RE submodule for a wind farm include:

- Generator type code
- Generating maximum capacity
- Wind speeds (i.e., cut-in speed, rated wind speed and cut-off speed)
- Characteristic generation parameters
- State number (to be explained below)
- Feature (to be explained below)
- Depreciable capital cost
- Yearly specified construction cost (to be explained below)
- Plan tunnels (to be explained below)

The inputs of the CANDI-RE submodule for a solar farm include:

- Generator type code
- Generating maximum capacity
- Characteristic generation parameters
- State number (to be explained below)
- Feature (to be explained below)
- Depreciable capital cost
- Yearly specified construction cost (to be explained below)
- Plan tunnels (to be explained below)

A user can specify this information in a JSON format file in the path of *'root/projects/project_1/user_input/CANDI-GEN.json'*, following the similar format of CANDI-TH submodule. An example is demonstrated in Figure 5-3, exemplify the case with one wind farm and one solar generator.

```
        {
        "code": "WIND_CANDI_1",
        "type": "wind",
        "capacity": [30,30,30,30,30,30,30,30],
        "para":[30, 30, 0, 0, 0, 0, 5, 40, 50, 0, 0, 3.67, 4.0, 0, 0, 0],
        "depreciable_capital_cost": [1094, 0, 26.0, 20, 5],
        "construction_cost_yearly_specific": [1094, 1080, 1000, 980, 900,
880, 870, 850, 840, 820, 800, 780, 770, 760, 740, 720, 700, 690, 680, 600],
        "forced_outage_rate":5,
        "speed": [3.5, 12.5, 30],
        "state_number": [10],
        "plans":{
            "number": [[1,0],[2,0],[4,0],[7,0],[9,0]],
            "year": [2001,2006,2010,2013,2016]
            },
        "feature":{
            "type": [0],
            "path":
"projects/project_1/user_input/data/wind2/winda_2007_5min.csv"
            }
        },
        {
        "type": "solar",
        "code": "SOLAR_CANDI_1",
        "capacity": [50,50,50,50,50,50,50,50],
        "para":[50, 50, 0, 0, 0, 0, 0, 0, 50, 0, 0, 0, 0, 0, 0, 0],
```

```
        "depreciable_capital_cost": [1400, 0, 26.0, 25, 5],
        "construction_cost_yearly_specific": [1400, 1280, 1110, 1080, 990,
890, 870, 850, 840, 820, 800, 780, 770, 750, 740, 720, 700, 696, 680, 650],
        "forced_outage_rate":0,
        "state_number": [10],
        "plans":{
            "number": [[1,0],[2,0],[5,0],[8,0]],
            "year": [1999,2003,2007,2015]
            },
        "feature":{
            "type": [0],
            "path":
"projects/project_1/user_input/data/solar1/solar_5min_1.csv"
            }
        }
```

Figure 5-3 Example of input to the CANDI-RE submodule

In the example above, the user specifies two types of wind power plants "0"; and one type of solar power plant "0". This can be explained as follows:

For wind or solar farms:

- "*code*": specifies a generator code, which can be any string. This example uses "WIND_CANDI_1" and "SOLAR_CANDI_1".

- "*capacity*": specifies the maximum capacity of each generator (in MW) in different time periods.

- "*speed*" (only for wind): specifies the cut-in speed, rated wind speed and cut-off speed (m/s). In the above example, these speeds of "WIND_CANDI_1" are 3.5 m/s, 12.5 m/s and 30 m/s, respectively.

- "*state_number*": specifies the number of intervals according to the wind/solar farm capacity. The larger state number can be used to represent larger wind/solar farm capacity. The larger the state number, the more accurate it is in the follow-on analysis, but with more time-consuming computation.

- "*feature*": specifies the input used to describe the renewable energy probabilistic distribution feature. Two input types are acceptable to describe output characteristics of wind/solar power plants. They are explained in detail below.

  `"type": [0] -` means the solar/wind output is provided as time-series fixed-interval data (e.g. hourly data or 5-min interval data). The path to the data input file is required. The input file must be in the CSV file format.
  - For wind, the data have the dimension of N*2; where N is the data size depending on the interval data used. For example, N=8,760 if hourly data are provided; N=105,120 if 5-minute interval data are used. The first column is the wind speed and the second column is the corresponding power generation.
  - For solar, the data only need to have a single column for the power generation. This data set has the dimension of N*1.

  `"type": [1] -` means the wind output is provided as a probability function.
  - For wind, the Ksi (ξ) and Sigma (σ) parameters representing the Weibull distribution of the wind speed data are required.

- Type [1] cannot be used for solar generation as no widely accepted probability distribution exists.

- "*plans*" has two attributes: number and year, which are used to constraint the minimal and maximal number of generator units allowed to be installed in the specified year. Some years that are not indicated follows the previous ones.

- "*depreciable capital cost*" specifies the construction related information, such as plant life, depreciation rate and construction years.

- "*forced outage rate*" specifies the forced outage rate.

- "*capacity*" specifies the generator unit capacity in different time periods.

- "*para*" specifies parameters describing the generator, as shown in Table 5-2. These parameters are always following the same order of sequence in variable matrix for keeping the physical meaning.

- "*construction_cost_yearly_specific*" specifies the year-by-year construction cost of renewable energy sources since this cost usually keep decreasing rapidly in the future

5.5 Output of CANDI-TH and CANDI-RE Submodules

Outputs of the CANDI-TH and CANDI-RE submodules are combined and unified as outputs of the CANDI_GEN module, which include:

- Generation capacity: an array containing the capacity of all generators in the system.
- Availability rate: an array containing the availability rates of all generators (including both thermal and renewable generators) in the system.
- Generation configuration information: reserve margin information, generator characteristics and various cost parameters

The variables above are used in other modules (e.g., CONFIG, OPTIMIZE, ELCC), and are saved in a '.mat' file under the path of *'root/projects/project_1/project_data/CANDI-GEN.mat'* for the CANDI-GEN module. The output information is also stored in the structure variable *system_info* as *candi_gen_info* and *candi_gen_conf*, which are depicted in Figure 5-4. The variable names are 'capacities' and 'availability_rate'. Availability rates are actually "one minus outage rate" for thermal generators, and availability probability for renewable energy sources. The candidate thermal generator characterized by the outage rate, in this way, is equivalent to two-states representation similar to multi-states representation renewable energy sources. This capacity availability information, together with reserve margin requirement information, is used by CONFIG module to produce the available candidate configurations of generation expansion planning. The cost information and characteristic parameters are used by OPTIMIZE module to calculate the total cost and cash flow of the potential candidate configuration of specific expansion plans. This is explained in Chapter 7.

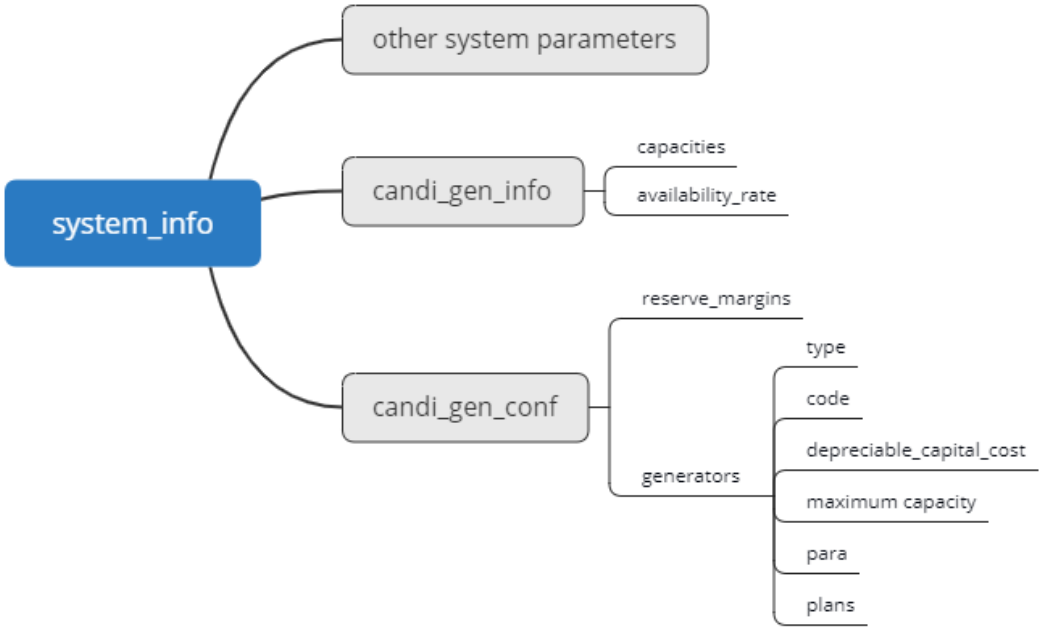


Figure 5-4 The output structure of candidate generator information

6.0 CONFIG Module

CONFIG is the expansion configuration module that generates additional constraints for the optimization problem in the OPTIMIZE module. It narrows down possible expansion configurations based on the constraint inputs form a user. Required input data include, for example, minimum/maximum reserve requirements, minimum/maximum number of units of a particular expansion candidate that can be installed in a given year, and the acceptable LOLP in the given study period.

6.1 Overview of CONFIG Module

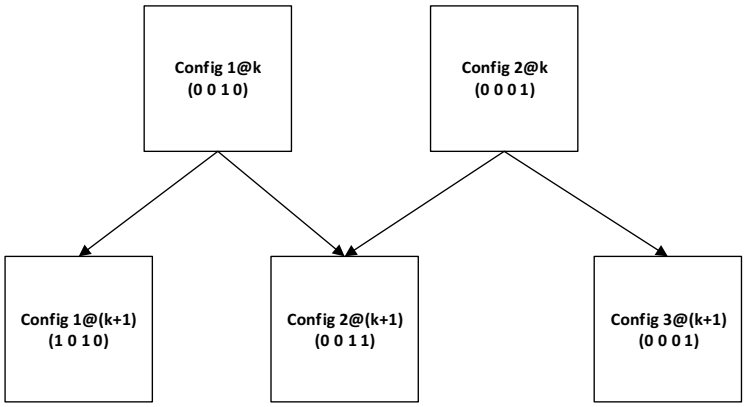
The CONFIG module is used to generate all valid configurations over the planning years. For a specific year, a configuration shows the cumulative number of installed candidate plants of each kind in this year since the start. **Error! Reference source not found.** demonstrates an example of a configuration in a specific year. in this year, the accumulative number of Generator G1 is four starting from the first year of the planning period.

Table 6-1 Configuration example for a specific year

Candidate Generator	G1	G2	G3	G4
Number	4	5	3	2

In general, configurations have the following features:

1) Evolving: Configurations are related to years, and each configuration must be able to evolve from at least one configuration of the previous year. **Error! Reference source not found.**-1 shows an example of configurations in Year k and k+1. The evolving feature of configuration gives them a parent-child relationship: 'Config 1@k' is the parent configuration of 'Config 1@(k+1)' and 'Config 2@(k+1)'. In addition, a configuration can have multiple different parent configurations and even itself as a parent configuration, as shown in **Error! Reference source not found.**-1. The root configuration 'Config 1@0' is always a 1-by-N vector, with N representing the total number of generators.



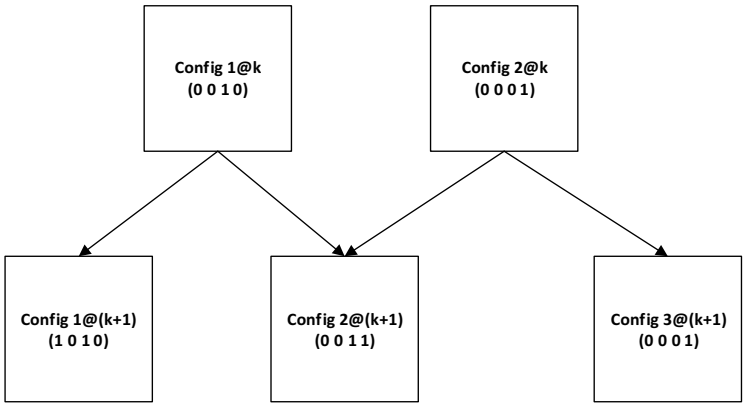


Figure 6-1 Example of configuration evolution

2) Cumulative: The number of each generation unit in a configuration represents how many such unit has been built since the first year of the planning period. As a result, the number can only be equal or larger than its parent configuration in the previous year. For example, (G1, G2, G3, G4) = (1 1 1 0) might be a possible child configuration of 'Config 1@(k+1)' while (G1, G2, G3, G4) = (0 1 1 0) cannot be since the number of Generator G1 is decreasing.

In addition to these features, there are also other requirements for a configuration to be valid (See Table 6-2). Configurations that satisfied the system reserve requirement are called valid configuration. All invalid configurations are discarded. In summary, the functionality of the CONFIG module is to generate valid configurations of all years during the planning period.

6.2 Inputs of CONFIG Module

The inputs and outputs of the CONFIG module are briefly summarized in **Error! Reference source not found.**.

Table 6-2 Inputs and outputs of the CONFIG module

Inputs	Outputs
1) Reserve requirements	Valid configurations in a well-organized data structure that can be conveniently accessed by the following modules.
2) Minimum number of units installation	
3) Tunnel width of units installation	

Since configuration is related to the installation of candidate generators, it makes sense to put all required inputs in the *CANDI-GEN.json* configuration file.

- ***Input 1: Reserve requirements***

System generation reserve are specified by two numbers: lower bound and upper bound. If the generation reserve is lower than the lower bound, the system reliability is jeopardized due to generation capacity

shortage; on the other hand, if it is over the upper bound, unnecessary asset investment incurs. As a result, at each year the total generation capacity from existing generators and candidate generators should fall between these boundaries.

