New Simulation Methods to More Effectively Integrate High Levels of Renewable Energy Resources

FINAL REPORT

Team Number: 21
Client: MISO Adviser:
James McCalley
Team Members: Jared Rickard, Jeremy Nash, Nidhi Rawell, Collins Ntwali
Team Email: sdmay20-21@iastate.edu
Team Website: http://sdmay20-21.sd.ece.iastate.edu/

Contents

Acknowledgment

We would like to thank James Okullo, Armando Figueroa-Acevedo, Yifan Li, David Severson, and Ryan Hay from the Midcontinent Independent System Operator (MISO) team for their experience and insights during our client meetings and email conversations. We would also like to thank Dr. James McCalley for his helpful suggestions during our discussions.

1. Introduction

1.1. Problem and Project Statement

MISO runs thousands of production cost simulations (year-long hour-by-hour simulations of the electric and economic performance of the MISO grid) every planning study cycle to investigate a host of topics including the efficacy of proposed transmission upgrades, the impact of federal policy, and the complexity of integrating large amounts of renewables to the system. These simulations, which model the entire Eastern Interconnect of the US power system, take large amounts of processing time due to high model dimension with a large number of load and generation profiles. The projected high growth of renewable penetration in the MISO footprint, and the resulting increase in modeling data, will only exacerbate the situation.

To integrate more renewable energy more efficiently and effectively onto the grid, new modeling techniques are needed. The goal of this study is to, therefore, research and implement various methods to appropriately reduce the fidelity of the data profiles while maintaining an adequate amount of the key production cost information. The study will validate the methods' reliability and quantify the effects that profile approximation has on simulation runtime and results. It is expected that the properly designed profile reduction method would make the normal 8760 hours production cost simulation more efficient. The increased efficiency has the potential to enable us to explore more ways to improve our current planning study processes by introducing cutting edge research in academia and industry.

1.2. Operation Environment

The final program developed will be used by MISO engineers and modelers to assist in running simulations. The results from testing will be reviewed by engineers and used in determining the best program inputs to fit their profiles.

1.3. Intended Users and Uses

Our intended user is a transmission engineer or modeler who plans to run production cost modeling simulations and needs the simulation to produce results faster without losing too much accuracy.

1.4. Assumptions and Limitations

Table 1: Assumptions and Limitations

Assumption	***Reason***
Input CSVs will have the format: • Row 1 is a nodal ID • Column 1 is sequential time stamps • Cells will state a MW value for that node at that time stamp. The value will be relevant to the data being represented by the CSV	This is the only format that we have been given so far
Python will be the only coding language used in this project	MISO engineers use Python more than other data analysis languages, such as R, and are very familiar with the libraries in Python.
K-means is the best clustering algorithm to design our program around	K-means provides a direct relationship between its main user input, the number of clusters, and the results runtime while providing an indirect relationship between user input and accuracy. This was preferable compared to other clustering algorithm's indirect relationships between user input and time or accuracy.
Our code must calculate and cluster around attributes instead of the full profiles	The process of clustering around the entire profiles would be too time consuming and we believe properly chosen attributes better capture the features of the profile that need to be preserved.
Limitation	*Reason*
There are currently no options for any other input CSV formats	We have not been given any other formats to design for.
Our program ignores the final days of the year if they do not make up a full subsection. For example, if the user wants ideal weeks, the program ignores the last day because it does not fit.	This was designed so as not to give those final days any unfair advantage during the comparison and clustering algorithm. These days can just be remade from the first days of the previous subsection.
Our study only looked at weekly clustering	Due to time constraints we tried to focus our search on only weekly clustering, however the program can allow for other sized subsets.

1.5. End Products and Deliverables

Below is a list of the major deliverables expected by our client.

- Program that allows the user to set the amount of representative data they want from profile and the attributes they want the program to use when determining this representative data.
- Analysis of both the chosen data and the results of simulating that chosen data.
- Instructions for using the program and recreating/expanding the results obtained
- A final presentation to the client summarizing these results.
- Detailed explanation of further research opportunities utilizing this program.

2. Implementation

The current method used by the program takes in a production cost model's set of renewable generation and load data in the form of CSV files. It then applies a four step calculation to the data: divide the input into subsets by time, calculate attributes related to each subset, find ideal, imaginary attributes clustered around what was just calculated, and find the subsets who are closest to these ideal attribute sets, making these the ideal subsets. Finally, the code outputs the selected data with the same format along with an indexing file that indicates which sections of the data are represented by which chosen data. This file also contains a summary of the attributes used in the calculation for review.

Input: Each CSV table in the profile must be formatted such that each row represents an interval of time and each column represents a node in the network. The profile data will be split among the CSVs so that similar data (e.g. wind generation, solar generation, nodal load) are grouped in the same files.

Divide the input into subsets by time: Once the code has read the data it will section off the rows of the data into evenly spaced time subsets. The size of these is defined by the user. The point of this process is to make the next step of applying attribute calculations to specific time slices easier.

Calculate attributes related to each subset: Next, the code will apply attribute calculations to each subset. These include finding basic attributes like the maximum generation of each node during the subset, total energy generated at each node, or maximum positive ramp from one measurement to the next. A full list can be seen in the table below. The user can set different attribute lists for each input table.

Table 2: Attribute Descriptions

Attribute	Description
max	Maximum value of each node
min	Minimum value of each node
total energy	Cumulative energy of each node
difference	Difference between maximum and minimum value of each node
max ramp	Largest positive change from one period to the next for each node
min ramp	Largest negative change from one period to the next for each node
percent high	Percentage of periods above 75% max generation for each node
percent low	Percentage of periods below 25% max generation for each node
percent extremes	Percentage of periods above 90% or below 10% max generation for each node

The program can also apply calculations based on the nodes' spatial information. To do this the user inputs a type of spatial data, where it can be found, what value to split across, and attributes to apply to this split. As an example, the user would supply an extra file containing the node IDs and their corresponding latitudes, the user would tell the program to group the nodes into sets above and below a certain latitude, say 45. Then the program would sum all the values across the nodes in each group. After that, the program would find the hourly difference between the two groups. The user then decides which attributes to apply to this difference, using the same possible attributes listed in the table above. This process can be repeated as many times with as many different choices as defined by the user.

