EVCSAP Set-up and Total Work Flow

Step 0: Set-up `Python` Env. (with Anaconda)

Find the `py_env_setup` folder in the repository, then:

- For advanced Python programmer:

  requirements.txt is provided to set up the EVCSAP project environment. Copy it in your system user folder (e.g., C:\Users\z004ffpm). The link below provides a way of using this for environment set up

  > https://stackoverflow.com/questions/48787250/set-up-virtualenv-using-a-requirements-txt-generated-by-conda

- For others: (anaconda is required for set up)
  1. Read the **Spec List** or **Environment.yml** section of this blog https://www.anaconda.com/blog/moving-conda-environments.
  2. Copy either `EVCSAP_env_list.yml`, or `OptPyomoSP.txt` (depending on which file you use) in your system user folder where the anaconda can find the environment file. (e.g., C:\Users\z004ffpm)

Furthermore:

> - If `geopy` is missing, execute `pip install geopy` in (anaconda) prompt
> - If `MPI-SPPy` is missing, use `conda install openmpi`. Then, `conda install mpi4py` and finally `pip install mpi-sppy` in (anaconda) prompt

In [ ]:
```
import os, sys
currentdir = os.path.dirname(os.path.realpath(''))
parentdir = os.path.dirname(currentdir)
sys.path.append(currentdir)
```

Step 1: Pre-process Geo Data (Grid Connections Excluded)

Pre-process to get the Set-up Dictionary for building up **MPDP** frame model which calculates expressions like distances between nodes and EESC, etc.

In [ ]:
```
# To import local scripts
from csap_packages_sp import sp_data_process as Dap

# To import Opensource packages
import numpy as np
import pandas as pd
import geopandas as gpd
from shapely.geometry import Point, MultiPoint
import time
import pickle
from datetime import datetime
```

Step 1.1: Pre-process Data of POIs and SSs

In [ ]:
```
#########################################################
#### Load Data from OSM, Sythetic, OCM, DE Bundesamt ####

# --- Load Schutter Wald OSM Data (Resource from hdgb API by Domenico) --- #
geojson_folder = r"C:\Users\z004ffpm\Work_Documents\CSallocModel\for_branch_copy_mpsp_evcsap\Data\geo_raw_data\osm\osm_raw_geojson"
geojson_filename = r"\SW_77746_with_SSs.geojson"

filepath = geojson_folder + geojson_filename
SW_gdf = gpd.read_file(filepath)

# --- Load Synthetic Substations Data for SW (Resource from Domenico) --- #
```

```python
path_synthetic_ss_SW =  r"C:\Users\z004ffpm\Work_Documents\CSallocModel\for_branch_copy_mpsp_evcsap\Data\geo_raw_data\osm\synthetic_SS_location\synth_SSs_SW_77746.csv"
df = pd.read_csv(path_synthetic_ss_SW)
df['element'] = 'substations' # We keep the 'tag' == 'transformers', for later reference that these data are synthetic
syn_ss_SW_gdf = gpd.GeoDataFrame(
    df,
    geometry=gpd.points_from_xy(df.lon, df.lat)
)

# --- Load existing old CSs from OCM --- #
# # There's no CS data for Schutterwald on OCM, thus jump this step. (Resouce from Sisi)
# df_ocm = pd.read_csv('./ocm.csv')

# --- Load existing old CSs from "Ladesaeulenregistered" by DE Bundesamt --- #
df_DE_Gov = pd.read_csv(
    r'C:\Users\z004ffpm\Work_Documents\CSallocModel\for_branch_copy_mpsp_evcsap\Data\geo_raw_data\Ladensaeulenregister\Ladesaeulenregister_CSV.csv',
    encoding = "ISO-8859-1", # "utf-8",
    header = 10,
    sep = ';'
    )
# df_DE_Gov.head(3)
oldCS_df_SW = df_DE_Gov[df_DE_Gov.Postleitzahl == 77746] # EDEKA Oberle Getränkemarkt
```

In [ ]:

```python
######################################################################################
#### Process loaded data, define single-period and multi-period deterministic data ######

# --- concat OSM and Synthetic data for SW --- #
SW_concat_gdf = gpd.GeoDataFrame(pd.concat([SW_gdf, syn_ss_SW_gdf], axis=0, ignore_index=True))

# --- Pre-process concatted data --- #
gdfcopy = SW_concat_gdf.copy()

# Calculate centroids by merging data with same id.
gdfcopy = Dap._get_centered_gdf(gdfcopy)

# # Get parking and charging_station info from `df['tag']` to define candidate locations for new CSs
gdfcopy = Dap._detect_parking_CS_SS(gdfcopy)
```

```
Extracting parking info from gdf["tag"]: 100%|██████████████████████████████████| 195/195 [00:00<00:00, 16210.59it/s]
```

In [ ]:

```python
gdfcopy.tail(3)
```

Out[ ]:

id	osm_id	element	tag	geometry	lat	lon	id_count	parking_capacity	max_extraCPsToInstall	charging_capacity
192	9887717403	amenity	{'amenity': 'car_pooling', 'bench': 'yes', 'na...	POINT (7.88045 48.44707)	48.447070	7.880451	1	NaN	NaN	NaN
193	9887717414	amenity	{'amenity': 'bench'}	POINT (7.88556 48.45341)	48.453412	7.885560	1	NaN	NaN	NaN
194	9887717415	amenity	{'amenity': 'bench'}	POINT (7.88556 48.45339)	48.453385	7.885562	1	NaN	NaN	NaN

In [ ]:

```python
# --- Insert Bundesnetzagentur data to concatted DataFrame --- #
lat_sw = float(oldCS_df_SW.loc[4012, 'Breitengrad'].replace(',', '.'))
lon_sw = float(oldCS_df_SW.loc[4012, 'Längengrad'].replace(',', '.'))
geo_array = np.array([lon_sw, lat_sw])
gdfcopy.loc[len(gdfcopy)] = {
    'osm_id': 'regisID_4012',
    'element': 'charging_station',
    'tag': "{'amenity': 'charging_station', \
            'Source': 'Bundesnetzagentur'}",
    'geometry': Point(lon_sw,lat_sw),
    'lat': lat_sw,
    'lon': lon_sw,
    'id_count': 1,
    'parking_capacity': None,
    'max_extraCPsToInstall': 2,
```

```
    'charging_capacity': 2,
    } # Useful Reference: https://gis.stackexchange.com/questions/345167/building-geodataframe-row-by-row
```

```
C:\Users\z004ffpm\Anaconda3\envs\OptPyomoSP\lib\site-packages\pandas\core\construction.py:762: ShapelyDeprecationWarning: The array interface is deprecated and will no longer work in Shapely 2.0. Convert the '.coords' to a numpy array instead.
  subarr = construct_1d_object_array_from_listlike(arr)
```

In [ ]:
```
gdfcopy.tail(3)
```

Out[ ]:

id	osm_id	element	tag	geometry	lat	lon	id_count	parking_capacity	max_extraCPsToInstall	charging_capacity
193	9887717414	amenity	{'amenity': 'bench'}	POINT (7.88556 48.45341)	48.453412	7.885560	1	NaN	NaN	NaN
194	9887717415	amenity	{'amenity': 'bench'}	POINT (7.88556 48.45339)	48.453385	7.885562	1	NaN	NaN	NaN
195	regisID_4012	charging_station	{'amenity': 'charging_station', 'S...	POINT (7.88779 48.45271)	48.452714	7.887786	1	None	2.0	2.0