In the *CANDI-GEN.json*, the reserve requirement is specified by the key '*reserve_margins*' with two key-value pairs as the contents: they are '*margins*' and '*year*', as shown in Figure 6-2. The values for 'margins' and 'year' should have the same length. In the example, it shows that the upper and lower bound of reserve margin is 15(%) and 50(%) since year 1998. This margin remains the same for the following years until new change specified: starting in 2004, the margin should be within 20(%) and 40(%).

```
    "reserve_margins":{
        "margins":[[15,50],[20,40]],
        "year":[1998,2004]
    }
```

Figure 6-2 Configuration of system reserve margin in the CANDI-GEN.json

- ***Input 2: Minimum and maximum number of units installation***

Human experts can specify the range of generators' number every year: for a specific type of generator, the number should fall within certain range. The ranges can be changed in the candidate generators' configuration file, with the key name '*plans*' in the candidate generator's hash map data structure. This module inherits a feature from WASP, so most of the users are familiar: the range is specified as minimum number of units and the tunnel width, which refers to how many more units can be added to the minimum number.

For instance, 6-3 shows an example of one candidate thermal generator. For this generator, according to the user defined configuration, the number should be between [0, 2] in the year 2001 and 2002; starting 2003, there should be at least one unit of this generator and the maximum number can be 3; starting in 2004, there should be at least two units and can reach up to 4 units, this remains the same till the end of the planning period. Before 2001, since there is no specification, it is assumed [0, 0] in place, that means this generator is unable to be installed prior to 2001

```
{
        "type": "thermal",
        "code": "V-CC",

        …
        …

        "para":[300, 600, 6, 1950, 1950, 0, 10, 28, 600, 0, 1200, 2.1, 4.0, 11000, 0.0, 0.5],
        "plans":{
            "number": [[0,2],[1,2], [2,2]],
            "year": [2001, 2003, 2004]
         }
    }
```

Figure 6-3 Configuration of an example candidate generator in the CANDI-GEN.json

6.3 Outputs of CONFIG Module

The output of the CONFIG module is an indexed matrix that stores all valid configurations over the planning period in the variable *CT.pStore* (*CT* is an instantization of class *ConfigurationTree*). Some examples of generated configuration candidates are depicted in Figure 6-4. Each column is corresponding to a specific generator type. These configuration candidates with indexes also form a tree-like structure that contains all possible system evolution paths, which is evaluated later to identify the optimal plan. In this fashion, the CONFIG module does not only generate the individual configuration candidate (e.g., 4 3 3 6 1) but also the evolution relationship of these configuration candidates through *HaspMap* and a sparse *connection* matrix. This kind of information is also stored in the instantization variable of class *ConfigurationTree* as shown in Figure 6-5.

CT.pStore

	1	2	3	4	5
79	4	3	3	6	1
80	4	3	1	6	1
81	4	3	4	6	1
82	4	3	2	6	2
83	4	3	3	6	2
84	4	3	4	6	2
85	4	3	4	7	1
86	4	3	4	6	2
87	4	3	4	7	2
88	4	3	2	7	2
89	4	3	3	7	2
90	4	4	4	8	1
91	4	3	5	8	1
92	4	4	5	8	1
93	4	4	4	7	2
94	4	3	5	7	2
95	4	4	5	7	2
96	4	3	4	8	2
97	4	4	4	8	2
98	4	3	5	8	2
99	4	4	5	8	2
100	4	3	3	8	2

Figure 6-4 Screen capture of generated configuration candidates index 79-100

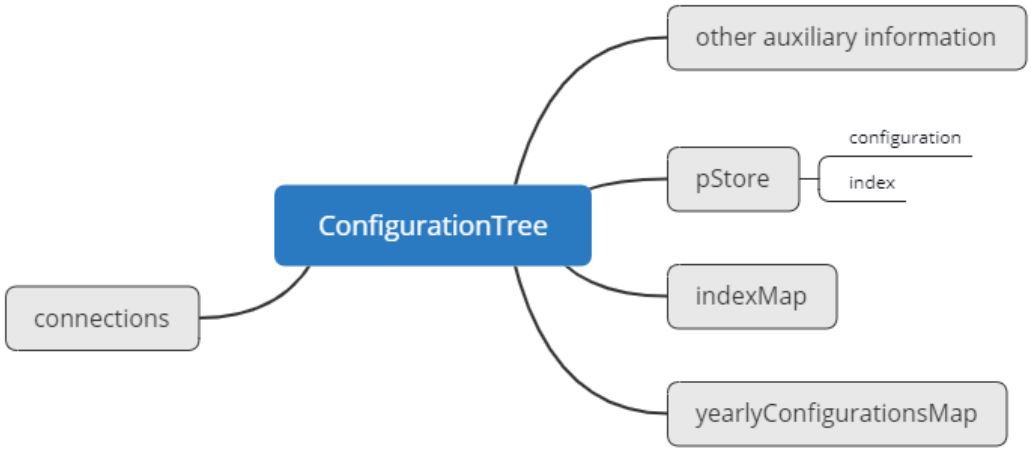


Figure 6-5 The output structure of configuration candidate information

6.4 Configuration Tree Data Structure

In order to keep the configuration candidate evolution information, all the valid generation expansion planning configurations candidates are kept connected and stored in a tree-like data structure. We do care about this evolution relationship because the path of evolution determine the different newly construction cost involved in the different generation expansion plans. This custom tree data structure is defined to hold all the permutations of configurations for each year and indicate their evolution relationship (i.e., parent configuration and child configuration connections). The basic structure of the configuration tree is presented as shown in Figure 6-6.

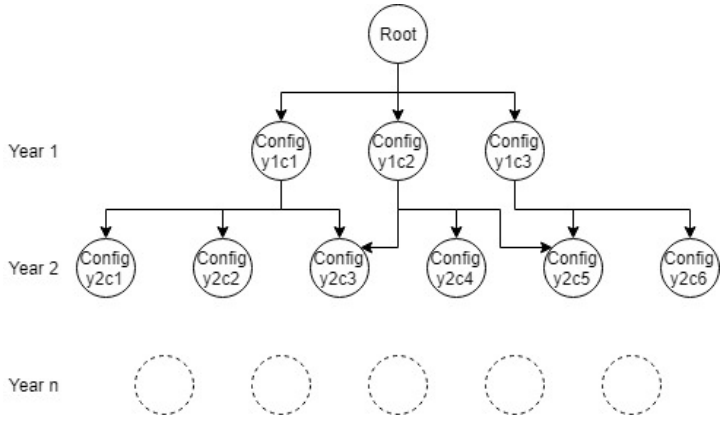


Figure 6-6 Tree data structure of configurations

For each year, starting the first year, we need to generate a set of valid configurations that can be reached from each configuration in the previous year, called the parent configuration. Valid configurations are those configurations which meet the reserve margin for that year, as previously discussed.

The basic class definition of our configuration tree data-structure is as follows:

```
classdef ConfigurationTree < handle
    properties
        pStore
        indexMap
        yearlyConfigurationsMap
        totalYears
        totalCandiGenerators
    end
    methods (Access = public)
        function obj = ConfigurationTree(studyYear,totalCandiGenerators)
            %constructor
        end

        function obj = addConfiguration(obj,year,parentIndex,configuration)
            %add a new configuration to the tree
        end

        function obj = addConfigurationList(obj,year,parentIndex,configurationList)
              %generalization of addConfiguration to add a list of
              configurations
         end
    end
    methods (Access = protected)
        function addChild(obj,parentIndex,childIndex)
            %set a specific configuration as a child to a parent config
        end
    end
end
```

Figure 6-7 Class of configuration tree structure in MATLAB code

Each of the member variables and the functions are explained below:

pStore

```
type: 2-d numeric matrix
size: NxC (N = number of nodes in the tree, C = number of candidate generators)
```

It is a 2-D matrix to hold all of the configurations (nodes) in the tree. Each row is a new configuration, and the columns is for number of units of different candidate generators. A typical value of *pStore* might look like this:

[0, 0, 0, 0, 0;
2, 2, 0, 1, 0;
2, 0, 0, 2, 0]

Which means, there are three configurations in total at the configuration tree. The first configuration, is 0, 0, 0, 0, 0. Which means, all five candidate generators are not included in this configuration. The second configuration is 2, 2, 0, 1, 0. Which means, 2 of the first, 2 of the second, and 1 of the fourth candidate generator are included in this configuration.

indexMap

```
type: hash-map
size: N (equal to number of unique yearly configurations)
key: string, configuration + year
value: int, index of the configuration in pStore
```

It is a hash map linking a particular configuration to its index in the *pStore*. A hash map of configuration is needed so that duplicate configurations aren't added to the configuration tree. Every time a new configuration is generated for a particular year, a test is done to check if that configuration already exists in *indexMap*, in which case, it is considered a duplicate and is not added to *pStore*. Instead, it's index at pStore is retrieved from indexMap, and that configuration is marked as a child of the configuration for which new configurations was being generated. Since configurations need to be unique for a given year only, the year number is appended to the configuration vector and converted to a string before adding to indexMap as a key.

yearlyConfigurationMap

```
type: hash-map
size: total number of study years
key: string, year
value: list of int, indices of the configurations in pStore for that year
```

It is a hash map to store and retrieve all the valid configurations for each year.

connections

```
type: matrix
size: NxN
```

This is the adjacency matrix to store the parent-child relationships of all the configuration nodes in the tree. It's size is equal to the total nodes in the configuration tree.

totalYears

```
type: int
```

Stores the total simulation years

totalCandiGenerators

```
type: int
```

Stores the total number of candidate generators.

Methods including:

ConfigurationTree(studyYear,totalCandiGenerators)

```
Input:
   studyYear: list of int, the list contigious years to conduct the study
   totalCandiGenerators: int, total number of candidate generators

Output:
   obj: A new configuration tree object, with the root configuration
```

This is the constructor method to initialize an empty configuration tree. It takes in *studyYear*, which is the list of study years, and *totalCandiGenerators* which is the total number of candidate generators. It then places the first configuration, which is just a zero vector as the root of the configuration tree.

addConfiguration(year,parentIndex, configuration)

```
Input:
   year: int, the year for which configuration is being added
   parentIndex: int, the index of the parent configuration in pStore
   configuration: vector, the configuration vector (1xC)

Output:
   obj: Modified configuration tree object, with the new configuration added
```

All configuration (except the root, which is automatically created by the constructor) needs to have a parent configuration from which it is derived. This function allows to add a configuration to the configuration tree, as a child of a certain existing configuration. If a configuration identical to the configuration being added alreay exists for the same year, no new configuration is added to the pStore. Instead, the existing duplicate configuration is marked as the child of the parentIndex configuration.

addConfigurationList(year,parentIndex, configurationList)

```
Input:
   year: int, the year for which configuration is being added
   parentIndex: int, the index of the parent configuration in pStore
   configurationList: matrix of T configurations (TxC)

Output:
   obj: Modified configuration tree object, with the new configurations added
```

This is an extension of *addConfiguration* function that allows to add multiple configurations at once.

addChild(parentIndex, childIndex)

```
Input:
  parentIndex: int, the index of the parent configuration in pStore
  childIndex: int, the index of the child configuration in pStore

Output:
  obj: Modified configuration tree object, with the parent-child relationship
  saved in the adjacency matrix
```

This is an internal function to mark the parent-child relationship in adjacency matrix.

Configuration tree generation:

A dynamic programming approach is used to generate the configuration tree for each of the study year. The process starts with generating configurations for the first year, and for each of the subsequent years, new configurations are generated based on each of the existing configuration in the previous year. Each of these new configuration needs to be tested for meeting the reserve requirement for each periods of the year, and if they meet the requirement, they are added into the configuration tree. The configuration tree data structure takes care of duplicate configurations.

The pseudo-code for configuration tree generation can be written as follows:

```
1.  CT = new ConfigurationTree()
2.  for each year in studyYears:
3.  prev_year = year – 1
4.  prev_configs ← configurations in prev_year. (Root config if year is starting year)
5.    for each config in prev_configs:
6.          new_configs = getNewConfigs(config, candidateGeneratorRangesForThisYear)
7.          configCapacities = getConfigCpacities(new_configs,candi_gen_capacities)
8.          minimum_capacity = getMinimumAditionCapacity(year)
9.          validConfigIndices = getValidIndices(configCapacities , minimum_capacity)
10.         acceptable_configs ← new_configs(validConfigIndices)
11.         CT.addConfigurationList(year, index_of_config, acceptable_configs)
```

Here is the description of the functions:

getNewConfigs(config, candidateGeneratorRangesForThisYear)

```
Input:
   config:1xC numeric vector, the parent configuration
   candidateGeneratorRangesForThisYear: 1xC cell-array. Element i gives is a list
of acceptable number of units for the candidate generatior i.
   #example: {[0,1,2],[0],[0],[1,2], [0]}. Candidate generator 1 can have 0, 1 or
two units for this year and candidate generator 4 can have 1 or 2 units

Output:
   new_configs: TxC numeric matrix, where T is the number of new configurations
generated
```

The function generates new configurations based on an input (parent) configuration and the user specified ranges for the number of units of candidate generators.

Example

```
Input:
candidateGeneratorRangesForThisYear = {[0,1,2],[0],[0],[1,2], [0]}
Config = ([1,0,0,0,0]

Output:
#remove units in the ranges which are less than that in config

new_candidateGeneratorRangesForThisYear = {[1,2],[0],[0],[1,2], [0]}
new_configs = cartesianProduct({[1,2],[0],[0],[1,2], [0]})
[1,0,0,1,0;
2,0,0,1,0;
1,0,0,2,0;
2,0,0,2,0]
```

getConfigCpacities(new_configs,candi_gen_capacities)

```
Input:
   new_configs:TxC numeric matrix, the new T configurations
   candi_gen_capacities: CxP numeric matrix of generator capacities, where P is the
number of periods in a year.

Output:
   configCapacities: TxP numeric matrix of total generator capacities for each
configuration and each period
```

The function performs matrix multiplication of the configurations and the capacities, to return a matrix of total generator capacity for each configuration for each period.