Once all the attributes are calculated for each input table, they are joined into one 2-dimensional matrix. The reason for this is so that the clustering algorithm compares everything as a unified profile.

Find ideal, imaginary attributes clustered around what was just calculated: Now that the program has attributes for each time subset, it passes that data through a clustering algorithm (in our case K-means). This algorithm iteratively decides the best way to cluster the time subsets into groups based on similarities in the attributes. The number of groups is a user defined number (at least for K-means).

Once they are grouped the clustering algorithm provides the average set of attributes for each group. In a sense these attributes represent the most ideal time subset to represent all the other subsets in the group.

Find the subsets who are closest to these ideal attribute sets, make these the ideal subsets: Unfortunately, these ideal sets of attributes cannot be backtracked into full data sets. We cannot design a full profile for each node based only on the max generation or load of that node. Therefore, we must instead find the subset from the input data whose attributes are closest to these ideal attributes. We then designate this close subset the ideal subset that represents all the other subsets in the group.

Output: Finally, the program outputs these chosen subsets in the same format as the input. It was designed this way to allow for quickly dropping in the file in the simulation software. The program also outputs a secondary file that states which subsets are represented by which subsets. This is to easily copy out the results from simulations of the chosen data to the entire profile. This file also contains the average value of each attribute applied for each subset.

3. Testing Process and Results

For testing purposes our client provided multi-year data for load, wind generation, and various solar generation types. These were used to develop the program. Later they provided a full Plexos profile with data files containing load, wind generation, photovoltaic (PV) solar generation, and distributed photovoltaic (DPV) solar generation. We will only be discussing results derived from the four data files in the Plexos profile so that comparisons remain consistent throughout this section. We will also only be discussing results from weekly comparisons (7 day/168 hour subsections). Finally, we will only be discussing results from selections made from unnormalized attributes. We have results from normalized data as well for sections 3.1 and 3.2. However, we ran out of time to simulate them and do not have any results for them in section 3.3. Therefore, we chose to only discuss results from unnormalized data to remain consistent.

This section will start with a brief visualization of the output of the program. From there the section contains results visualizing and comparing the attributes, effects that attributes had on week choices, and comparisons of simulation output.

First, to better visualize what this program is meant to do, below is a graph showing an approximation of a profile's data set by weeks with different numbers of clusters. The x axis indicates the actual weeks of the year and the y axis indicates which week was chosen to represent the actual week. The colors show the progression of choices for representation when more clusters can be chosen.

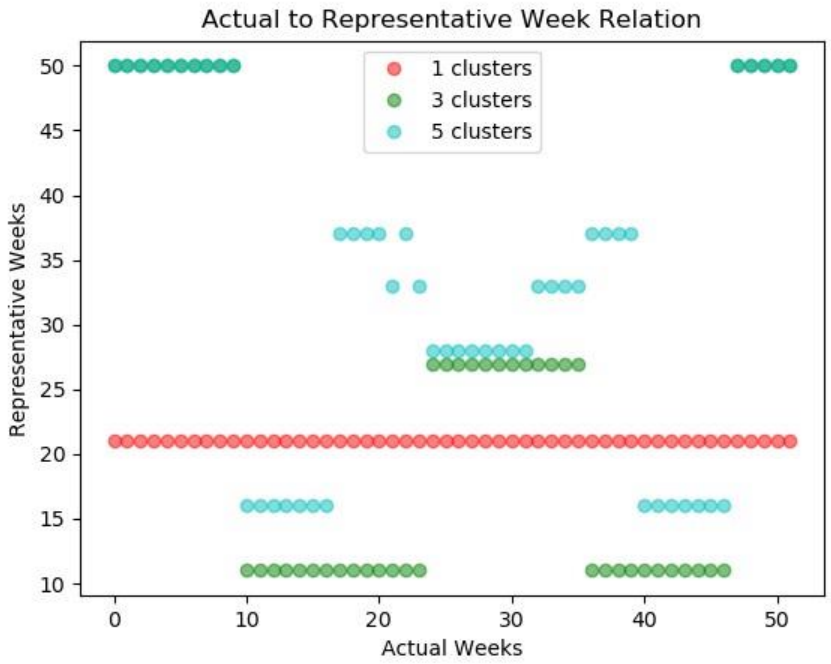

Figure 1: Comparison of actual weeks and the weeks chosen to represent them with increasing clusters

Obviously, with only one cluster a single week was chosen, week 21. This week then represents the entire year for being the most average. If we were to progress this graph to 52 clusters, each week would be represented by itself. Of note is the false downward trend shown with more the higher cluster counts. This is most likely due to the cyclic nature of this data and that fall data will look very similar to spring data.

3.1. Attribute Comparisons

Our first task was to understand how the attributes 'looked' across the all the data files so that we could determine which ones would provide the most effective clustering. While there are nine attributes, because we have 4 different data types there are really 36 attributes that can be applied to this profile. For example, maximum load is not comparable to maximum wind generation, which is also not comparable to maximum PV solar or maximum DPV solar. Therefore, these similar calculations can be considered different attributes.

With that being said, we wanted to visualize the attributes across the year. To start, we took the average of every attribute across all nodes and graphed them. We also wanted to show how effective the clustering was at capturing these attributes, which can indicate how effective the attribute was at influencing the selection.