In [ ]:
```
# Visualize GeoDataFrame
Dap.geo_visualizer(gdfcopy)
```

In [ ]:
```
# --- Build multi-period POI statistics Dataframe  --- #
### Define statistics by sampling from Poisson/Beta/Normal distributions.
### These data are treated as known info to EVCSAP model
np.random.seed(151222)
dict_POI_Statistics = {
    'day_normal':{
        'amenity': {
            'NrArrival_poissonDistr_lam': 80, # For poisson distribution, mean = lambda
            'Arrival_SOC_betaDistr_(alpha,beta)': (22, 4),  # For beta distribution, mean = alpha/(alpha+beta)
            'Visiting_duration_normalDistr_(loc,scale)': (0.45, 0.1)
        },
```

```
        'historic': {
            'NrArrival_poissonDistr_lam': 20, # For poisson distribution, mean = lambda
            'Arrival_SOC_betaDistr_(alpha,beta)': (25, 3),  # For beta distribution, mean = alpha/(alpha+beta)
            'Visiting_duration_normalDistr_(loc,scale)': (0.25, 0.05)
        },
        'leisure': {
            'NrArrival_poissonDistr_lam': 60, # For poisson distribution, mean = lambda
            'Arrival_SOC_betaDistr_(alpha,beta)': (21, 4),  # For beta distribution, mean = alpha/(alpha+beta)
            'Visiting_duration_normalDistr_(loc,scale)': (0.75, 0.1)
        },
        'shop': {
            'NrArrival_poissonDistr_lam': 130, # For poisson distribution, mean = lambda
            'Arrival_SOC_betaDistr_(alpha,beta)': (22, 2),  # For beta distribution, mean = alpha/(alpha+beta)
            'Visiting_duration_normalDistr_(loc,scale)': (0.55, 0.1)
        },
        'sport': {
            'NrArrival_poissonDistr_lam': 60, # For poisson distribution, mean = lambda
            'Arrival_SOC_betaDistr_(alpha,beta)': (21, 4),  # For beta distribution, mean = alpha/(alpha+beta)
            'Visiting_duration_normalDistr_(loc,scale)': (1, 0.15)
            }
    },
    'day_peak':{
        'amenity': {
            'NrArrival_poissonDistr_lam': 30, # For poisson distribution, mean = lambda
            'Arrival_SOC_betaDistr_(alpha,beta)': (22, 4),  # For beta distribution, mean = alpha/(alpha+beta)
            'Visiting_duration_normalDistr_(loc,scale)': (25/60, 6/60)
        },
        'historic': {
            'NrArrival_poissonDistr_lam': 7, # For poisson distribution, mean = lambda
            'Arrival_SOC_betaDistr_(alpha,beta)': (25, 3),  # For beta distribution, mean = alpha/(alpha+beta)
            'Visiting_duration_normalDistr_(loc,scale)': (15/60, 3/60)
        },
        'leisure': {
            'NrArrival_poissonDistr_lam': 20, # For poisson distribution, mean = lambda
            'Arrival_SOC_betaDistr_(alpha,beta)': (21, 4),  # For beta distribution, mean = alpha/(alpha+beta)
            'Visiting_duration_normalDistr_(loc,scale)': (45/60, 6/60)
        },
        'shop': {
            'NrArrival_poissonDistr_lam': 40, # For poisson distribution, mean = lambda
            'Arrival_SOC_betaDistr_(alpha,beta)': (22, 2),  # For beta distribution, mean = alpha/(alpha+beta)
            'Visiting_duration_normalDistr_(loc,scale)': (35/60, 6/60)
        },
        'sport': {
            'NrArrival_poissonDistr_lam': 25, # For poisson distribution, mean = lambda
            'Arrival_SOC_betaDistr_(alpha,beta)': (21, 4),  # For beta distribution, mean = alpha/(alpha+beta)
            'Visiting_duration_normalDistr_(loc,scale)': (60/60, 9/60)
        }
    },
    'night':{
        'amenity': {
            'NrArrival_poissonDistr_lam': 5, # For poisson distribution, mean = lambda
            'Arrival_SOC_betaDistr_(alpha,beta)': (22, 4),  # For beta distribution, mean = alpha/(alpha+beta)
            'Visiting_duration_normalDistr_(loc,scale)': (420/60, 45/60)
        },
        'historic': {
            'NrArrival_poissonDistr_lam': 3, # For poisson distribution, mean = lambda
            'Arrival_SOC_betaDistr_(alpha,beta)': (25, 3),  # For beta distribution, mean = alpha/(alpha+beta)
            'Visiting_duration_normalDistr_(loc,scale)': (360/60, 45/60)
        },
        'leisure': {
            'NrArrival_poissonDistr_lam': 5, # For poisson distribution, mean = lambda
            'Arrival_SOC_betaDistr_(alpha,beta)': (21, 4),  # For beta distribution, mean = alpha/(alpha+beta)
            'Visiting_duration_normalDistr_(loc,scale)': (420/60, 45/60)
        },
        'shop': {
            'NrArrival_poissonDistr_lam': 3, # For poisson distribution, mean = lambda
```

```
            'Arrival_SOC_betaDistr_(alpha,beta)': (22, 2),  # For beta distribution, mean = alpha/(alpha+beta)
            'Visiting_duration_normalDistr_(loc,scale)': (330/60, 75/60)
        },
        'sport': {
            'NrArrival_poissonDistr_lam': 3, # For poisson distribution, mean = lambda
            'Arrival_SOC_betaDistr_(alpha,beta)': (21, 4),  # For beta distribution, mean = alpha/(alpha+beta)
            'Visiting_duration_normalDistr_(loc,scale)': (420/60, 60/60)
        }
    }
}
```

In [ ]:
```
### Build POIs stat df
df_SW_POI_stat_mpd = Dap._assign_mp_POI_statistics(
    gdf = gdfcopy, dict_POI_Statistics = dict_POI_Statistics
)
```

```
Assigning POI statistics: 100%|██████████████████████████████████████████████████████████| 3/3 [00:00<00:00, 91.73it/s]
```

In [ ]:
```
df_SW_POI_stat_mpd.head()
```

Out[ ]:

id	time	element	Nr_DailyArrivals	Arrival_SOC	Visiting_duration
15	**day_normal**	leisure	65.0	0.947905	0.660298
	day_peak	leisure	18.0	0.752797	0.759943
	night	leisure	2.0	0.885695	7.28749
16	**day_normal**	leisure	67.0	0.716903	0.710016
	day_peak	leisure	22.0	0.959579	0.736742

In [ ]:
```
# --- Build multi-period Substations Statistics Dataframe --- #
### Define statistics by sampling from normal distributions.
### These data are treated as known info to EVCSAP model
avg_SS_full_load_cap = 400
dict_mpd_SW_SS_Statistics = { # The only substation statistics to define is the available power load (on avg.) during different periods
    'day_normal': (0.2*avg_SS_full_load_cap, 0), # means the available power load follows a normal distribution with (mean = 80, std = 0)
    'day_peak': (0.12*avg_SS_full_load_cap, 0),
    'night': (0.3*avg_SS_full_load_cap, 0)
}
### Build SSs stat df
df_SW_SS_stat_mpd = Dap._assign_mp_SS_statistics(
    gdf = gdfcopy, dict_mpd_SS_Statistics = dict_mpd_SW_SS_Statistics
)
```

```
Assigning SS statistics: 100%|██████████████████████████████████████████████████████████| 3/3 [00:00<00:00, 499.30it/s]
```

In [ ]:
```
df_SW_SS_stat_mpd.head()
```

Out[ ]:

id	time	element	available_power_load
0	**day_normal**	substations	80.0
	day_peak	substations	48.0
	night	substations	120.0
1	**day_normal**	substations	80.0
	day_peak	substations	48.0

Step 1.2: Define Strategy Dictionary of Optimization Model (Budget, Hyperparamters like distance thresholds, etc.)

In [ ]:
```
############################################################
####      Get Model Setup Strategy dictionary      ######
# Define Strategy dictionary for other non-osm/ocm/synthetic... parameters
strategy_dict = {
    # ----- CS ----- #
    'cost_buildNewCS': 3000, # float, annuity of updating an old opened CS
    'cost_updateOldCS': 2500, # float, annuity of updating an old opened CS
    'cost_installCP': 2250, # float, annuity of installing a CP

    'profit_charge_fee': (0.51, 0.00), # set as normal Distrib. (mean, std), charging fee at CS euro/kWh
    'budget_max_N_newBuildCS' : 8, # int, max amount of new CSs the investors want to open
    'budget_max_N_updateCS' : 1, # int, max amount of old CSs investors want to update
    'budget_max_N_totalCSs' : 8, # int, max amount of CSs (old or new) to update or open in total:
    'budget_max_N_new_CPs' : 80, # int, max amount of new CPs investors want to install
    'rule_Min_dist_Between_CSs' : 2/60, # float in hour, min distance allowed between two CSs
    'config_CPpower' : 22, # float, in kW, Power rating of a CP,

    # ----- SS ----- #
    'cost_expandSS': 500, # float, annuity of substation powerload expansion per KW.
        # Since domain of grid decision variables can be: [0, 50, 100, 150, 200,...],
        # every expansion cost then should be in the unit of €/50kW
    'cost_backstopTech': 5, # float, one time cost of using backstop tech per KW to prevent overload blackout
        # This is one time cost, and domain of backstop tech usage decision variables is: [0, 50, 150,...],
        # which means every usage of backstop tech is in the unit of (365 * €/50kW )
    ## ---- Stochastic scenario generation ---- ##
    'max_dist_cs_ss_connection': 6/60, # float in hour, walking distance [hour] with walking speed (5km/hour) -> 6/60 * 5000 = 500 m
    'CS_neighbour_dist_decay_threshold': 4/60, # float in hour, walking distance [hour]
        # # used for calculating dist_decay between CSs to adjust "neighbours' connection reward weights"
        # # 'CS_neighbour_dist_decay_threshold' > 'rule_Min_dist_Between_CSs'

    # ----- CD ----- #
    'rule_dist_decay_threshold' : 5/60, #10/60 # float in hour, max walking dist EV drivers can tolerate to charge their EVs
    'config_Battery_cap': 50, # float, in kWh, battery capacities of EVs
    'config_periods_length': {
        'day_normal': 10, # 08:00~17:30; 19:30~20:00
        'day_peak': 2, # 17:30~19:30
        'night': 12 # 20:00~08:00
    },
}
```

Step 1.3: Created the Dictionary for Model Setup .