Example

```
getNewConfigs([1,1,0,0,0; 2,1,0,0,0], [100,120;30,40;120,110;50,55;20,22])

Input:
new_configs = [1,1,0,0,0; 2,1,0,0,0] (T = 2, C = 5)
candi_gen_capacities = [100,120;30,40;120,110;50,55;20,22] (C =5, P = 2)

Output:
configCapacities = new_configs * candi_gen_capacities
[130, 160;
230, 280]
```

getMinimumAditionCapacity(year)

```
Input:
   year: int, the year for which minimum capacity is to be determined

Output:
   minimum_capacity: 1xP double, the minimum value of total candidate generator
capacity to be added this year to meet reserve margin for each period
```

The function determines the amount of new candidate generator capacity that needs to be added in a given year to meet the reserve requirement. For this the function takes help of the load-calc module to determine the peak load for the given year (for different periods), and the different existing generator plans to determine existing generation capacity for that year. Then, the unmet generation to meet the minimum reserve margin is returned as the *minimum_capacity*.

getValidIndices(configCapacities , minimum_capacity)

```
Input:
   configCapacities: TxP numeric matrix of config capacities
   minium_capacity: 1xP double, the minimum total candidate generator capacity
required to meet the reserve requirement for each period

Output:
   validConfigIndices: 1xT logical array of indices in configCapacities, which
meet the minium_capacity
```

The function performs a basic logical comparison to determine which of the configuration meet the minimum reserve requirement.

Example

```
Input:
configCapacities = [130, 160; 230, 280]
minium_capacity = [140, 150]

Output:
#all makes sure all columns passes the test
validConfigIndices = all(configCapacities > minium_capacity)
= [0, 1] (logical array)
#Only the second row passes the test
```

Verification with WASP:

The developed CONFIG module was run with a test case (without renewables) to verify the output with WASP. The used test case was the default demonstration case study with variable expansion plan in WASP, with the hydro plants removed from fixed, existing and candidate plans.

For the year 2017, the configurations generated by WASP is:

STATE	IC	CAP	ACCEPTED CONFIGURATION					
66	1	18113.	4	5	7	10	3	0
67	2	18133.	4	5	7	9	4	0
68	3	18153.	4	5	5	10	4	0
69	4	18153.	4	4	6	10	4	0
70	5	18433.	4	5	6	10	4	0
71	6	18153.	4	3	7	10	4	0
72	7	18433.	4	4	7	10	4	0
73	8	18713.	4	5	7	10	4	0
CONFIGURATIONS THIS YEAR							8	
CONFIGURATIONS THROUGH THIS YEAR							73	

Figure 6-8 Screen capture of generated configuration in 2017 from WASP

For the same year 2017, the configuration generated by CONFIG muddle is:

```
>> CT.pStore(CT.yearlyConfigurationsMap('2017'),:)

ans =

     4     5     7    10     3
     4     5     7     9     4
     4     5     6    10     4
     4     5     7    10     4
     4     4     6    10     4
     4     4     7    10     4
     4     5     5    10     4
     4     3     7    10     4
```

Figure 6-9 Screen capture of generated configuration in 2017 from MATPLAN

It can thus be seen that the configurations match exactly with WASP.

7.0 OPTIMIZE Module

7.1 Overview of OPTIMIZE Module

OPTIMIZE is the optimization module that combines probabilistic and optimization techniques to determine the optimal system expansion policy based on inputs defined in the other modules. The objective function of minimal generation expansion cost is optimized with taking into account probabilistic model of generator availability, including renewable energy sources, and different types of cost. All economic and cash flow calculations are included, such as present value calculations and escalation of fuel prices. OPTIMIZE module provides users the final decision-making reference of optimal generation expansion plan in different years.

7.2 Cost Calculation

Once all the configurations satisfying given constraints are generated, a cost function is evaluated to find the optimal (least cost) sequence of configurations throughout the study period.

This cost function is an objective function be minimized. It comprises of the followings:

- *Depreciable capital investment costs: equipment, site installation and construction costs (I)*
- *Salvage value of investment costs (S)*
- *Fuel costs (F)*
- *Non-fuel operation and maintenance costs (OM)*
- *Cost of the energy-not-served (ENS)*

The objective function is expressed in Equation (7-1):

$$\text{Minimize} \qquad C_j = \sum_{t=1}^{T}\left[\overline{I}_{j,t} - \overline{S}_{j,t} + \overline{F}_{j,t} + \overline{OM}_{j,t} + \overline{ENS}_{j,t}\right] \tag{7-1}$$

Here,

C_j : objective function attached to an expansion plan $\boldsymbol{j}$.
t : time in years (1, 2, …, T).
T : total length of the study period, given in years.

The bars given above each term in Equation (7-1) mean that these are discounted values based on a reference date and discount rate $\underline{i}$.

Therefore, the optimization problem can be formulated as:

$$\text{Minimum } C_j \qquad \text{for } \forall \text{j} \tag{7-2}$$

The distribution of expenses throughout the entire expansion plan is depicted, as shown in Figure 7-1.

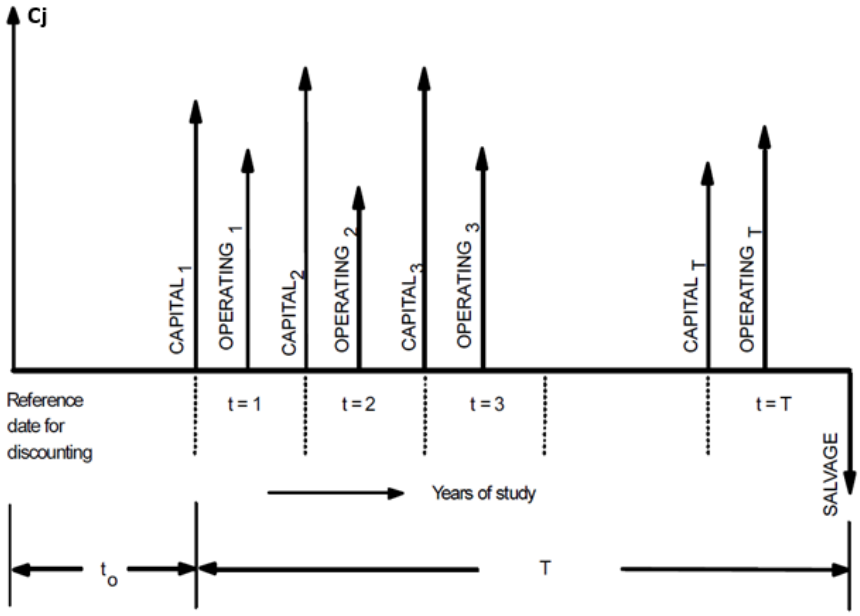


Figure 7-1 Schematic diagram of cash flows for an expansion plan.

In the figure above,

$CAPITAL_1$	: sum of the investment costs of all units added in the first year of study.
$OPERATING_1$	: sum of all system operating costs (fuel, O&M, energy not served) in the first year of study.
SALVAGE	: sum of the salvage values at the horizon, of all plants added during the study period.
t_0	: number of years between the reference date for discounting and the first year of the study.
T	: total length of the study period, given in years.

a. Depreciable capital investment costs

$$\overline{I}_{j,t} = (1+i)^{-(t+t_0-1)} * \sum[UI_k * MW_k] \quad (7\text{-}3)$$

Here,

$\sum$: sum calculated considering all units added in year t by expansion plan ***j***.
UI_k : capital investment cost of unit k, expressed in monetary units per MW.
MW_k : capacity of unit k in MW.
i : discount rate.
t : time in years (1, 2, ..., T).
t_0 : number of years between the reference date for discounting and the first year of the study.

Equation (7-3) was implemented in our optimization module using the code given below:

```
108                 configuration_parent = system_info.CT.pStore(parents(k),:);
109                 new_added = configuration - configuration_parent;
110                 COST_CON_vector = new_added .* unitCapacity_candi .* UI;
111                 COST_CON{countConfig}(k) = (1+discount)^(-tt) * sum(COST_CON_vector);
```

For a given year of the study (line 108), the parent nodes of a particular acceptable configuration are traced back. Using this information (line 109), the difference between the parent configuration and newly generated configuration is calculated to find which generator units are added into the new configuration. Lines 110-111 implement Equation (3), where **COST_CON** = depreciable capital investment costs (construction costs) . The cost for each new plant added in a given year is then calculated and the values are stored in the vector **COST_CON**. The process is repeated for all possible configurations in that year.

b. Salvage value of investment costs

$$\overline{S}_{j,t} = (1+i)^{-(T+t_0)} * \sum\left[\delta_{k,t} * UI_k * MW_k\right] \tag{7-4}$$

Where,

$$\delta_{k,t} = \frac{1-(1+i)^{-L_k+y_k}}{1-(1+i)^{-L_k}} \tag{7-5}$$

Here,

$\delta_{k,t}$: salvage value factor at the horizon for unit *k*.
T : total length of the study period, given in years.
L_k : Life of unit *k* in years.
y_k : $T-(t+1)$

Using the costs calculated in the previous section, the salvage costs are calculated next.

```
100         for k = 1:length(plant_life)
101             salvageFactor(k) = (1-(1+discount)^(-plant_life(k)+(length(study_years)-offset+1)))/(1-(1+discount)^(-plant_life(k)));
102         end
```

The first step is to calculate the salvage factor, which is done by implementing Equation (5) in our optimization module. This is shown in the code snippet above (lines 100-102), where a for loop is used to calculate the salvage factor of each of the plants, for each year during the study period.

```
113                 COST_SAL_vector = salvageFactor .* COST_CON_vector;
114                 COST_SAL{countConfig}(k) = (1+discount)^(-length(study_years)) * sum(COST_SAL_vector);
```

Once the salvage factor has been calculated, it is used to calculate the salvage value of each of the plants in a configuration for a particular year. This is the implementation of Equation (4) in our Matlab module shown in lines 113-114. The variable **COST_SAL{countConfig}(k)** is the output, which represents the salvage cost for each new plant added in a particular configuration year. This calculation is repeated for each of the newly added plants in each of the possible configurations for that year, and the values for each plant are stored in the vector **COST_SAL**.

c. Fuel costs

$$\overline{F}_{j,t} = (1+i)^{-t-0.5} * (FC_k * 10^{-5}) * \sum_k \left[H_k^b * E_k^b * H_k^c * \left(E_k - E_k^b\right)\right] \quad (7\text{-}6)$$

Here,

FC_k : Fuel Cost of unit k, given in cents/million kcals.
H_k^b : Heat rate at minimum operating level (kcal/kWh) of unit k.
H_k^c : Average incremental heat rate (kcal/kWh) of unit k.
E_k : Total energy (GWh) generated by unit k.
E_k^b : Base energy (GWh) of unit k.

Fuel costs are calculated for each possible configuration in a given year by implementing Equation (6) in our optimization module as shown in the code snippet below:

```
88          COST_FUEL_vector = (heatRate .* sum(baseEnergy_config{countConfig}) + heatRate_increase .* ...
89              sum(energy_config{countConfig}-baseEnergy_config{countConfig})) .* fuelCost;
90
91          COST_FUEL(countConfig) =  (1+discount)^(-tt-0.5) * sum(COST_FUEL_vector);  % Discounting by year
```

Lines 88-91 implement Equation (7-6) in two steps for each possible configuration in a given year, with the variable **COST_FUEL(countConfig)** storing the value. Once this calculation is repeated for all the possible configurations, the fuel costs for each of them is stored in the **COST_FUEL** vector.

d. Non-fuel operation and maintenance costs

$$\overline{OM}_{j,t} = (1+i)^{-t-0.5} * \sum [UFO\&M_k + UVO\&M_k * E_k] \quad (7\text{-}7)$$

Here,

$UFO\&M_k$: Unitary fixed O&M cost of unit k, expressed in monetary units per MW-year.
$UVO\&M_k$: Unitary variable O&M cost of unit k, expressed in monetary units per kWh.

Using information from the possible configurations for a given year and calculating the total generation capacity for each configuration, the operation and maintenance costs can then be calculated as follows:

```
93          totalCapacity = [number_of_units  configuration] .* unitCapacity;
94          COST_OM_vector = FOM .* totalCapacity *12*1000 + VOM .* sum(energy_config{countConfig})*1000;
95          COST_OM(countConfig) = (1+discount)^(-tt-0.5) * sum(COST_OM_vector);
```

The code snippet lines 93-95 show the steps involved in the calculation of operation and maintenance costs. The first step is shown in line 93, where the total generation capacity of a possible configuration is calculated. Using this information in lines 94-95, Equation (7-7) is implemented where the variable **FOM** in the code represents the variable $\boldsymbol{UFO\&M_k}$ and the variable **VOM** in the code represents the variable $\boldsymbol{UVO\&M_k}$ from the original equation. The vector **totalCapacity** on line 93 is multiplied by 12 to account for each month of the year and then multiplied by 1000 to convert from $/kW to $/MW. The sum of the entry in vector **energy_config** is multiplied by 1000 to account for the energy generated in GWh instead of MWh. The calculations are repeated for each possible configuration in a particular year and the results are stored in the **energy_config** vector.

e. Cost of the energy-not-served

$$\overline{ENS}_{j,t} = (1+i)^{-t'-0.5} * \left[a + \frac{b}{2} * \left(\frac{N_{t,h}}{EA_t}\right) + \frac{c}{3} * \left(\frac{N_{t,h}}{EA_t}\right)^2\right] * N_{t,h} \quad (8)$$

Here,

$N_{t,h}$: Amount of energy-not-served in year *t*.

EA_t : Energy demand (kWh) of the system in year *t*.

a, b and c : Polynomial coefficients for incremental cost of ENS defined by the user input.

The final component of the objective function is the energy-not-served cost. This has been implemented in our optimization module using the code snippet given below:

```
97        COST_ENS_vector = CF1 .* system_info.CT.eensStore(countConfig+1,:) .* periodHours' .* system_info.load_conf.peak(offset);
98        COST_ENS(countConfig) = (1+discount)^(-tt-0.5) * sum(COST_ENS_vector) * 10^3;   % $/kWh --> $/MWh
```

In our model, we used only one polynomial coefficient **a** which is represented by **CF1** in our code, line 97. As we consider the coefficients **b** and **c** to be zero, part of Equation (7-8) simplifies to the expression shown in line 97. Using calculated values of energy not served (kWh), period hours and peak load values for each possible configuration in a given year, the total ENS cost is calculated. This calculation is carried out for all possible configurations and is stored in the vector **COST_ENS**.

