ABSTRACT

LIANG, MING. A Machine Learning-Based Approach for Synthetic Distribution Feeder Generation. (Under the direction of Dr. Ning Lu).

Test systems are widely used by researchers and engineers to test conceptual designs, optimize parameter settings, and validate performance. However, developing high-fidelity distribution feeder models requires access to utility network models and customer data, which is a major barrier for the research community to have unrestrictive, unlimited number of customizable, realistic test systems for research and development purpose. So far, there has been very little attempt made towards the manual and static test system design principles, making creating an ensemble of test systems from actual feeder models a daunting task. Motivated by this, the dissertation aims at developing an end-to-end, machine-learning-based approach for automated, customizable test feeder generation using actual feeder models as inputs.

This dissertation presents a novel, automated, generative adversarial networks (GAN) based synthetic feeder generation mechanism, abbreviated as FeederGAN. FeederGAN digests real feeder models represented by directed graphs via a deep learning framework powered by GAN and graph convolutional networks (GCN). Information of a distribution feeder circuit is extracted from its model input files so that the device connectivity is mapped onto the adjacency matrix and the device characteristics, such as circuit types (i.e., 3-phase, 2-phase, and 1-phase) and component attributes (e.g., length and current ratings), are mapped onto the attribute matrix. Then, Wasserstein distance is used to optimize the GAN and GCN is used to discriminate the generated graphs from the actual ones. A greedy method based on graph theory is developed to reconstruct the feeder using the generated adjacency and attribute matrices. After the feeder topologies and attributes are generated, we then use a statistical, rule-based method to generate the load transformers. The rules make the transformers follows line capacity constraints, line attributes constraints and topology

constraints. Finally, we write the generated feeder file with load information into OpenDSS format and run combined case studies. The results show that the GAN generated feeders resemble the actual feeder in both topology and attributes verified by visual inspection and by empirical statistics obtained from actual distribution feeders.

In the second part of the dissertation, a synthetic load generation method is developed via a novel sequential energy disaggregation (SED) algorithm. The SED algorithm is presented for extracting heating and cooling energy consumptions from residential and small commercial building loads using low-resolution (i.e. 15-minute, 30-minute, and 60-minute) smart meter data. The method is validated using data collected from 137 households in the PECAN street project. Results show that the proposed SED method is computationally efficient, simple to implement, and robust in performance. Based on the SED algorithm developed, case study is conducted on buildings with photovoltaic (PV) systems and electric vehicles (EVs). Among the cases, load database of zero-net energy (ZNE) cases, ZNE ready cases, ZNE with EV cases is built up. Therefore, those load data can be then used together with the generated synthetic feeders to test different planning and operational strategies.

A Machine Learning-Based Approach for Synthetic Distribution Feeder Generation

by
Ming Liang

A dissertation submitted to the Graduate Faculty of
North Carolina State University
in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

Electrical Engineering

Raleigh, North Carolina

2020

APPROVED BY:

_____	_____
Dr. Ning Lu Committee Chair	Dr. David Lubkeman
_____	_____
Dr. Mesut Baran	Dr. Min Chi

DEDICATION

To my mother HU Youshu, my father LIANG Xianyu and my brother LIANG Hong.

BIOGRAPHY

Ming Liang was born in Sichuan, China. He received his B.S. degree in Electrical Engineering and Automation from Chongqing University, Chongqing, China in 2014 and the M.S. degree in Electrical Engineering from North Carolina State University, Raleigh, NC, USA in 2019. He is currently pursuing the Ph.D. degree in electrical engineering at North Carolina State University. His research interests include developing and applying data analytics and machine learning in power distribution system modeling and planning.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1 INTRODUCTION

1.1 Research Work Overview

Test systems are widely used by researchers and engineers to test conceptual designs, optimize parameter settings, and validate performance [1]–[6]. A test system can be either a real feeder model or a generalized, derived, or extended versions of an actual feeder. For example, a number of new test systems can be obtained by varying the network topology, adjusting the placement and parameters of apparatus (e.g., adding new devices, removing old ones, retrofitting, etc.), and changing user patterns in an actual utility feeder model. A realistic test system needs to faithfully reproduce important distribution system behaviors in different operation conditions so that researchers and engineers can benchmark their algorithms and conduct studies on a wide range of operation conditions with less development time, lower testing cost, and zero risk to interrupt actual system operation.

In recent years, the needs for such realistic distribution feeder test systems are increasing drastically because of the integration of distributed energy resources (DERs) [7, 8], energy storage systems [9, 10] and electric vehicles (EVs) [11, 12]. To model the uncertainty and variability in DER operation as well as the variations in its operation environment (e.g., changes in topology, variation in model parameters, user patterns, etc.), an ensemble of realistic distribution test feeders are often needed.

However, developing high-fidelity distribution feeder models requires access to utility network models and customer data, which is a major barrier for the research community to have unrestrictive, unlimited number of customizable, realistic test systems for research and development purpose. Meanwhile, domain experts, such as operation and planning engineers, have conventions to follow and specific design criterions to meet when designing actual engineering

systems. Those inherent characteristics are very hard to be captured by researchers even when they can access actual system models.

Faced with the great demand of testing feeders, this dissertation aims to develop high fidelity, realistic synthetic feeders for distribution system simulation and research. The goal is not only to develop methodologies and algorithms to generate realistic synthetic distribution feeder topology and hierarchy, but also to generate realistic synthetic load data. Hence, comprehensive studies, e.g. renewable integration study, can be conducted using a large amount of synthetic feeders and data. In the following sections, the current research status of synthetic feeder generation and load disaggregation is introduced.

1.2 Synthetic Feeder Generation

Since 1991, 10 IEEE test feeders were developed to meet researcher's needs, approximately. In 2009, led by a group of researchers at Pacific Northwest National lab, 24 taxonomy feeders were developed from 575 actual feeders from 17 utilities across the US [3, 23]. At Texas A&M, researchers are focusing on developing synthetic electric grid models that are designed to be statistically and functionally similar with actual electric grids while containing no confidential critical energy infrastructure information [4].

However, so far, there has been very little attempt made towards the manual and static test system design principles, making creating an ensemble of test systems from actual feeder models a daunting task. In recent years, researchers develop a few methods that use street maps to align and generate feeder layouts [24]–[26]. For example, in [25], Saha et al. use an open source street map tool to create individual synthetic distribution feeders and groups of feeders in an area with the same ZIP code using a small number of input data, while in [26], Domingo et al. use a reference model to generate large-scale radial feeders using street maps as inputs. In [27], [28], Birchfield et

al. summarize topological and statistical criterion to validate the realism of synthetic feeders. In [29], Shweitzer et al. conduct a two-stage study to generate synthetic feeders where they treat the feeder as a graph with nodes and edges. In the first stage, a thorough analysis is conducted to summarize the topological and physical characteristics where divergence is used to measure the statistical similarity of real and synthetic feeders. In the second stage, summarized statistics of the feeders are used to generate synthetic feeders having similar characteristics with those Netherland feeders. The shortcoming of these approaches is that they are semi-automatic and lack of the intelligence to learn feeder typology or devices attributes from ingesting actual feeder models, making them unsuitable for generating a large amount of realistic, up-to-date test feeders.

In lieu of this, our aim is to develop an end-to-end, machine-learning-based approach for automated, customizable test feeder generation using actual feeder models as inputs. In [30], Kipf et al. propose the graph convolutional networks (GCN) for classifying graph data, such as social network and chemical molecules. Based on GCN, many follow-up studies choose to use a generative adversarial networks (GAN) with GCN to generate graph data. For instance, in [31], Fan and Huang use labeled graph in GAN to generate citation graph and protein graph. A recurrent network based model is used in [32] by You et al. to generate synthetic graphs.

Motivated by those approaches, a GAN-based synthetic feeder generation mechanism is developed, abbreviated as FeederGAN. FeederGAN digests real feeder models using a deep learning framework powered by GAN and graph convolutional networks (GCN). From the information provided in the input files of a power system feeder model, first the device connectivity is mapped to the adjacency matrix, and then the device characteristics such as circuit types (i.e., 3-phase, 2-phase, and 1-phase) and component attributes (e.g., length and current ratings) are mapped to the attribute matrix, making GCN training possible. Wasserstein distance

is used to optimize the GAN for discriminating the generated graph from the actual. A method based on graph theory is introduced to reconstruct feeder topology from the obtained adjacency and attribute matrix.

The FeederGAN has three major characteristics. *Frist*, the topology is learnt from ingesting actual feeder models instead of street maps, making it possible to extract inherent distribution system design features. *Second*, to our best knowledge, it's the first time to use a GAN-based end-to-end and plug-to-play model for generating synthetic feeders. *Third*, FeederGAN is scalable and customizable so its user can generate either a whole feeder or substructures of a feeder at user selected scale and size.

1.3 Load Disaggregation

A common goal of load disaggregation algorithms is to extract time series end use load profiles from a total load profile. In general, load disaggregation (LD) methods can be classified into two categories: power disaggregation (PD) and energy disaggregation (ED). The main difference between PD and ED is that ED calculates the average power consumption in a period of time instead of the exact power consumption at each time instance. For example, if a building load is metered at 1-minute or higher data sampling rates, PD algorithms can be applied to obtain the minute-by-minute power consumption of an end use load. On the other hand, if the building load is metered every 30-minute, ED algorithms are used to estimate the energy consumption of an end use load in each 30-minute interval. As can be seen in this example, ED algorithms normally do not calculate the power consumption of an end use load nor its on/off sequence. This is because the end use load may only be on for part of the 30-minute interval (e.g. 5 out of 30 minutes).

PD algorithms are often used to detect the power consumption and on/off status of end use loads for power management purposes. An important category of PD is Non-intrusive Load

Monitoring (NILM). NILM algorithms identify power demand of each appliance within an aggregated load profile [13, 14] using a set of rules or parameters that uniquely define the electrical behavior of an individual device or appliance when it is in operation. The data required for NILM is normally collected by power quality meters (sampling rate in kHz) or high-resolution (<5-minute) smart meter data [14, 15].

In distribution planning, the separation of building heating and cooling loads are critical for assessing the impacts of different energy efficiency and demand response programs because the heating and cooling loads are widely used in those programs. NILM methods are normally sufficient to extract the heating and cooling loads when high-resolution smart meter data is available. However, although recently installed smart meters allow customer consumptions to be metered at the sub-minute level, most utilities only meter 15-, 30-, or 60-minute customer energy consumptions. There are two main considerations from the utilities' perspective. First, costs of communication, storage, and computing of high-resolution smart meter data for millions of customers are still prohibitively high. Second, collecting high-resolution smart meter data has raised serious privacy concerns from customers all over the world [16] because high-resolution data reveal too much private information, such as occupancy and customer lifestyles. Thus, in practice, most NILM-based methods cannot be used by utility engineers to perform load disaggregation studies because the smart meter data resolution is much lower than 5 minutes.

So far, very few research projects have been conducted to develop ED-based load disaggregation methods using low-resolution (i.e. 15-, 30-, or 60-minute) smart meter data. Albert and Rajagopal proposed in [17] a Hidden Markov Model (HMM) to convert hourly kWh data into thermal states for identifying suitable demand response resources using thermostatically controlled appliances. Chin and Tanaka proposed a states-based method similar to HMM in [18] to

disaggregate temperature-sensitive load and segment customer into different groups for energy saving plans. In [19], Ji and Xu used Fourier time series to disaggregate hourly HVAC end-use for commercial building loads. The above methods focus mostly on extracting fuzzy thermal states of appliances and their applications are limited to certain building types.

If an ED method can only fit for certain types of buildings, its application in distribution system planning is very limited. Hence, a more stream-lined and computationally efficient method with robust performance across different buildings is needed for ED using low-resolution smart meter data collected from thousands or millions of buildings. Thus, in this research, a novel sequential ED algorithm was presented. The algorithm consists of three key process: large, infrequently used load detection and removal, day type detection, and average value subtraction, for disaggregating end use energy consumptions using smart meter data with 15-, 30- or 60- minute resolution.

The goal of the sequential energy disaggregation (SED) algorithm developed is to estimate energy consumptions of major building loads so utility engineers can assess impacts of different energy (e.g. cooling/heating) efficiency and demand response programs on revenue and grid operation using low-resolution smart meter data. Note that data with a sampling rate at one minute or less is considered as high resolution and at five minutes and above is considered as low resolution. In this research, the focus is on extracting the energy consumption of the water heater load and the building heat, ventilation, and air conditioning (HVAC) loads, electrical water heater load and baseline load. Hence, synthetic load database on each typical appliance can be built. Then, based on the synthetic appliance load, zero-net energy (ZNE) building cases study is conducted to obtain synthetic load data in different scenarios, especially with the consideration of high penetration of photovoltaic (PV) and electric vehicles (EVs).

A Zero-Net-Energy (ZNE) building refers to a building whose total amount of energy used on an annual basis is roughly equal to the amount of renewable energy generated on site. The US Department of Energy sets a goal to achieve marketable zero energy homes in 2020 and commercial zero energy buildings in 2025 [20]. California becomes a pioneer in promoting ZNE buildings in its effort to achieve carbon emissions reduction by 80% by the year 2050 compared to 1990. The first groups of new residential, government, and commercial ZNE buildings will be built in 2020, 2025 and 2030, respectively [21].

A few steps can be taken to retrofit non-ZNE buildings to ZNE. First, energy efficient appliances are used to replace older and less efficient appliances. Such retrofit will make a building "ZNE ready". Then, installing photovoltaic (PV) systems so that self-generated solar power can approximately meet the annul energy consumption of the buildings, which makes the building ZNE [22]. Energy storage devices (ESD) can also be installed to maximize self-consumption and reduce backfeeding power.

To study the impact of ZNE on customer energy savings and on utility operation, in 2015 an experimental ZNE community was built at Fontana, California. Energy consumptions of the ZNE building and energy efficient appliances were recorded and processed by EPRI. Based on those measurements, EPRI published a report [21] discussing the impact of deploying ZNE buildings on electric distribution systems operation. However, although the EPRI report has quantified impact of ZNE buildings for the Fontana community in detail using actual measurements, the size of the Fontana community is too small for its load consumption patterns to be representative. In addition, the impacts of the ZNE buildings on utility feeder operations are highly correlated with network topology, solar irradiance, load mix, and voltage control mechanisms. Therefore, in this research, a customer-oriented, data-driven method is presented for

quantifying the impact of Zero-Net-Energy (ZNE) buildings on customer savings and on distribution system operation. Smart meter data collected from 5000 buildings in North Carolina on three distribution feeders are used to conduct this study.

In the study, a tool based on Matlab is developed to conduct the sequential energy/load disaggregation and build up a synthetic load database for different ZNE cases, with high penetration of PV and EVs.

CHAPTER 2 SYNTHETIC FEEDER GENERATION VIA DEEP GRAPH LEARNING

This chapter introduces the study on utilizing deep learning based framework to generate synthetic distribution feeders. Synthetic feeder test systems are widely used by researchers and engineers to test conceptual designs, optimize parameter settings, and validate performance. However, developing high-fidelity distribution feeder models requires access to utility network models and customer data, which is a major barrier for the research community to have unrestrictive, unlimited number of customizable, realistic test systems for research and development purpose. Hence, a mechanism to automatically generate a large number of synthetic feeders is in great demand.

In this chapter, we present a novel, automated, generative adversarial networks (GAN) based synthetic feeder generation mechanism, abbreviated as FeederGAN. FeederGAN ingests real feeder models represented by directed graphs via a deep learning framework powered by GAN and graph convolutional networks (GCN). From power system feeder model input files, device connectivity is mapped to the adjacency matrix while device characteristics such as circuit types (i.e., 3-phase, 2-phase, and 1-phase) and component attributes (e.g., length and current ratings) are mapped to the attribute matrix. Then, Wasserstein distance is used to optimize the GAN and GCN is used to discriminate the generated graph from the actual. A greedy method based on graph theory is developed to reconstruct the feeder from the generated adjacency and attribute matrix. The results show that FeederGAN can generate feeders of high quality that resemble the actual feeder in both topology and attributes verified by visual inspection and by empirical statistics obtained from actual distribution feeders. It would be a great contribution to the community of power distribution research [43].

After the feeder topologies and attributes are generated, we then use a statistical, rule-based

method to generate the load transformers. The rules make the transformers follows line capacity constraints, line attributes constraints and topology constraints. Finally, we write the generated feeder file with load information into OpenDSS format and run a combined case study.

2.1 Feeder Generation Problem Formulation and Algorithm

This section introduces basic problem formulation of feeder generation, the deep learning concepts used in the method, the preparation of training data, and the problem formulation of GAN and GCN for generating synthetic feeder topology and attributes.

2.1.1 Graphical Representation of Feeders

In this subsection, we introduce two main considerations when converting a power system feeder model to a machine-learning friendly directed graph model and then discuss the problem formulations of the directed graph model.

The *first* consideration is to omit the geographical information and use the length of the feeder line to represent distance. This is because, in power system network models, the electrical distance used for calculating line impedance is of interest instead of the geographical distance. For visualization purpose, a greedy method (introduced in Section 2.3.4) is developed to calculate pseudo coordinates for each bus. The substation (i.e., the head of a feeder) is placed at the origin (0, 0). Then, coordinates of subsequent buses are calculated one by one based on electrical distance and connectivity so that the generated synthetic feeder stretches forward as straight as possible, making it easy to traverse feeder topology and identify the relationships between feeder components. As shown in Figure 2.1, this approach removes the need for using sensitive customer geographic information, such as street maps and geographic coordinates. From the machine learning standpoint, masking geographical information forces the algorithm to use only the relative distance between line segments, preventing the algorithm from learning from non-generic,

geographic information.

The *second* consideration is to replace the traditional bus-as-node and device-as-edge feeder topology representation with a device-as-node representation. Using the device-as-node approach, each device (e.g., a line segment instead of a bus) is defined as a node so the edge will only serve as an indicator to show the direction pointing from the feeder head to a load node. By doing so, the device attributes, such as length, current ratings, phase, can be defined as node attributes.