Produce an inintial model setup dictionary for Auxiliary Model.

This auxiliary model is used for droping those candidate locations for CSs which are outside the range of grid connection to SSs known in the GeoDataFrame extracted in Step 1.1.

In [ ]:
```
from csap_packages_sp import sp_sce_generation as Sceg
```

```
[    0.00] Initializing mpi-sppy
```

In [ ]:
```
# Get initial setup dictionary for auxilliary model
csap_setup_dict_4_auxi, gdf_4_auxi = Dap._get_ModelSetupDict(
    gdf = gdfcopy,
    strategy_dict = strategy_dict,
    mpd_poi_stats_df = df_SW_POI_stat_mpd,
    mpd_SS_stats_df = df_SW_SS_stat_mpd
)
# Set up auxilliary model
# and get (gdf, substations_stat_df) after droping grid-isolated nodes
auxi_model_2_drop_islt_nodes = Sceg.get_auxi_model(
    setup_dict = csap_setup_dict_4_auxi
    )
```

```
Created a setup dictionary for pyomo auxi/frame/formal models and a processed regional geodataframe
Set up an auxilliary model and calculate distances and distance decay factors between CSs and SSs to either deal with isolated nodes or generate connection scenarios for stochastic EVCSAP
Done! Process took 0.53 seconds.
```

Created the final version of model set-up dictionary.

In [ ]:
```
gdfcopy_dropped_isolated, df_SW_SS_stat_mpd_dropped_isolated = \
    Dap._get_dfs_with_isolated_nodes_dropped(
        auxilliary_model = auxi_model_2_drop_islt_nodes,
        origin_gdf = gdfcopy,
        origin_SS_stat_df = df_SW_SS_stat_mpd
    )

# Get the formal Set-up dictionary for generating scenarios and set-up model
final_csap_setup_dict, final_gdf = Dap._get_ModelSetupDict(
    gdf = gdfcopy_dropped_isolated,
    strategy_dict = strategy_dict,
    mpd_poi_stats_df = df_SW_POI_stat_mpd,
    mpd_SS_stats_df = df_SW_SS_stat_mpd_dropped_isolated
)
print('Final Set-up Dictionary Created.')

#
```

```
Created a setup dictionary for pyomo auxi/frame/formal models and a processed regional geodataframe
Final Set-up Dictionary Created.
```

Store the Set-up Dictionary Created

In [ ]:
```
# Define wher to store the Set-up Dict
# storage_path = r"Define_your_path"
# # storage_path = r"C:\Users\z004ffpm\Work_Documents\CSallocModel\for_branch_copy_mpsp_evcsap\Data\cleaned_model_data_for_test\final_set_up_dict.pickle"
# with open(storage_path, 'wb') as f:
#     pickle.dump(final_csap_setup_dict, f)

# Read the file:
# storage_path = r"C:\Users\z004ffpm\Work_Documents\CSallocModel\for_branch_copy_mpsp_evcsap\Data\cleaned_model_data_for_test\csap_setup_dict_thesis_version.pickle"
# # Read the stored dictionary
# final_csap_setup_dict = pd.read_pickle(storage_path)
```

Step 2: Generate Grid Connection Scenarios

In [ ]:
```
# To make a copy to make sure original data will not be modified
final_setup_dict_copy = final_csap_setup_dict.copy()
```

In [ ]:
```
# Create an auxilliary model containg only nodes of CSs and SSs
cs_ss_con_sce_auxi_model = Sceg.get_auxi_model(
    setup_dict = final_setup_dict_copy
    )

# Generate 10 grid connection scenarios from 2D-C-S-NET algorithm
all_connection_sces_dict = Sceg._generate_cs_ss_connection_sces(
    num_scenarios = 10,
    auxi_model = cs_ss_con_sce_auxi_model,
    save_prob_matrices = False
    )

# # Set `save_prob_matrices = True` to get an additional Connnection probability matrix
# all_connection_sces_dict, con_probs_dict = Sceg._generate_cs_ss_connection_sces(
#     num_scenarios = 10,
#     auxi_model = cs_ss_con_sce_auxi_model,
#     save_prob_matrices = False
#     )
```

```
Set up an auxilliary model and calculate distances and distance decay factors between CSs and SSs to either deal with isolated nodes or generate connection scenarios for stochastic EVCSAP
Done! Process took 0.15 seconds.
generating connection scenario: 100%|██████████████████████████████████████████████████| 10/10 [00:00<00:00, 15.28it/s]
```

In [ ]:
```python
# # Save generated connection scenarios
# storage_path = r"define_your_path"
# with open(storage_path, 'wb') as f:
#     pickle.dump(all_connection_sces_dict, f)# # Read the stored dictionary
```

In [ ]:
```python
# Check name of generated scenarios
all_connection_sces_dict.keys()
```

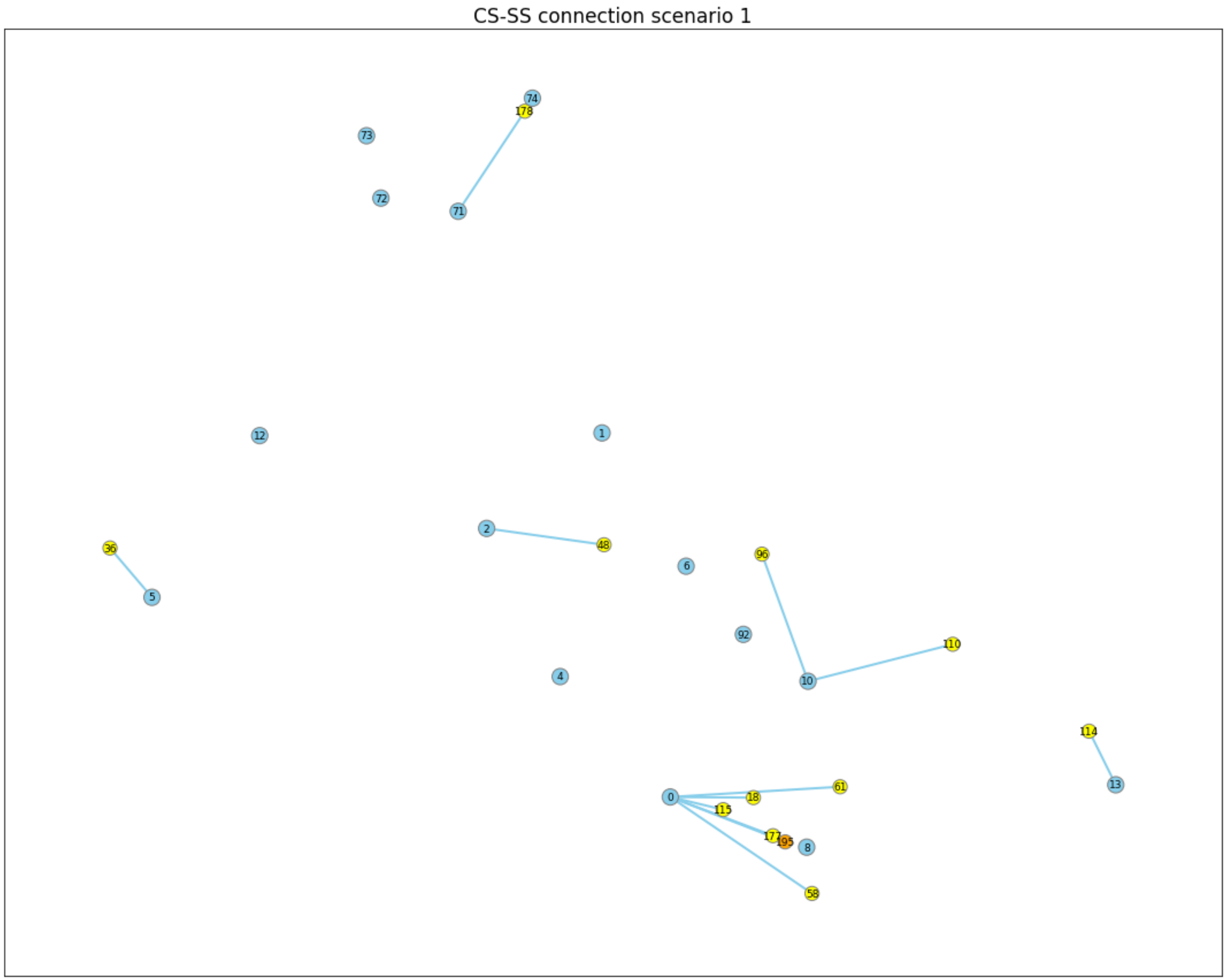
Out[ ]:
```
dict_keys(['1', '2', '3', '4', '5', '6', '7', '8', '9', '10'])
```