Once all the different cost components of Equation (1) have been calculated for each possible configuration of a given year, the optimization module moves onto the next year in the study and repeats these set of calculations. This process is repeated until the final year of the study is reached, and the costs for all possible configurations starting from the first year of the study are calculated.

7.3 Maintenance Mechanism

Prior to performing actual production costing, an estimated maintenance schedule must be prepared since it affects equipment availability in each of the time periods. It is not reasonable to ignore maintenance since the maintenance of units has a significant effect on the system's operating cost. To a large extent, the maintenance of generation equipment can be scheduled at times when system capacity reserves are greatest. The time requirements for scheduled maintenance outages depend on the type and size of a unit. A reasonable procedure is to schedule maintenance for the largest items of equipment when reserves are the greatest, schedule maintenance for the next largest items of equipment when remaining reserves are the greatest, etc. This procedure tends to levelize the operable equipment reserves for the system during the year. The approach to schedule the maintenance energy block is illustrated in Figure 7-2.

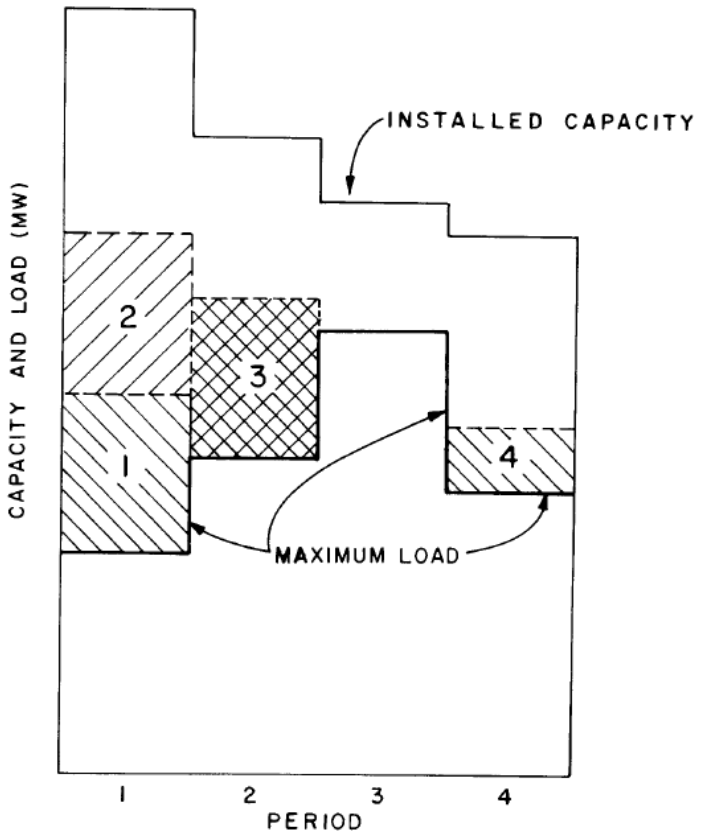


Figure 7-2 Illustration of maintenance schedule with energy blocks

The general steps can be summarized as follows:

- Determine the maintenance space in each period and maintenance class for each generator;
- Categorize all generating units in different sizes and classes with maintenance requirements organized as energy block;
- Schedule the energy block in each maintenance space slot with the principle: Largest energy block scheduled to largest maintenenace space in the rest unsatisfied maintenance requirements;
- Convert maintenance days and energy amount into availability (%).

The code snippet to perform the above steps are presented below:

```
function availability = getAvailability(year, configuration, system_info)
%   getAvailability Summary of this function goes here
%   The calculation procedure follows the mechanism described in pp.43-49
%   https://www.osti.gov/servlets/purl/5208341 to consider maintenance schedule.

... Do some initialization and parameter retriving HERE ...

    mainSpace = installedCapacity - maxLoad;
```

```
    mainSize = para(9,:);
    [class,~,ic] = unique(mainSize);

    for i = 1:length(class)
        largest = length(class)-i+1;
        classSize = class(largest);

        MWDAYS =
sum(para(2,ic==largest).*para(8,ic==largest).*number_of_gens(ic==largest));
        MAINBK = classSize * 91;
        NO = MWDAYS/MAINBK;
        NO_periods = zeros(1,number_periods);

        REMAIN = MWDAYS;
        while REMAIN > MAINBK
            [~, idx] = max(mainSpace);
            NO_periods(idx) = NO_periods(idx)+1;
            mainSpace(idx) = mainSpace(idx)- MAINBK/days(idx);
            REMAIN = REMAIN - MAINBK;
        end

        [~, idx] = max(mainSpace);
        NO_periods(idx) = NO_periods(idx) + REMAIN/MAINBK;
        mainSpace(idx) = mainSpace(idx)- REMAIN/days(idx);

        main_prob = NO_periods / NO;

        units = find(ic==largest);
        for k = 1:length(units)
            main_days = main_prob * para(8,units(k));
            main_rate = main_days ./ days;
            availability(:,units(k)) = (1-para(7,units(k))/100) * (1-main_rate)';
        end
    end

    availability = repmat(availability,2,1);
end
```

Figure 7-3 Implementation code of the maintenance mechanism

The above partial code snippet works for transforming the energy block shifting into availability of each generator unit. In other words, their capacity is discounted by this availability probability (%).

7.4 Energy Dispatch

The energy dispatch information is key to calculate all associated cost related to fuel and O&M determined by total energy generation (MWh or GWh).

```
function [LOLP, EENS, baseEnergy, energy] =
getEnergy(year,periodHours,system_info,loadingOrder,configuration)
```

The energy dispatch is calculated using the function *getEnergy* following some integral procedure for load duration curves. It is noteworthy that the energy integral should consider two-block representation similar to WASP, which distinguishes base energy and peak energy in the overall calculation. The loading

order is only valid among each sub category, implying all base energy with base capacity are preceeding peak energy and peak capacity scheduling.

7.5 Graph Structure and Search

Using the calculated cost results, we associated the total cost for each candidate configuration and mapped the expansion planning cost of generator configuration evolution to a graph structure that contains nodes and edges. In the graph structure mapping, each node stands for the destination configuration of that year, and each edge stands for the generator configuration evolution path from parent configuration to child configuration as described in the CONFIG module. The overall workflow of OPTIMIZE module can be presented in Figure 7-4 and summarized as follows.

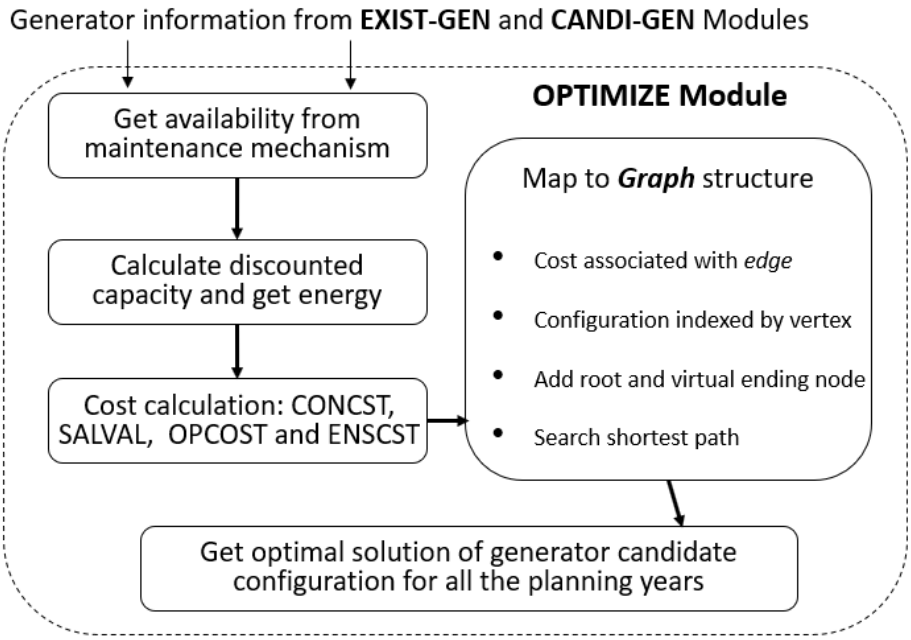


Figure 7-4 The workflow of OPTIMIZE module

- ***STEP 1***: Determine the loading order (through calculation of full-load-cost, FLC) and get the availability according to the maintenance mechanism, preparing this information for later energy dispatch calculation;
- ***STEP2***: Calculate the energy and base energy generated by each unit in each type of power plant, including renewables, using discounted capacity (considering availability) and shifted LDC curve (considering loading order) in each period;

- ***STEP3***: Process the cost calculation using the formulas presented in the last section, assign the cost for each possible configuration evolution rather than configuration itself. The same configuration coming from different parents does have different costs involved;
- ***STEP4****:* Map the cost and configuration index to a graph structure, the cost is weight for edge and the configuration index is for node;
- ***STEP5****:* Perform the dynamic programming or tree search algorithm to find the optimal solution that can be retrieved and stored for ELCC module.

The reason why the total cost of each configuration is associated with each edge, as weight, instead of each node is that we do care about the evolution path of each configuration rather than the simple configuration destination. Because in some parts of the cost calculation, for example CONSTRUCTION COST and SALVAGE VALUE, the cost or value should be calculated based on the newly added units. Only the added units in child configuration that are different from the parent configuration are considered and discounted in that year. In orther words, the same configuration may have different CONSTRUCTION COST and SALVAGE VALUE if it evolves from different configurations in the previous year.

In the following code snippet (lines 143-183), we built the graph structure by using the MATLAB built-in function *digraph* and track configuration in each layer (depth) of year. The method *addedge* is also used to assign the cost as weight to each edge linking parent-child configurations.

`G = digraph` creates an empty directed graph object, `G`, which has no nodes or edges

`G = addedge(G,s,t)` adds an edge to graph `G` between nodes `s` and `t`. If a node specified by `s` or `t` is not present in `G`, then that node is added. The new graph, `H`, is equivalent to `G`, but includes the new edge and any required new nodes.

```
143  %% Print the cost calculation results & build graph of the configuration tree
144  G = digraph;
145
146  countConfig = 0;
147  for year = study_years
148      offset = year-starting_year+1;
149      pStoreIndex = system_info.CT.yearlyConfigurationsMap(num2str(year));
150      yearlyConfig = system_info.CT.pStore(pStoreIndex,:);
151
152
153      for iConfig = 1:length(pStoreIndex)
154          countConfig = countConfig+1;
155          configuration = yearlyConfig(iConfig,:);
156
157          fprintf("YEAR %d: PRESENT WORTH COST ( K$ ) FOR CONFIGURATION: ",year);
158          disp(configuration);
159
160          parents = find(system_info.CT.connections(:,countConfig+1)==1);
161          for k = 1:length(parents)
162              configuration_parent = system_info.CT.pStore(parents(k),:);
163              fprintf("If evolving from configuration: ");
164              disp(configuration_parent);
165              fprintf("CONCST %.0f, SALVAL %.0f, OPCOST %.0f, ENSCST %.0f, TOTAL %.0f \n\n",...
166                  COST_CON{countConfig}(k)/1000,...
167                  COST_SAL{countConfig}(k)/1000,...
168                  (COST_FUEL(countConfig)+COST_OM(countConfig))/1000,...
169                  COST_ENS(countConfig)/1000,...
170                  (COST_CON{countConfig}(k)-COST_SAL{countConfig}(k)...
171                  +COST_FUEL(countConfig)+COST_OM(countConfig)+COST_ENS(countConfig))/1000);
172
173              G = addedge(G,parents(k),countConfig+1,...
174                  (COST_CON{countConfig}(k)-COST_SAL{countConfig}(k)...
175                  +COST_FUEL(countConfig)+COST_OM(countConfig)+COST_ENS(countConfig))/1000);
176          end
177
178          % Add the virtual ending node to facilitate the tree search
179          if year == ending_year
180              G = addedge(G,countConfig+1,size(system_info.CT.pStore,1)+1,1);
181          end
182      end
183  end
```

Figure 7-5 Screen capture of building graph structure

It is also noteworthy that we added a virtual ending node after the last year in order to facilitate the tree search process introduced in the code snippet (lines 179-181). All the evolved configuration candidates at the last year are assumed to connected to this virtual ending node. The resultant graph structure is as shown in Figure 7-6. The numbers (1-79) are the index of each valid configuration candidate. The configurations at the same layer belong to the same year.