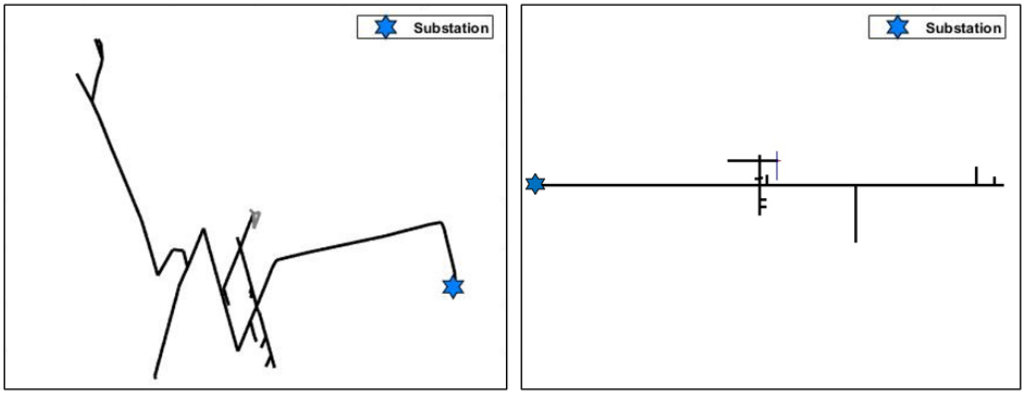


Figure 2.1: Topology representation: feeder using geographical coordinates (left) and using electrical distance (right)

The device-as-node representation allows us to represent each device by a node $v_i \in \mathcal{V}$ and a node attribute vector $\mathbf{x_i}$. The feeder can be represented as a directed graph $\mathcal{G}$ with a set of nodes $\mathcal{V}$ and edges $\mathcal{E}$ so that each edge $(v_i, v_j) \in \mathcal{E}$ connects two nodes and shows the graph direction. Then, the feeder topology can be represented by a directed graph by an adjacency matrix $\mathbf{A} \in \mathbb{R}^{m\times m}$and an attribute matrix $\mathbf{X} = [\mathbf{x_1}, \dots, \mathbf{x_m}]^T \in \mathbb{R}^{m\times d}$, where m is the number of nodes, $\mathbf{A_{i,j}} \in \{0, 1\}$ and d is the dimension of the attributes. After converting the feeder topology to a directed graph, the learning algorithm only needs to learn from **A** and **X**, significantly reducing the problem complexity, as shown in Figure 2.2.

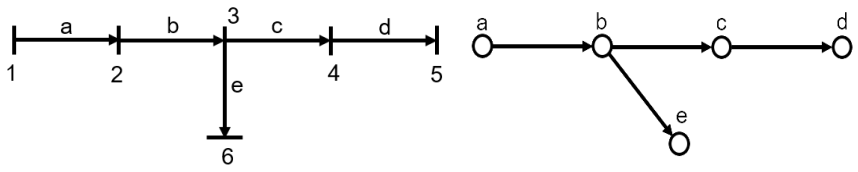


Figure 2.2: Distribution feeder topology representation: Bus-as-node and device-as-edge (left) and device-as-node (right)

Assumption 1: Because traditional distribution feeders are designed as radial networks, we assume that the feeder graph is a directed radial graph and there is no loop in the graph.

Property 1: If Assumption 1 holds, the in-degree is 1 for all nodes except the feeder head node, whose in-degree is 0. This property suggests that there should be one and only one non-zero element in each column of the adjacency matrix **A**, except for the column representing the feeder head.

A traditional distribution feeder is designed as a radial network. Loops in the feeder via tie lines are only used in emergency operation or feeder reconfiguration for balancing the load among phases. Thus, in normal operating conditions, the tie switch is open. Thus, Assumption 1 holds. Without any loop in the network, for each node, there should be one and only one edge goes into it, as defined in the graph theory, the in-degree of the node is 1, i.e., there is only 1 non-zero entity in each column of the adjacency matrix. Because there is no edge goes into the feeder head node, its in-degree is therefore, 0.

Once a feeder model is converted to **A** and **X,** we can use a GAN-based framework to learn from the real feeder and generate the topology $\widehat{\mathbf{A}}$ and attributes $\widehat{\mathbf{X}}$ for a synthetic feeder.

2.1.2 The Deep Learning Framework

FeederGAN digests real feeder models using a deep learning framework powered by GAN and GCN. Deep learning refers to deep neural networks built on several layers of neutrons. If the layer is linearly and fully connected, it is called a fully connected (FC) layer. A model built on

several FC layers is called multilayer perceptron (MLP). To model the non-linearity in the dataset, activation functions are used. In the proposed method, two major activation functions are rectified linear units (ReLU) and softmax. ReLU [44] is formulated as $f(x) = \eta x$, where $\eta = 1$ when $x > 0$; $\eta = 0$ when $x \leq 0$. If the negative part η is a small but non-zero value, the method is called leaky ReLU [44]. Softmax [45] is formulated as $f(x_i) = e^{x_i} / \sum_j e^{x_j}$. Softmax is often used in classification or approximating discrete one-hot values, where a discrete variable is encoded as a vector consisting of binary values (i.e., 0/1). When using Softmax, the output of each layer is normalized as a probability and the sum of all outputs is one.

As shown in Figure 2.3, a GAN [46] consists of two main components: Generator and Discriminator. Discriminator distinguishes whether the input data is real or fake (i.e., generated) while Generator generates fake data sets resembling the real data to exploit weakness in Discriminator. Thus, GAN learns to generate high-quality fake data sets by letting Generator and Discriminator play a minimax game, which ends when Discriminator can no longer distinguish the generated and real data. This process can be formulated as

$$\min_G \max_D V = \mathbb{E}_x[\log (D(x))] + \mathbb{E}_z[\log (1 - D(G(z)))] \tag{2.1}$$

where $D(x)$ is Discriminator outputs, $G(z)$ represents Generator output, x represents the real data, and z represents a random noise. For Discriminator, the first part of Eq. (2.1) is to predict the real as real, and the second part is to predict the fake as fake. $D(x)$ outputs are interpreted as probability with 1 for real and 0 for fake. To make the Discriminator output a probability, a sigmoid function [46] is added to the Discriminator output layer to normalize $D(x)$ into [0, 1]. For Generator, the first part of Eq. (2.1) is not used because the Generator goal is to make Discriminator "think" the fake is real.

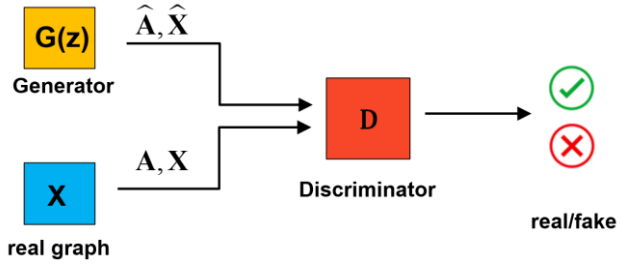


Figure 2.3: Framework of a typical GAN

GCN [30] uses the adjacency matrix **A** to represent the connectivity between nodes and the attribute matrix **X** to store nodal device information. The GCN representation of a directed graph is

$$\bar{\mathbf{A}} = \mathbf{D}^{-1}(\mathbf{A} + \mathbf{I}) \tag{2.2}$$

$$\mathrm{GCN}(\bar{\mathbf{A}}, \mathbf{X}) = \sigma(\bar{\mathbf{A}}\mathbf{X}\mathbf{W}) \tag{2.3}$$

where **I** is an identity matrix, **D** is a diagonal degree matrix with $\mathbf{D}_{ii} = 1 + \sum_j \boldsymbol{A}_{ij}$, and **W** is the network parameter matrix to be learnt. $\sigma(\cdot)$ is an activation function (e.g., ReLU). Note that by using Eq. (2.2), a row-wise normalization is executed to **A**. To simplify future derivations, we let **A** be the normalized adjacency matrix in the rest of the sections.

2.1.3 Feature Engineering

Feeder topology information, i.e., the connectivity between feeder nodes, are coded into **A**, a $m \times m$ matrix. Let $\mathbf{A}_{i,j} = 1$ if there is a connection from node i to j (unilateral) and let $\mathbf{A}_{i,j} = 0$ otherwise. After the whole network is traversed, use Eq. (2.2) to normalize **A**.

As shown in Table 2.1, there are two types of features in **X**: organic and topological. Organic features include device length, current rating, and phase, the information of which can be directly extracted from device definition files. Topological features include the distance between the device and the feeder head, pseudo loads the device carries, and the device level in the overall feeder topology hierarchy. Topological features are obtained by traversing the feeder topology

using either the depth-first search (DFS) or the breadth-first search (BFS).

In Table 2.1, the first four features have continuous, numerical values and are normalized by their extreme values to uniformly map them into [-1, 1]; the last two features are categorical features that bear discrete values and are encoded by one-hot values. Thus, numerical attributes are represented by $\mathbf{X}_{num} \in \mathbb{R}^{m\times 4}$ and categorical attributes by $\mathbf{X}_{cat} = [\mathbf{X}_{cat}^{1}, \mathbf{X}_{cat}^{2}] \in \mathbb{R}^{m\times(d_1+d_2)}$, where m is the number of nodes, and d_1, d_2 are the dimension of each categorical feature, i.e., the number of categories for each feature.

Table 2.1: A summary of the attributes

$\mathcal{O}$: organic, $\mathcal{T}$: topological, $\mathcal{N}$: numerical, $\mathcal{C}$: categorical

Name	Definition	Type	Source
Length	The length of a device.	$\mathcal{N}$	$\mathcal{O}$
Norm Amps	Normal condition conductor amps, an indicator for the conductor materials.	$\mathcal{N}$	$\mathcal{O}$
Distance	Distance from feeder head to the device.	$\mathcal{N}$	$\mathcal{T}$
Pseudo Load	The sum of the capacity of all downstream customer side transformers.	$\mathcal{N}$	$\mathcal{T}$
Level	Start as Level 0 at the feeder head. When encountered a bifurcation leading to several children branches, level+1 if "norm amps" or "phase" of the child is different from that of the parent.	$\mathcal{C}$	$\mathcal{T}$
Phase	1 of the 7 options: *a, b, c, ab, ac, bc, abc*	$\mathcal{C}$	$\mathcal{O}$

2.1.4 FeederGAN Algorithm

In our experiments, the minimax GAN problem formulated in Eq. (2.1) suffers convergence issues caused by gradient vanishing, where the loss of Discriminator decreases quickly when reaching to an optimal solution, causing backward gradients of the Generator to drop to 0. As a result, the Generator can no longer be trained properly. To solve this problem, we apply

the Wasserstein GAN (WGAN) introduced by Arjovsky in [47]. WGAN shares the same model structure with GAN, but by removing the sigmoid output layer of the GAN, WGAN directly outputs a logit, called the critic score, instead of a probability. Thus, the gap of the critic score between a real graph and a generated graph, called the Wasserstein distance, can be used to train the Generator and Discriminator.

The WGAN formulation is

$$\min_G \max_D V = \mathbb{E}_x[D(x)] - \mathbb{E}_z[D(G(z))] \tag{2.4}$$

where x is the real graph, i.e. $\{\mathbf{A}, \mathbf{X_{num}}, \mathbf{X_{cat}}\}$, z is a random noise matrix, and $G(z)$ is the generated graph $\{\widehat{\mathbf{A}}, \widehat{\mathbf{X}}_{\boldsymbol{num}}, \widehat{\mathbf{X}}_{\boldsymbol{cat}}\}$, $\mathbb{E}_x[D(x)]$ and $\mathbb{E}_z[D(G(z))]$ is the critic score the Discriminator gives to the real graph and the generated graph, respectively. Now, the Discriminator's objective is to widen the Wasserstein distance while the Generator's objective is to narrow it.

When there is no constraint on the Discriminator, the value of the Discriminator weights will gradually grow larger and larger and thus fail to train Generator properly. Hence, the Discriminator will be subject to a constraint called the *k*-Lipschitz condition such that

$$|D(x_1) - D(x_2)| < k \times |x_1 - x_2| \tag{2.5}$$

To implement the Lipschitz condition, we conduct weights clipping so that the weights of Discriminator are clamped within $(-c, c)$, where c is the clipping limit. The FeederGAN algorithm is summarized in Table 2.2 (Algorithm 2.1).

Lines 3-4 illustrate the strategy used to train Discriminator as strong as possible. As shown in Figure 2.4, in the first few iterations, the Generator will be trained once every n_1 Discriminator training cycles.

Table 2.2: FeederGAN Algorithm

Algorithm 2.1. FeederGAN. All experiments used the default training parameters α=0.0001, c=0.1, n=20, n_0=500, n_1=1000, n_2=5.

Require: α, the learning rate; c, the weight clipping limit; n_1, n_2, the number of iterations for Discriminator at different stages; n, n_0, the numbers separate different training stages; ω_0, initial Discriminator parameters; θ_0, initial Generator parameters.

$i = 0$
while θ not converge **do**
  $iter \leftarrow n_1$, **if** $i < n$ or $i \,\%\, n_0 = 0$
  $iter \leftarrow n_2$, **else**
  for t=1, 2, …, $iter$ **do**
    Draw a sample $\{\mathbf{A}, \mathbf{X}_{num}, \mathbf{X}_{cat}\}$ from the real dataset
    Generate a random matrix $\boldsymbol{z}$ from the prior distribution
    $\{\widehat{\mathbf{A}}, \widehat{\mathbf{X}}_{num}, \widehat{\mathbf{X}}_{cat}\}_{\boldsymbol{\theta}} \leftarrow G_\theta(\mathbf{z})$
    $g_\omega \leftarrow \nabla_\omega [D_\omega(\{\mathbf{A}, \mathbf{X}_{num}, \mathbf{X}_{cat}\}) - D_\omega(\{\widehat{\mathbf{A}}, \widehat{\mathbf{X}}_{num}, \widehat{\mathbf{X}}_{cat}\}_{\boldsymbol{\theta}})]$ (2.6)
    $\omega \leftarrow \omega + \alpha \cdot \mathrm{RMSProp}(\omega, g_\omega)$
    $\omega \leftarrow clipping(\omega| - c, c)$
  end for
  Generate a random matrix $\boldsymbol{z}$ from prior distribution
  $\{\widehat{\mathbf{A}}, \widehat{\mathbf{X}}_{num}, \widehat{\mathbf{X}}_{cat}\}_{\boldsymbol{\theta}} \leftarrow G_\theta(\mathbf{z})$
  $g_\theta \leftarrow -\nabla_\theta [D_\omega(\{\widehat{\mathbf{A}}, \widehat{\mathbf{X}}_{num}, \widehat{\mathbf{X}}_{cat}\}_\theta)]$ (2.7)
  $\theta \leftarrow \theta - \alpha \cdot \mathrm{RMSProp}(\theta, g_\theta)$
  $i \leftarrow i + 1$
end while

After the Generator has been trained n times, the frequency will be reduced to train Generator once every n_2 Discriminator training cycles. Then, at the n_0, $2n_0$, $3n_0$, … Generator training cycles, we will enforce an aggressive training snapshot to train the Discriminator n_1 times, before training the Generator. By doing so, we can train the Discriminator as strong as possible to

widen the Wasserstein distance, making it easy to distinguish the real graph from the generated ones. If the Generator can be trained to narrow the Wasserstein distance, the generated graphs eventually will be realistic enough to cheat the Discriminator.

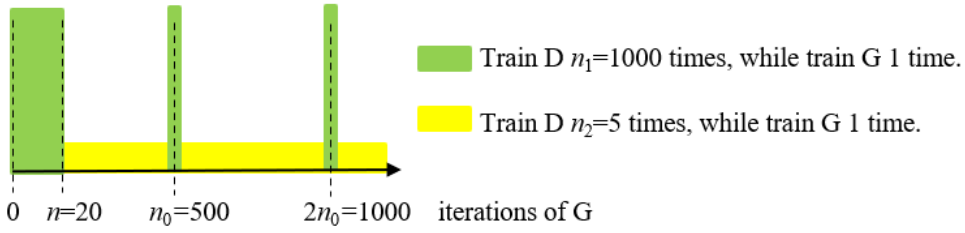


Figure 2.4: FeederGAN training schedule

Lines 5-12 illustrate the Discriminator training process. At each training session, a real graph combined with a generated graph are used to obtain the Wasserstein distance. Then, we fix the parameters of the Generator and trace backward to calculate the gradients using Eq. (2.6). After that, we use the RMSProp algorithm [48] to adaptively update the weights of the Discriminator. In the end, we conduct a weight clipping so that for weights smaller than $-c$, set them at $-c$; for weights larger than c, set them at c. Similarly, lines 13-16 describe how to calculate the Generator gradients and update its weights.

2.2 FeederGAN Implementation Details

This section introduces the architecture of the Generator and the Discriminator as well as hyperparameters used in the FeederGAN algorithm.