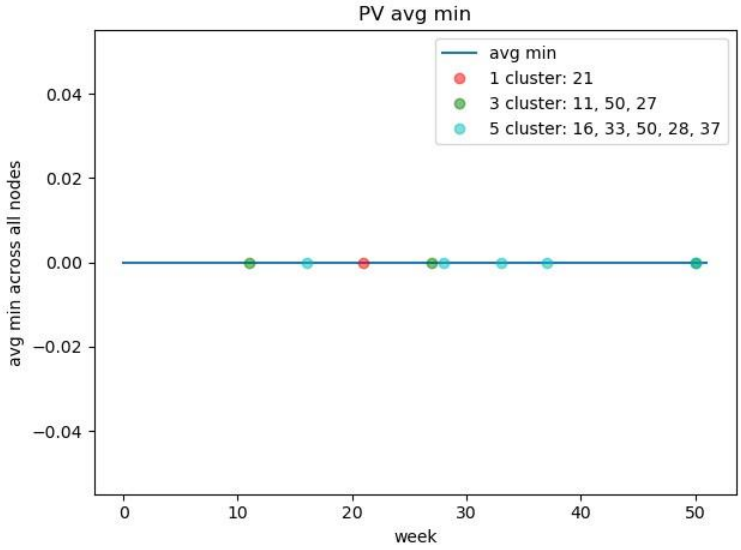

Figure 2: Photovoltaic solar average minimum value across all nodes for every week along with indicators for selected weeks

This first graph shows a completely ineffective attribute. As stated in the attribute description table, the minimum attribute returns the minimum value of each node for each week. It then compares the minimum of each node across all the weeks. Obviously, the minimum of every week is going to be zero because every week experiences nighttime. Therefore, the average minimum of every week is zero. This attribute is useless because it does not add anything to the selection to compare 52 zeros for every PV solar node.

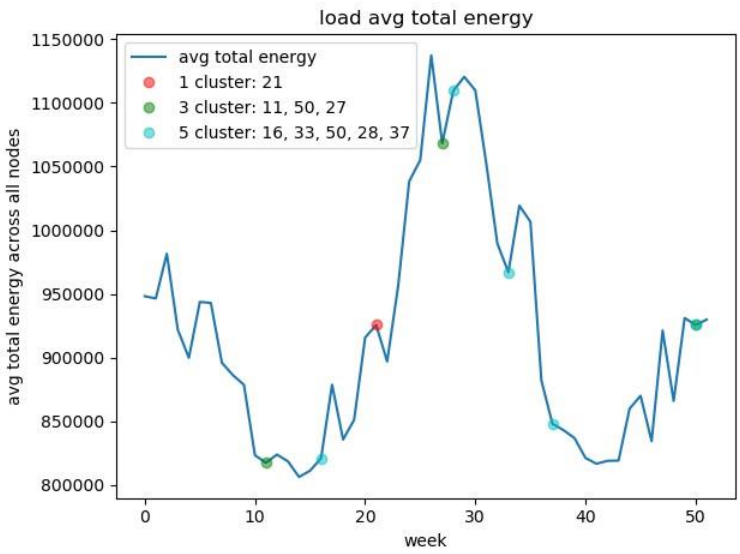

Figure 3: Load data average total energy across all nodes for every week along with indicators for selected weeks

This next graph shows an effective attribute. Not only does it have a range of values, unlike the previous attribute, the clustering selections do a good job capturing that range. As shown by the dots, when one cluster is selected to try and represent the entire data, the week chosen has an about average value for this attribute. As more clusters are added, the new ones move to capture the extremes even better.

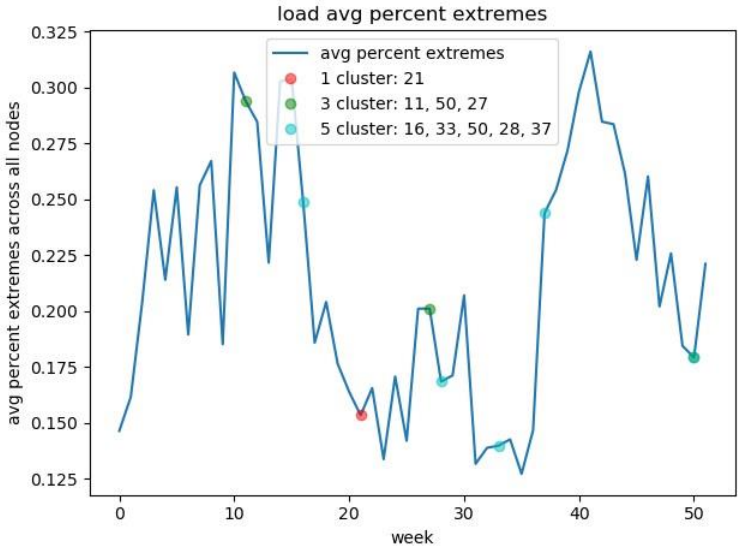

Figure 4: Load data average amount of hours in the extremes across all nodes for every week along with indicators for selected weeks

The graph of the average percent extreme values for load data shows an attribute that is not very effective for low cluster numbers but might become more effective for higher cluster numbers. With one cluster the week chosen does not represent this attribute very well. However, as more clusters are allowed, they begin to spread out over the range of data. This is a common theme among many of the

attributes and shows that as more clusters are allowed it would be a good idea for the user to include more attributes in the calculation to optimize the representation.

Another way to visualize these results is by looking at the percent difference between the attributes of the actual weeks and the attributes of the weeks they are chosen to be represented by. Each box graph was calculated by finding the percent difference between the average value of each week and average value of the chosen week to represent it.