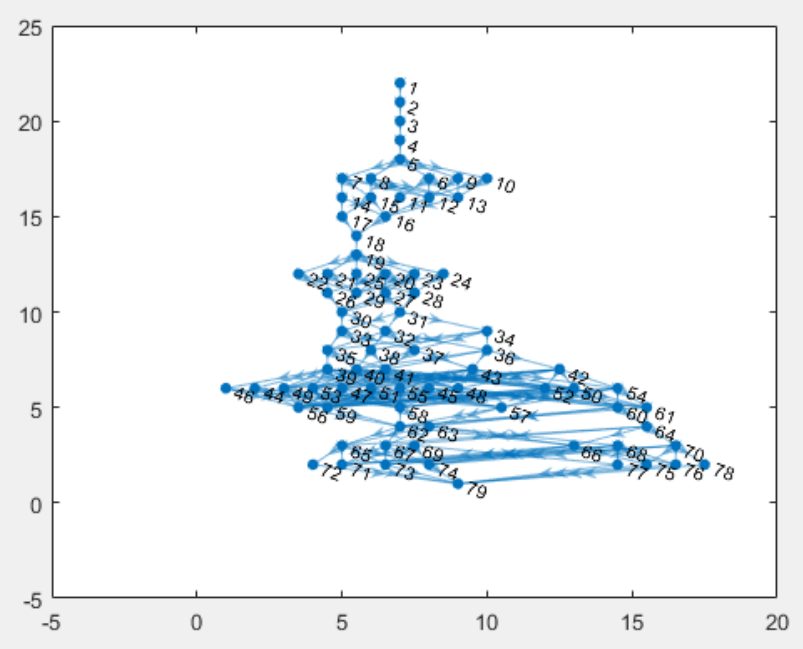


Figure 7-6 Graph structure of example configuration candidates

After the graph structure is built, the least cost expansion plan (configuration evolution) across all the studied years can be mathematically equivalent to finding a shortest path with the least distance that represents the total weight (cost) over all the pass-by edges. Again, we used the MATLAB built-in function *shortestpathtree* to help us achieve such a goal.

```
TR = shortestpathtree(G,s,t) computes the tree of shortest paths between multiple source or target nodes.
```

The functionality is shown in the following code snippet (lines 190-210) to check the final cost results. The start node is node 1 (i.e., the first study year), and the destination node is the virtual ending node (indexed as #last configuration + 1).

```
190  % Use the graph structure and MATLAB built-in function to find the optimal
191  % solution mapping analogous to searching the shortest path
192  [TR, D] = shortestpathtree(G,1,size(system_info.CT.pStore,1)+1);
193  %figure(2)
194  %pTR = plot(TR,'Layout','layered');
195
196
197  disp('--------------------------------------------------------------------------------');
198  disp('---------------**********************************************---------------');
199  disp('---------------* OPTIMAL GENERATION EXPANSION PLAN GNERATED *---------------');
200  disp('---------------**********************************************---------------');
201  disp('--------------------------------------------------------------------------------');
202
203  optimal_configuration = zeros(length(study_years),totalCandiType);
204  sucID = 2;
205 for year = study_years
206      fprintf("THE OPTIMAL CONFIGURATION FOR YEAR %d: ",year);
207      disp(system_info.CT.pStore(sucID,:));
208      optimal_configuration(year-starting_year+1,:) = system_info.CT.pStore(sucID,:);
209      sucID = successors(TR,sucID);
210  end
```

Figure 7-7 Screen capture of tree search code structure for optimal solution

The returned variable TR (line 192) is also a graph structure that can be further retrieved by method *successors* for the optimal solution (line 209). It is noteworthy that we begin retrieving our expansion plan from configuration ID=2 since it is actually the first valid configuration at first study year after root (ID=1) configuration.

7.6 Input of OPTIMIZE Module

Inputs to OPTIMIZE module are all the outputs from previous introduced modules, and can be listed as follows:

- Energy-not-served penalty cost information from LOAD-CALC module
- Existing generator capacity information from EXIST-GEN module
- Existing generator configuration information from EXIST-GEN module
- Candidate generator capacity information from CANDI-GEN module
- Candidate generator configuration information from CANDI-GEN module
- Valid configuration candidates and their construction & operation cost information from CONFIG module

7.7 Output of OPTIMIZE Module

Outputs of the OPTIMIZE module include:

- Optimal generation expansion plans in different years
- The total cost and separate cost elements for each configuration evolution in different years
- All the associated LOLP and ENS information for further decision-making

The variables above is saved in a '.mat' file under the path of *'root/projects/project_1/project_data/OPTIMIZE.mat'* for the *OPTIMIZE* module. The output

information is also stored in the structure variable *system_info* as *optimal_configuration* that is depicted in Figure 7-8.

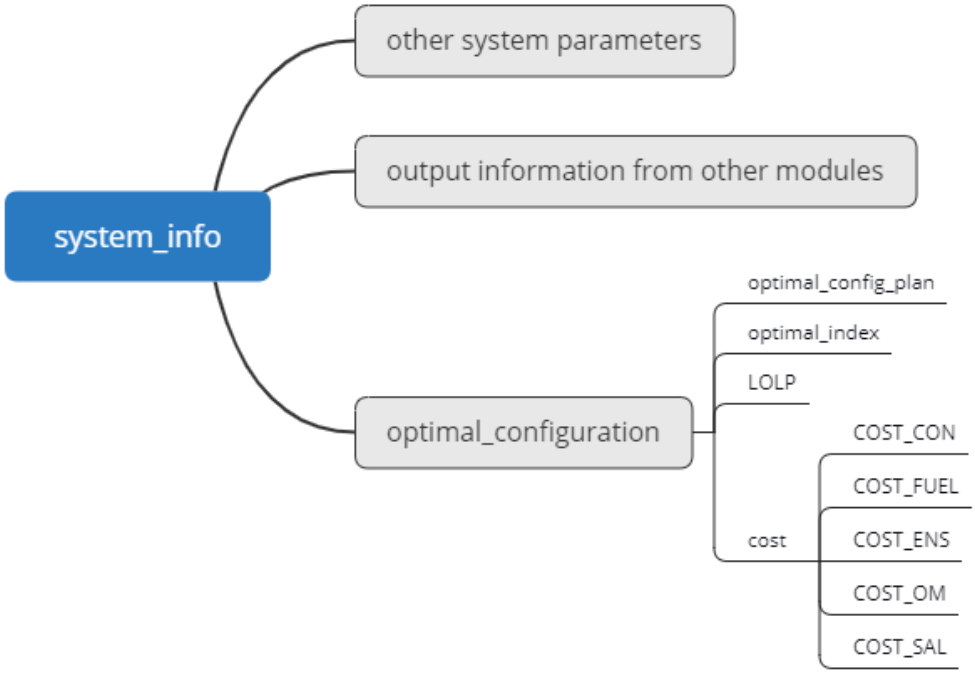


Figure 7-8 The output structure of optimal generation expansion plan information

The following Figure 7-9 also shows the cost calculation result for several example configurations, considering five types of conventional power plants and two types of renewables (solar PV and wind farm). It is noteworthy that even the single one configuration (4 4 5 9 3 1 2) at year 2015 can have different cost values (CONCST, SALVAL) if it evolves from different parent configurations (4 4 4 9 2 1 2), (4 4 5 9 2 1 2) or (4 3 5 9 2 1 2).

```
YEAR 2015: PRESENT WORTH COST ( K$ ) FOR CONFIGURATION:      4     4     6     8     3     1     2

If evolving from configuration:      4     4     6     8     2     1     2

CONCST 288814, SALVAL 212624, OPCOST 348720, ENSCST 6051, TOTAL 430961

YEAR 2015: PRESENT WORTH COST ( K$ ) FOR CONFIGURATION:      4     4     4     9     3     1     2

If evolving from configuration:      4     4     4     9     2     1     2

CONCST 288814, SALVAL 212624, OPCOST 347776, ENSCST 5884, TOTAL 429850

YEAR 2015: PRESENT WORTH COST ( K$ ) FOR CONFIGURATION:      4     4     5     9     3     1     2

If evolving from configuration:      4     4     4     9     2     1     2

CONCST 364208, SALVAL 267362, OPCOST 349702, ENSCST 3352, TOTAL 449900

If evolving from configuration:      4     4     5     9     2     1     2

CONCST 288814, SALVAL 212624, OPCOST 349702, ENSCST 3352, TOTAL 429244

If evolving from configuration:      4     3     5     9     2     1     2
```

Figure 7-9 Print-out of several example configuration evolution

The Figure 7-10 shows the final optimal generation expansion planning solution that stands for the least cost expansion plan over studied years (1998-2017). The seven columns indicate five types of conventional power plants and two types of renewables (solar PV and wind farm). The numerical values are actually cumulative results. For instance, compared with year 2016, one unit of third type (column) and one unit of fifth type (column) are installed at the final studying year 2017.

-------- PRESENT WORTH COST OF THE YEAR (K$)--------

YEAR	OPCOST	ENSCST	LOLP	CONFIGURATION						
2017	355689	13420	0.269%	4	5	5	10	4	1	2
2016	380316	18016	0.327%	4	5	4	10	3	1	2
2015	397112	25131	0.411%	4	4	4	9	3	1	2
2014	429349	21989	0.345%	4	4	4	9	2	1	2
2013	445816	19476	0.292%	4	3	3	8	2	1	2
2012	464159	22917	0.318%	4	3	2	7	2	1	2
2011	488192	15201	0.209%	4	3	2	6	2	1	2
2010	522448	30814	0.367%	4	3	1	6	1	1	2
2009	546480	32681	0.365%	4	3	1	5	1	1	2
2008	563519	32379	0.342%	4	3	0	4	1	1	2
2007	611988	40197	0.392%	4	3	0	4	0	1	2
2006	638362	50725	0.456%	4	3	0	3	0	1	2
2005	663883	32208	0.290%	4	3	0	2	0	1	2
2004	701371	22026	0.195%	4	2	0	2	0	0	0
2003	730130	35727	0.285%	3	2	0	2	0	0	0
2002	758152	61121	0.438%	2	1	0	2	0	0	0
2001	694902	6199839	17.499%	0	0	0	0	0	0	0
2000	727081	5066751	14.055%	0	0	0	0	0	0	0
1999	757964	3844066	10.700%	0	0	0	0	0	0	0
1998	758300	6216943	15.717%	0	0	0	0	0	0	0

Figure 7-10 Print-out of the final optimal solution of expansion plan

8.0 ELCC Module

8.1 Overview of ELCC Module

ELCC is effective load carrying capability module that is responsible to calculate ELCC value of each power plant, including renewables selected as expansion planning candidates. The general idea is to define the capacity credit of each new unit as the capacity added to a system, allowing the load to increase without compromising the generation adequacy. For implementation, the functionality of this module can be explained as follows: first, take a note of the original system installed capacity (MW) and LOLP from the OPTIMIZE module as benchmark values; second, remove the generator unit of interest and rerun the OPTIMIZE module to locate the revised system installed capacity (MW) that is able to keep the original LOLP. The difference between the benchmark capacity and the revised system capacity is the ELCC of the generator in question. The module is rerun to obtain ELCC of all the studied generation units.

8.2 Theoretical Analysis of ELCC values

ELCC stands for effective load carrying capability module. ELCC represents the generator's contribution to system reliability. It is the additional firm load that can be met by an incremental generator while maintaining the same level of system reliability. ELCC measures the contribution of an individual generator to system capacity with and without the generator of interest. This module is responsible for calculating ELCC value of each power plants, including renewable generation units, that are selected as expansion candidates. Theoretically, the most well-known method for ELCC calculation is Garver's method. The general idea is to define the capacity credit of each new unit as the capacity added to a system, allowing the load to increase without compromising the generation adequacy. When unit g is added to the system, the risk of power deficit decreases from $LOLP_{g-1}$ to $LOLP_g$. The load carrying capability of unit g is defined as the largest constant load, C_{ELCC}, which can be added to the system without the risk of power deficit exceeding the earlier level $LOLP_{g-1}$. The equivalent load duration curve including unit g and the constant load is given by the following Equation (8-1) and (8-2) and illustrated in Figure 8-1.

$$\begin{aligned}\tilde{F}_{ELCC}(x) &= P(E_g + C_{ELCC} > x) \\ &= P(E_g > x - C_{ELCC}) = \tilde{F}_{E_g}(x - C_{ELCC})\end{aligned} \tag{8-1}$$

$$C_{ELCC} = \hat{G}_{tot} + \hat{G}_g - \tilde{F}_{E_g}^{-1}(LOLP_{g-1}) \tag{8-2}$$

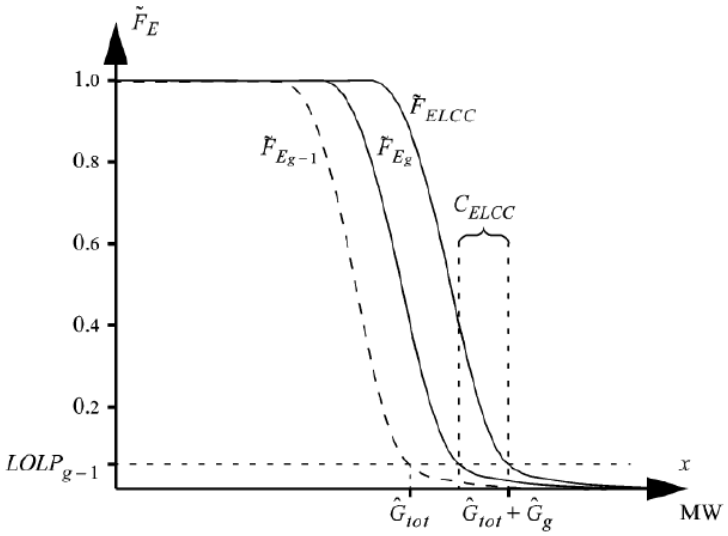


Figure 8-1 ELCC calculation illustration

In the implementation, the ELCC calculation process can be explained as follows: first, take a note of the original system installed capacity (MW) and LOLP from the OPTIMIZE module as benchmark values. Second, remove the generator of interest and rerun the OPTIMIZE module with a new constraint to keep the original LOLP. Then, take a note of the revised system installed capacity (MW). The difference between the benchmark capacity and the revised system capacity is the ELCC of the generator in question. The module is rerun to obtain ELCC of all power plants.

```
84 -            ldccurves{period} = ldccurves{period}.process(normalized_capacities{period,columnEnd},...
85                  probabilities{period,columnEnd}, STEP);
86 -            xi = 0:STEP:2;
87 -            [yi, index] = unique(ldccurves{period}.poly(xi));
88 -            newCap = interp1(yi,xi(index),lolp,'spline');
89 -            ELCC_periods(period) = totalCapacity - newCap;
```

In the above code snippet (lines 84-89), the MATLAB built-in function *interp1* was used to retrive x value by setting y value in a functional relationship.

`vq = interp1(x,v,xq)` returns interpolated values of a 1-D function at specific query points using linear interpolation.