2.2.1 Generator Architecture

As shown in Figure 2.5, a multilayer perceptron (MLP) that has several FC layers is used to generate $\widehat{\mathbf{A}}$ and $\widehat{\mathbf{X}}$. The random noise matrix $\mathbf{Z}$ is generated using a prior Gaussian distribution and the latent matrix $\mathbf{L}$ is generated using FC1. FC2 is used to generate the numerical attributes $\widehat{\mathbf{X}}_{num}$ and a hyper tangent activation function is used to scale $\widehat{\mathbf{X}}_{num}$ within [-1, 1], the same value

range as the real attributes. The process is described as

$$\mathbf{L} = \mathrm{FC1}(\mathbf{Z}),\ \widetilde{\mathbf{X}}_{\boldsymbol{num}} = \mathrm{FC2}(\mathbf{L}),\ \widehat{\mathbf{X}}_{\boldsymbol{num}} = \tanh(\widetilde{\mathbf{X}}_{\boldsymbol{num}}). \tag{2.8}$$

To generate the categorical features, we first use FC3 to generate **V**, the latent matrix for all categorical features. Then, a separate FC4-*i* layer is used to generate the i^{th} categorical feature. After that, a row-wise softmax activation function is used to approximate the one-hot encoded categorical feature before outputting it to $\widehat{\mathbf{X}}^{\boldsymbol{i}}_{\boldsymbol{cat}}$. The use of *softmax* makes the network differentiable. Note that if a hard encoding method that is not non-differentiable (e.g., *argmax*) is used, the gradient backward pass will be broken. This process for generating the categorical features is described as

$$\mathbf{V} = \mathrm{FC3}(\mathbf{L}),\ \widetilde{\mathbf{X}}^{\boldsymbol{i}}_{\boldsymbol{cat}} = \mathrm{FC4} - i(\mathbf{V}),\ \widehat{\mathbf{X}}^{\boldsymbol{i}}_{\boldsymbol{cat}} = \mathrm{softmax}(\widetilde{\mathbf{X}}^{\boldsymbol{i}}_{\boldsymbol{cat}}) \tag{2.9}$$

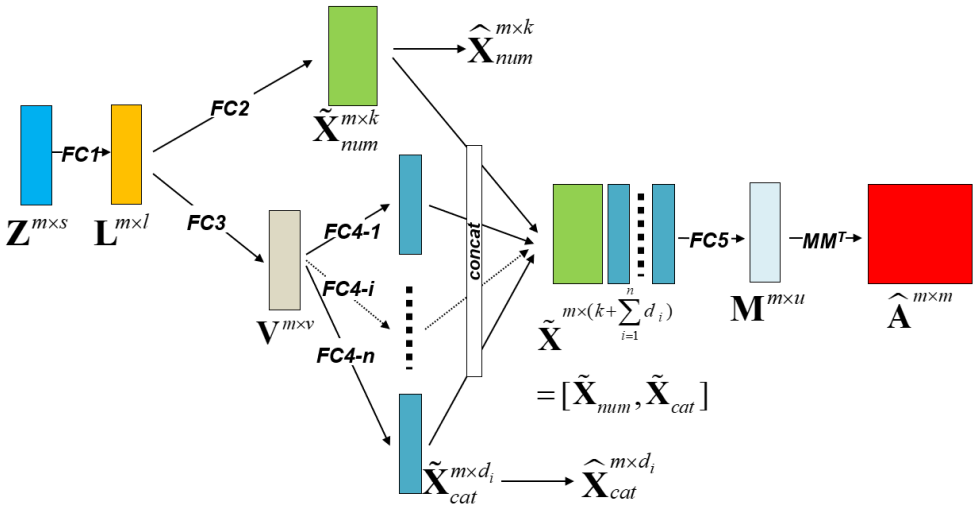


Figure 2.5: Generator architecture

The numerical and categorical features (before activation) are then concatenated to form the latent matrix $\widetilde{\mathbf{X}}$. Since the two features have different value scales, they will first be normalized then go through FC5 to generate **M**. **M** is then multiplied by its transpose to form an $m \times m$ matrix, which will then go through a column wise softmax activation function to obtain $\widehat{\mathbf{A}}$. Using column-wise softmax, we can map all $\widehat{\mathbf{A}}$ entries to [0, 1] while enforcing Property 1 (i.e., only 1

non-zero entry in each column), making those potentially non-zero entries stand out. This process is formulated as

$$\widetilde{\mathbf{X}} = \text{norm}\left(\left[\widetilde{\mathbf{X}}_{num}, \widetilde{\mathbf{X}}_{cat}\right]\right),\ \mathbf{M} = \text{FC5}\left(\widetilde{\mathbf{X}}\right),\ \widehat{\mathbf{A}} = \text{softmax}(\mathbf{M} \times \mathbf{M}^{T}) \qquad (2.10)$$

2.2.2 Discriminator Architecture

The objective of the Discriminator is to distinguish the real and generated graphs. The discriminative process for both the real and the generated graphs is the same, as shown in Figure 2.6. Hence, only the generated set of data $\widehat{\mathbf{A}}$, $\widehat{\mathbf{X}}_{num}$, $\widehat{\mathbf{X}}_{cat}$ are used to illustrate the discriminative process.

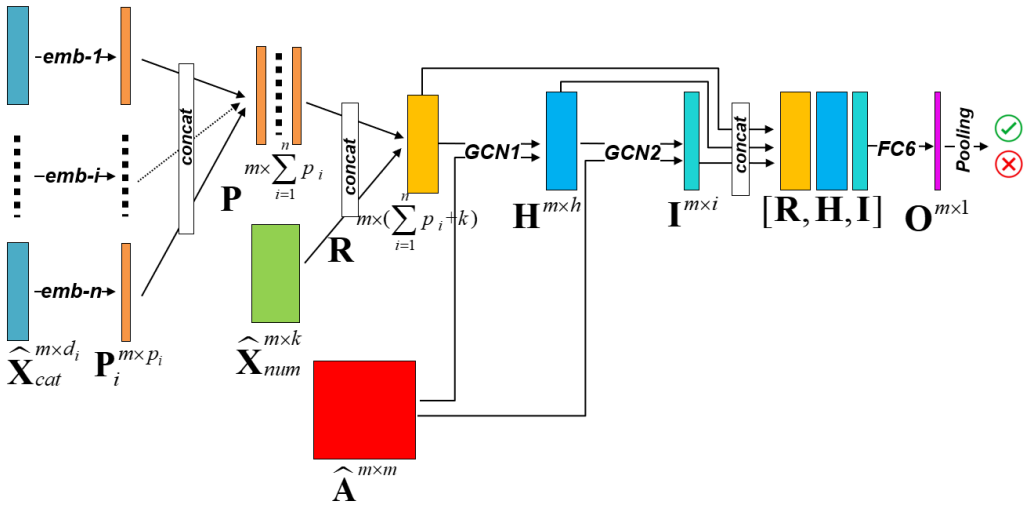


Figure 2.6: Discriminator architecture

Categorical features are either one-hot encoded or softmax approximated, so an embedding layer emb-*i* is used to project those discrete values into a lower dimension matrix, $\mathbf{P}_i$. The reasons are threefold. *Frist*, it projects sparse discrete features into a dense, continuous domain so categorical features can be used together with numerical features. *Second*, this serves as a dimension-reduction process. For instance, 4 numerical features only account for 4 dimensions, but the "phase" feature consumes 7 dimensions (3 for 1-phase, 3 for 2-phase, and 1 for 3-phase). If we don't conduct embedding, the feature space would be dominated by such categorical features.

Third, through embedding, we can calculate the similarity among features in different categories. One-hot values are orthogonal so when they are multiplied, the results are 0, making the calculation of similarities impossible.

Symmetrically as in the generative process, all the embedding results are concatenated as a latent representation, $\mathbf{P}$, for categorical features. Then, $\mathbf{P}$ is concatenated with the numerical features and go through a layer normalization to get the latent matrix $\mathbf{R}$ for all the attributes. The attributes processing is formulated as

$$\mathbf{P_i} = \mathrm{emb} - i(\widehat{\mathbf{X}}_{\boldsymbol{cat}}^{\boldsymbol{i}}),\ \mathbf{R} = \mathrm{norm}([\mathbf{P}, \widehat{\mathbf{X}}_{\boldsymbol{num}}]) \tag{2.11}$$

Next, we use the adjacency matrix to filter the attributes via two GCN layers. Since GCN is prone to gradient vanishing problem, a residual structure is used to aggregate the latent matrices $\mathbf{R}, \mathbf{H}, \mathbf{I}$ together. Hence, the gradient backward pass becomes a three-way pass instead of a single one to boost the gradients back and benefit the training of the Generator.

In the second to the last layer, FC6 processes the concatenated residual matrices to obtain a vector $\mathbf{O}$ with m entries that reveals the latent information for each node. Finally, a global pooling method, e.g. max-pooling, is used to output a single value serving as the critic score. The discriminative process is formulated as

$$\mathbf{H} = \mathrm{GCN1}\big(\widehat{\mathbf{A}}, \mathbf{R}\big),\ \mathbf{I} = \mathrm{GCN2}(\widehat{\mathbf{A}}, \mathbf{H}) \tag{2.12}$$

$$\mathbf{O} = \mathrm{FC6}([\mathbf{R}, \mathbf{H}, \mathbf{I}]),\ \mathrm{score} = \mathrm{pooling}(\mathbf{O}) \tag{2.13}$$

Note that besides softmax and hyper tangent, we use ReLU after certain FC layers and leaky ReLU after GCN to introduce non-linearity.

2.2.3 Hyperparameters

Learning rate α and clipping rate c are the two most important hyperparameters in the FeederGAN model. To guarantee the convergence of Wasserstein distance, experiments need to

be conducted to tune the learning rate. In our experiments, we choose a small learning rate α=0.0001. Clipping limit is also determined through experiments. A few experiments are conducted when c=0.01, 0.1, 0.5 and 1. The gradients of the top and bottom layers of the generator, i.e. FC5 and FC1, are plotted in Fig 2.7, which shows that a small clipping limit may result in gradient vanishing that lead to failures in training; a large clipping limit will result in gradient exploding that causes the training to go unstable or the memory to overflow. Thus, c is selected to be 0.1.

The dimension of the random noise matrix $\mathbf{Z}$ is $m \times s$, where m is the number of nodes that is adaptative to the real graphs; s is the parameters to be tuned, and we set as 20 in our experiments. The rest of the parameters are listed in the first row of Algorithm 1. The hidden layers' size is subject to the size of real graphs and we use typical values of 128 and 64.

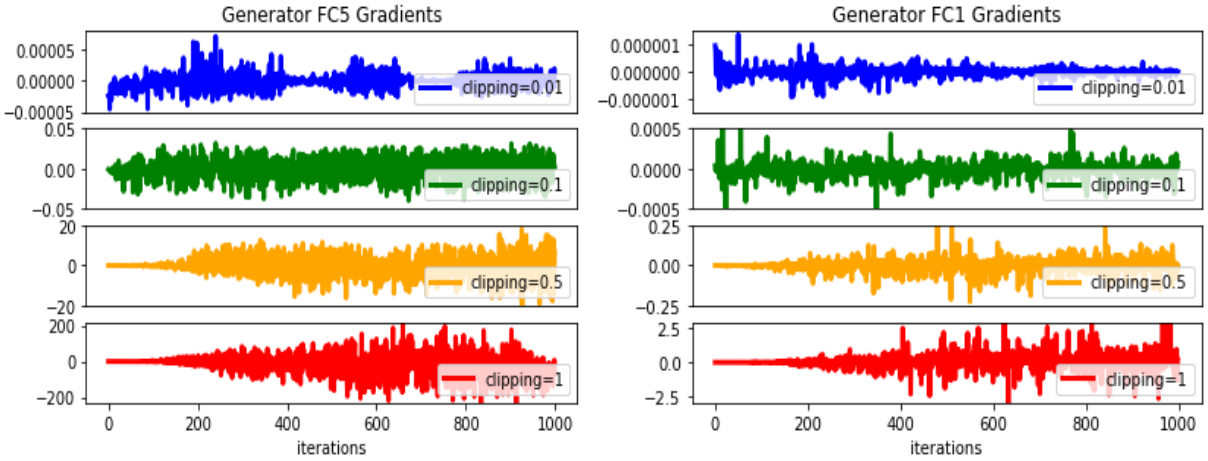


Figure 2.7: Generator gradients when using different clipping limits

2.3 Feeder Reconstruction

Once the training of Generator finished, we can use it to generate $\widehat{\mathbf{A}}$ and the attributes matrix $\widehat{\mathbf{X}}_{num}$, $\widehat{\mathbf{X}}_{cat}$, which are either softmax-approximated to be within [0, 1] or hyper tangent approximated to be within [-1, 1]. Then, we use the reconstruction methods to map them back to

the actual values.

2.3.1 Adjacency Matrix Reconstruction

The entries in the generated adjacency matrix $\widehat{\mathbf{A}}$ have continuous values in [0, 1]. Algorithm 2.2 (Table 2.3) is used to reconstruct the binary directed adjacency matrix for the feeder from $\widehat{\mathbf{A}}$.

Table 2.3: Adjacency Matrix Reconstruction Algorithm

Algorithm 3.2. Adjacency Matrix Reconstruction.

Require: $\widehat{\mathbf{A}}$, the generated soft adjacency matrix; *row*, the number of rows in the adjacency matrix; *col*, the number of columns in the adjacency matrix.

1: **for** i=1, 2, …, *row* **do**
2: **for** j=i, i+1, ..., *col* **do**
3: $\widehat{\mathbf{A}}_{j,i} \leftarrow 0$, **if** $\widehat{\mathbf{A}}_{i,j} \geq \widehat{\mathbf{A}}_{j,i}$
4: $\widehat{\mathbf{A}}_{i,j} \leftarrow 0$, **if** $\widehat{\mathbf{A}}_{i,j} < \widehat{\mathbf{A}}_{j,i}$
5: **end for**
6: **end for**
7: $\widetilde{\mathbf{A}} \leftarrow \widehat{\mathbf{D}}^{-1}\widehat{\mathbf{A}}$
8: **for** j=1, 2, ..., *col* **do**
9: $k \leftarrow \text{argmax}(\widetilde{\mathbf{A}}_{.,j})$, $\widetilde{\mathbf{A}}_{k,j} \leftarrow 1$
10: **for** i=1, ..., *row* **and** $i \neq k$ **do**
11: $\widetilde{\mathbf{A}}_{i,j} \leftarrow 0$
12: **end for**
13: **end for**
14: **return** $\widetilde{\mathbf{A}}$

In lines 1-6, to make a directed graph out of $\widehat{\mathbf{A}}$, for any symmetrical pair $\widehat{\mathbf{A}}_{\boldsymbol{ij}}$ and $\widehat{\mathbf{A}}_{\boldsymbol{ji}}$, there should be one non-zero entry at most. Hence, we set the smaller one as 0 and keep the larger one as it is. In line 7, we conduct a row-wise normalization, which is also used in the feature

engineering process as an import procedure to guarantee sparsity. In lines 8-13, we conduct Property 1, i.e. in each column we set the largest entry as 1 and the rest as 0. After that, $\widetilde{\mathbf{A}}$ becomes a binary, sparse adjacency matrix describing a directed graph.

2.3.2 Topology Reconstruction

In graph theory, node ordering can be problematic, because for a specific adjacency matrix, the different ordering of nodes can represent different topologies. For the Discriminator, because GCN is ordering invariant, node ordering is not an issue. However, for Generator, MLP is ordering variant, so node ordering needs to be addressed. Note that using permutation to resolve the node ordering is impossible as a graph with m nodes has m-factorial ($m!$) possibilities.

To reduce the complexity, one can permutate only the position of the feeder head based on the reconstructed $\widetilde{\mathbf{A}}$, which has only 1 non-zero entry in each column. The algorithm is illustrated in Algorithm 2.3 (Table 2.4). The node-to-edge transformation (in line 5) is an inverse procedure of Figure 2.2. Therefore, for each generated adjacency matrix, theoretically, m topologies can be obtained at the most.

Table 2.4: Feeder Head Permutation Algorithm

Algorithm 2.3. Feeder Head Permutation.
Require: $\widetilde{\mathbf{A}}$, the reconstructed binary directed matrix; m, the node number.
1: **for** $i = 1, 2\ldots, m$ **do**
2: **if** row i with non-zero entries **do**
3: Set the corresponding column i as all 0s;
4: Let node i be the feeder head and traverse the adjacency matrix to build the graph;
5: Conduct a node-to-edge transformation to build the feeder topology.
6: **end if**
7: **end for**

The following example illustrates the reconstruction process via feeder head permutation. As shown in Figure 2.8, the reconstructed $\widetilde{\mathbf{A}}$ has only 1 non-zero entry in each column with only row 1, 2, and 3 has non-zero entries. Hence, nodes 1, 2 and 3 are the candidates for the feeder head node. If node 1 is chosen, set all entries in column 1 as 0. Traverse all the nodes to build the directed graph, i.e. $1 \rightarrow 2, 2 \rightarrow 3/4/5$. Then, convert a node-to-edge transformation so a feeder topology is obtained following the power system conventions. It's same for node 2 and 3 to be the feeder head node.

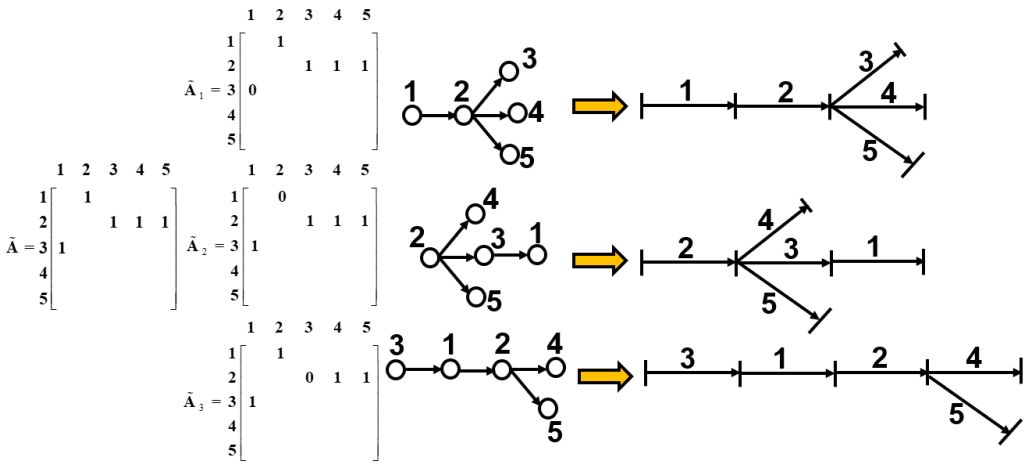


Figure 2.8: Topology Reconstruction

2.3.3 Attributes Reconstruction

After reconstructing the feeder topology, we need to assign attributes to each device so that a distribution feeder model in the targeted format (e.g., OpenDSS, CYME) can be produced. Among all attributes, topological features are just used to facilitate the discriminative inference, which are not necessary to be reconstructed. Hence, we only need to reconstruct the organic features.

For "length", we map the generated value from [-1, 1] to [0, 800] uniformly, where 800 m is the nominal maximum value of a feeder line segment. Note that one can pick a suitable

maximum value based on his modeling need. Although the "norm amps" is used as a continuous value, it's used to represent the conductor material information, e.g. line code, thus the category of the values is limited to about 10. Hence, based on the generated value, we use a lookup table for line codes to assign the nearest "norm amps" value. Among all attributes, "phase" is the most important one because it affects how realistic a generated feeder is. As *softmax* is used to approximate the one-hot encoding, an *argmax* function is used to map the phase type to the 7 choices.

2.3.4 Feeder Visualization

As the geographical coordinates have been omitted in the learning process, we need to generate pseudo coordinates for each bus to plot out the feeder topology for inspection. A greedy method, as described in Algorithm 2.4 (Table 2.5), is developed to generate the pseudo coordinates with two objectives: making the feeder lines as straight as possible while reducing the chance of overlapping and crossing.

Table 2.5: Feeder Topology Visualization Algorithm

Algorithm 2.4. Feeder Topology Visualization
1: Set the feeder head as the origin (0,0) and traverse the rest of the feeder;
2: Set the direction as straight right and make the direction angle as $\theta = 0$;
3: If no bifurcation, keep the current direction, calculate the coordinate of the next bus using the line length;
4: When there is a bifurcation, rank the priority of children branches based on the number of its downward devices. The large the number, the higher the ranking;
5: The 1st child branch keeps the same direction as parent's, the 2nd and 3rd children's angle plus a delta angle δ ($\pi/2$) and (-$\pi/2$) respectively. If there are more children in the branch, use more angles, e.g. ($\pi/4$), ($3\pi/4$), ($-\pi/4$), ($-3\pi/4$), ($\pi/8$), ...;
7: Update the children branch's direction angle, calculate the corresponding bus coordinates with the angle, the length of the line, and the parents' coordinates.