In [ ]:
```python
# Check one of the scenario
all_connection_sces_dict['1']
# (row: CSs, col: SSs)
```

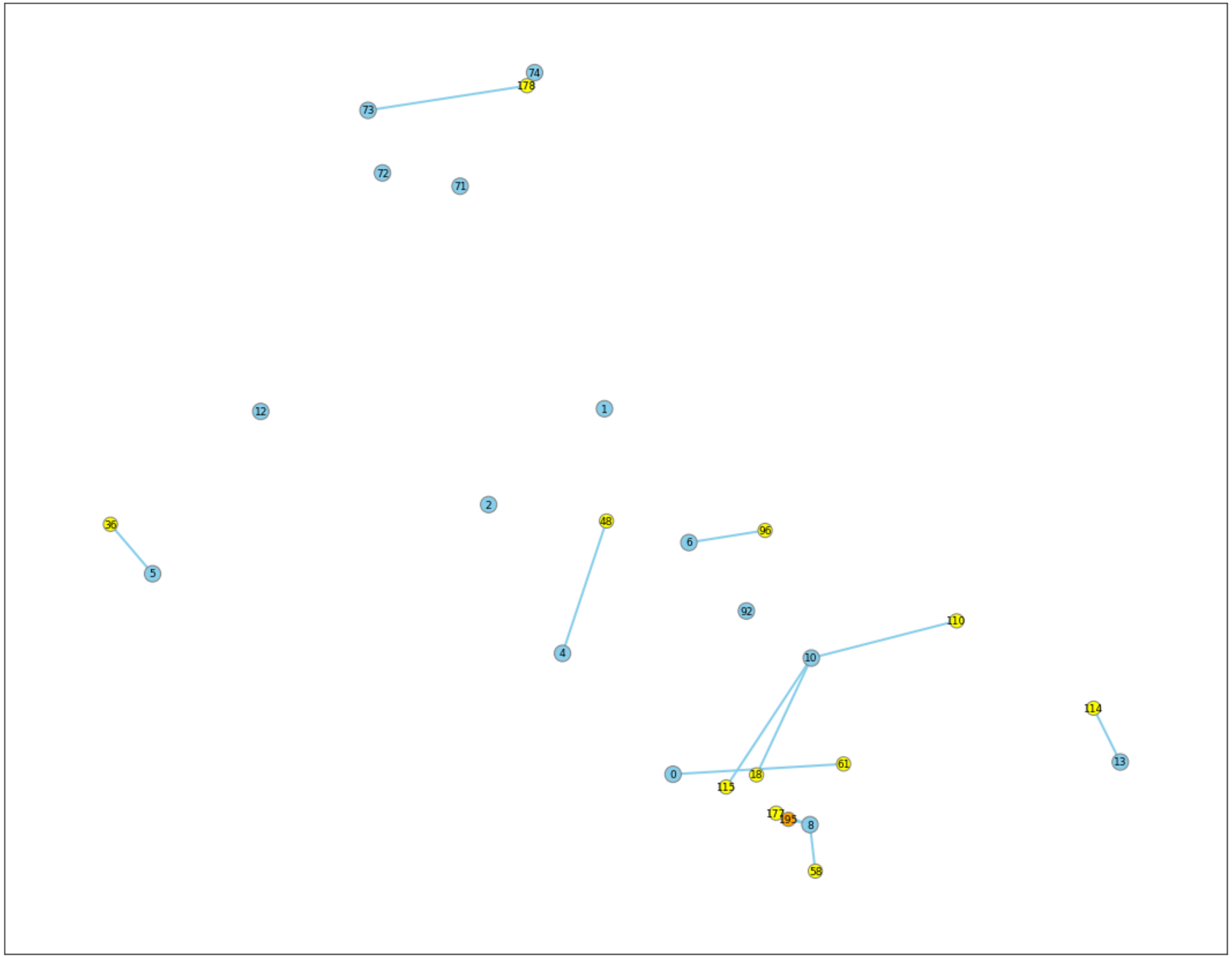
Out[ ]:

	0	1	2	4	5	6	8	10	12	13	71	72	73	74	92
36	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
195	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
58	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
48	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
110	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
61	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
114	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
96	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
177	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
18	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
115	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
178	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0

In [ ]:
```python
# Plot some scenarios of grid connection
for sce_id in [str(i) for i in range(1,3)]:
    Sceg._plot_cs_ss_connection_graph(
            auxi_model = cs_ss_con_sce_auxi_model,
            cs_ss_connection_dict = all_connection_sces_dict,
            sce_Nr = sce_id
        )
```


CS-SS connection scenario 1
74
178
73
72
71
12
1
2
48
36
5
6
96
92
110
4
10
114
13
0
18
61
115
177
195
8
58

CS-SS connection scenario 2



Calculated the MPDP connection scenario by a 20,000 generated scenarios example

In [ ]:
```
# Read stored file. E.g., Load the file with 20,000 generated scenarios
generated_scenarios_20k = pd.read_pickle(r"C:\Users\z004ffpm\Work_Documents\CSallocModel\for_branch_copy_mpsp_evcsap\Data\cleaned_model_data_for_test\20k_con_sces_dropped_isl.pickle")
```

In [ ]:
```
total_connection_matrix, mpdp_connection_sce = \
    Sceg._get_total_con_matrix_and_mpdp_con_sce(
        generated_connections = generated_scenarios_20k
    )
```

`total_connection_matrix` reflects the CS-SS connection frequency among 20,000 scenarios

(row: CSs, col: SSs)

In [ ]:
```
total_connection_matrix
```

Out[ ]:

	0	1	2	4	5	6	8	10	12	13	71	72	73	74	92
18	7638	0	0	0	0	0	8376	3986	0	0	0	0	0	0	0
36	0	0	0	0	19926	0	0	0	74	0	0	0	0	0	0
48	0	4342	5112	2563	0	5623	0	0	0	0	0	0	0	0	2360
58	5981	0	0	0	0	0	13535	484	0	0	0	0	0	0	0
61	5918	0	0	0	0	0	8865	5217	0	0	0	0	0	0	0
96	0	0	0	0	0	8115	0	4308	0	0	0	0	0	0	7577
110	0	0	0	0	0	0	0	19609	0	0	0	0	0	0	391
114	0	0	0	0	0	0	0	0	0	20000	0	0	0	0	0
115	8359	0	0	0	0	0	8843	2798	0	0	0	0	0	0	0
177	8208	0	0	0	0	0	10150	1642	0	0	0	0	0	0	0
178	0	0	0	0	0	0	0	0	0	0	5312	2915	4482	7291	0
195	8257	0	0	0	0	0	10389	1354	0	0	0	0	0	0	0

`mpdp_connection_sce` Calculated by taking the most frequent SS connection of each CS as the connection status for MPDP scenario