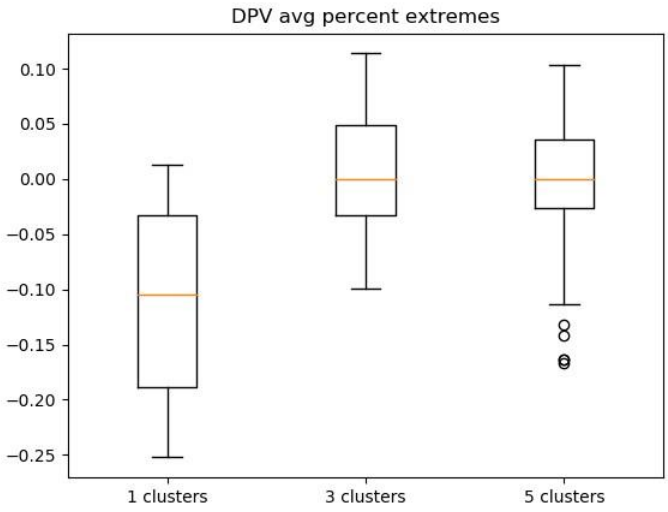

Figure 5: Percent difference between the calculated value of the percent extremes attribute and the value of the selected weeks for each week

Here we see an attribute that improves its average as the cluster count goes up but starts to lose accuracy in its outliers as well. This could indicate that this attribute is not as effective as the first two box plots would indicate, but that its improvement is coincidental.

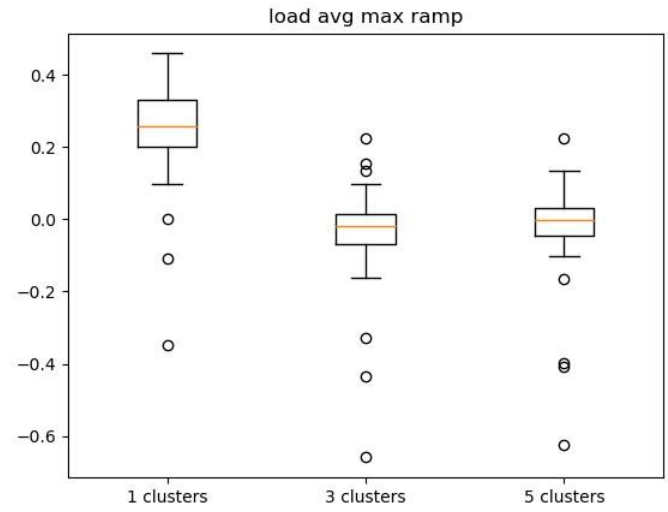

Figure 6: Percent difference between the calculated value of the maximum ramp attribute and value of the selected weeks for each week

The above graph shows an attribute that not only improves its approximation of the average with more clusters, it also pulls in its outlying percent difference. This indicates that as more clusters are added, the attribute has enough influence to make sure that its variability is also captured.

The purpose of these graphs is to show how some attributes are more influential in the selection process than others. This should better explain some of the selections made in the results later. This also shows that if the user wishes to capture the variability of certain attributes, they may have to sacrifice the consideration of other attributes.

3.2. Selection Comparisons

Moving away from the in-depth analysis of all the attributes together, we are going to take a more general look at different attribute combinations. Due to computational limitations, it would be impossible to test every possible combination of attributes, seeing that there are nine across four files so 36 attributes. Instead we look at the attributes on a smaller scale, specifically by file.

Our next step was to try every combinations of attributes for a single file, 512 combinations. We then grouped the results so to compare whether including one attribute changed any of the selected weeks. For example, when we clustered only the wind file with the attributes percent low and percent high, the chosen week was 15. However, if we included maximum so that the attribute list was maximum, percent low, and percent high, the chosen week was 21. We can say from this that the attribute maximum made a difference on the outcome of the selection. If instead we looked at the list percent low and total energy, adding maximum did not change the outcome because each attribute lists selected week 21. We can say that maximum did not make a difference here. Repeating this step for all attributes in a file, across all files, produces the following graph.

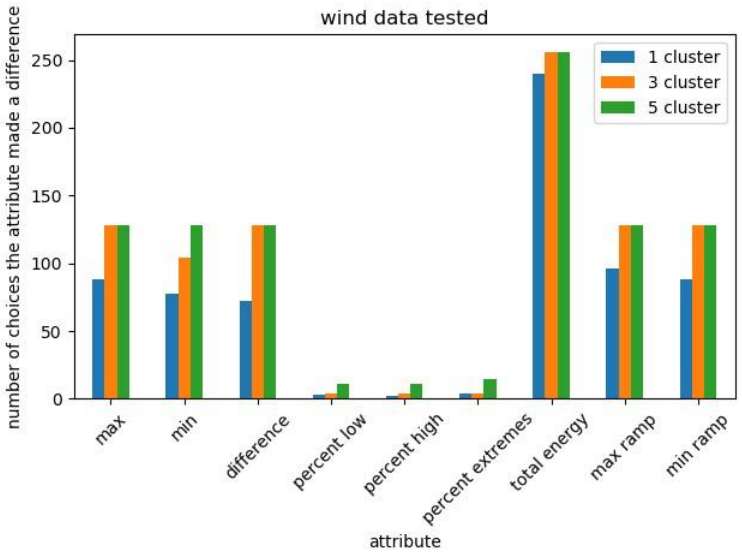

Figure 7: Comparison of wind attributes' influence on week selection due to increasing clusters

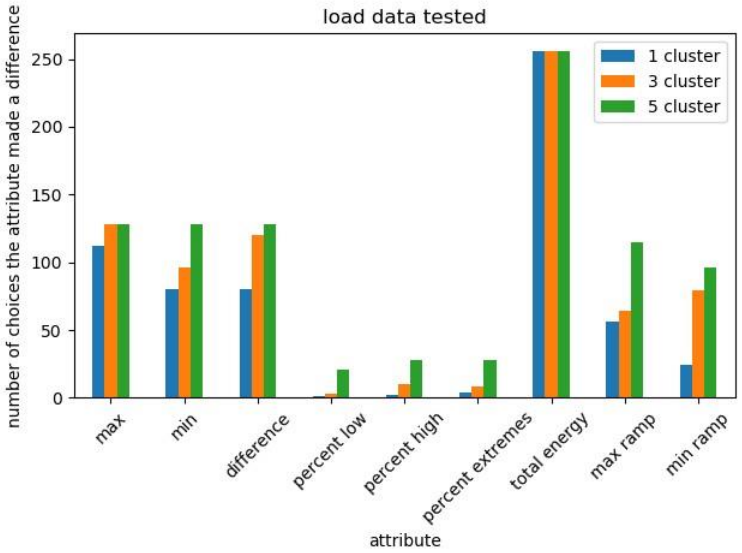

Figure 8:Comparison of load attributes' influence on week selection due to increasing clusters