In Figure 2.9, we illustrate how to calculate the pseudo coordinates. Suppose we have already traversed to bus j with a coordinate at (x_j, y_j) and a branch direction of θ. The bifurcation leads to 4 children branches. The device linking bus j and k with fewest downward devices will have the lowest priority ranking with an angle assigned as δ. Hence, for this branch, the direction angle is updated as $\theta + \delta$. The coordinate of bus k is (x_k, y_k), calculated by $x_k = x_j + l \cdot \cos(\theta + \delta)$, $y_k = y_j + l \cdot \sin(\theta + \delta)$, where l is the device length between bus j and k.

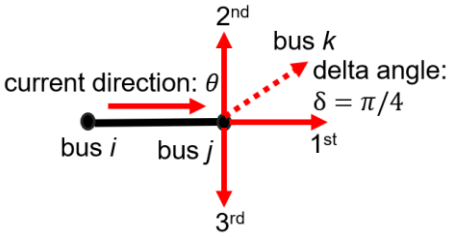


Figure 2.9: Pseudo Coordinates calculation

2.4 FeederGAN Results and Evaluation

In this section, we will discuss the training dataset used to train the FeederGAN model, the training results, and the mode collapse problems we found in the GAN based deep learning framework. We then introduce the early-stop strategy and human guidance rules to release phase mode collapse and topology mode collapse. Finally, we use empirical statistics gained from real feeders to exam the realisticness of the generated feeders.

2.4.1 Training Dataset and Augmentation

We use real feeder models with 1500 to 2000 overhead lines and cables as inputs in this study. The topology of a typical distribution feeder is shown in Figure 2.10.

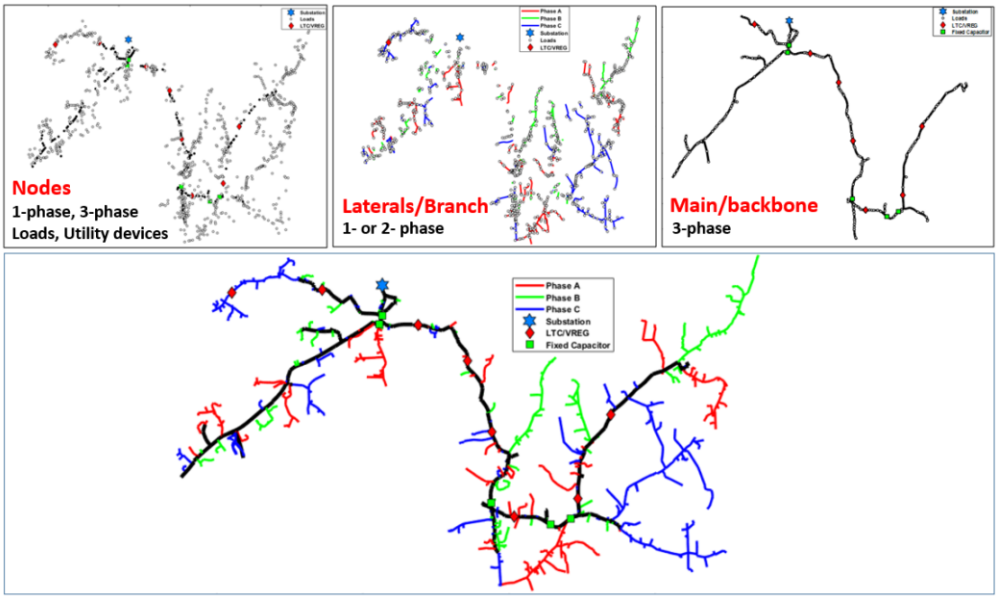


Figure 2.10: Topology of a distribution feeder

First, all tie switches are set as "open" to satisfy Assumption 1 (i.e., no loops). Second, we sample sub-feeder structures from full feeder networks to augment the training database using Algorithm 2.5 (Table 2.6). Then, the full feeder network and the sampled sub-feeder structures are used as the training dataset, in which we have 664 graphs in total.

Table 2.6: Feeder Graph Subsampling Algorithm

Algorithm 2.5. Subgraph Sampling.
Require: A pool of real feeders served as real graphs
1: Select a start node from level 0 or level 1 nodes;
2: Extract all its downstream nodes (all the way to loads) to form a subgraph;
3: If number of nodes is less than 100, resample;
4: If the number of nodes is more than 50% of the original graph, resample;
5: Repeat for desired times.

2.4.2 Training Loss and Generated Feeders

In Figure 2.11, the training losses of the Discriminator when feeding with real and generated (fake) graphs, and the Wasserstein distance are shown on the left; the zoom-in plots for

each training stage are shown on the right. The *x*-axis is the number of iterations for Generator. Note that one Generator iteration corresponds to multiple Discriminator iterations.

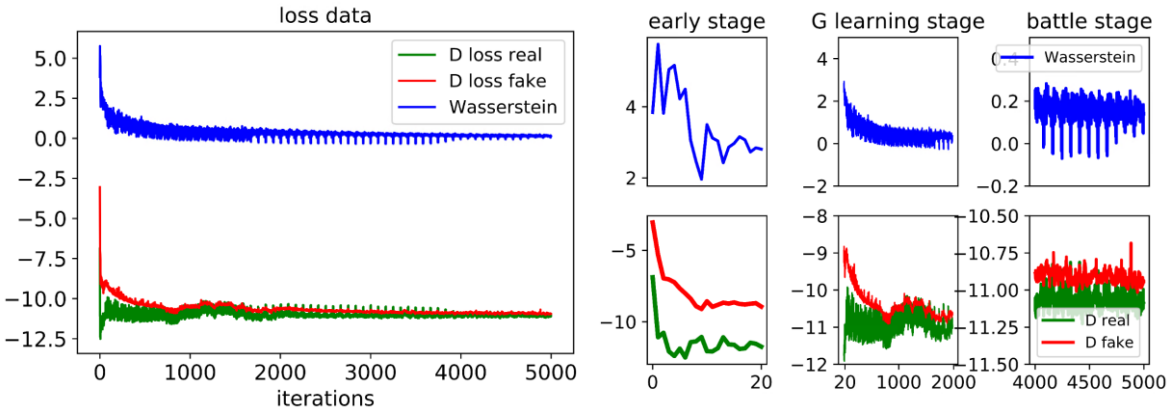


Figure 2.11: Loss and Wasserstein distance

In the first few iterations, we train Discriminator to be very strong so it can easily widen the Wasserstein distance. When the training proceeds, although the Discriminator is trained more times than the Generator, the Generator learning process will gradually narrow the Wasserstein distance until the distance converges to a very small but non-zero oscillatory equilibrium, as shown in the zoom-in plots. Meanwhile, the critic score for the real and the fake vibrates around an equilibrium point. It shows that the Generator generates graphs of good quality that the Discriminator can no longer tag them.

Figure 2.12 shows examples of the feeder topologies generated by the FeederGAN, from which, an expert can examine the feeder layout and exclude the ones that do not meet their requirements. Note that we only generate line segments, i.e., overhead lines or cables in current stage. Reclosers or regulators can be added later based on different modeling needs by the user.

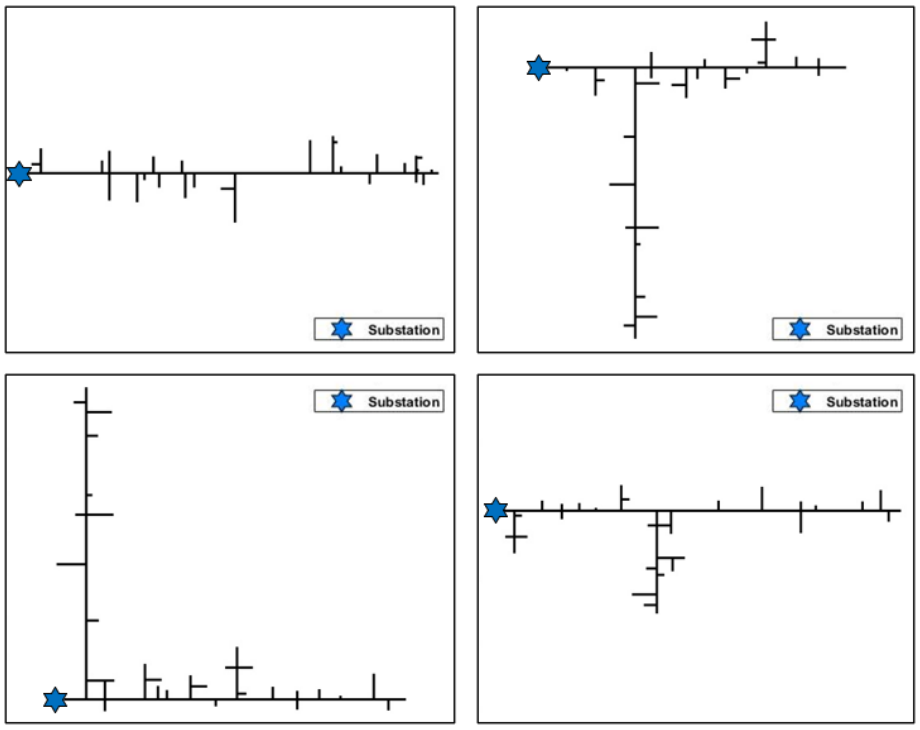


Figure 2.12: Generated feeders

2.4.3 Performance, Mode Collapse and Human Guidance

Table 2.7: Definition of the Success and Perfect metrics

Success definition		Perfect definition	
phase	subsequent neighbor's phase	phase	subsequent neighbor's phase
3 phases	3 phases, 2 phases, single phase	*abc*	*abc, ab, ac, bc, a, b, c*
		ab	*ab, a, b*
		ac	*ac, a, c*
2 phases	2 phases, single phase	*bc*	*bc, b, c*
		a	*a*
single phase	single phase	*b*	*b*
		c	*c*

We define 3 performance metrics in Table 2.7 to evaluate the generating process. The first metric is to pass a connectivity check. An adjacency matrix is *connected* when the matrix contains

fully connected graph without isolated partitions. So an adjacency matrix containing isolated blocks will fail the connectivity check because it causes the generated graph to break into a few disconnected pieces. The second and third metrics are *Success* and *Perfect*, which are related with the continuity of feeder phase transitions. For example, only *a*, *b* or *ab* line segments can be branched out from a line segment having the *ab* phase attribute.

Recall that there are *m* possible topologies for each reconstructed adjacency matrix $\widetilde{\mathbf{A}}$ when applying the feeder-head permutation method. So the 3 types of *performance rates* are defined as the ratio of the fully *connected*, *success* and *perfect* generated graphs among all *m* options. In training, after every 50 Generator iterations, we output a set of generated graphs and calculate the performance rates. The rates can vary often because they are calculated using the results of a snapshot. Therefore, we average those rates in a 500-iteration window, making each data point represent an average value of 10 results during 500 iterations. As shown in the left plot in Figure 2.13, the model performance improves to 25% after 4000 iterations, meaning that 25% of the *m* generated feeder topologies are connected, success, and perfect.

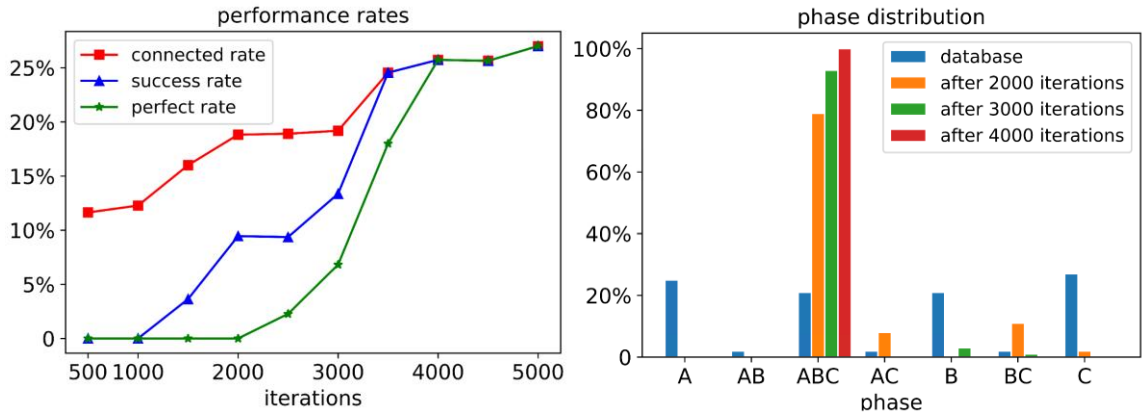


Figure 2.13: Performance rates (left) and phase mode collapse (right)

During the training, we observed two major problems related with mode collapse [49], where the Generator locked into one or a few limited modes to exploit the Discriminator. The first

is phase mode collapse. As shown in the right plot in Figure 2.13, at the beginning of the training, 3-phase and single phase circuits are mixed evenly. However, after 2000 iterations, 3-phase circuits start to dominate with the presence of a small number of 2-phase (i.e., *ac* and *bc*) circuits; while towards the end, all circuits converge to 3-phase. To solve this problem, the early-stop-strategy is used to terminate the training after approximately 3000 to 4000 iterations, when the phases generated are not all 3-phase circuits. Then, we use the human guidance to assign the phase information to line segments as illustrated in Algorithm 2.6 (Table 2.8). The advantage of applying human guidance are twofold: achieve a user-defined phase diversity and convert previously imperfect generated circuit topology to a perfect one.

Table 2.8: Human Guidance for Phase Assignment Alogrithm

Algorithm 2.6. Human Guidance for Phase Assignment.
1: Traverse the feeder from the feeder head with 3-phase attribute, i.e., *abc*;
2: Keep the phase attribute until reaching a bifurcating point;
3: Check for possible phase options via perfect definition in Table II;
4: Select from the possible choices the one with the highest score in the generated *softmax* approximated phase results, i.e. $\widehat{\mathbf{X}}_{\boldsymbol{cat}}^{\mathbf{2}}$.

The second is topology mode collapse. We have plotted a few graphs to show the topologies of a few generated circuits at different locations in the Wasserstein loss curve, as shown in Figure2.14. At the very beginning, the feeder generated is dense and has shorter backbones. After 2000 iterations, the feeder shows a longer backbone with branches. But after 4000 iterations, the backbone remains but the branches disappear, showing a single line without bifurcation. This problem can be addressed by an early stop. In our experiments, we stop the training after 3000-4000 iterations to get satisfactory results.

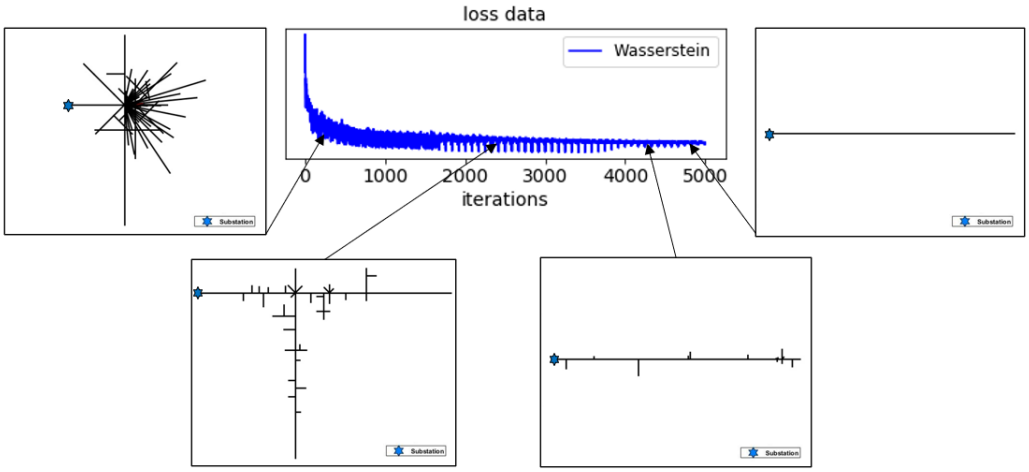


Figure 2.14: Topology mode collapse

2.4.4 Quality Evaluation and Empirical Statistics

When using FeederGAN to generate synthetic feeders, we can generate *m* possible candidates each time for a feeder with *m* devices via feeder head permutation (Algorithm 2.3). Then, we use the empirical statistics method for measuring realisticness, in which the statistical characteristics of real distribution feeder models (See Table 2.9) are compared with those of the generated ones. If the statistics of a generated feeder are outside the empirical range, it will be discarded. The first measurement is "level", which measures the feeder topology hierarchy from the feeder head to the lowest downward load node. If a generated feeder has more than 10 levels, it is discarded. The second measure is the ratio of 3-phase, 2-phase, and 1-phase circuits because a realistic feeder usually has more 1-phase circuits and 3-phase with only a few being 2-phase. The third measure is the out-degree of a node defining the number of its subsequent neighbors. For a realistic feeder topology, 99% times the out-degree is less than 5. What's more, if the probability distribution of the length of generated line segments differs a lot from the real one (as shown in Figure 2.15), we can either discard it or use the probability distribution of real feeders to conduct a 2nd round "length" re-generation, which will make sure that the length of each generated line

segment is realistic.

Table 2.9: Empirical statistics

metrics	empirical value						
levels	4 ~ 7						
phase distribution	*a*	*ab*	*abc*	*ac*	*b*	*bc*	*c*
	18% ~ 28%	1% ~ 3%	20% ~ 25%	1% ~ 3%	18% ~ 28%	1% ~ 3%	18% ~ 28%
out-degree distribution	0	1	2	3	4	≥5	
	20% ~ 40%	25% ~ 45%	18% ~ 26%	5% ~ 7%	1% ~ 3%	<1%	

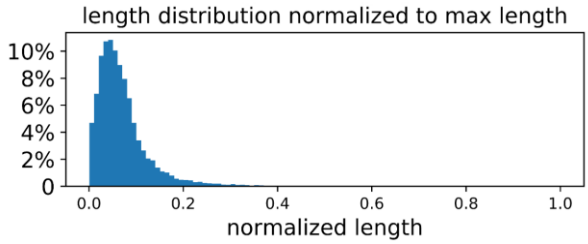


Figure 2.15: Probability distribution of the length of the line segments

2.5 Load Transformer Generation

Feeder topologies and line segments attributes, such as line length, conductor type and phase connections, are generated via the FeederGAN tool in previous sections. To make the generated feeder files a runnable testing system, what left to do is to add on load data. In feeder models, loads are connected to load transformers or customer side transformers. In this section, we use a statistical, rule-based method to generate the load transformer. The time series load data generation or synthesis will be introduced in the next chapter via load disaggregation algorithms.