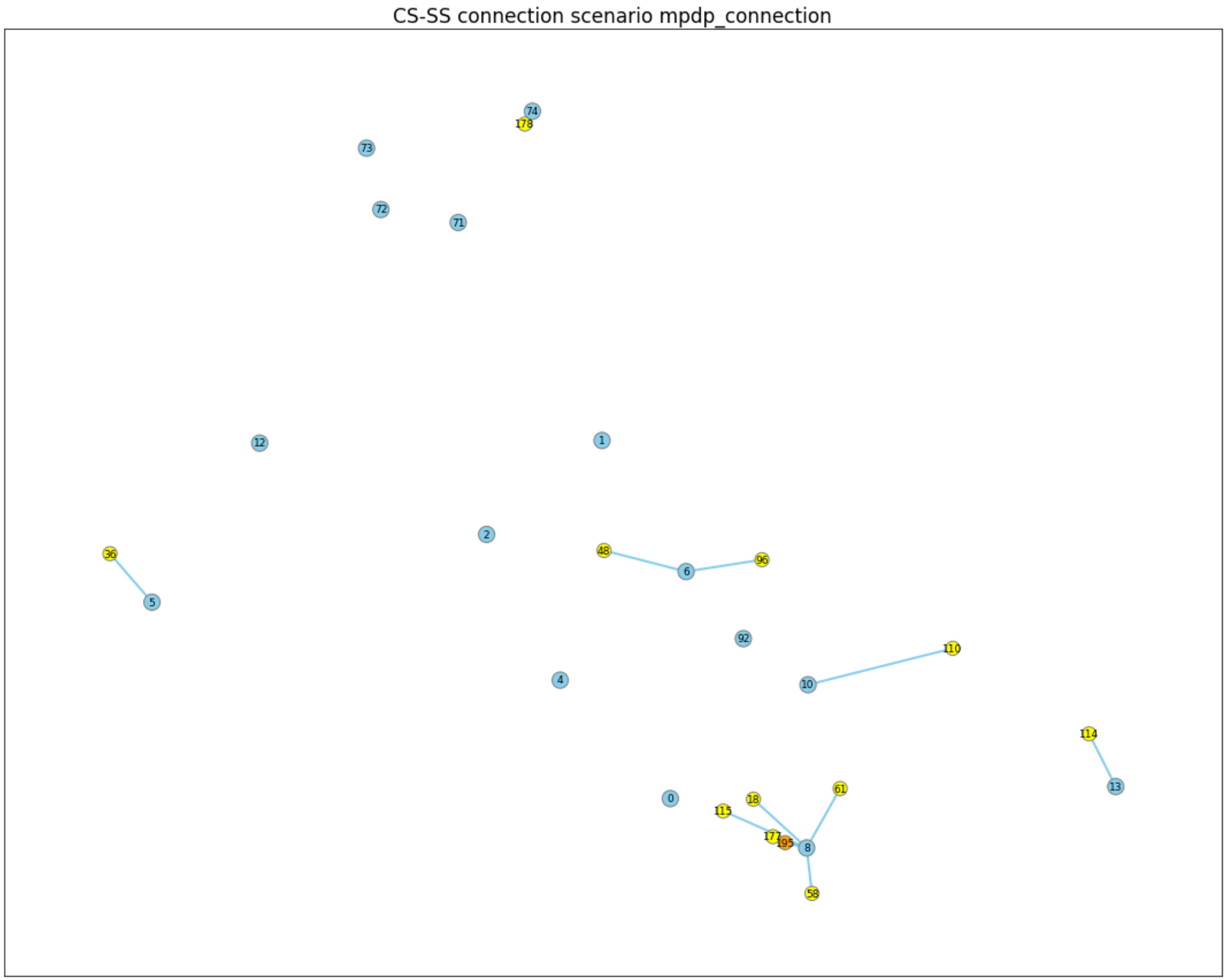
In [ ]:
```
mpdp_connection_sce
```

Out[ ]:

	0	1	2	4	5	6	8	10	12	13	71	72	73	74	92
18	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
36	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
48	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
58	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
61	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
96	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
110	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
114	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
115	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
177	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
178	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
195	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0

Plot MPDP connection

In [ ]:
```
mpdp_connection_dict = {'mpdp_connection': mpdp_connection_sce}
Sceg._plot_cs_ss_connection_graph(
        cs_ss_con_sce_auxi_model,
        mpdp_connection_dict, 'mpdp_connection'
        )
```



Return the number of all possible connection scenarios

(Just for getting a feeling of the stochastics. Irrelavent for model set up.)

In [ ]:
```
combinatorics_num_possible_sces = Sceg._get_total_num_sces(total_connection_matrix)
combinatorics_num_possible_sces
```

Out[ ]:
```
174960
```

Step 3: Set up and solve a MPDP frame model

for calculating parameters such as walking distances and EESC, which will be fed to MPSP later.

(Model Set-up by set-up dictionary (`final_setup_dict_copy`) and grid connection scenario of **MPDP** (`mpdp_connection_sce`))

In [ ]:
```
import pyomo.environ as pyo
from csap_packages_sp.sp_mpd_frame_model_setup import _build_mpdp_csap_frame
```

Step 3.1 Set up a MPDP frame model

In [ ]:
```
# Assign name to the model, optional. Default name set in the setup function is 'CSAP'
model_name = 'mpdp_frame_model'
csap_is_solved = False
linearize_csap = True
# sce_id = '1'
# connection_sce = all_connection_sces_dict[sce_id].stack().to_dict()

mpdp_frame_model = _build_mpdp_csap_frame(
    setup_dict = final_setup_dict_copy,
    # convert pd.DF mpdp_connection_sce to dict for building Pyomo model
    cs_ss_connect_sce = mpdp_connection_sce.stack().to_dict(),
    m_name = model_name,
    linearized = linearize_csap
)
```

```
Built an empty concrete pyomo model named mpdp_frame_model.
Defining sets   ...
Parameters_setup 1: Feeding basic parameters to mpdp_frame_model ...
Parameters_setup 2: Feeding CS parameters  to 'mpdp_frame_model' ...
Parameters_setup 3: Feeding SS parameters to 'mpdp_frame_model' ...
Parameters_setup 4: Feeding CD parameters to mpdp_frame_model ...
 ...
   Calculating walking distances between candidate locations and CD centers: model.d ...
     Done! Process took 1.33 seconds.
All CD parameters fed!

Feeding cs-ss connection scenario to model ...
Feeding decision variables ...
Defining Expression of Exogenous (Charging) Energy Supply Capability of CSs ...
 Done! Process took 57.83 seconds.
Feeding objective to mpdp_frame_model ...

          There are two objectives 'obj_total_profit' and 'obj_profit_no_grid' in frame model.
          Before resolution, use method:
    `model.objective_name.deactive()` to deactive one of them and solve the model with the remained active objective function.
Feeding constraints to mpdp_frame_model ...
  Linearizing constraints of CD coverage lower bound ...
All constraints fed! Process took 1.44 seconds.
Successfully built model mpdp_frame_model! Please use solver to solve it.

HINT: to check all properties of mpdp_frame_model, please use
    'mpdp_frame_model.display()'
HINT: to check the properties of parameters of mpdp_frame_model, please use:
    'mpdp_frame_model.param_name.display()'

Done, model set-up took 63.07 seconds in total.
```

Check model components by `.display`

> If data as `dictionary` is needed for further analysis. Refer to the usage of `.get_values()` for decision variables and `.extract_values()` for parameters in the Thesis `Model Implementation` Appendix Chapter.

In [ ]:
```
# Enumerate names of components in Pyomo
compo_categories = [pyo.Var, pyo.Param, pyo.Expression, pyo.Objective, pyo.Constraint]
compo_category_name_list = ["Variable", "Param", "Expression", "Objective", "Constraint"]
compo_dictionary = {comp_cat_name: list() for comp_cat_name in compo_category_name_list}
for compo_cat_name, compo_cat in zip(compo_category_name_list, compo_categories):
    for compo in mpdp_frame_model.component_objects(compo_cat, active=True):
```

```
        compo_dictionary[compo_cat_name].append(str(compo))
#         print(f"{compo_cat_name}:", v)
```