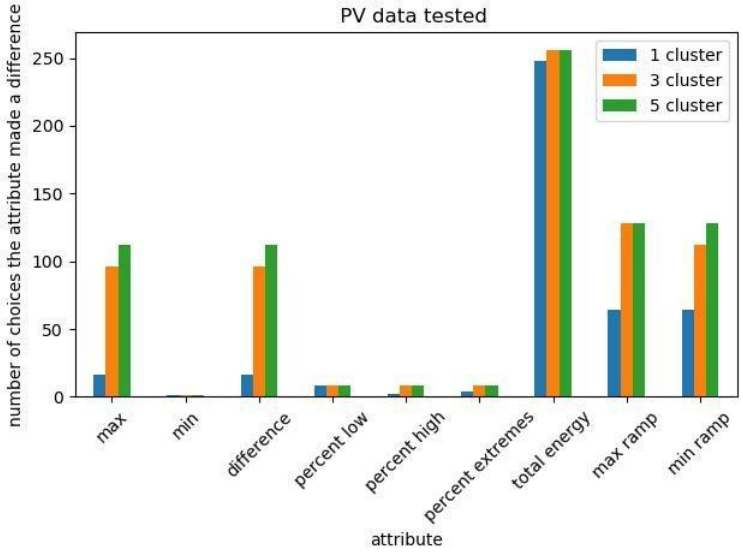

Figure 9: Comparison of PV solar attributes' influence on week selection due to increasing clusters

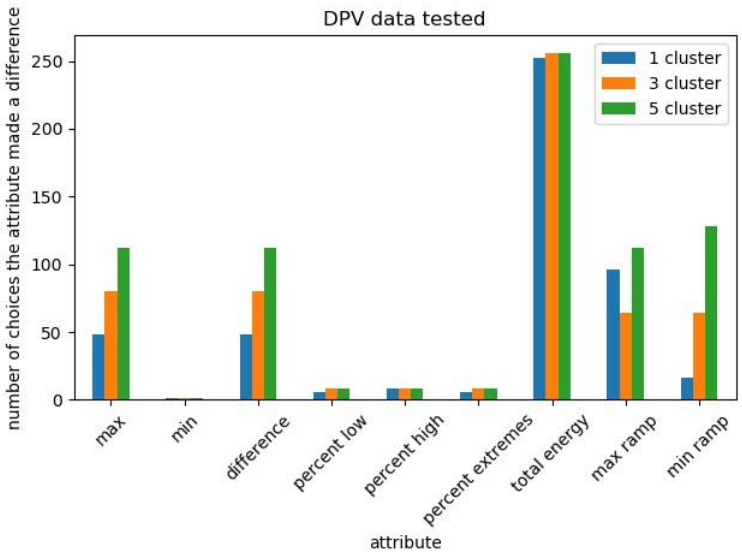

Figure 10: Comparison of distributed PV solar attributes' influence on week selection due to increasing clusters

From this process we can determine which attributes affect the outcome the most in the individual files. For example, we can draw the same conclusions about the minimum attribute for solar data as we did in the previous section. Because minimum almost never makes a difference in the combinations it's included in, we can say that it either has no range of values or its variability isn't being represented properly. From the previous section we know that it is the former. Other attributes, such as total energy for every file, show that they are almost always, if not always, the deciding factor in the decision for which weeks are selected.

Then there are attributes, such as maximum for the solar files, that show increasing influence with more clusters. This is probably due to the fact that as more clusters are added, there is more room to capture the variability of less influential attributes.

It should be noted that this comparison does not capture all the effectiveness of the attributes when more than one cluster is used. This is because the comparison does not capture the edges of the groups. For example, while it is true that percent low very rarely affects the selected weeks, it could move weeks that were at the border of one group into another group.

3.3. Simulation Result Comparisons

We ran three comparison tests on Plexos output data, comparing full year output with our approximated output. These tests included locational marginal pricing (LMP), Production Cost, and capacity factor. We used least squares error when comparing the 2 outputs, meaning that as the gap between the numbers increases, the error exponentially increases. Since running 52 Plexos runs would take too long, we chose random weeks to test as well as the weeks our code chose. The chosen week's Plexos output was appended to an array 52 times to form a full year array of identical weeks, and this was compared to the full year run.

The LMP shows what the price of a unit of energy was at that specific hour over the MISO footprint. The error values ranged from 1.38*10^5 to 4.93*10^5 with the chosen week's error being 1.72*10^5. This

was the second-best error value of the group and proved to us that the chosen week was a good one. For the graph below, our chosen week was week 21.

Figure 11: Comparison of LMP error between selected weeks and the full data profile

Production Cost is a measure of how much it costs to produce a unit of energy at that given time. We summed up all the generators in the MISO footprint for each hour of the year, and then formed full years of identical weeks for our testing weeks and chosen week. After comparison, the error values ranged from 3.61*10^16 to 6.42*10^16. Our code's chosen week had the lowest error of the group. The error was so big here because the inputted values were huge, typically in the millions of dollars per hour, and squaring large values means large errors.

Finally, we looked at capacity value which is just each hour of generation divided by the max generation that **could** be achieved at that hour. We used the full year max capacity for both the individual weeks and the full year comparison. Our chosen week's capacity factor was 42.45% capacity, very close to the yearly rate of around 40%. This further proved that our chosen week was a reliable pick.

4. Context

There are four aspects of our design project where we decided to go in a certain direction as opposed to taking the alternative approaches/views/perspectives. The four aspects are: clustering methods, criteria for selecting representative data samples, weighting certain attributes, and methods for validating results.