In line 89, the *totalCapacity* is the actual installed total capacity including newly added generator, and *newCap* is the newly obtained capacity by satisfying the same reliability level. Finally, all the generator unit's ELCC value were printed out in different years and different periods under the optimal configuration scenario. For renewables, like wind (second last column) and solar (last column), they may have significant different ELCC values in different periods due to the change of weather conditions. These are shown in Figure 8-2.

0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
475.8686	0	0	485.1103	0	0	26.3021
482.8511	0	0	490.1450	0	0	26.2348
473.4623	239.5317	0	482.4943	0	0	26.3335
470.7748	238.2949	0	480.5402	0	0	26.2204
477.2142	240.2112	0	485.6368	0	0	26.3317
474.8579	238.9931	0	483.5143	0	0	26.1026
476.8271	240.0053	0	485.3450	476.8271	0	25.9891
476.9975	239.3873	239.3873	485.1225	476.9975	0	26.1080
478.3492	239.9817	239.9817	486.2825	478.3492	13.0833	26.1148
477.7957	240.0484	240.0484	485.5640	477.7957	13.0992	26.1552
477.6978	240.5623	240.5623	485.8177	477.6978	13.1283	26.0083
477.2022	239.9507	239.9507	485.1745	477.2022	13.0952	26.1831
479.8316	240.8548	240.8548	487.1181	479.8316	13.1273	26.1191
481.6077	240.3337	240.3337	488.5301	481.6077	13.0699	26.1169
479.9194	240.1997	240.1997	487.2954	479.9194	13.0864	26.0516
479.1508	240.1345	240.1345	486.6511	479.1508	13.0968	26.0348

Figure 8-2 Print-out of ELCC values for the first period over 20 years

Additionally, because of the ELCC of each incremental resource depends on the whole portfolio of renewable resources, the ELCC function in WASP including renewable parts is actually a multidimensional surface. According to some studies, the marginal ELCC of one renewable resource technology declines as its penetration increases. For solar PV, high capacity credit at low penetrations while this rapidly decreases as additional capacity is added. A renewable portfolio that contains a diverse set of power technologies, such as thermal, hydro, wind can mitigate the decline in ELCC.

8.3 Input of ELCC Module

Inputs to ELCC module are outputs from previous introduced modules, and can be listed as follows:

- LDC curve information from LOAD-CALC module
- Existing generator capacity information from EXIST-GEN module
- Candidate generator capacity information from CANDI-GEN module
- Valid configuration information from CONFIG module

8.4 Output of ELCC Module

Outputs of the ELCC module are the ELCC values calculated for each valid configuration candidate in each time period in different years. The variables above is saved in a '.mat' file under the path of *'root/projects/project_1/project_data/ELCC.mat'* for the *OPTIMIZE* module. The output information is also stored in the structure variable *system_info* as *ELCC* that is depicted in Figure 8-3.

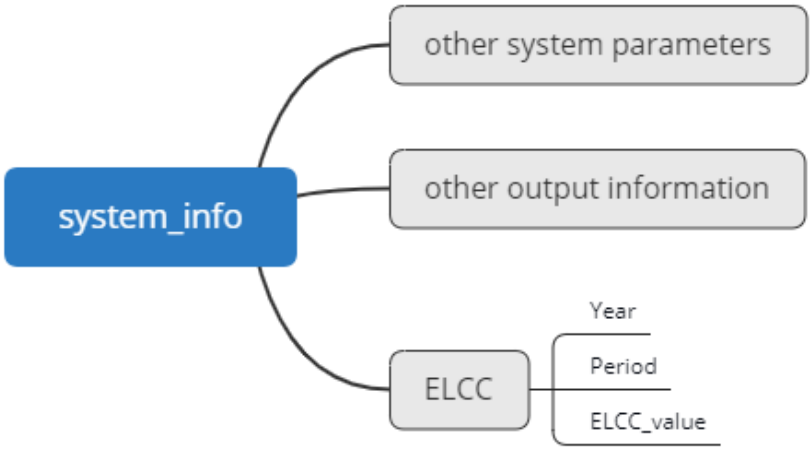


Figure 8-3 The output structure of ELCC value information

9.0 Validation and Case Study

9.1 Validation using WASP

The developed MATPLAN package was validated with the well-known Wien Automatic System Planning (WASP) package. To validate MATPLAN, results from MATPLAN were compared with that of WASP. This allowed us to verify the validity of MATPLAN in performing power system expansion planning with traditional power plants. All studies were carried out in a PC – Intel i5 with 8GB RAM running Windows 10. The newest version of WASP-IV package was used.

Validation results confirm that MATPLAN is able to deal with expansion planning with conventional power plants. The WASP demo example was used as the validation case study. Under this case study, the characteristics of six existing conventional power plants in the case study are listed in Table 9-1. These existing conventional power plants are: two lignite plants (FLG1 and FLG2), one coal plant (FCOA), one oil plant (FOIL), one gas turbine plant (FGT) and one natural gas plant (F-CC). Table 1 also summarizes the characteristics of five candidate power plants. These candidate power plants include the following types: natural gas plant (VCC), lignite plant (VLG1 and VLG2), coal plant (VCOA) and nuclear power plant (NUCL).

Table 9-1 Important parameters of the existing conventional power plants

Name of plants	FLG1	FLG2	FCOA	FOIL	FGT	F-CC	V-CC	VLG1	VLG2	VCOA	NUCL
No. of units	4	9	1	7	4	1	-	-	-	-	-
Base capacity (MW)	150	150	400	80	50	87	300	150	150	400	300
Max capacity (MW)	270	276	580	145	50	174	600	280	280	580	600
Forced outage rate (%)	10.0	8.9	8.0	7.3	6.0	15.0	10.0	10.0	10.0	8.0	10.0
Maintenance days	56	56	48	42	42	28	28	56	56	48	42
Fixed O&M cost ($/kW-month)	4.06	1.91	2.92	4.57	8.35	2.10	2.10	2.70	2.70	2.92	2.50
Variable O&M cost ($/MWh)	4.90	2.00	5.00	1.60	1.60	5.00	4.00	6.00	6.00	5.00	0.50
Fuel cost (cent/million kcals)	600	495	800	833	420	1266	1200	710	1100	800	194

During the 20-year planning horizon from 1998 to 2017, Table 9-2 summarizes the unit retirement/addition information of generating plants, together with their tunnel width and yearly peak load. The unit retirement/addition is indicated by -1/+1, respectively, in each year of the planning horizon. The tunnel width restricts the minimum and maximum number of units allowed to be installed. For example, [1,2] for VLG1 in year 2005 implies that the minimum number of VLG1 to be added is one (1) and the maximum increment is two (2) – this means in 2005, the number of VLG1 that can be added are 1, 2 and 3. If not specified, the tunnel width is as the same as that of the previous year.

Table 9-2 Yearly information for the studied system

Year	FLG1	FLG2	FCOA	FOIL	FGT	F-CC	V-CC	VLG1	VLG2	VCOA	NUCL	Peak load (MW)
	Retirement/Addition						Tunnel width					
1998	/	/	/	/	/	/	/	/	/	/	/	6000.00
1999	/	/	+1	/	/	/	/	/	/	/	/	6333.00
2000	/	/	/	/	/	+1	/	/	/	/	/	6725.00
2001	/	/	/	/	/	+1	[0,2]	/	/	/	/	7109.01
2002	/	/	/	/	/	/	/	[0,2]	/	[0,2]	/	7496.45
2003	-1	/	/	/	/	/	[1,2]	/	/	/	/	7897.51
2004	/	/	/	/	/	/	[2,2]	/	/	/	/	8304.23
2005	/	/	/	/	/	/	/	[1,2]	/	/	/	8702.83
2006	/	-1	/	/	/	/	/	/	/	[1,2]	[0,1]	9120.57
2007	/	/	/	/	/	/	/	/	[0,2]	[2,2]	/	9558.36
2008	/	/	/	/	/	/	/	/	/	/	/	10017.20
2009	/	-1	/	/	-1	/	/	/	/	[3,2]	/	10488.00
2010	/	/	/	/	/	/	/	/	[1,2]	[4,2]	/	10980.90
2011	/	/	/	/	/	/	/	/	[2,2]	/	[0,2]	11497.00
2012	/	/	/	-1	/	/	/	/	/	[5,2]	/	12025.90
2013	/	/	/	-1	/	/	/	[2,2]	[3,2]	[6,2]	/	12579.10
2014	-1	-1	/	/	/	/	/	/	[4,2]	[7,2]	/	13157.70
2015	/	/	/	/	/	/	/	/	/	/	[1,2]	13749.80
2016	/	/	/	/	/	/	/	[3,2]	/	[8,2]	/	14368.50
2017	/	/	/	/	/	/	/	/	[5,2]	/	[2,2]	15015.10

Considering only conventional power plants, each year was divided into four periods according to different seasons. Peak load in each period has a different peak load ratio, which is 0.90, 0.87, 0.93, 1.00 of the annual peak loads for each season. By using the same constraints on reserve margin and LOLP, MATPLAN can produce the same feasible candidates and the same optimal solution as WASP, as shown in Table 9-3.

Hence, it can be concluded that the developed MATPLAN has been successfully validated with WASP, and both of which generate the same optimal configurations in different study years. Some slight mismatch of LOLP values was observed, which was expected to be caused by numerical issues associated with different LDC representation methods. This can be improved by using a detailed modeling of LDC curves with more accurate point-wise load data.

Table 9-3 Optimal solutions from MATPLAN vs. WASP

	MATPLAN						**WASP**					
Year	V-CC	VLG1	VLG2	VCOA	NUCL	LOLP (%)	V-CC	VLG1	VLG2	VCOA	NUCL	LOLP (%)
1998	0	0	0	0	0	15.717	0	0	0	0	0	14.924
1999	0	0	0	0	0	10.700	0	0	0	0	0	10.485
2000	0	0	0	0	0	14.055	0	0	0	0	0	12.899
2001	2	0	0	0	0	3.599	2	0	0	0	0	3.349
2002	2	0	0	1	0	2.766	2	0	0	1	0	2.595
2003	3	0	0	1	0	3.062	3	0	0	1	0	2.905
2004	4	1	0	2	0	0.397	4	1	0	2	0	0.537
2005	4	3	0	2	0	0.332	4	3	0	2	0	0.459
2006	4	3	0	3	1	0.116	4	3	0	3	1	0.212
2007	4	3	0	3	1	0.427	4	3	0	3	1	0.555
2008	4	3	0	4	1	0.384	4	3	0	4	1	0.510
2009	4	3	1	5	1	0.407	4	3	1	5	1	0.525
2010	4	3	1	6	1	0.407	4	3	1	6	1	0.517
2011	4	3	2	6	2	0.234	4	3	2	6	2	0.337
2012	4	3	2	7	2	0.353	4	3	2	7	2	0.458
2013	4	3	3	8	2	0.323	4	3	3	8	2	0.421
2014	4	4	4	9	2	0.380	4	4	4	9	2	0.472
2015	4	4	5	9	3	0.277	4	4	5	9	3	0.369
2016	4	4	5	10	3	0.359	4	4	5	10	3	0.449
2017	4	5	5	10	4	0.296	4	5	5	10	4	0.384

In addition, MATPLAN was also tested using realistic field data to showcase its applicability in a real-world environment considering renewable energy (both solar PV and wind farms) in the generation mix. This is discussed below.

9.2 Case 1: Low Penetration of Renewable Energy Sources in the Generation Mix

This case study describes the use of MATPLAN for generation expansion planning with low penetration of renewable energy, including wind and solar power. The study years were from 2019 to 2038. In order to fully consider the volatile contribution of renewable energy to the overall system capacity in different periods, especially for solar plants (which produce no output during the nighttime), year-division-by-day and day-division-by-hour attributes were added in the setting file to let the user define how a year/a day could be divided into different sections. For example, a year could be divided into four different sections, as follows: days 1-91, days 92-183, days 184-274, and lastly days 275-365. Currently, there is no limit on the number of sections but higher number of sections result in longer computation time. On the other hand, the day-division-by-hour attribute is intended to let users split each day into two different periods: day and night. For instance, for the first section of the year (days 1-91), the daytime period is 07:00 to 18:00, and the nighttime period is 18:00 to 07:00 of the next day. The above division results in eight periods, as summarized in Table 9-4. Each period has different LDC representations and different renewable generation profiles.

Table 9-4 Time period division for renewable energy

Index No.	Time period
Period 1	07:00 - 18:00 in days 1 - 91
Period 2	06:00 - 19:00 in days 92 - 183
Period 3	07:00 - 20:00 in days 184 - 274
Period 4	08:00 - 17:00 in days 275 - 365
Period 5	18:00 - 07:00 in days 1 - 91
Period 6	19:00 - 06:00 in days 92 - 183
Period 7	20:00 - 07:00 in days 184 - 274
Period 8	17:00 - 08:00 in days 275 - 365

In the case study, the annual wind production data were obtained from the U.S. National Renewable Energy Laboratory (NREL) wind integration data sets. The solar production data were collected through realistic field measurements for on-site PV panels. Their normalized power output in a typical example year is depicted in Figure 9-1. Their cost characteristics are provided in Table 9-5. The parameters of conventional generators are kept the same as those in the validation case.