In [ ]:
```
for key, value in compo_dictionary.items():
    print(f"{key}: {value}")
```

```
Variable: ['x', 'y', 'z', 'h', 'eta', 'x_hat', 'chi', 'calE']
Param: ['nodes', 'N_update', 'N_newBuild', 'N_totalCSs', 'N_newCPs', 'DeltaT', 'rho', 'phi_IJ', 'c_x', 'c_y', 'pi', 'm', 'n', 'u', 'Pi', 'c_h', 'c_eta', 'phi_IK', 'phi_II', 'B_cap', 'calA', 'delta', 'calT', 'Z']
Expression: ['d', 'd_pScaled', 'tau', 'calD', 'single_CP_eff_CC_frac_i2j', 'w_cs2poi', 'w', 'AvgCD', 'N_EVs', 'Big_M', 'calD_poi', 'fullcd_POI_static', 'psi']
Objective: ['obj_profit_no_grid', 'obj_total_profit']
Constraint: ['MaxUpdateCSs', 'cons_MaxNewCSs', 'cons_MaxTotalCSs', 'cons_TotalMaxCPs', 'cons_x_hat_rules_1', 'cons_x_hat_rules_2', 'cons_x_hat_rules_3', 'cons_linearized_NewCSsNotTooClose', 'cons_NewCSnotCloseToOld',
'cons_MaxCP_atCS', 'cons_NoEmptyCS', 'cons_AssignedCDfracLessOne', 'cons_CSchargeCap', 'cons_CS_service_time_Cap', 'cons_CalX_UpB_FullPOI_CD', 'cons_CalX_LowB_FullPOI_CD', 'cons_CalX_UpB_allCDcoverByCSInNeigh', 'cons_
CalX_LowB_allCDcoverByCSInNeigh', 'cons_CDcoverageLB', 'cons_no_grid_connected_no_build', 'cons_grid_load_efficientCD', 'cons_grid_load_NrCPs']
```

In [ ]:
```
mpdp_frame_model.y.display()
```

```
y : Number of CPs to be installed at location i ∈ I.
    Size=12, Index=I
    Key : Lower : Value : Upper : Fixed : Stale : Domain
     18 :     0 :     0 :  None : False : False : NonNegativeIntegers
     36 :     0 :     0 :  None : False : False : NonNegativeIntegers
     48 :     0 :     0 :  None : False : False : NonNegativeIntegers
     58 :     0 :     0 :  None : False : False : NonNegativeIntegers
     61 :     0 :     0 :  None : False : False : NonNegativeIntegers
     96 :     0 :     0 :  None : False : False : NonNegativeIntegers
    110 :     0 :     0 :  None : False : False : NonNegativeIntegers
    114 :     0 :     0 :  None : False : False : NonNegativeIntegers
    115 :     0 :     0 :  None : False : False : NonNegativeIntegers
    177 :     0 :     0 :  None : False : False : NonNegativeIntegers
    178 :     0 :     0 :  None : False : False : NonNegativeIntegers
    195 :     0 :     0 :  None : False : False : NonNegativeIntegers
```

In [ ]:
```
mpdp_frame_model.obj_total_profit.display()
```

```
obj_total_profit : Size=1, Index=None, Active=True
    Key  : Active : Value
    None :   True :   0.0
```

In [ ]:
```
mpdp_frame_model.cons_MaxTotalCSs.display()
```

```
cons_MaxTotalCSs : Size=1
    Key  : Lower : Body : Upper
    None :  None :    0 :     8
```

In [ ]:
```
mpdp_frame_model.w.display()
```

```
w : Size=36
    Key                  : Value
     (18, 'day_normal') :                0.0
       (18, 'day_peak') :                0.0
          (18, 'night') :                0.0
     (36, 'day_normal') :                0.0
       (36, 'day_peak') :                0.0
          (36, 'night') :                0.0
     (48, 'day_normal') :                0.0
       (48, 'day_peak') :                0.0
          (48, 'night') :                0.0
     (58, 'day_normal') :                0.0
       (58, 'day_peak') :                0.0
          (58, 'night') :                0.0
     (61, 'day_normal') :                0.0
       (61, 'day_peak') :                0.0
          (61, 'night') :                0.0
     (96, 'day_normal') :                0.0
       (96, 'day_peak') :                0.0
          (96, 'night') :                0.0
    (110, 'day_normal') :                0.0
      (110, 'day_peak') :                0.0
         (110, 'night') :                0.0
    (114, 'day_normal') :                0.0
      (114, 'day_peak') :                0.0
         (114, 'night') :                0.0
    (115, 'day_normal') :                0.0
      (115, 'day_peak') :                0.0
         (115, 'night') :                0.0
    (177, 'day_normal') :                0.0
      (177, 'day_peak') :                0.0
         (177, 'night') :                0.0
    (178, 'day_normal') :                0.0
      (178, 'day_peak') :                0.0
         (178, 'night') :                0.0
    (195, 'day_normal') : 235.26993886952584
      (195, 'day_peak') : 48.422024619348875
         (195, 'night') : 29.389906597104506
```

Step 3.2 Retrieve all data from the `mpdp_frame_model` for further MPSP Setup

From the set-up message one can see that building up the frame model can take almost one minute for this small Schutterwald case. If we use such frame model to construct MPSP, it will be very time consuming. Thus we will retrieve the calculated data from frame_model, and later when we want to build a model based on the same statistics of POIs and SSs and CSs, we can then utilize this frame model data to build new models very fastly

In [ ]:
```
from csap_packages_sp import sp_model_setup_by_fm_data as SupSP
```

In [ ]:
```
data_mpdp_frame = SupSP._get_data_from_frame_model(frame_model = mpdp_frame_model)
# Usage of this data of frame model for # quicker
# model set-up will be illustrated in Step 4.
```

In [ ]:
```
# Check structure of data_mpdp_frame
data_mpdp_frame.keys()
```

Out[ ]:
```
dict_keys(['I_update', 'I_newBuild', 'I', 'J', 'K', 'T', 'nodes', 'N_update', 'N_newBuild', 'N_totalCSs', 'N_newCPs', 'DeltaT', 'rho', 'phi_IJ', 'c_x', 'c_y', 'pi', 'm', 'n', 'u', 'Pi', 'c_h', 'c_eta', 'phi_IK', 'phi_II', 'B_cap', 'calA', 'delta', 'calT', 'd', 'd_pScaled', 'tau', 'calD', 'single_CP_eff_CC_frac_i2j', 'w_cs2poi', 'w', 'AvgCD', 'N_EVs', 'Big_M', 'calD_poi', 'fullcd_POI_static', 'psi', 'x', 'y', 'z', 'h', 'eta'])
```