4.1 Alternative clustering methods

In our project, our input is a generation/load profile that we apply calculations to study and analyze it carefully. These calculations provide key information about the data. Our approach then was to group similar sections of the data by clustering around this key information. To do this, we are using a clustering algorithm called K-means. K-means forms a fixed user-set number (k) of clusters (collections of data points) in a dataset like ours, based on certain information contained in the data points. From our research, we found that K-means was the simplest, yet most efficient, method best suited for our needs and assumptions.

However, the K-medoids clustering algorithm was a good alternative method. Its inputs are like K-means, so it could easily be implemented in our code without requiring more work from the user. K-Medoids also has the potentially important characteristic which is that it finds a cluster and represents it with one of its members, rather than the mean of its members as in the case of K-means. Our use of the K-means does something similar because we find ideal points then pick the points closest to ideal. However, since K-medoids does this in one step that could be an advantage over K-mean. A topic for further research could be to try K-medoids and see if it picks the same 'centers', or if doing it in a 2-step process like ours produces a better (or worse) solution.

A recommendation would be to further investigate the affects each clustering algorithm has on the dataset and then select a main clustering algorithm. We are making this recommendation because we did not have time to implement other clustering algorithms in the project, as this would require reevaluating what we want the user inputs to be and finding how those affect the final product. Therefore, it might be beneficial to see how changing the clustering algorithm will affect the results. We think this might be beneficial because it looks for our results from a different, and possibly better, viewpoint.

4.2 Selecting Representative Data Samples

Our main objective is to reduce computation time of PCM while maintaining result accuracy within a certain margin. To address this issue, we needed to narrow the datasets computed by finding fewer sample sections of the data representative of the original dataset and simulate those instead, hence getting the same vital information in a fraction of the time. Since the original dataset is always going to be in the format - time periods x node, our approach was to subdivide the periods into either days, weeks, or months and then calculating attributes of those subsets; i.e. total energy, max, min, difference, average, etc.

But there were other approaches that we found while researching the topic. “The 8760 hours that make up a year are broken down into time blocks (referred to here as ‘time slices’) that capture seasonal, weekly, and daily variations” (Planning for the renewable future: Long-term Modelling and tools Expand Variable Renewable Power in Emerging Economies by International Renewable Energy Agency (IRENA))[5]. The representative time slices were within the range of 12 to 64. In the paper, Hierarchical Clustering to

Find Representative Operating Periods for Capacity-Expansion Modeling by Yixian Liu and Ramteen Sioshansi selected represented days[6]

4.3 Validating Results

An essential part of our study is to validate the developed method and demonstrate its effectiveness from a statistical perspective. In our project, we validate our software tool by simulating actual production cost models using Plexos and comparing the results with those found after simulating approximated production cost models. The margin of error statistically tells us how effective the chosen attributes, cluster count, and subset size combination is.

Given more time, we could have tested as many combinations of clusters against attributes as possible to get a better statistical sense of reliability. This was not possible because of too many questions and not enough time. We can better validate our results by using the elbow method to determine the best attribute/cluster/subset size combination to use. We tried doing this but didn't get enough data to create it. An IEEE study found that the elbow method could be useful in determining the maximum number of clusters that represents most of the variance of the data[7]. Therefore, increasing the number of clusters beyond this point will show significantly less variance of the data. In the code written for the project, the number of clusters can be changed to any number. This makes the code flexible in adjusting the number of clusters.

4.4 Weighting Certain Attributes

The first misunderstanding is that attributes can't be weighted. In clustering terms there are samples and features. Relating this to our work each week (or day/month) is a sample and each attribute for each node constitutes a feature. The distance calculation used in K-means and other clustering algorithms considers each feature a dimension in space. When finding the cluster center, they generally compute the center of mass of all the samples in a cluster. This center of mass equation:

$$(R = \frac{1}{M}\sum_{i=1}^{n} m_i r_i = (\frac{1}{M}\sum_{i=1}^{n} m_i x_i \ , \frac{1}{M}\sum_{i=1}^{n} m_i y_i \ , \frac{1}{M}\sum_{i=1}^{n} m_i z_i \ , \dots))$$

calculates the center of each equation independently of each other. Therefore, weighting a dimension, or a feature, or an attribute, won't do anything to the final calculation.

However, those m_i show that we could add weights to certain samples, or weeks (days/months). And the choosing of what weeks get what weights can depend on what values they get for certain attributes. i.e. if a week shows high ramp across many nodes (certain features), then we could add weight to that week (sample).

Initial reading didn't show any work on this subject; however, they are probably using a different term for this concept and further reading might shed more light. A few reasons we ignored this concept is because we misunderstood it as weighting the attributes and we didn't have time to add this new dimension of possible opportunities to the work.

In the paper, Planning for the Renewable Future: Long-term Modelling and Tools Expand Variable Renewable Power in Emerging Economies by International Renewable Energy Agency (IRENA) highlighted the usage of representative time slices to break down the 8760 hours that make up a year. This is different from the work done in the project because the 8760 hours were broken into down into representative weeks to capture variations in the project. The issue of selecting representative data subsets was an issue looked at in this project and in this paper. Also, weighting attributes was looked at

as an issue for the project, and IRENA decided to add "capacity-credit" values. In the project, weights could be added to certain samples, or weeks (days/months). Moreover, the choosing of what weeks get what weights can depend on what values they get for certain attributes I.e. if a week shows high ramp across many nodes (certain features), then we could add weight to that week (sample). In this paper, capacity credit was added to the models in order to reflect the contribution of the alignment of demand and supply. The paper, Hierarchical Clustering to Find Representative Operating Periods for Capacity-Expansion Modeling by Yixian Liu, and Ramteen Sioshansi used the hierarchical clustering method to find clusters in the data, whereas in our project, K-means was used to cluster the data. All clustering methods have the goal of minimizing the sum of the distances between every object or periods of time and the cluster's centroid or median.