Table 9-5 Characteristics of candidate renewable energy generators

	Wind	Solar
Capacity (MW)	30	50
Forced-outage-rate (%)	5.0	5.0
Plant life (years)	20	25
Constructions cost ($/MW)	1094	1400
Fixed O&M cost ($/kW-month)	3.67	0
Variable O&M cost ($/MWh)	4.0	0

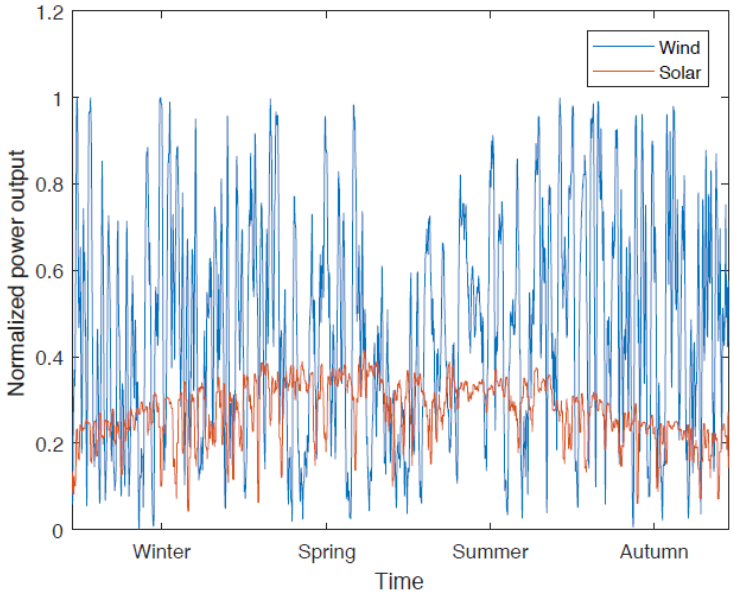


Figure 9-1 Normalized power output of PV and wind farms during one year (using a 24-hour moving average window)

Using MATPLAN, the optimal solution of the generation expansion evolution path is presented in Figure 9-2. We can observe that the renewable energy generators are incorporated into the expansion plan in early years due to their economic advantages, including low maintenance costs and no fuel cost.

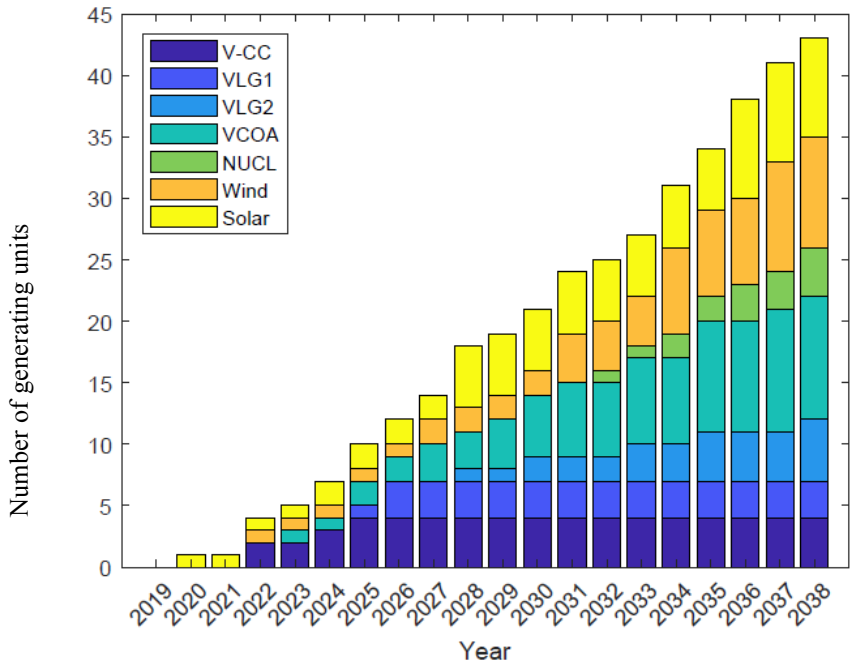


Figure 9-2 Optimal expansion plan – the case of low renewable energy penetration

The ELCC values of each candidate generator in different periods of the last study year 2038 are provided in Table 9-6.

Table 9-6 ELCC of candidate generators in different periods

ELCC	V-CC		VLG1		VLG2		VCOA		NUCL		Wind		Solar	
	MW	%	MW	%	MW	%	MW	%	MW	%	MW	%	MW	%
Period 1	448.3	74.7	216.9	77.5	216.9	77.5	451.3	77.8	448.3	74.7	10.2	34.0	21.3	42.6
Period 2	432.3	72.1	209.3	74.8	209.3	74.8	435.6	75.1	432.3	72.1	8.9	29.6	20.0	40.0
Period 3	465.9	77.7	225.6	80.6	225.6	80.6	468.5	80.8	465.9	77.7	11.6	38.7	23.2	46.4
Period 4	503.9	84.0	243.7	87.1	243.9	87.1	505.7	87.2	503.9	84.0	15.0	50.1	25.7	51.5
Period 5	449.4	74.8	218.0	77.7	218.0	77.6	451.9	77.8	449.4	74.7	10.2	34.0	0	0
Period 6	433.1	72.2	209.1	74.7	209.1	74.7	436.3	75.2	433.1	72.2	8.9	29.6	0	0
Period 7	466.1	77.7	227.6	81.6	225.6	80.6	468.5	80.8	465.9	77.7	11.6	38.7	0	0
Period 8	504.0	84.0	244.2	87.1	243.9	87.1	506.7	87.3	504.9	84.1	15.0	50.1	0	0

It can be observed that the renewable energy generators usually have much less ELCC values compared to those of conventional generators due to their intermittent nature. This is especially for solar power plants that have no capacity credit at nighttime. Most generators also have different ELCC values in different periods due to the different shapes of ELDC curves in these periods while considering the same reliability levels. It is noteworthy that the ELCC values of conventional generators during the daytime (Period 1 - Period 4) and nighttime (Period 5 - Period 8) are not necessarily the same if LDC curves are modeled separately, and hence they are different during daytime and nighttime.

9.3 Case 2: High Penetration of Renewable Energy Sources in the Generation Mix

This case scenario accounts for 30%-35% of the total system capacity at the end of the planning horizon. In this case study, MATPLAN was used to determine the optimal expansion plan with high level (30-50%) of renewable energy penetration. Meanwhile, the LOLP was kept at the same level or less by calibrating the reasonable capacity contribution of renewable energy generators. The resulting MATPLAN's optimal expansion plan is depicted in Figure 9-3. Figure 9-4 shows how the 30-35% renewable energy penetration target has been reached in the final study year of 2038.

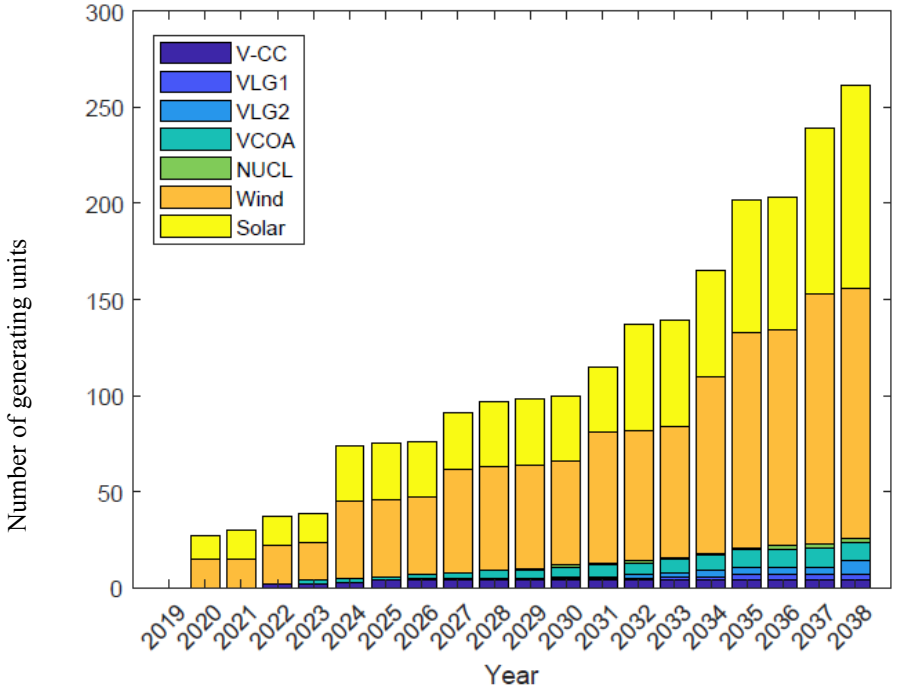


Figure 9-3 Optimal expansion plan – the case of high renewable energy penetration

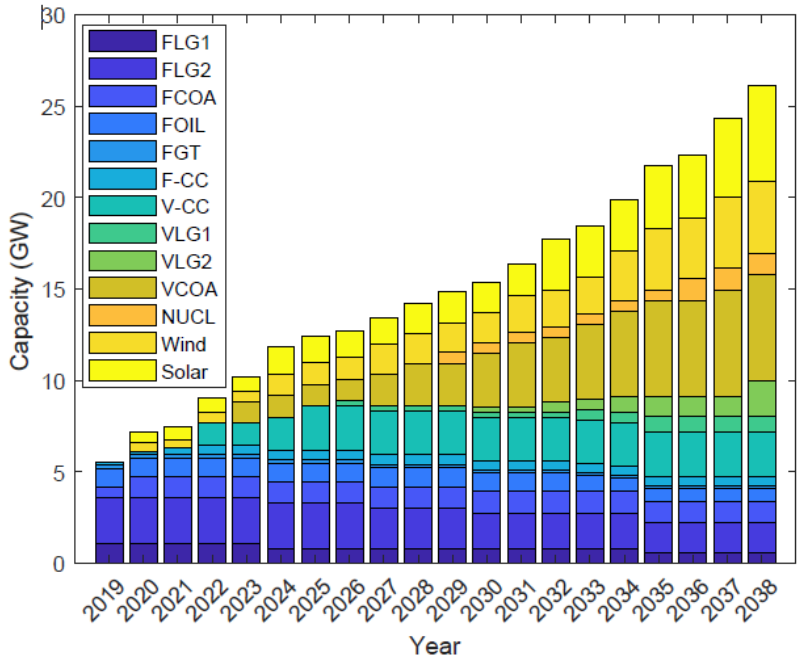


Figure 9-4 The total system installed capacity - the case of high renewable energy penetration

It is observed in Figure 9-5 that even with the increase in renewable energy penetration over the study years, the LOLP values were still kept at a quite low level if all renewable energy sources were treated as positive power output generators (not using negative load methods). It can also be observed that the nighttime LOLP values were larger than the daytime LOLP values due to the volatile capacity contribution of solar power.

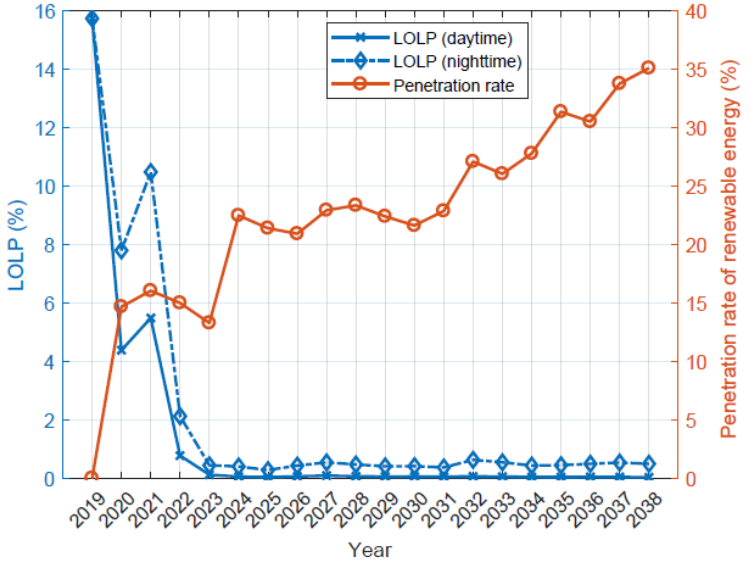


Figure 9-5 LOLP – the case of high renewable energy penetration

9.4 Case 3: Accounting for Multiple Locations of Renewable Energy Sources.

In this test case, different installed capacities of wind/solar power plants located in multiple locations were studied to showcase the flexibility of MATPLAN in dealing with diverse input datasets. The datasets came from (i) the NREL Grid Modernization project that provides national-wide wind/solar datasets; (ii) the Eastern Wind Integration Data Set and Western Wind Integration Data Set (https://www.nrel.gov/grid/wind-integration-data.html) that include meteorological conditions and turbine power for more than 126,000 sites in the continental United States for the years 2007–2013; and (iii) the Solar Power Data for Integration Studies (https://www.nrel.gov/grid/solar-power-data.html), which consist of 1 year (2006) of 5-minute solar power and hourly day-ahead forecasts for approximately 6,000 simulated PV plants.

The renewable energy sources, including their locations and characteristics used in this case study, are summarized in Table 9-7.

Table 9-7 Multiple locations of selected renewable energy sources

Name	WIND1	WIND2	SOLAR1	SOLAR2
Type	Wind	Wind	Solar	Solar
Coordinate	38.477, -79.675	38.174, -84.408	36.650, -78.250	36.350, -115.950
State	Virginia	Kentucky	Virginia	Nevada
County	Highland	Fayette	Albemarle	Nye
Capacity (MW)	30	20	50	150

The result of MATPLAN's optimal expansion planning considering multiple locations of renewable energy sources is presented in Figure 9-6, in which renewable energy sources, including two wind farms (WIND1 30MW and WIND2 20MW) and two solar farms (SOLAR1 50MW and SOLAR2 150MW) dominate the newly added generating units. Since thermal plants usually have larger capacity than renewable energy sources, they make up majority of the system capacity, which can be observed in Figure 9-7.