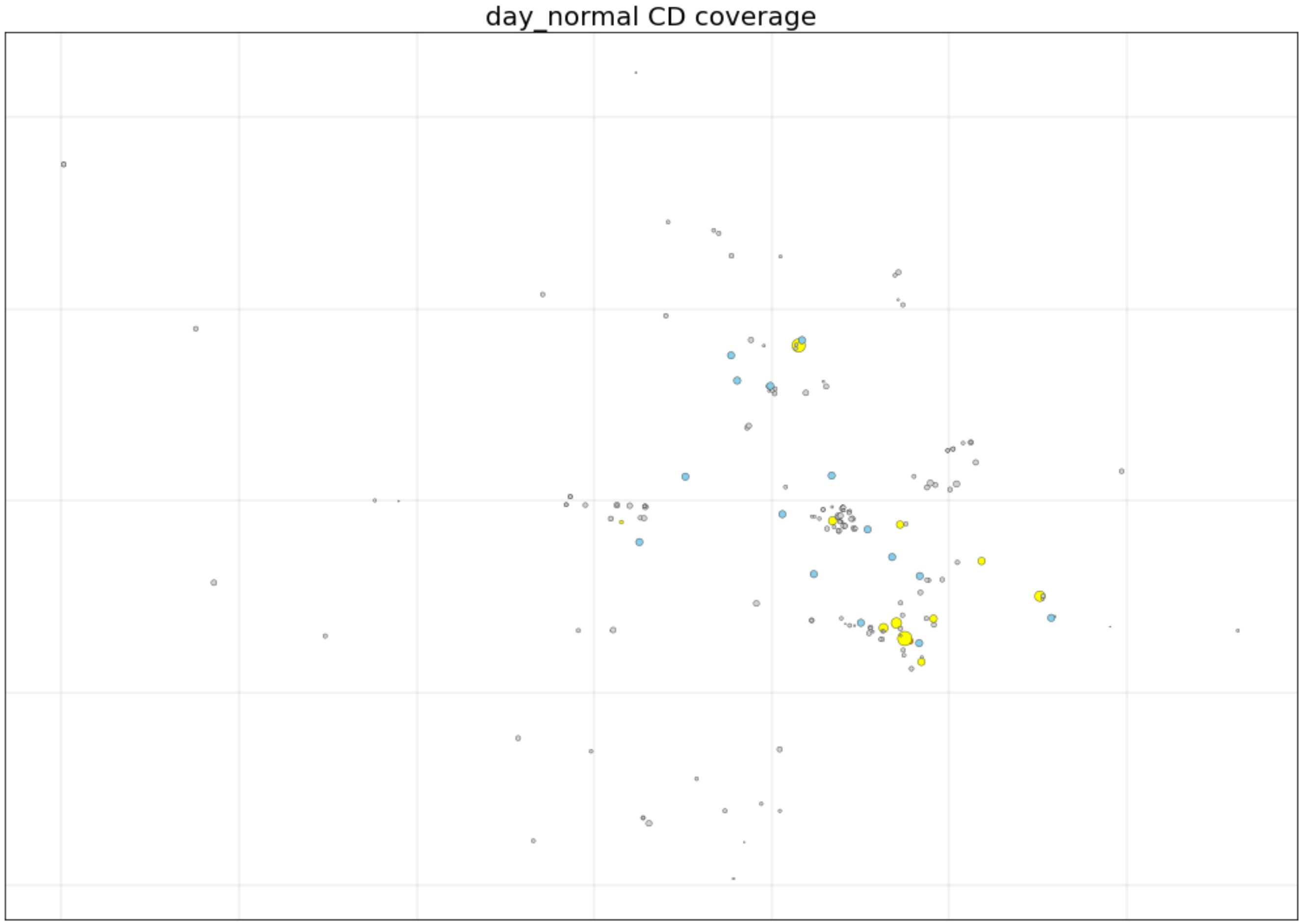
Print a short case description

In [ ]:
```
case_model = mpdp_frame_model
print(f'''- {len(case_model.I)} candidate locations ({len(case_model.I_newBuild)} for building new CSs, {len(case_model.I_update)} for updating old CSs),
- {len(case_model.K)} substations, and
- {len(case_model.J)} locations with charging demand
- Without considering the budget, the maximum amount of CPs one can install in those candidate locations is: {sum(pyo.value(case_model.m[i]) for i in case_model.I)}''')
```

```
- 12 candidate locations (11 for building new CSs, 1 for updating old CSs),
- 15 substations, and
- 158 locations with charging demand
- Without considering the budget, the maximum amount of CPs one can install in those candidate locations is: 146.0
```

Plot the unsolved model

In [ ]:
```python
Dap._plot_model_graph(
    model = mpdp_frame_model,
    cd_period = 'day_normal',
    model_is_solved = csap_is_solved,
    save_fig = False # Set True to save a high resolution .svg file
)
```



Step 3.3 Solve the frame model

In [ ]:
```python
from pyomo.opt import SolverFactory
# Initialize solver
Solver = SolverFactory(
    'cplex', # 'glpk',
    tee = True, # Decide whether to print detailed solver message
)
```

Deactivate the obj which does NOT consider grid cost

In [ ]:
```
mpdp_frame_model.obj_profit_no_grid.deactivate()
# mpdp_frame_model.obj_total_profit.activate()
```

Deactivate the tighter constraint which use TESC as power load at CSs and will result in much more expensive power expansion decisions

In [ ]:
```
mpdp_frame_model.cons_grid_load_NrCPs.deactivate()
# mpdp_frame_model.cons_grid_load_efficientCD.activate()
```

In [ ]:
```
# Use the solver to solve frame model
solver_results = Solver.solve(mpdp_frame_model)
csap_is_solved = True
print("Solver Message:", solver_results)
```

```
Solver Message:
Problem:
- Name: tmpq_ec19gb
  Lower bound: 1599429.40246346
  Upper bound: 1599429.40246346
  Number of objectives: 1
  Number of constraints: 3486
  Number of variables: 6842
  Number of nonzeros: 14704
  Sense: maximize
Solver:
- Status: ok
  User time: 0.19
  Termination condition: optimal
  Termination message: MIP - Integer optimal solution\x3a Objective = 1.5994294025e+06
  Statistics:
    Branch and bound:
      Number of bounded subproblems: 0
      Number of created subproblems: 0
  Error rc: 0
  Time: 0.39281797409057617
Solution:
- number of solutions: 0
  number of solutions displayed: 0
```

In [ ]:
```
from csap_packages_sp import sp_stat_compu as Stac
```

In [ ]:
```
print_decision = True
if print_decision:
    Stac._print_decision(mpdp_frame_model)
#     _print_decision(mpdp_frame_model_4_sp)
```

```
Decision to build new CSs,
(loc_id, Nr_CP):
[(36, 2.0), (48, 10.0), (58, 8.0), (61, 8.0), (96, 8.0), (110, 7.0), (115, 12.0), (178, 25.0)]

Decision to update old CSs,
(loc_id, Nr_CP, Nr_total_CPs):
[]
Decision to expand SSs,
(loc_id, size_expansion):
[(6, 200.0), (8, 300.0), (74, 100.0)]

Decision to use expensive backstop tech at SSs,
((loc_id, period), amount_backstop usage):
[]
```

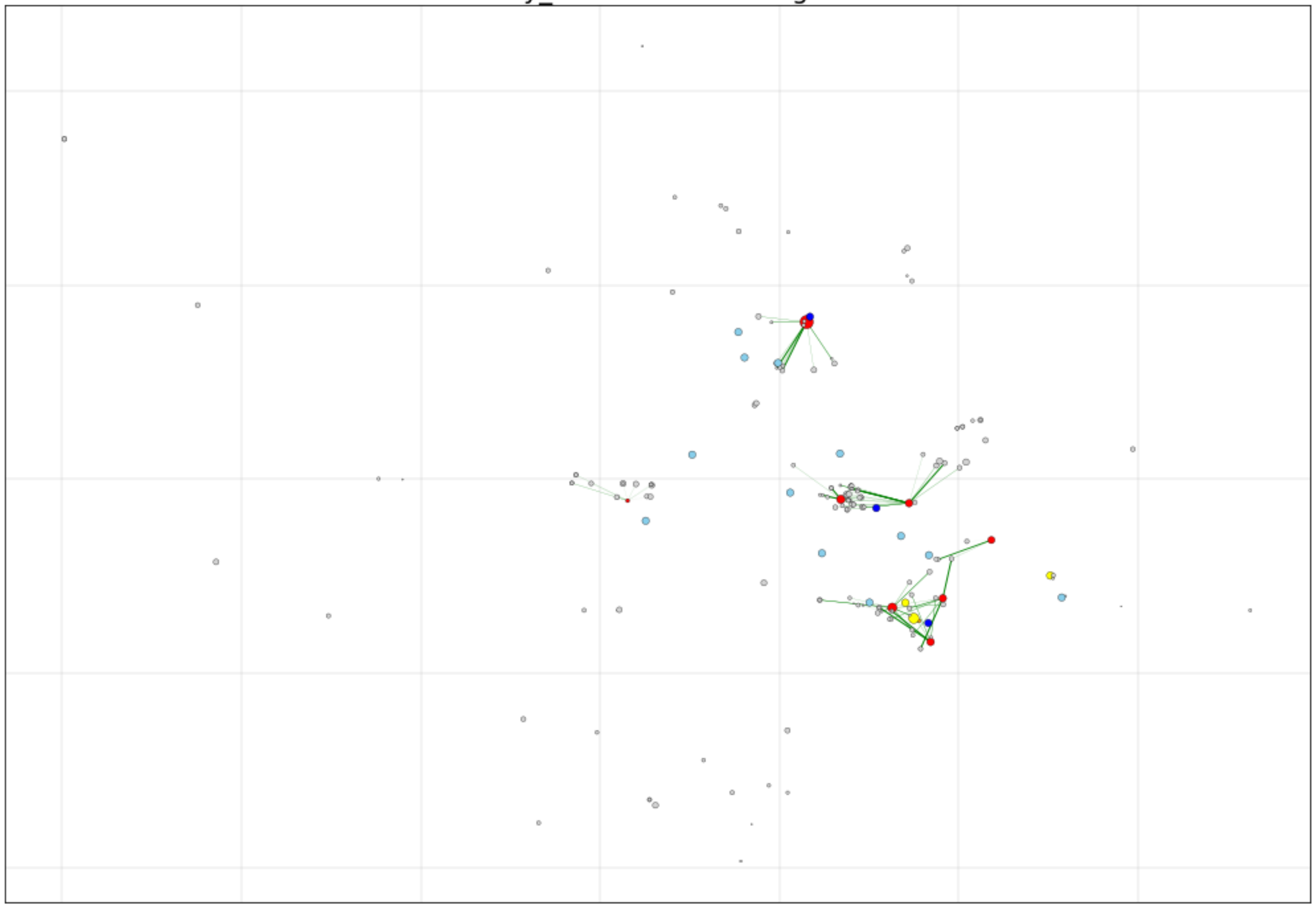
Plot the solved model

In [ ]:
```
mpdp_frame_model.cons_MaxTotalCSs.display()
```

```
cons_MaxTotalCSs : Size=1
    Key  : Lower : Body : Upper
    None :  None :  8.0 :     8
```

In [ ]:
```
Dap._plot_model_graph(
    model = mpdp_frame_model,
    cd_period = 'day_normal',
    model_is_solved = csap_is_solved,
    save_fig = False # Set True to save a high resolution .svg file
)
```

day_normal CD coverage



Step 4: MPDP quick set-up by data retrieved from frame_model

The model set-up method below only takes around 2 seconds. Much faster than the set-up time of one minute before.