4.5 References

[1] Dr. Aishwarya Batra, "Analysis and Approach: K-Means and K-Medoids Data Mining Algorithms"

[2]Fränti, Pasi, and Sami Sieranoja. "How Much Can k-Means Be Improved by Using Better Initialization and Repeats?" *Pattern Recognition*, vol. 93, 1 Sept. 2019, pp. 95–112., doi:10.1016/j.patcog.2019.04.014.

[3]Sun, Mingyang, et al. "Data-Driven Representative Days Selection for Investment Decisions: A Cost-Oriented Approach." *IEEE TRANSACTIONS ON POWER SYSTEMS*, 4 July 2019.

[4]Almaimouni, Abeer, and Atinuke Ademola-Idowu. "Selecting and Evaluating Representative Days for Generation Expansion Planning." *Selecting Representative Days for Capturing the Implications of Integrating Intermittent Renewables in Generation Expansion Planning Problems - IEEE Journals & Magazine*, 2018, ieeexplore.ieee.org/document/7527691/.

[5]IRENA. "Planning for the Renewable Future Long-Term Modelling and Tools to Expand Variable Renewable Power in Emerging Economies." *IRENA â International Renewable Energy Agency*, 2017, www.irena.org/publications/2017/Jan/Planning-for-the-renewable-future-Long-term-modelling-and-tools-to-expand-variable-renewable-power.

[6]Y. Liu, R. Sioshansi, and A. Conejo, "Hierarchical Clustering to Find Representative Operating Periods for Capacity-Expansion Modeling," IEEE Transactions on Power Systems, Vol. 33, No. 3, May 2018.

[7]Poncelet, Kris, et al. "Selecting Representative Days for Capturing the Implications of Integrating Intermittent Renewables in Generation Expansion Planning Problems." *IEEE Transactions on Power Systems*, vol. 32, no. 3, May 2017, pp. 1936–1948., doi:10.1109/tpwrs.2016.2596803.

Appendices

A. Manual

This manual will be broken into 3 sections: Input file formats, GUI walkthrough and option explanation, and instructions for expansion.

For this code to work, the user must have “approx_steps.py”, “attribute_calculations.py”, “spat_att_calculations.py”, and “GUI.py” in the working folder. They must also have the python libraries “numpy”, “pandas”, and “scikit-learn” installed.

All questions and error reports can be emailed to jared.rickard357@gmail.com

A.1. Input File Format

Nodal measurement data: These files contain data obtained every time period from nodes in the electric grid. This data must be organized in a .CSV file such that the first column contains datetime stamps and the first row contains node identifiers (names, numbers, etc.). A cell at the intersection of a node and datetime should contain the value measured at that node at that time.

It is assumed that all measurements in a single file will be related to each other (e.g. A .CSV will contain all load measurements in the grid, a .CSV will contain all wind generation in the grid, etc.). It is also assumed that the datetime stamps will be the same period and will be sequential.

Nodal spatial data: These files contain data pertaining to a node’s location, such as latitude, longitude, surrounding population, etc. This data must be organized in a .CSV file such that the first column contains node identifiers (the same identifiers used in nodal measurement data) and the first row contains headers labeling the type of spatial data. A cell at the intersection of a header and node should contain that header’s type of information at that node.

It is assumed that all data contained in a column will be the same type and will have a meaningful less than/greater than comparison. It is also assumed that these node identifiers can be matched to the node identifiers in nodal measurement data.

A.2. GUI Walkthrough and Option Explanation

After starting the program, the user will be greeted with a window containing many textboxes and options. To use this tool, please fill out Input File, Output Destination, and Output Name. If the user wishes to cluster by comparing nodal attributes, please fill out Attributes. If the user wishes to cluster by comparing groups of nodes organized by their spatial data, please fill out as many rows of Spatial type, Spatial value, and Spatial attributes as required. If the user wishes to cluster by comparing both they may fill out both. If the user has multiple nodal data files to cluster, they should press More Files and fill out the new text boxes. The user must also fill out all text boxes below the More Files button as well. When all options have been correctly set, press the approximate button. When done close the window to end the program.

Below is an explanation of the various options.

Input File: This text box should contain the location of a nodal measurement data file (e.g. D:/Downloads/DPV.csv). A button to the right, ‘Find File’, will open a file explorer to search for the file. If

an input file section is not needed, leaving this text box blank will make sure the code ignores all fields related to this input file.

Output Destination: This text box should contain the directory to store the approximated version of the input file (e.g. D:/Downloads/temp). A button to the right, 'Set Directory', will open a directory explorer to search for the directory.

Output Name: This text box should contain the name for the file(s) storing the approximated version of the input file (e.g. DPV_cluster_#.csv). If 'Write data into one CSV with consecutive dates' is unchecked and a '#' is contained in the name, then the cluster number will replace the '#' in the final files. If 'Write data into one CSV with consecutive dates' is unchecked but there is no '#' in the name, then the cluster number will be inserted before the '.csv' of the name. If 'Write data into one CSV with consecutive dates' is checked then one file is written and the cluster numbers are ignored.

Attributes: This text box should contain the attributes the user wishes to apply to the input file. This list will be in a comma-space delimited format (e.g. max, min, total energy, difference). The list of currently useable attributes is given below:

Table 3: Attribute Descriptions

Attribute	Description
max	Maximum value of each node
min	Minimum value of each node
total energy	Cumulative energy of each node
difference	Difference between maximum and minimum value of each node
max ramp	Largest positive change from one period to the next for each node
min ramp	Largest negative change from one period to the next for each node
percent high	Percentage of periods above 75% max generation for each node
percent low	Percentage of periods below 25% max generation for each node
percent extremes	Percentage of periods above 90% or below 10% max generation for each node

Spatial Data Location: This text box should contain the location of the nodal spatial data related to the input file (e.g. D:/Download/Spatial Bus Mapping Data.csv). A button to the right, 'Find File', will open a file explorer to search for the file. If no spatial data is needed this text box can be left blank.