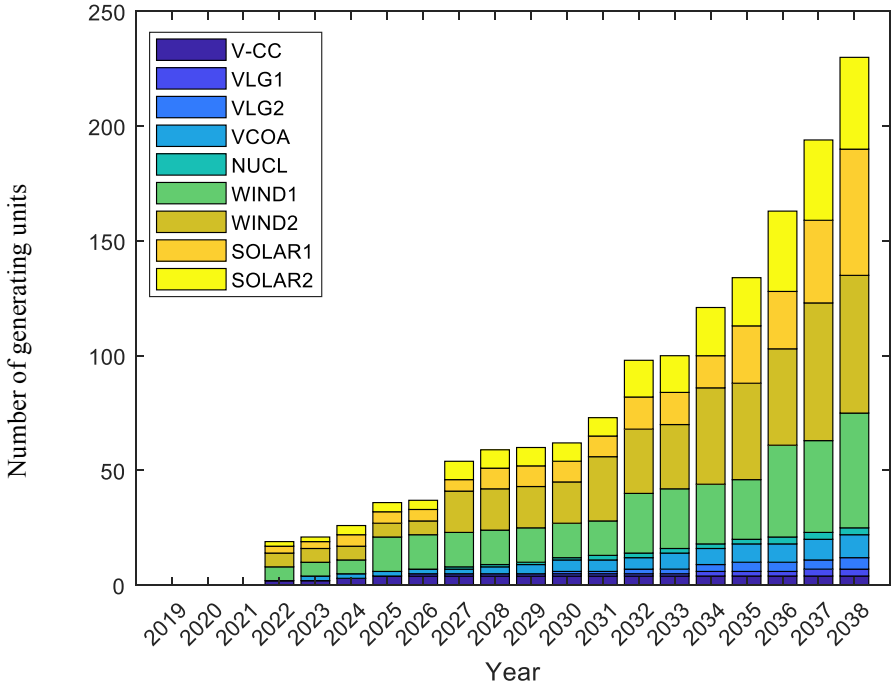


Figure 9-6 Optimal expansion plan – the case of multiple locations of renewable energy sources

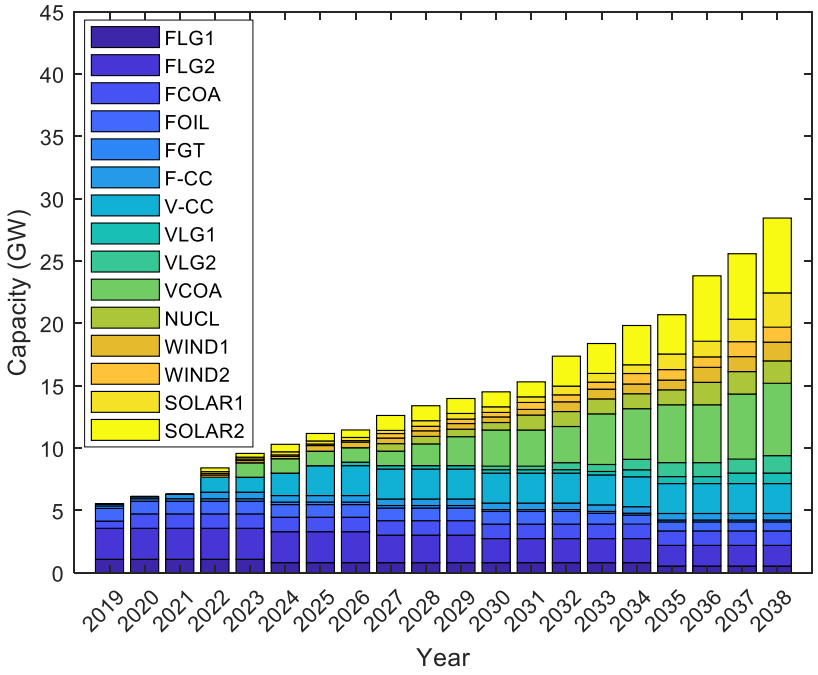


Figure 9-7 Total system installed capacity – the case of multiple locations of renewable energy sources

9.5 Case 4: Different Cost Structure of Different Types of Solar Panels

Different cost structures of different types of solar panels (e.g., monocrystalline, polycrystalline, fixed-tilt, one-axis-tracking) were considered by taking into account different cost specifications. In order to see the impact of different types of solar panels (e.g., monocrystalline, polycrystalline, fixed-tilt, one-axis-tracking) with different cost structures on MATPLAN's output, specifications of year-by-year construction costs were added as one of the inputs to MATPLAN. The cost data were extracted from NREL technical report: Cost-Reduction Roadmap for Residential Solar Photovoltaics (PV), 2017–2030. (https://www.nrel.gov/docs/fy18osti/70748.pdf). Some general cost information associated with different types of solar panels are summarized in Table 9-8.

Table 9-8 Cost structures (2017) of different types of solar panel

Type	Residential	Commercial	Fixed-tilt utility-scale	One-axis-tracking utility-scale
Size Range	3-10kW	10kW-2MW	>2MW	>2MW
Cost	2.80 $/W	1.85 $/W	1.03 $/W	1.11 $/W
Soft costs portion (e,g, land acquisition, tax)	65%	50%	30%	32%
Labor costs portion	10%	5%	10%	8%
Hardware	10%	20%	25%	25%
Inverter	5%	5%	5%	5%
Module	10%	20%	30%	30%

Thus, by considering the different cost structures of different types of PV panels and the year-by-year cost values, the increasing number of fixed-tilt PV units and one-axis tracking PV units in the expansion planning plans could be seen in Figure 9-8. However, it is worth mentioning that the cost is not the only factor that determines the final optimal expansion plan, which is also affected by various other factors, such as specific tunnel width, relative cost of other energy sources and PV capacity size.

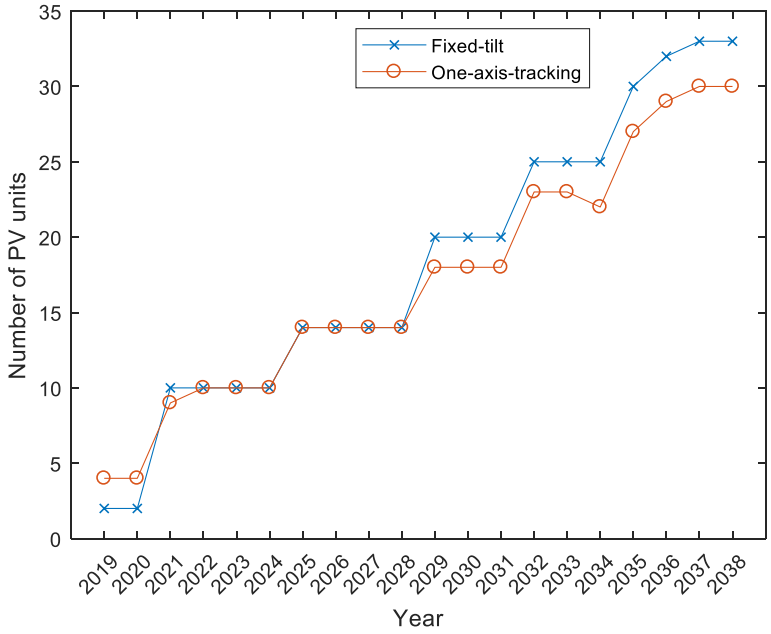


Figure 9-8 The increasing number of PV units in the expansion planning plan

9.6 Test the Implementation using Parallel-computing Toolbox

In some case studies, very long computational time was required to run MATPLAN in order to complete the iteration involving a large amount of expansion configuration options. Hence, parallel computing toolbox in MATLAB was used in loops to run independent iterations in parallel on multi-core CPUs. Specifically, *parfor loops* were used, in which *parfor* automates the creation of parallel pools and manages file dependencies.

> `parfor` loopVar = initVal:endVal; statements; end executes the loop body commands in statements for values of loopVar between initVal and endVal. loopVar specifies a vector of integer values increasing by 1. If we have Parallel Computing Toolbox™, the iterations of statements can execute on a parallel pool of workers on your multi-core computer or cluster.

For example, in one case study of 1205 candidate configurations in total (including wind1, wind2, solar1, solar2), four working cores were used in parallel pool to reduce the total computing time from 150 min to 49min on a desktop with Intel Core i5-3340 CPU @ 3.10GHz and 8.00 GB RAM.

10.0 MATPLAN Software Access

The latest version of MATPLAN source code (version 1.0) has been made publically available on Github, URL: https://github.com/wasp2019/MATPLAN, together with its Wiki that provides description about MATPLAN overview, features, use guides as well as developer resources.

The screen capture of MATPLAN 1.0 page on Github is shown in Figure 10-1.

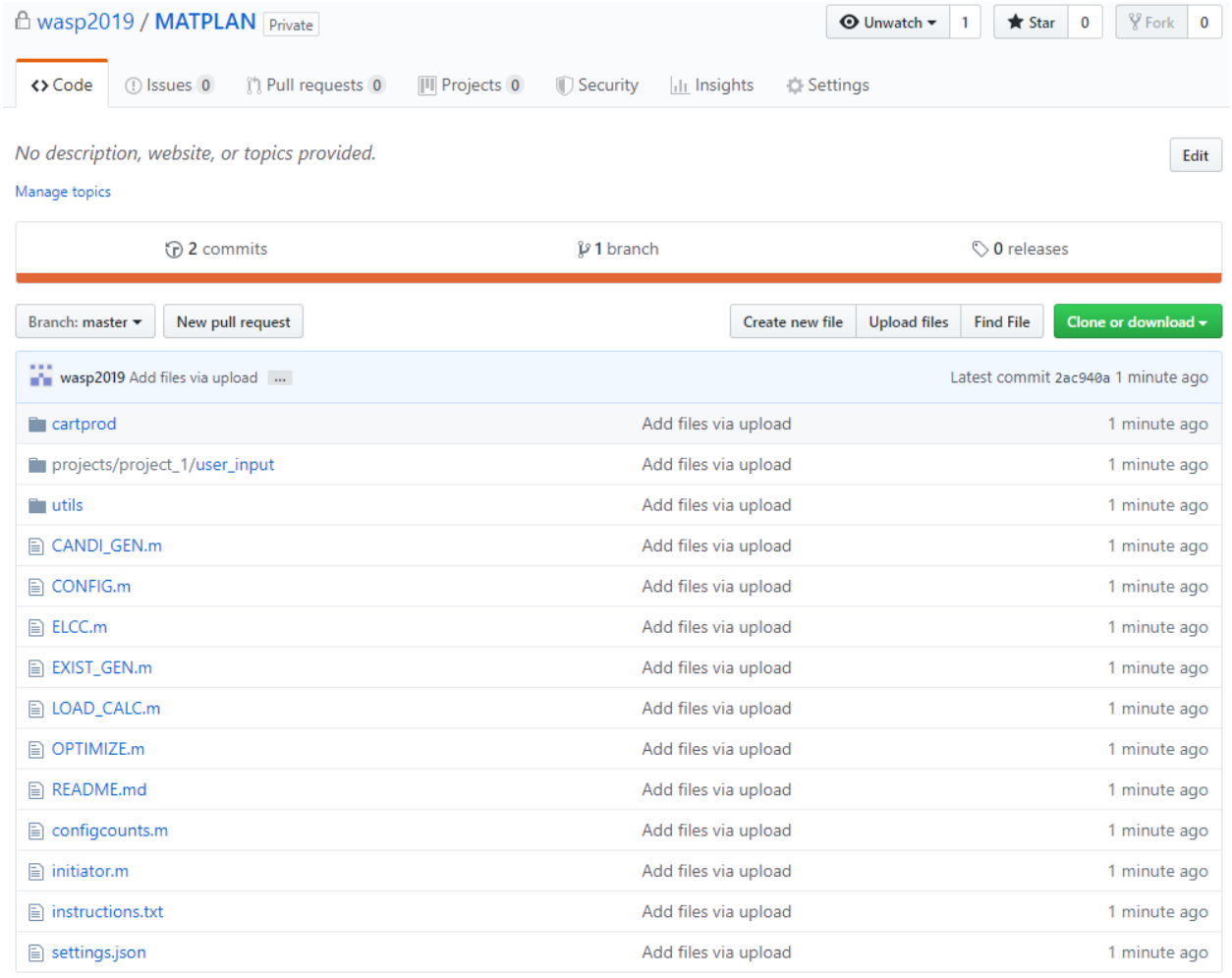


Figure 10-1 Screen capture of MATPLAN 1.0 page on Github

The screen capture of MATPLAN wiki page on Github is shown in Figure 10-2.

Welcome to the MATPLAN wiki!

MATPLAN stands for **MAT**lab and probability-based **PLAN**ning tool that is developed for resource adequacy evaluation and production costing. As an open-source open-architecture software tool, MATPLAN was engineered to provide lightweight and flexible modules to deal with generation expansion planning (GEP) problems with consideration of renewable energy sources.

MATPLAN offers the following key features:

- **Flexibility** – MATPLAN can account for different types of thermal generator plants, as well as the variable nature of renewable energy sources (both solar PV and wind farms). It combines both probabilistic and optimization techniques to allow the determination of optimal system expansion policy and is in contrast to the current practice that treats these variable sources as negative loads.
- **Modularization** – MATPLAN comprises six modules to reduce its computation complexity. These modules can work either coordinately or independently to provide useful intermediate results for expansion planning. This kind of modularization design allows users to reorganize and modify the decoupled modules for their own implementation purpose.
- **Scalability** – MATPLAN can handle up to 8760 time-division intervals, thousands of generation expansion planning configuration candidates and long-term planning time horizon (e.g., 15-30 years).
- **Source code available for public access** – MATPLAN's source codes – built on top of the widely used MATLAB – have been made available for public access.
- **Parallel Computing Capability** – MATPLAN provided an optional parallel computing version that allows the users to leverage the MATLAB Parallel Computing Toolbox ™ for higher computational efficiency and speed up the calculation process.

Developer Resources

Figure 10-2 Screen capture of MATPLAN Wiki page on Github

Appendix

Publications List

1. Zhang, X., Pipatanasomporn, M. and Rahman, S. (2018, October). A Comprehensive Analysis of Renewable Energy Representations in Power System Generation Expansion Planning. In *2018 International Conference and Utility Exhibition on Green Energy for Sustainable Development (ICUE)* (pp. 1-6). IEEE.

2. Chen, T., Pipatanasomporn, M., Rahman, I., Jing, Z., Adhikari. R., Zhang, X. and Rahman, S. (2019) MATPLAN: A Probability-based Planning Tool for Cost-effective Integration of Renewable Energy into the Electricity Grid. *IEEE Transactions on Sustainable Energy*. (under review)

3. Chen, T., Pipatanasomporn, M. and Rahman, S. (2019) Mainstreaming Renewable Energy Sources into Generation Expansion Planning Using MATPLAN. (in preparation)