2.5.1 Load Capacity and Its Distribution

Different distribution feeders may have different load components in terms of transformer capacities. We collect about 10, 000 load transformers from 14 real feeders. Figure 2.16 shows the load transformer components in percentage in each feeder. For feeders mainly with residential loads, typical loads are 25 kVA and 50 kVA. For old feeders in past decades, 10 kVA and 15 kVA are very common, which may not common in the modern newly built feeders.

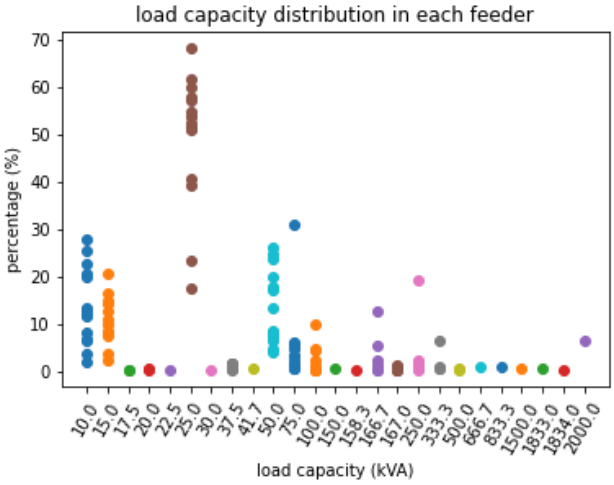
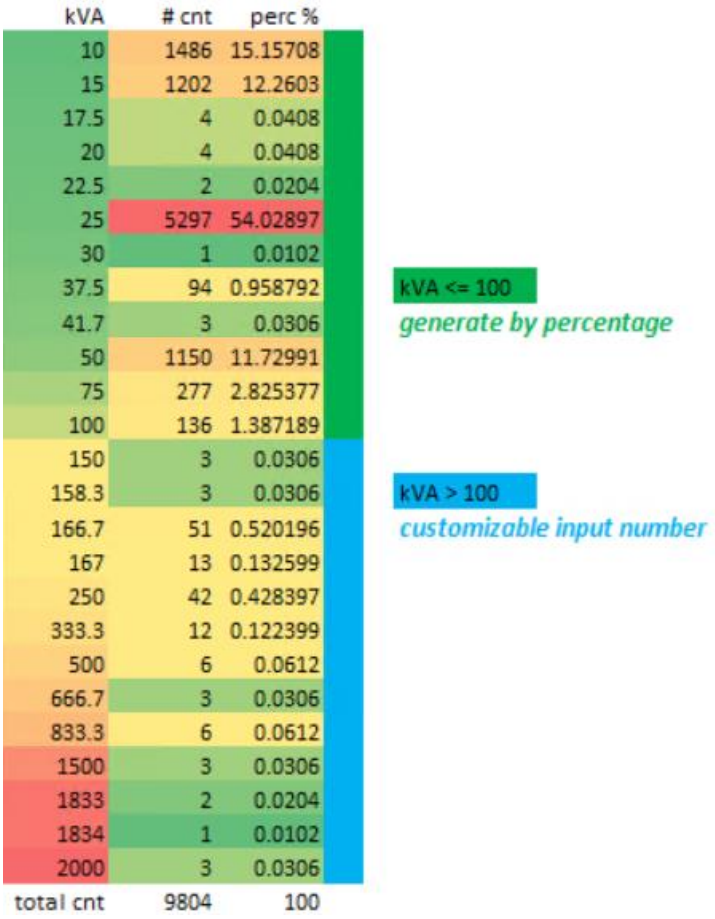


Figure 2.16: Load transformer capacity in each feeder

Among all the load transformers, Figure 2.17 shows the summary statistics. In general, for feeders mainly with residential loads in suburban or rural areas, 99% of the load transformers are not larger than 100 kVA. Among them, half of the loads are about 25 kVA. Hence, to generate the load transformer capacities, we can refer to those statistics. For example, as shown in Figure 2.17, we can characterize the percentage of the kVA components for loads no larger than 100 kVA. For loads larger than 100 kVA, we can specify the number of each type, say we put 10 transformers with 500 kVA in the feeder system.

Hence, in the following generation process, we should let the load capacity distribution as a customized input. Users, who want to generate the feeder with customized load data, should

follow the example of Figure 2.17 to customize his/her specific feeder model. If no customized input given, we can choose a default value from real feeders we collected as shown in Figure 2.16.



kVA	# cnt	perc %
10	1486	15.15708
15	1202	12.2603
17.5	4	0.0408
20	4	0.0408
22.5	2	0.0204
25	5297	54.02897
30	1	0.0102
37.5	94	0.958792
41.7	3	0.0306
50	1150	11.72991
75	277	2.825377
100	136	1.387189
150	3	0.0306
158.3	3	0.0306
166.7	51	0.520196
167	13	0.132599
250	42	0.428397
333.3	12	0.122399
500	6	0.0612
666.7	3	0.0306
833.3	6	0.0612
1500	3	0.0306
1833	2	0.0204
1834	1	0.0102
2000	3	0.0306
total cnt	9804	100

Figure 2.17: Load capacity distribution summary

2.5.2 Capacity Statistics and Constraints

A. Load Node Type

We furtherly divide the nodes (or buses) in a feeder as in-between node (bus) or ending node (bus). In-between buses are those having subsequent connections, while ending buses are those at the end of a lateral and no subsequent components. The statistics in Figure 2.18 shows that 83% buses are in-between buses and 17% buses are ending buses, and about 65% loads are connected to in-between buses, while about 35% loads are connected to ending buses. However,

as an ending bus, one has 98% chance to connect to a load, and only 2% probability to be a hanging bus. But only about 38% in-between buses have load on them, while 62% of them would only be a transition or connection component.

				perc %		
total bus #	20273	total load #	9804	100		
in-between bus #	16822	on in-between bus #	6418	65.46308	in-betwwen load node perc %	38.15242
ending bus #	3451	on ending bus #	3386	34.53692	ending load node perc %	98.11649

Figure 2.18: Load node type and probability

To make it simple, in the generation process, we set a quota for in-between buses and ending buses as Eq. 2.14.

$$q(C_i) = \begin{cases} \alpha, \ C_i \in in\_between \\ \beta, \ C_i \in ending \end{cases} \tag{2.14}$$

where we set the in-between loads quota to be $\alpha = 65\%$ and ending bus loads quota to be $\beta = 35\%$, i.e. if there are 100 load transformers, we set 65% of them on in-between buses and 35% of them on ending buses.

B. Load Degree Probability List

We have divided the load node into in-between node and ending node. Furtherly, we character them via their out-degree, i.e. the number of edges going out from the node. Figure 2.19 shows the load degree probability list for different feeders. For example, ending nodes are those with 0 out-degrees. In average, we know that 98% of the ending buses are collected to load transformer. Since different feeders may have different chances to connected to a load, we use median instead of average to reduce bias. Equation 2.15 summarizes the chance a node with different out-degree to collect to a load.

$$p(d_i) = \begin{cases} a, & d_i = 0 \\ b, \ d_i \in (1,2,3) \\ c, & d_i > 3 \end{cases} \tag{2.15}$$

where d_i is the out-degree for the i-th node, $p(d_i)$ is defined as its node degree probability list. We divide them into 3 groups, for out-degree equal to 0, we set the chance to be $a = 80\%$ to connect to a load; while for out-degree equal 1, 2 and 3, we set the set to be $b = 33\%$ to connect to a load; for nodes with out-degree larger than 3, we set its chance to connect to a load as $c = 0$.

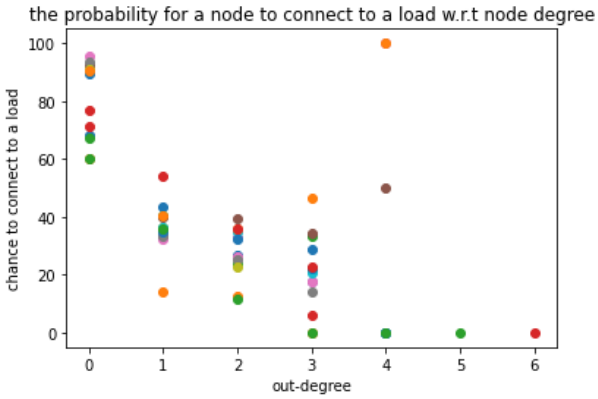
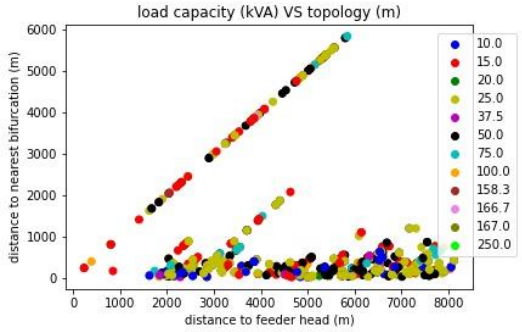


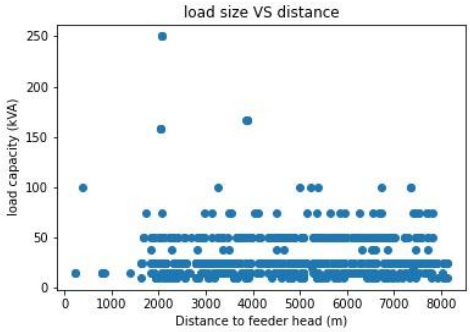
Figure 2.19: Out-degree probability list

C. Topology Ratio Constraint

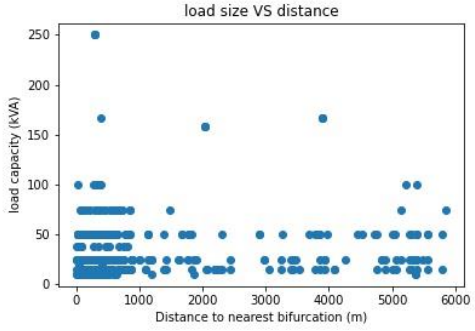
From our study, we found that load capacity and its location in the feeder system have some relationships. Figure 2.20 shows the load capacity relationship with topology information. Subplot 2.20 (a) shows the load capacity from feeder head and its nearest bifurcation point. Different colors represent different load capacities, the x-axis is the distance to the feeder head, while the y-axis is the distance to the nearest bifurcation point. Points on a straight-line show that these transformers are on the same lateral (or on the main backbone). Subplot 2.20 (b) show the capacity relationship with distance to the feeder head, while Subplot 2.20 (c) shows the capacity relationship with distance to the nearest bifurcation point.



(a) Load capacity VS topology



(b) Load capacity VS distance to feeder header



(c) Load capacity VS distance to nearest bifurcation point

Figure 2.20: Load capacity and topology relationship

The conclusion we can get from the above analysis is summarized as following.

(1) There are no apparent patterns for residential loads no larger than 100 kVA and their load location distribution is rather random;

(2) For large loads, larger than 100 kVA, they can be anywhere from the feeder head, but

it won't be very far away from its nearest bifurcation points, i.e. it can either be on a backbone or if it's on a lateral, it shouldn't be far away from the bifurcation.

Based on the above discovery, we introduce the concept of topology ratio for each load node, which is defined as the ratio between distance to its nearest bifurcation and its distance to the feeder head. Figure 2.21 shows the load capacity with its corresponding topology ratio. The topology ratio plot can be divided into 2 main regions, the green and red regions. The green one shows the load on the main backbone, where the topology ratio is always 1. The red region shows loads on laterals, where the topology ratios are smaller than 1.

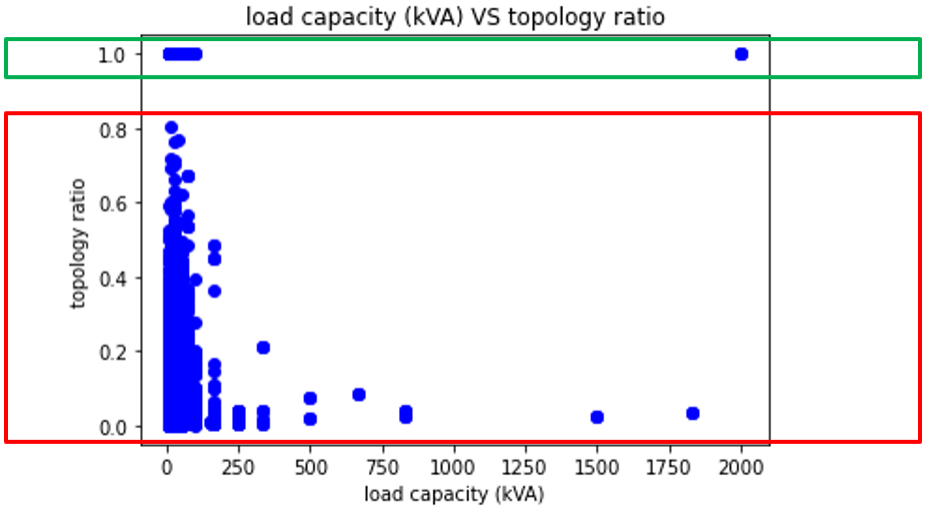


Figure 2.21: Load capacity VS topology ratio

Based on the conclusion we get from the above, we further divide the topology ratio into 4 regions as Figure 2.22. They are summarized as follows.

(1) For loads on the main backbone or loads no larger than 100 kVA, we don't set the constraint on where it should be located at;

(2) for loads in between 100 and 200 kVA, we set the topology ratio as 0.5, i.e. the distance to the nearest bifurcation point should be no larger than half of its distance to the feeder head;

(3) for loads in between 200 and 300 kVA, we set the topology ratio as 0.2, i.e. the distance

to the nearest bifurcation point should be no larger than 20% of its distance to the feeder head;

(4) for loads larger than 500 kVA, we set the topology ratio as 0.1, i.e. the distance to the nearest bifurcation point should be no larger than 10% of its distance to the feeder head;

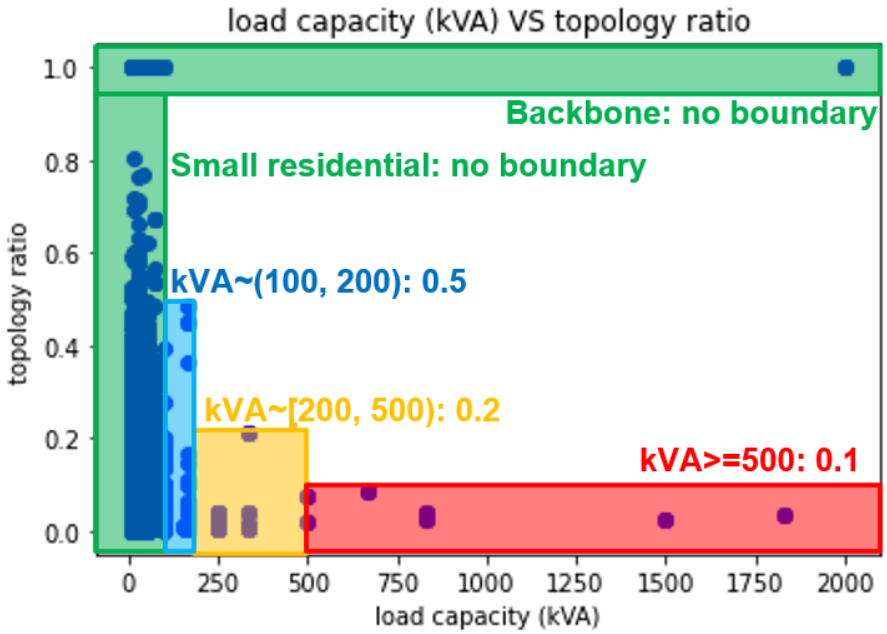


Figure 2.22: Topology ratio constraints

D. Other Constraints

When our FeederGAN tool generate the feeder topology, the conductor material also being indicated with the normal conditional current ampere. Hence, we can use the normal conditional ampere and base voltage (120V) to set the upper boundary of the kVA capacity can be connected to a line segment. Equation 2.16 shows the conductor capacity constraint.

$$C_i \leq I\_norm \times V\_base \tag{2.16}$$

where C_i is the load capacity for the *i*-th load, I_norm is the normal condition ampere current the line segment connected to the *i*-th load bus, V_base is the base voltage. For instance, a conductor with 485 A normal ampere can support up to 58 kVA capacity in normal operation condition.

In general, larger loads trend to on an underground cable as shown in Figure 2.23, but there

are outliers. Since the line segments of the feeder in the feederGAN tool are generated, we cannot make sure at certain place it should really be a cable or overhead line. Hence, we don't include this constraint in the implementation. Instead, we force large load to be on backbones, and ignore whether it's cable or overhead line.

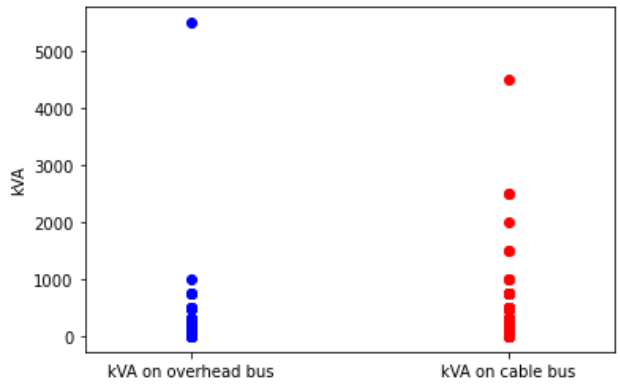


Figure 2.23: Loads on overhead lines and underground cables

What's more, large loads also trend to be on 3-phase buses as shown in Figure 2.24. However, since the range of kVA is a little bit large, we don't directly use phase constraint. Instead, topology constraint is used indirectly, i.e. making large load near backbone to satisfy this constraint.

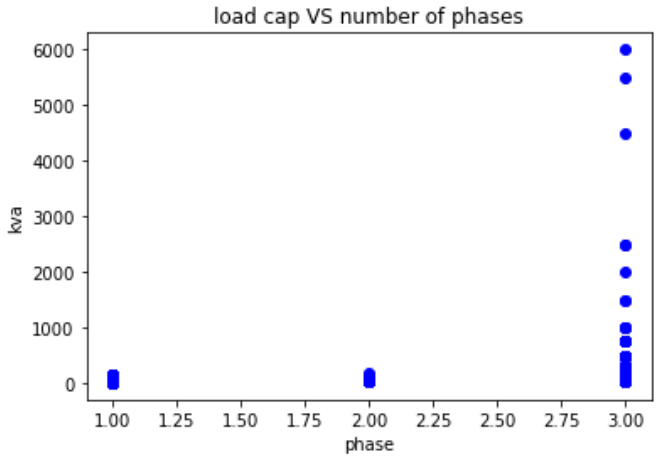


Figure 2.24: Load capacity VS phase

2.5.3 Load transformer generation

In this part, we will summarize the algorithm used to generate load capacities with its location along the feeders. As mentioned above, we need a customized load capacity distribution as input, where loads no larger than 100 kVA are given as the percentage distribution and loads larger than 100 kVA are given as their specific numbers. If no inputs are given, we would use a default value as we collect. Table 2.10 shows the algorithm to generate the large load capacity and its corresponding locations. We traverse the network once for each large load, and making the largest load find its satisfiable location first. During the process, we first meet the conductor capacity constraint and topology ratio constraint. And then among all the feasible locations, we use the node degree probability list to choose the ideal location.