In [ ]:
```
mpdp_by_frame_data = SupSP._build_mpsp_csap_from_mpdp_frame(
    mpdp_frame_data = data_mpdp_frame, # test_results_dict[f'phi_IJ_{i}/60']['solved_model_data'],
```

```
    cs_ss_connect_sce = mpdp_connection_sce.stack().to_dict()
)
```

```
Done! set up MPSP_CSAP took 2.07 seconds.
```

The Solution to `mpdp_by_frame_data` will be exactly the same as that of `mpdp_frame_model`.

In [ ]:
```
solver_results_fast_mpdp = Solver.solve(mpdp_by_frame_data)
csap_is_solved = True
print("Solver Message:", solver_results_fast_mpdp)
```

```
Solver Message:
Problem:
- Name: tmpzte5xc1o
  Lower bound: 1599429.4024634599
  Upper bound: 1599429.4024634599
  Number of objectives: 1
  Number of constraints: 3486
  Number of variables: 6842
  Number of nonzeros: 14704
  Sense: maximize
Solver:
- Status: ok
  User time: 0.14
  Termination condition: optimal
  Termination message: MIP - Integer optimal solution\x3a Objective = 1.5994294025e+06
  Statistics:
    Branch and bound:
      Number of bounded subproblems: 0
      Number of created subproblems: 0
  Error rc: 0
  Time: 0.3211369514465332
Solution:
- number of solutions: 0
  number of solutions displayed: 0
```

In [ ]:
```
print_decision = True
if print_decision:
    Stac._print_decision(mpdp_by_frame_data)
#     _print_decision(mpdp_frame_model_4_sp)
```

```
Decision to build new CSs,
(loc_id, Nr_CP):
[(36, 2.0), (48, 10.0), (58, 8.0), (61, 8.0), (96, 8.0), (110, 7.0), (115, 12.0), (178, 25.0)]

Decision to update old CSs,
(loc_id, Nr_CP, Nr_total_CPs):
[]
Decision to expand SSs,
(loc_id, size_expansion):
[(6, 200.0), (8, 300.0), (74, 100.0)]

Decision to use expensive backstop tech at SSs,
((loc_id, period), amount_backstop usage):
[]
```

Step 5: set-up and solve MPSP (An example with 10 Scenarios)

In [ ]:
```
from mpisppy.opt.ef import ExtensiveForm
```

In [ ]:
```
# 0. Define solver options:
num_threads = 2
solver_options = {"solver": "cplex",
```

```
    "threads": num_threads, # Define the max. number of CPU threads used. MPI-SPPy suggested small number, such as 2.
    "warmstart": False
}
```

Step 5.1: Set up Extensive Form (EF)

through `csap_scenario_creator` , `data_mpdp_frame` (Generated in **Step 3.2**) `all_connection_sces_dict` with 10 scenarios (Created in **Step 2**).

In [ ]:

```
scenario_names = list(all_connection_sces_dict.keys()) # `all_connection_sces_dict` Created in Step 2
tic = time.perf_counter()
print(f"""Building Stochastic Model with {len(all_connection_sces_dict)} Scenarios.
    You will see {len(all_connection_sces_dict)} times the same model set-up message. \n""")
MPSP_ef = ExtensiveForm (options = solver_options ,
    all_scenario_names = scenario_names,
    scenario_creator = Sceg.csap_scenario_creator,
    scenario_creator_kwargs = {
        "mpdp_frame_data": data_mpdp_frame, # Generated in Step 3.2
        "all_sces_dict": all_connection_sces_dict, # Created in Step 2
        }
    )
toc = time.perf_counter()
print(f"Succeed! SP Model set-up took {round(toc - tic,2)} seconds in total.\n")
# # 2. Pass EF to the solver to solve :
# solver_results = MPSP_ef . solve_extensive_form ()
```

```
Building Stochastic Model with 10 Scenarios.
    You will see 10 times the same model set-up message.

[  117.95] Initializing SPBase
Done! set up EVCSAP_MPSP took 2.81 seconds.
Done! set up EVCSAP_MPSP took 3.01 seconds.
Done! set up EVCSAP_MPSP took 3.42 seconds.
Done! set up EVCSAP_MPSP took 3.02 seconds.
Done! set up EVCSAP_MPSP took 4.0 seconds.
Done! set up EVCSAP_MPSP took 2.51 seconds.
Done! set up EVCSAP_MPSP took 3.56 seconds.
Done! set up EVCSAP_MPSP took 2.58 seconds.
Done! set up EVCSAP_MPSP took 2.17 seconds.
Done! set up EVCSAP_MPSP took 2.57 seconds.
Succeed! SP Model set-up took 32.13 seconds in total.
```

Step 5.2 Solve EF

In [ ]:

```
print('Solving MPS-MILP')
tic = time.perf_counter()
solver_results = MPSP_ef.solve_extensive_form()
toc = time.perf_counter()
print(f"  Done! It took {round(toc - tic, 2)} seconds to solve the extensive form of SP. \n")
# print(solver_results)
```

```
Solving MPS-MILP
  Done! It took 23.91 seconds to solve the extensive form of SP.
```

Retrieve data of solved MPSP and show results

In [ ]:

```
test_results = Stac._get_data_from_solved_MPSP(
    ef_solver_results = solver_results,
    solved_MPSP_extensive_form = MPSP_ef,
    num_scenarios = 10,
    ids_scenarios = scenario_names,
    )
```

In [ ]:
```
test_results.keys()
```

Out[ ]:
```
dict_keys(['num_scenarios', 'ids_scenarios', 'objective_value', 'prob_description', 'solver_info', 'gap', 'mpsp_csap_decisions', 'subsce_1_data'])
```

In [ ]:
```
pd.Series(test_results)
```

Out[ ]:
```
num_scenarios                                                     10
ids_scenarios                            [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
objective_value                                       1280203.519633
prob_description       {'Lower bound': 1280203.5196326366, 'Upper bou...
solver_info            {'User time': 1.0, 'Termination message': 'MIP...
gap                                                              0.0
mpsp_csap_decisions    {'x': {'48': 1.0, '58': 1.0, '61': 1.0, '96': ...
subsce_1_data          {'obj_total_profit': 1316703.5196326352, 'cs_p...
dtype: object
```

In [ ]:
```
test_results['prob_description']
```

Out[ ]:
```
{'Lower bound': 1280203.5196326366,
 'Upper bound': 1280203.5196326366,
 'Number of constraints': 96015,
 'Number of variables': 68411}
```

In [ ]:
```
test_results['solver_info']
```

Out[ ]:
```
{'User time': 1.0,
 'Termination message': 'MIP - Integer optimal solution\\x3a Objective = 1.2802035196e+06',
 'Statistics': {'Branch and bound': {'Number of bounded subproblems': 13, 'Number of created subproblems': 13}, 'Black box': {}},
 'Time': 2.3301239013671875}
```

In [ ]:
```
for deci_var, dv_values in test_results['mpsp_csap_decisions'].items():
    if deci_var != 'z':
        print(f"{deci_var}: {dv_values}")
    else:
        print(f"z:\n {pd.Series(dv_values)}")
```

```
x: {'48': 1.0, '58': 1.0, '61': 1.0, '96': 1.0, '110': 1.0, '114': 1.0, '115': 1.0, '178': 0.9999999999999996}
y: {'48': 10.0, '58': 8.0, '61': 7.000000000000014, '96': 8.0, '110': 8.0, '114': 16.0, '115': 12.0, '178': 10.999999999999995}
h: {'0': 5.0, '2': 0.9999999999999997, '6': 0.9999999999999997, '8': 5.0, '10': 3.000000000000001, '13': 1.0, '92': 1.0}
z:
 48,15,day_normal      0.113531
48,15,night           1.000000
48,46,day_normal      0.485299
48,46,day_peak        0.041068
48,46,night           0.096736
                        ...
195,111,day_peak      0.097585
195,144,night         0.014131
195,160,night         0.180419
195,176,day_peak      0.078758
195,182,day_normal    0.136105
Length: 272, dtype: float64
```

END