New Spatial Choice: This button opens a new row of text boxes corresponding to 'Spatial type', 'Spatial value', and 'Spatial attributes'.

Spatial type: This text box should contain a type of spatial data the user wants to split the measurement data by. The text here must correspond to a header in the nodal spatial data file (e.g. A header in the nodal spatial data says 'Lat' so this text box could contain Lat).

Spatial value: This text box should contain the value the user wishes to split the nodes between, in terms of the values under the spatial type data (e.g. nodes in the grid could range across latitudes 22 to 54, so the user might pick spatial type = Lat and spatial value = 35 to apply attribute calculations to the difference between nodes above and below latitude 35). This value must be in the same format as the data in the nodal spatial data.

Spatial attributes: This text box should contain the list of attributes the user wants to apply to the difference found from spatial type and spatial value. This list can contain the same attributes found in the attribute table.

More Files: This button will create a new box containing another set of options for a new input file. This button can be used as many times as needed to create input file options for all the data files related to the simulation profile.

Where to store indices: This text box the directory to store the indices data (e.g. D:/Downloads/temp/). A button to the right, 'Set Directory', will open a directory explorer to search for the directory.

What to call indices file: This text box should contain the name for the file storing the indices (e.g. indices.csv). This file outlines the representation calculated and metadata statistics.

Number of desired clusters: This text box should contain an integer determining how many representative subsets to calculate. This value should be in the range 0 : (number of periods / size of subsets).

Size of subsets: This text box should contain an integer determining the length of subsets which will be compared (e.g. if the user wishes to compare weeks against each other to find representative weeks, they should type 168 for hourly data, because 168 hrs = 1 wk).

clustering algorithm: This dropdown list contains all the useable clustering algorithms. NOTE: Currently only kmeans is available. However, if more options become available, they could change the meaning of previous text boxes, like 'Number of desired clusters'. Please make sure this stays up to date should more options be added later.

Normalize metadata: If this box is checked, during the metadata calculations all the columns will be divided by their maximum values to normalize the metadata before clustering. Default: unchecked

Show progress messages: If this box is checked, during the approximation process, descriptive messages will appear in the python console describing what section of code just finished and how long it took to accomplish. Default: checked

Write the chosen data to CSVs: If this box is checked, the chosen representative data will be written to CSVs in the same format as the input file. Default: checked

Write indices CSV: If this box is checked, the file containing representative information and metadata statistics will be written. Default: checked

Write data into one CSV with consecutive dates: If this box is checked, all the chosen subsets will be stored in one CSV with datetime stamps starting at the beginning of the profile (e.g. 4 weeks are chosen to represent an entire year. The data from these chosen weeks is stored in one CSV, the end of one week leading into the beginning of the next week. All there datetimes are replaced with consecutive timestamps beginning Jan 1st).

approximate: This button will start the python code approximating all the input files listed with the options chosen.

A.3. Instructions for Expansion

Adding more attribute calculations: All the attribute calculation functions are found in 'attribute_calculation.py'. They are defined such that their input is a 1D array of generation or load data and their output is a single value. This 1D array is exactly 'subset_size' long (24 for daily, 168 for weekly, 720 for monthly). This input corresponds to one node's data for one subset term. The output is whatever single value you want to calculate for one node during the subset term.

To create your own attributes, start by creating a function definition with one input (the 1D array of generation or load data for the subset term). Inside the function perform whatever calculations you wish to find the attribute you want. Finally return the single attribute value you want for that generation for that subset term. To link it to the rest of the code, go to the definition of "att_set()" and start a new line inside the "switcher" dictionary variable. In this line write "[name of attribute]" : [function name]. Finally, to use this attribute, write out [name of attribute] in the attribute list inside the GUI.

What happens when the code runs is that the attribute list is passed to the metadata building code, which goes node by node, attribute by attribute, subset by subset, calculating the attributes. When the attribute name gets passed to "att_set()" it matches it to the function in "switcher" and performs that calculation.

Adding more clustering algorithms: The clustering algorithms are contained in functions defined in 'approx_steps.py'. They are defined such that their inputs are the number of clusters and the metadata matrix of the input data and their outputs are a 2D array of distance measurements and a 1D array of cluster labels. The array of distance measurements is between the subsets and the ideal attribute sets. Each row represents a subset of the data and each column represents an ideal data set. The array of labels is the list stating which column of the distance array is smallest for each row.

To create your own clustering algorithm wrap, create a function definition with two inputs (the clusters and metadata). From here the internals depend drastically on what python has for implementations of the clustering algorithm of choice. The next part of this is function needs to return a 2D array of distances and 1D array of labels, outlined previously. To link this new function to the code go to the definition of "calculate_dist_and_labels()" and start a new line inside the "switcher" dictionary variable. In this line write "[name of algorithm]" : [function name]. To create a new option in the GUI, go to line 42 of 'GUI.py', the line defining "algorithm_choices", and add "[name of algorithm]" to the list.

Similar to what happens in the attribute calculation, the algorithm string gets passed to "calculate_dist_and_labels()" and uses it to pass on the arguments to the proper function.

NOTE: These instruction gloss over a lot about creating the internals of the function. This is because the implementation is going to differ drastically depending on what is added. A very likely issue to arise is that many algorithms will require different inputs than just data and cluster count. A good example is 'DBSCAN()' from sklearn.cluster library. This function and the algorithm behind it doesn't allow the user to set the cluster number, however it does let the user define things like "the maximum distance between two samples for one to be considered as in the neighborhood of the other." (taken from sklearn.cluster.DBSCAN help page). To pass these through is left as an exercise to the reader, however a good start would be reviewing how arguments are passed or ignored by python functions. In the same vein, a lot of user inputs that could be set for kmeans and kmedoids have been left as default in our implementation. If someone wanted to change these around, they would have to change them in

the function definition or rewrite to “approximate()”, “calculate_dist_and_labels()”, and the wrapper function of choice to pass on the relevant arguments. Not difficult, the task is just making sure it gets passed down correctly.