Table 2.10: Large Load Capacity Generation

Algorithm 2.7. Large Load Capacity Generation.

Require: *n*, the number of load larger than 100 kVA; the load capacity distribution customized input as shown in Figure 2.17.

1: **for** *i*=1, 2, …, *n* **do** *# For each load, traverse the network once*
2: Pop out the largest load capacity in the load distribution input list; *# largest first*
3: Find feasible location s.t. conductor capacity constraint as shown in Eq. 2.16 and topology ratio constraint as shown in Figure 2.22;
4: If more location than desired is found, among the feasible locations, use the node degree probability list as shown in Eq. 2.15 to choose a node that is near the backbone;
5: **end for**

For loads smaller than 100 kVA, we use Algorithm 2.8 in Table 2.11 to illustrate the generation process. Since there are many small loads and their constraints are looser, we just need to traverse once for all the loads. When we traverse to a node, we randomly choose it as a potential load node via node degree probability list in Eq. 2.15. If it's choosing as a load node, we further

check whether there is quota left for its node type. If there is quota left, we check it via conductor capacity constraint to finally determine whether there should add on this load transformer.

Table 2.11: Small Load Capacity Generation

Algorithm 2.8. Small Load Capacity Generation.
Require: the load capacity distribution customized input as shown in Figure 2.17.
1: Traverse the network from feeder head *# Traverse once for all the loads*
2: Use the node degree probability list as shown in Eq. 2.15 to determine whether a bus can be a load node;
3: If so, based on it's an in-between bus or ending bus, then check whether its corresponding quota as shown in Eq. 2.14 has been completed
4: If not, generate a load capacity from the capacity distribution s.t. conductor capacity as shown in Eq. 2.16. If so, skip this node and traverse the following node.
5: If both in-between and ending bus quota finished, stop and summarize.

Therefore, we can combine Algorithm 2.7 and 2.8 together to generate all the load transformers. The overall flowchart is shown below as Figure 2.25.

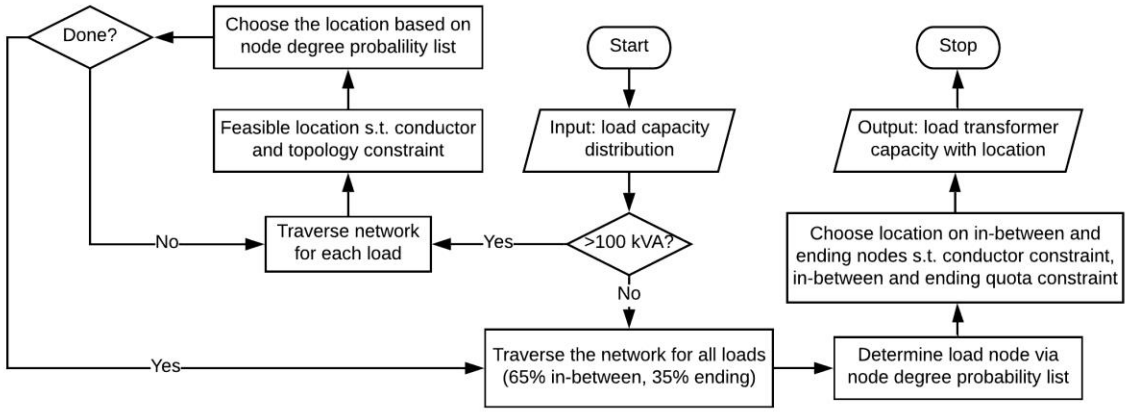


Figure 2.25: Load transformer generation flowchart

Finally, we validate the algorithm using a real distribution feeder. The load transformers in the original real feeders are shown in the left of Figure 2.26. We first summarize the load transformer number and capacity distribution. Then use the capacity distribution as the input of

the transformer generation algorithm. The results of the generated load transformers are shown in the right of Figure 2.26. It's clear that the generated load transformers are more concentrated on feasible locations due to tight constraints.

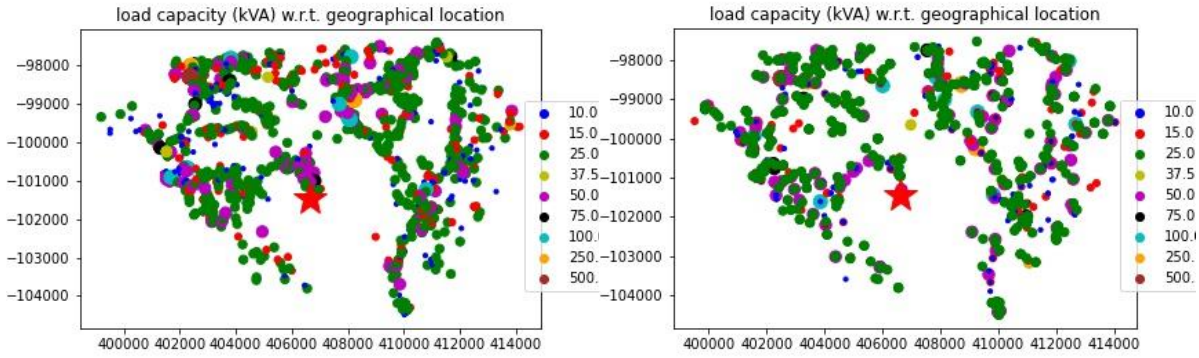


Figure 2.26: Geographical location of load transformers in the original feeder (left) and the generated feeder (right)

Figure 2.27 is shown below to further investigate the location and capacity statistics of the original and generated feeders. For loads smaller than 100 kVA, the location is rather random and the constraints are rather loose. We especially investigate the location of the 3 transformers larger than 100 kVA. In the original feeder, these 3 are on a lateral that is not far away from the backbone. In the generated feeder, the distance to the feeder head and distance to the nearest bifurcation point is the same, i.e. the topology ratio is 1, which indicates the algorithm would force the large load on the main backbone.

2.5.4 Combined Case Study

Once the generation process is finished, the FeederGAN tool automatically converts feeder topology and attributes as well as the load information back into OpenDSS input file formats for running power flow studies. Figure 2.28 shows the flowchart of the combined simulation process. Figure 2.29 shows an example of the nodal voltage distribution along the FeederGAN-generated feeder and Figure 2.30 shows the voltage profile of each phase with respect to the distance to the substation.

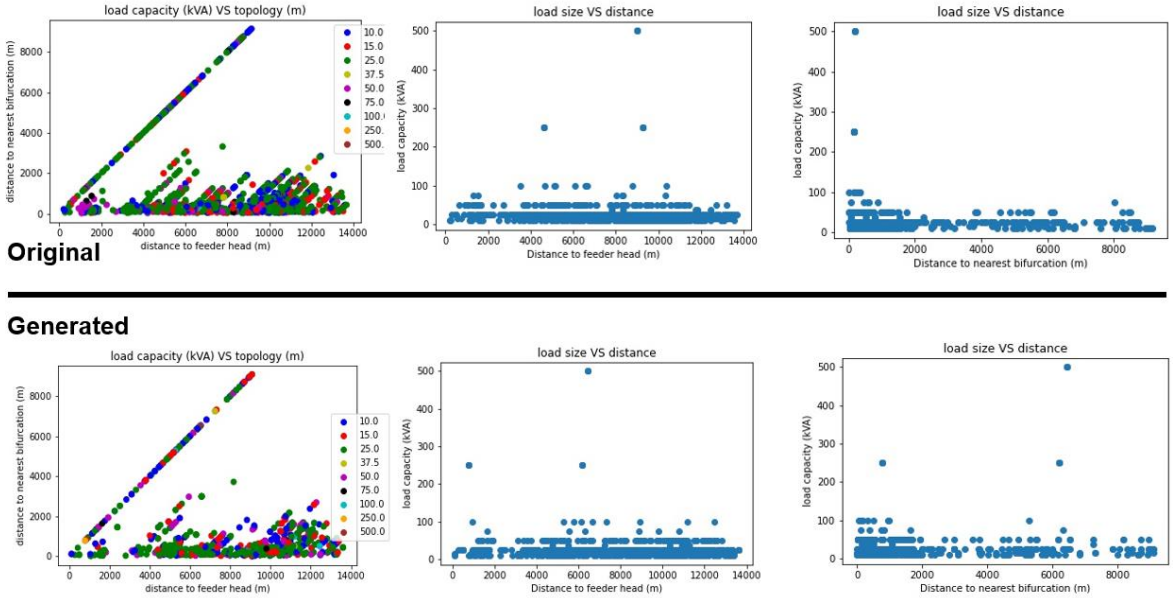


Figure 2.27: Location and capacity statistics of original (top) and generated (bottom) load transformers

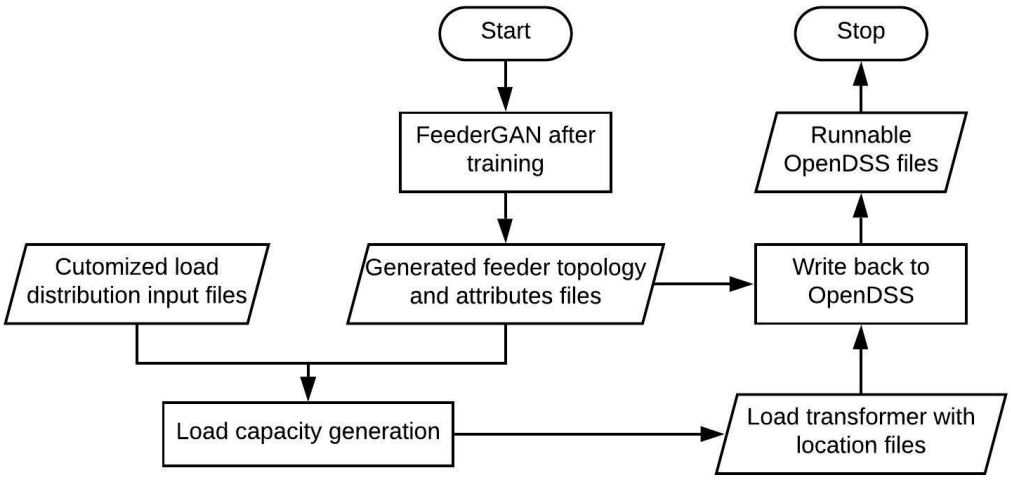


Figure 2.28: Combined simulation flowchart

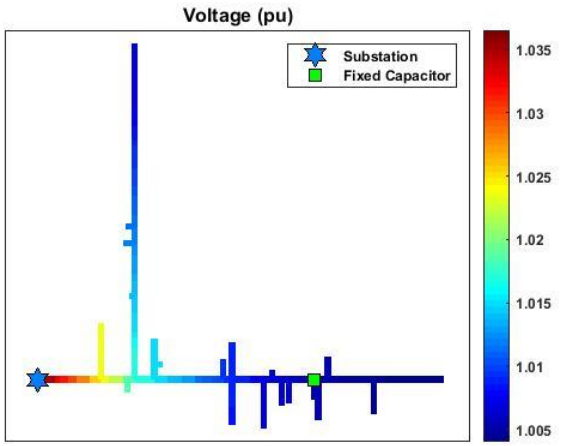


Figure 2.29: Nodal voltage along the generated feeder

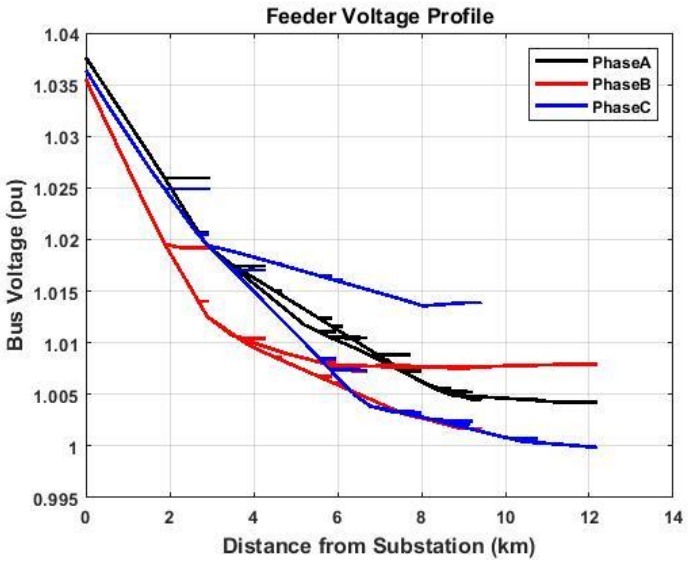


Figure 2.30: Voltage profiles of nodes on phase a, b, and c (Nodes arranged ascendingly according to their distance to the substation)

2.6 Conclusion

In this chapter, we introduced FeederGAN, a novel GAN-based method, to enable automated, high fidelity synthetic feeder generation. FeederGAN ingests power system distribution feeder models as directed graphs using a device-as-node representation. Feeder topology and device characteristics are coded into the adjacent matrix and the attribute matrix to allow GCN-based methods to learn topology and attributes from actual power system feeder model input files. This approach will make FeederGAN *expandable* (handling feeders at different scales

with any number of customized attributes) and having superb learning flexibility (learning from a full network structure or substructures). To solve the mode collapse problem, *early stop* and *human guidance* are proposed. The realisticness of the generated feeder topology can be validated by visual inspection and empirical statistics. Our simulation results have shown that even an expert cannot distinguish the generated topologies from the real ones. By comparing the empirical statistics, the generated synthetic feeders match well with the actual. This research aims at meeting the research needs of researchers for conducting distribution system planning studies and for developing control and energy management algorithms, where a large amount of realistic test feeders are needed.

After the feeder topologies and attributes are generated, we then use a statistical, rule-based method to generate the load transformers. The rules make the transformers follows line capacity constraints, line attributes constraints and topology constraints. Finally, we write the generated feeder file with load information into OpenDSS format and run a combined case study.

CHAPTER 3 SYNTHETIC LOAD DATA GENERATION VIA LOAD DISAGGREGATION

This chapter introduces the study on utilizing load disaggregation method to generate synthetic load data. The state-of-the-art non-intrusive load monitoring (NILM) methods performs well with high resolution smart meter data ($\leq$1 Hz), making it difficult to deal with the smart meter data collected and stored at most utilities, e.g. granularities of 15-, 30-, 60- min a data point. Hence, a novel sequential energy disaggregation (SED) algorithm is presented for extracting heating and cooling energy consumptions from residential and small commercial building loads using low-resolution (i.e. 15-minute, 30-minute, and 60-minute) smart meter data. The inputs of the algorithm include smart meter data and the corresponding outdoor temperature data. Large, infrequently used loads are first detected and removed from the total building energy consumption. Next, the heating method (i.e. gas-heating or electrical-heating) is identified. Then, base energy consumption curves, defined as the energy consumption without heating and cooling loads, are identified using the mild-day method. After that, the heating and cooling loads are extracted using a Monte-Carlo-based Average-Value Subtracting method. The SED-based method [33] is developed using a data-driven approach and will replace conventional empirical-value based methods currently used in distribution system planning. The method is validated using data collected from 137 households in the PECAN street project. Results show that the proposed SED method is computationally efficient, simple to implement, and robust in performance.

Based on the SED algorithm developed, case study is conducted on buildings with photovoltaic (PV) systems and electric vehicles (EVs). Among the cases, load database of zero-net energy (ZNE) cases, ZNE ready cases, ZNE with EV cases is built up. Therefore, those load data can be then used together with the generated synthetic feeders.

3.1 Background

This section we briefly introduce the background information of smart meter load disaggregation. This section is organized as follows. Sub-section 2.1.1 describes the challenges brought about by low resolution smart meter data available at utilities. Sub-section 2.2.2 presents the dataset we use to develop and validate our methodology.

3.1.1 Challenges for Traditional Methods

An example of the smart meter dataset with different resolution (granularity) is shown in Figure 3.1. As mentioned in Chapter 1, traditional NILM methods would use power or energy features/characteristics to extract the signal transition, either directly capture the signal changing values, e.g. signal processing based methods, or indirectly encode the transition states, e.g. Markov based methods. These methods are very sensitive to the data resolution/granularity. However, the smart meter data collected and stored at most utilities are rather low granularity, e.g. granularities of 15-, 30-, 60- min a data point, making it difficult for the state-of-the-art NILM methods, which performs well with high resolution data ($\leq$1 Hz). As shown in Figure 3.1 the low-resolution data is shaped not only by its nominal operating values, but also by its operation duration. Hence, this reality makes the fancy state-of-the-art papers fails to deal with the challenge faced by most utilities.

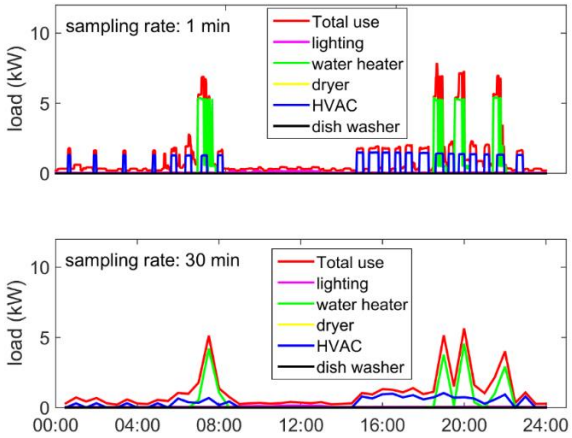


Figure 3.1: Load profiles of a residential household with different data resolutions

This dissertation conducted a thorough literature review of the state-of-the-art load disaggregation or NILM methods during 2014 to 2019 in IEEE Xplore and Science Direct database for transaction papers, journal papers and IEEE early access papers. Keywords used are *load disaggregation, energy disaggregation, non-intrusive load monitoring (NILM)*. In total, there are 158 papers (IEEE 76 and Science Direct 82). Among all the IEEE papers, a summary is made based on the signal they use, i.e. whether load or energy signal, and the data granularities. The results are shown as Table 3.1. For Science Direct papers, the searching result is similar and there are barely any papers talking about their results on energy (kWh/Wh) dataset with resolution larger than 30 minutes.

Only two papers handle with energy (kWh) data with data resolution larger than 10 minutes. However, researchers in [34] conduct quantitative study mainly on high resolution data and claim that their method also works good with dataset with 30-, 60-minute data. But they don't provide quantitative results for these low-resolution datasets. Researchers in [17] conduct thermal energy profiling on smart meter data. Their results are first tested on high resolution data with 15 kHz, then they validate in the real data with 1 hour resolution. However, their major concerns in

the paper are to model the demand response ability of a building using thermal loads. However, the purpose here is to conduct load disaggregation and then build up synthetic load database. Hence, this research is not only interested in the thermal load, but also wants to divide the such loads into individual appliance loads, e.g. whether it's a water heater load, HVAC load, or the base load that appears every day. Faced with the demand and request of the utilities, we have to develop our own SED methods.

Table 3.1: Summary of load disaggregation papers

Granularity / Type		< 1 sec	≥1 sec, < 1 min	≥ 1 min, < 10 min	≥10 min, < 30 min	≥ 30 min
Load	Power (kW/W)	14 papers	29 papers	15 papers	2 papers	0 paper
	Current (A)	18 papers	7 papers	3 papers	1 paper	0 paper
Energy	Energy (kWh/Wh)	0 paper	0 paper	1 paper	1 paper [34]	2 papers [17], [34]

3.1.2 Dataset for Load Disaggregation

Two data sets are used as inputs to the SED algorithms: low-resolution smart meter data and the corresponding weather data. Therefore, two types of data are required to develop the SED algorithm and verify its performance.

In this study, we used the energy consumption data set collected from 137 houses located at Austin, Texas in the PECAN street project [35]. Sub-meters were installed in those houses, so major appliance loads, such as washer and dryer, heating and cooling, water heater, cooking, lighting, were also available for download. We used the data collected in 2015 to conduct this study. The raw data is in 1 min resolution. We down-sampled the raw data to the required data

resolutions: 15-minute, 30-minute and 60-minute. A performance comparison is made for the three data resolutions to decide the best data resolution range for the SED to produce satisfactory results.

Using PECAN street data set allows us to have access to the sub-metered end use data. This makes it possible for us to verify the load disaggregation results, such as the accuracy in predicting the end use load shapes, daily/monthly/annual energy consumptions, load peaks and valleys, etc.

The second data set is the historical weather data collected in a weather station at Austin, Texas in 2015. The data can be downloaded using the National Oceanic and Atmospheric Administration (NOAA) Climate Data Online Tools [36]. In this research, only the temperature-load sensitivity is used for heating and cooling load extraction. As shown in Figure3.2, humidity, solar irradiance, and wind speed are either weakly correlated or not correlated with building electricity consumptions compared with the outdoor temperature. Our study also showed that ignoring the other weather variables only marginally influences the accuracy of the SED results. This is because when energy disaggregation is the main concern, infrequent and short-lived influences on appliance power consumptions cannot skew the results. As our previous thinking, humidity has something to do with human comfort feeling. However, the data we can collect is the outdoor humidity, thus it shows no correlation with the energy usage. In order to take humidity influence on human behavior into consideration, we use wet-bulb temperature as the weather input, which measures the temperature while taking into account evaporation and humidity.

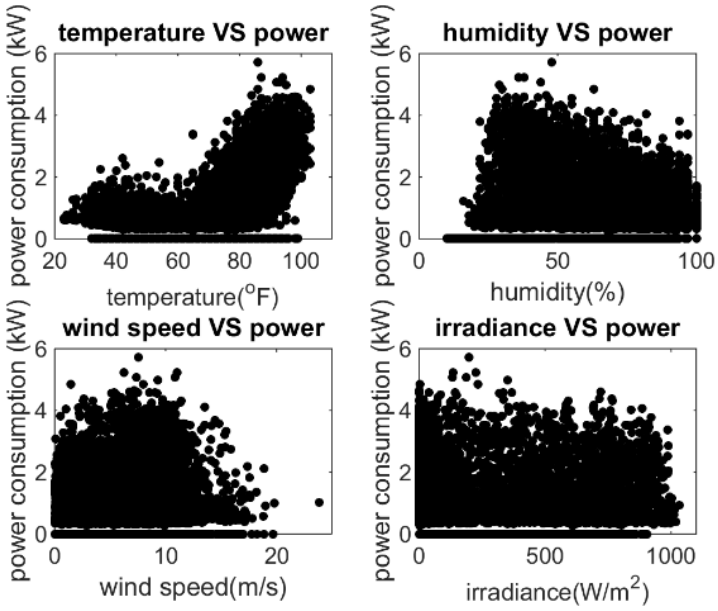


Figure 3.2: Hourly weather variables vs power consumption

3.2 Sequential Energy Disaggregation (SED) Algorithm

Figure 3.3 shows the flow chart of the SED algorithm. In the first step, a hump detection and removal algorithm are used to detect large, infrequently-used loads (LIUL) and remove them from the total load consumption. LIUL loads can be electric water heaters, dryers or other loads with power ratings higher than 4 kW and "on" durations longer than 5 minutes. As shown in Figure 3.4, when a water heater turns "on" and remains "on" for ten to fifteen minutes before turning off, it normally creates a distinct "hump" in the total load profile. After the removal of those humps, the total energy consumptions (the green line in Figure 3.4) will be smoother.

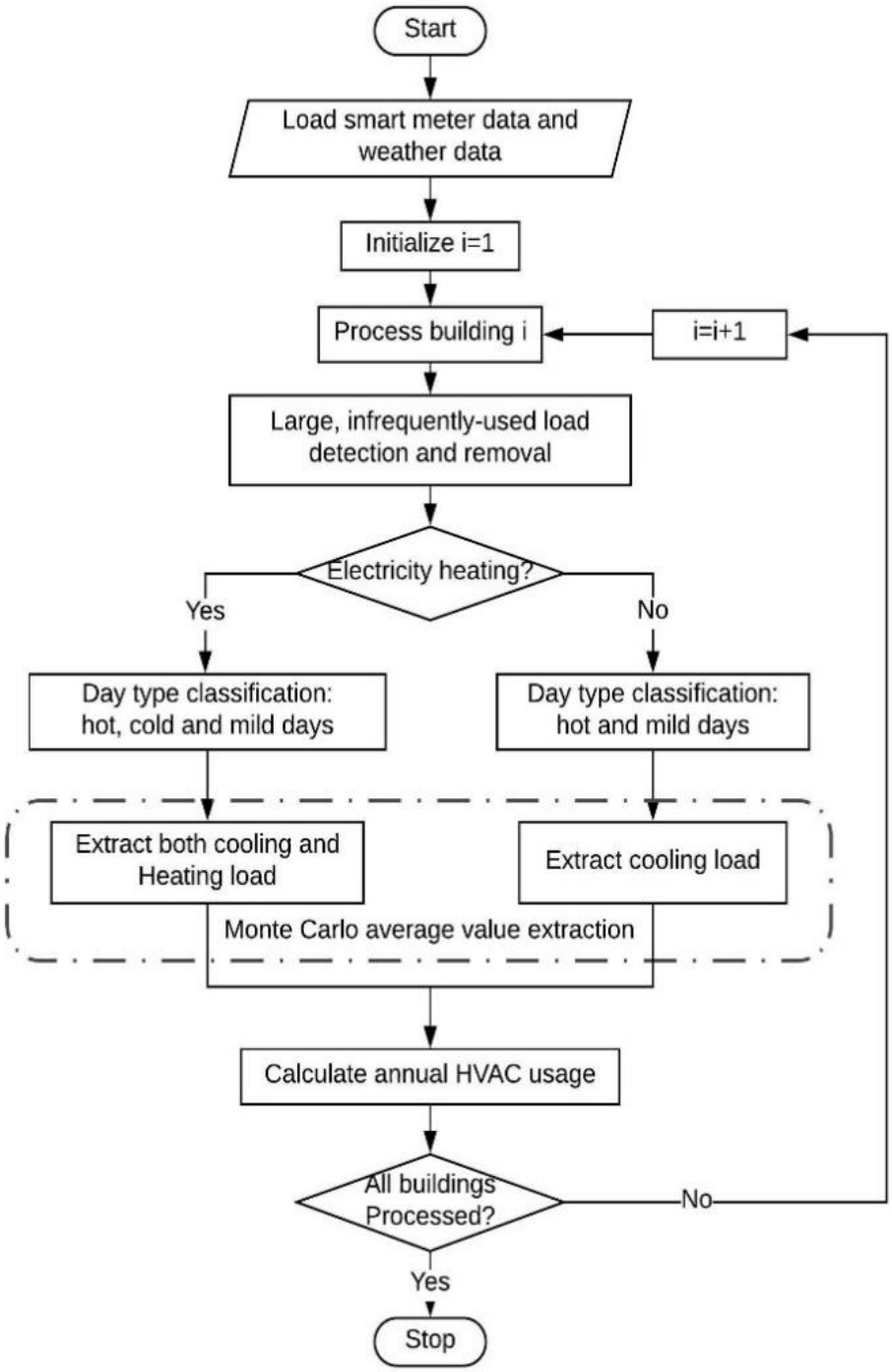


Figure 3.3: Flow chart of the SED algorithm

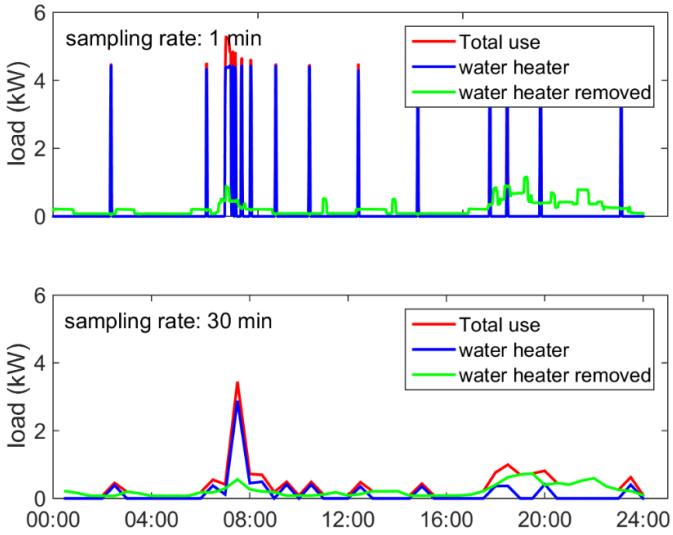


Figure 3.4: Load shape of electric water heater

In the second step, the heating method of a building will be identified by the building daily energy consumptions in winter days. As shown in Figure 3.5, compared with electrical-heating buildings, gas-heating buildings have much lower daily electricity consumptions, E_{daily}, in cold days (defined as days with daily average temperature below 50°F), making the identification possible using low-resolution smart meter data.In the third step, we observed that because of the cycling nature of the heating/cooling loads, it can be treated as an always-on load when using low-resolution smart meter data, especially in extreme days (i.e. the coldest days in winter or the hottest days in summer). Thus, in this step, the day type classification (DTC) algorithm uses outdoor temperatures and daily energy consumptions to identify "mild" days (defined as days with no or very little heating or cooling loads); "hot" days (defined as days with cooling loads); and "cold" days (defined as days with heating loads). Then, the Monte Carlo average value subtraction (MC-AVS) method can be used to find the daily average cooling and heating energy consumption of the house.

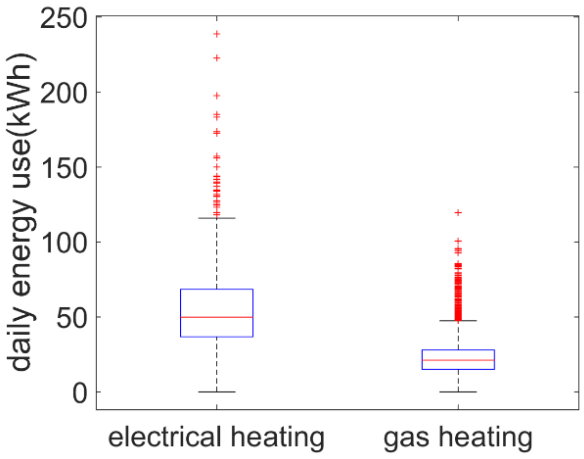


Figure 3.5: Daily energy use in winter

3.2.1 Hump Detection and Removal Algorithm

Let P_i be the i^{th} smart meter reading, δ_{rise} and δ_{drop} be the ramping up and down thresholds for identifying the start and end of a power hump, and D be the maximum duration of a power hump. The algorithm for large, infrequently-used load extraction is illustrated as follows. As shown in Figure 3.6, once a rising edge is detected, it is crucial to detect the falling edge of the hump so that a hump can be separated from the increasing part of the load curve (highlighted in blue dots), or a large hump caused by other types of loads (highlighted in yellow dots)

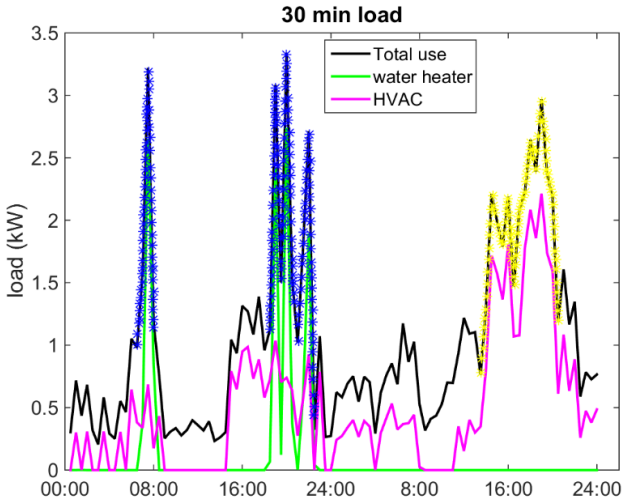


Figure 3.6: LIUL load extraction

The algorithm is summarized in Algorithm 3.1 (Table 3.2) as follows.

Table 3.2: Power Hump Detection and Removal Algorithm

Algorithm 3.1. Power Hump Detection and Removal Subroutine

Import 30-minute smart meter data and the corresponding outdoor weather data. Note that the data length is 52 weeks and is from Jan. 1 2015 – Dec 30, 2015. Set Timer = 0; Flag_hump = 0
for $i = 2{:}\,N$
  $R_i = P_i - P_{i-1}$
  if $R_i > \delta_{rise}$ & Flag_hump = 0
    Timer = Timer + Δt; hump_flag = 1
  elseif $R_i > \delta_{rise}$ & Flag_hump = 1
    Timer = 0; Timer = Timer + Δt;
  elseif $-R_i < \delta_{drop}$ & Flag_hump = 1
    Timer = Timer + Δt;
  elseif $-R_i > \delta_{drop}$ & Flag_hump = 1
    Timer = Timer + Δt, Flag_hump = 0;
    if Timer < D // hump detected, start the removal subroutine
      $j = i - Timer + \Delta t, cnt = Timer - \Delta t, temp = 0,$
      while (cnt)
        $tmp = tmp + R_j$; $P_j = P_j - tmp$ // remove the hump
        $cnt = cnt - \Delta t$; $j = j + \Delta t$;
      end while
      Timer = 0;
    end if
    Flag_hump = 0, Timer = 0; // not a power ramp
  end if
end for

3.2.2 Day Type Classification Algorithm

The DTC algorithm identifies three day types (i.e. mild, cold, and hot days) based on temperature-load sensitivities and daily energy consumptions. In the past, civil engineers use "degree days" [37] to determine building heating or cooling needs. First, the daily temperature mean is calculated by dividing the sum of the high and low temperatures of the day by two. If the daily temperature mean is above 65°F, the day is a Cooling Degree Day (CDD); if the temperature mean is below 65°F, the day is a Heating Degree Day (HDD). Thus, using CDD and HDD, one can estimate the heating and cooling loads of a building. However, 65°F is rather an arbitrary number that draws the line between CDDs and HDDs because types of buildings, occupancy, and customer comfort preferences all have significant influences on building cooling and heating loads. Thus, using CDD and HDD identified by conventional degree days for cooling and heating load identification often causes large errors.

Figure 3.7 shows a scattered plot of daily energy consumption, E_{daily}^{Total}, versus daily average temperature, T_{daily}^{mean}. We name such a plot as the Energy vs Temperature (EvT) plot. As shown in the left of Figure 3.7, the EvT plot of an electrical-heating building with air conditioning loads resembles a "U" shape. As shown in the right of Figure 3.7, the EvT plot of a gas-heating building with air conditioning loads resembles an inverse "L" shape. Note that for buildings with only heating load and no cooling load, the EvT curve will resemble an "L" shape. In this dissertation, the DTC algorithm is developed and validated using data collected in Texas, where all houses have electric cooling loads, so we only discussed the process of "U" shape and inverse "L" shape EvT curves. However, the same methodology can be readily extended to process the "L" shape EvT curves.

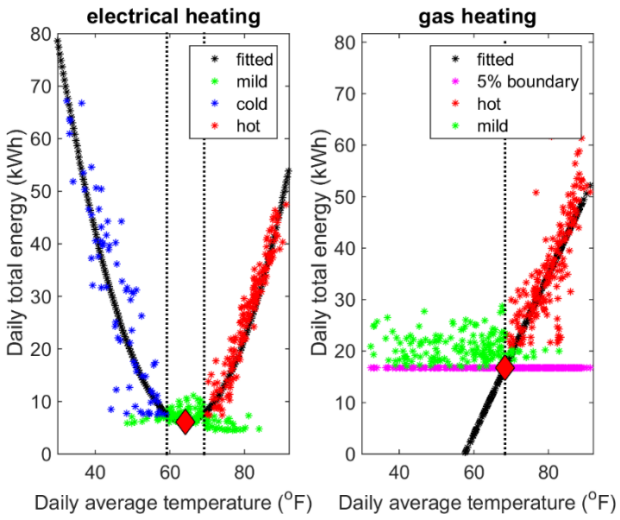


Figure 3.7: Daily energy consumption versus daily temperature mean

The "mild days" can be first identified as the data points located at the bottom of the "U" or near the turning points of the inverse "L". Then, the "hot days" can be identified as data points on the right side of the "U" or the inverse "L", where the cooling load dominates the building energy consumptions. In the end, the "cold days" for buildings with electric-heating can be identified to be the data points on the left side of the "U" shape EvT curve.

Note that although the shape of the EvT curve can also be used for detecting building heating methods, we used the building daily energy consumptions in winter months as an identifier for building heating type detection, as shown in Figure 3.5. Thus, before the execution of the DTC algorithm, the building heating and cooling load types of each smart meter data set have already been labeled.

The identifiers of the DTC algorithm for buildings with the "U" shape EvT curve is formulated as follows:

$$\boldsymbol{M}^T: \{d | T(d) \in [T_{min} - 5, T_{min} + 5]\} \tag{3.1}$$

$$\boldsymbol{H}^T: \{d | T(d) > T_{min} + 5\} \tag{3.2}$$

$$\boldsymbol{C}^T: \{d | T(d) < T_{min} - 5\} \tag{3.3}$$

$$\boldsymbol{M}^{mis}: \{d | E(\boldsymbol{M}^T) > median[E(\boldsymbol{H}^T)] \cup E(\boldsymbol{M}^T) > median[E(\boldsymbol{C}^T)]\} \tag{3.4}$$

$$\boldsymbol{H}^{mis}: \{d | E(\boldsymbol{H}^T) < median[E(\boldsymbol{M}^T)]\} \tag{3.5}$$

$$\boldsymbol{C}^{mis}: \{d | E(\boldsymbol{C}^T) < median[E(\boldsymbol{M}^T)]\} \tag{3.6}$$

$$\boldsymbol{M}^{reclCold}: \{d | d \in \boldsymbol{M}^{mis} \cap T(d) < T_{min}\} \tag{3.7}$$

$$\boldsymbol{M}^{reclHot}: \{d | d \in \boldsymbol{M}^{mis} \cap T(d) > T_{min}\} \tag{3.8}$$

$$\boldsymbol{M} = (\boldsymbol{M}^T - \boldsymbol{M}^{mis}) \cup \boldsymbol{H}^{mis} \cup \boldsymbol{C}^{mis} \tag{3.9}$$

$$\boldsymbol{H} = (\boldsymbol{H}^T - \boldsymbol{H}^{mis}) \cup \boldsymbol{M}^{reclHot} \tag{3.10}$$

$$\boldsymbol{C} = (\boldsymbol{C}^T - \boldsymbol{C}^{mis}) \cup \boldsymbol{M}^{reclCold} \tag{3.11}$$

where $\boldsymbol{M}$, $\boldsymbol{H}$, and $\boldsymbol{C}$ is the set of mild, hot, and cold days, respectively. Superscript *T* represents the set detected by using T_{daily}^{mean}; Superscript *mis* represents the misclassified set; Superscript *recl-* represents the reclassified set. $[T_{min}, E(T_{min})]$ represents the minimum point of the quadratic curve used to fit the "U" shape. Note that the detection band in Eq. (3.1)–(3.3) is set as 5°F to create a zone for detecting the mild days [38].

Similarly, the identifiers of the DTC algorithm for buildings with the inverse "L" shape EvT curve is formulated as follows:

$$\boldsymbol{M}^T: \{d | T(d) < T_{inter}\} \tag{3.12}$$

$$\boldsymbol{H}^T: \{d | T(d) > T_{inter}\} \tag{3.13}$$

$$\boldsymbol{M}^{mis}: \{d | E(\boldsymbol{M}^T) > median[E(\boldsymbol{H}^T)]\} \tag{3.14}$$

$$\boldsymbol{H}^{mis}: \{d | E(\boldsymbol{H}^T) < median[E(\boldsymbol{M}^T)]\} \tag{3.15}$$

$$\boldsymbol{M} = (\boldsymbol{M}^{T} - \boldsymbol{M}^{mis}) \cup \boldsymbol{H}^{mis} \tag{3.16}$$

$$\boldsymbol{H} = (\boldsymbol{H}^{T} - \boldsymbol{H}^{mis}) \cup \boldsymbol{M}^{mis} \tag{3.17}$$

The steps of the DTC algorithm processing buildings with U-shape EvT curves are illustrated as follows.

Table 3.3: DTC Algorithm for Buildings with U-shape EvT curves

Algorithm 3.2. DTC Algorithm for Buildings with U-shape EvT curves

1: Import 30-minute smart meter data processed by algorithm 1, the building heating and cooling types, and the corresponding outdoor weather data. Note that the data length is 52 weeks and is from Jan. 1, 2015 – Dec 30, 2015. So there are 17472 data points and 364 days.

2: Calculate E_{daily}^{Total} and T_{daily}^{mean} for 364 days, generate the EvT scatter plot

3: Fit a quadratic cure to the "U" shape scatter plot (see Figure 3.7(a)) and find the minimal point of the quadratic curve, $[T_{min}, E(T_{min})]$.

4: Apply Eq. (3.1) - (3.3) to identify $\boldsymbol{M^T}$, $\boldsymbol{H^T}$, $\boldsymbol{C^T}$ based on $T_{min} \pm 5$°F.

5: Apply Eq. (3.4) - (3.6) to identify the misclassified day sets, $\boldsymbol{M^{mis}}, \boldsymbol{H^{mis}}, \boldsymbol{C^{mis}}$, based on median daily energy consumptions of the day groups.

6: Apply Eq. (3.7) and (3.8) so that T_{min} is used as a threshold to reclassify the days in the misclassified mild day set to the cold and hot day sets, $\boldsymbol{M}^{reclCold}$ and $\boldsymbol{M}^{reclHot}$, respectively.

7: Apply Eq. (3.9) - (3.11) to obtain the final sets for the mild, hot and cold day groups, $\boldsymbol{M}$, $\boldsymbol{H}$, and $\boldsymbol{C}$, respectively.

The steps of the DTC algorithm processing buildings with the inverse L-shape EvT curves are illustrated as follows. Note that the 5th percentage as the mild day boundary is chosen because we need to exclude some low-occupancy days which cannot provide us with enough baseline load information.

Table 3.4: DTC Algorithm for Buildings with L-shape EvT curves

Algorithm 3.3. DTC Algorithm for Buildings with L-shape EvT curves

1: Import 30-minute smart meter data processed by algorithm 1, the building heating and cooling types, and the corresponding outdoor weather data. Note that the data length is 52 weeks and is from Jan. 1 2015 – Dec 30, 2015. So there are 17472 data points and 364 days.

2: Calculate E_{daily}^{Total} and T_{daily}^{mean} for 364 days, generate the EvT scatter plot.

3: Fit two linear curves (the magenta and black curves in Figure 3.7(b)) to the inverse "L" shape scatter plot) and find the intersection of these two curves $[T_{inter}, E(T_{inter})]$.

4: Apply Eq. (3.12) - (3.13) to identify $\boldsymbol{M^T}$ and $\boldsymbol{H^T}$ based on T_{inter}.

5: Apply Eq. (3.14) - (3.15) to identify the misclassified day sets, $\boldsymbol{M^{mis}}$ and $\boldsymbol{H^{mis}}$, based on median daily energy consumptions of the mild and hot day groups.

6: Apply Eq. (3.16) - (3.17) to get the final sets for the mild and hot day groups, $\boldsymbol{M}$ and $\boldsymbol{H}$, respectively.

3.2.3 Monte Carlo Average Value Subtraction Algorithm

After the mild, cold and hot day types are classified, we group the daily load profiles into mild, hot, and cold load groups, represented by D_{mild}, D_{hot} and D_{cold}.

$$D_{mild} = \{P(m,j) | \, j \in [1,2,\dots,n], m \in \boldsymbol{M}\} \tag{3.18}$$

$$D_{hot} = \{P(h,j) | \, j \in [1,2,\dots,n], h \in \boldsymbol{H}\} \tag{3.19}$$

$$D_{cold} = \{P(c,j) | \, j \in [1,2,\dots,n], c \in \boldsymbol{C}\} \tag{3.20}$$

where P is the smart meter data for a building after the LIUL removed; *m*, *h*, *c* are the date index in a year; j is the index of smart data points in a day so that $n = 48$ if the smart meter data resolution is 30 minutes. Figure 3.8 shows daily load curves of the three load groups for an electrical heating house.

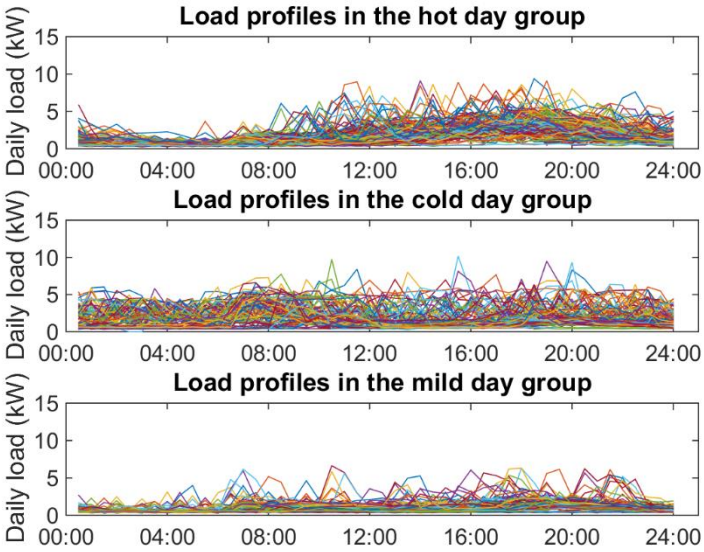


Figure 3.8: Hot, cold and mild day groups

A. Direct Average Value Subtraction Algorithm

A straight forward method for calculating the HVAC loads is as follows. First, calculate the average daily load curves for the mild, hot, and cold day types. $\overline{\boldsymbol{P_m}}(j), \overline{\boldsymbol{P_h}}(j), \overline{\boldsymbol{P_c}}(j)$. Then, an estimation of averaged cooling and heating loads, $\widehat{\boldsymbol{P}}_{\boldsymbol{cooling}}(j)$ and $\widehat{\boldsymbol{P}}_{\boldsymbol{heating}}(j)$, can be calculated, the result of which can be used to estimate the yearly energy consumption of the HVAC load, $\hat{E}_{HVAC}$. This process is formulated in Eq. (3.21) - (3.26).

$$\overline{\boldsymbol{P_m}}(j) = \frac{1}{M}\sum_{m=1}^{M} P(m,j) \tag{3.21}$$

$$\overline{\boldsymbol{P_h}}(j) = \frac{1}{H}\sum_{h=1}^{H} P(h,j) \tag{3.22}$$

$$\overline{\boldsymbol{P_c}}(j) = \frac{1}{C}\sum_{c=1}^{C} P(c,j) \tag{3.23}$$

$$\widehat{\boldsymbol{P}}_{\boldsymbol{cooling}}(j) = \overline{\boldsymbol{P_h}}(j) - \overline{\boldsymbol{P_m}}(j) \tag{3.24}$$

$$\widehat{\boldsymbol{P}}_{\boldsymbol{heating}}(j) = \overline{\boldsymbol{P_c}}(j) - \overline{\boldsymbol{P_m}}(j) \tag{3.25}$$

$$\hat{E}_{HVAC} = u^C \frac{24}{n} H \sum_{j=1}^{n} \widehat{\boldsymbol{P}}_{\boldsymbol{cooling}}(j) + u^H \frac{24}{n} C \sum_{j=1}^{n} \widehat{\boldsymbol{P}}_{\boldsymbol{heating}}(j) \tag{3.26}$$

where u^H and u^C represent heating and cooling load types (1 with and 0 without). $M = card(\boldsymbol{M})$, $H = card(\boldsymbol{H})$ and $C = card(\boldsymbol{C})$, representing the cardinality of mild, hot and cold days. Because this method directly subtracts the average load profile of each load group, we call it the direct-average value subtraction (D-AVS) method. D-AVS is simple and straightforward but because it over-simplifies the relationships between different load groups, the accuracy of the results is limited and not consistent for different buildings.

B. Monte Carlo Average Value Subtraction Algorithm

The MC-AVS algorithm is illustrated as follows. The main difference between the MC-AVS and D-AVS is that instead of directly subtracting the means between hot/mild and cold/mild daily load profiles, we randomly select the load curves from the hot/mild and cold/mild day pairs for subtraction so that a number of cooling/heating load curves can be calculated and the average of those cooling/heating load curves will converge close to the actual average curve. Note that when executing Algorithm 3.4 (Table 3.5), one can either run Monte Carlo simulation until the value converges or simply permute all the possible combinations of the hot/mild and cold/mild day pairs. Assuming there are 60 days in the mild day group and there are 150 days in the hot day group, then the permutation method requires 9000 runs.

An example of the extracted heating and cooling load profiles (red lines) using MC-AVS is shown in Figure 3.9. As shown in Figure3. 9, MC-AVS has much better performance than D-AVS, showing that the random subtraction between hot/mild and cold/mild day pairs can successfully remove the infrequently used loads and preserve the invariant, cycling loads in the progressive averaging process. This is because D-AVS uses point-based average value subtraction, letting cross- and auto- correlations unaccounted for, while MC-AVS uses load shape subtraction so that both the auto-correlations among the time series data points and the cross-correlations between

load curves are considered.

Table 3.5: MC-AVS Algorithm

Algorithm 3.4. MC-AVS Algorithm

1: Import the three pools of 24-hour load profiles: D_{mild}, D_{hot} and D_{cold}.

2: k = 1, flag = 1;

3: **while** (flag)

4: Randomly draw a mild day curve $\boldsymbol{P}(m, j)$ from D_{mild}

5: Randomly draw a hot day curve $\boldsymbol{P}(h, j)$ from D_{hot}

6: $$\boldsymbol{P}^{\boldsymbol{k}}_{\boldsymbol{cooling}}(j) = \boldsymbol{P}(h, j) - \boldsymbol{P}(m, j) \quad (3.27)$$

7: $$\widehat{\boldsymbol{P}}_{\boldsymbol{cooling}}(j) = \frac{1}{k}\sum_{j=1}^{k} \boldsymbol{P}^{\boldsymbol{j}}_{\boldsymbol{cooling}}(t) \quad (3.28)$$

8: **if** electrical heating

9: Randomly draw a cold day curve $\boldsymbol{P}(c, j)$ from D_{cold}.

10: $$\boldsymbol{P}^{\boldsymbol{k}}_{\boldsymbol{heatng}}(j) = \boldsymbol{P}(c, j) - \boldsymbol{P}(m, j) \quad (3.29)$$

11: $$\widehat{\boldsymbol{P}}_{\boldsymbol{heating}}(j) = \frac{1}{k}\sum_{i=1}^{k} \boldsymbol{P}^{\boldsymbol{i}}_{\boldsymbol{heating}}(j) \quad (3.30)$$

12: **end if**

13: **if** $\widehat{\boldsymbol{P}}_{\boldsymbol{cooling}}(j)$ and $\widehat{\boldsymbol{P}}_{\boldsymbol{heating}}(j)$ converged

14: flag = 0

15: **else** k=k+1

16: **end if**

17: **end while**

18: Calculate the annual HVAC usage use Eq. (3.26)

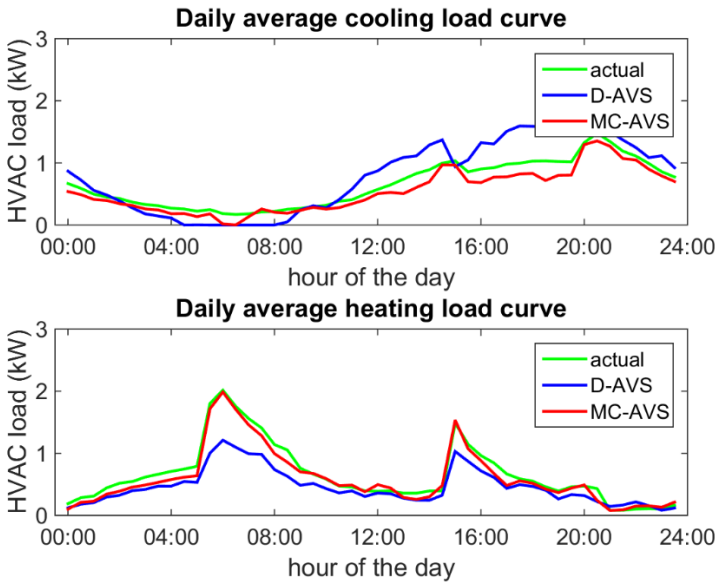


Figure 3.9: Comparison of MC- and D-AVS results

C. SED Algorithm

The SED algorithm is summarized as follows.

Table 3.6: SED Algorithm

Algorithm 3.5. SED Algorithm

1: Import 30-minute smart meter data processed by algorithm 3.1, the building heating and cooling types, and the corresponding outdoor weather data. Note that the data length is 52 weeks and is from Jan. 1 2015 – Dec 30, 2015. So there are 17472 data points and 364 days.

2: Run the hump detection and removal algorithm for LIUL extraction

3: Run the day-type detection algorithm to obtain the mild, hot, and cold day groups.

4: Run the Monte Carlo average value subtraction algorithm to calculate the average hot and cold load curves.

5: Calculate the total HVAC cooling and heating load consumptions.

3.3 Performance Evaluation

In this section, the performance criterion for evaluating the SED algorithm is first presented, followed by the impact study of using different data resolutions and a comparison between MC-AVS and D-AVS for extracting the heating/cooling loads.

3.3.1 Performance Evaluation Criterion

Three accuracy criteria are used for assessing the results of the SED algorithm: accuracy of hot/cold/mild day classification, accuracy of cooling/heating load profile estimation, and accuracy of total heating/cooling energy consumption calculation.

Since the sub-meter data cannot tell us whether the HVAC usage is cooling or heating. We treat it as a binary classification problem (i.e. mild days for HVAC OFF days and hot/cold days for HVAC ON days). Thus, the macro $f1$ score [39] can be used to test the accuracy.

$$Macro\ f1 = 2Pre \times Rec/(Pre + Rec) \tag{3.31}$$

$$Pre = \frac{1}{K} \sum_{i=1}^{K} Pre^i \tag{3.32}$$

$$Rec = \frac{1}{K} \sum_{i=1}^{K} Rec^i \tag{3.33}$$

where $Pre^i = \frac{TP^i}{TP^i+FP^i}$ and Rec= $\frac{TP^i}{TP^i+FN^i}$ are the precision and recall for building i respectively, K is the number of buildings, and TP, FP and FN are true positive, false positive and false negative in the confusion matrix.

Root mean square errors (RMSE) are used to compare the calculated with the actual load curves.

$$RMSE^i = \sqrt{\frac{1}{n}\sum_{j=1}^{n}\left(\hat{P}(j) - P(j)\right)^2} \tag{3.34}$$

$$overall\ RMSE = \frac{1}{K}\sum_{i=1}^{K} RMSE^i \tag{3.35}$$

where $RMSE^i$ is the RMSE value for building i, $\hat{P}(j)$ is the calculated cooling/heating daily load curve, and $P(j)$ is the actual cooling/heating load averaged over hot/cold days via sub-meter data. Note that $n=48$ if the data resolution is 30 minutes. The overall *RMSE* measures the average accuracy of the SED method for estimating the HVAC load profiles for K buildings.

ED-based algorithms are often used by utility planning engineers to estimate energy consumptions by end use at different granularity. In those cases, the total energy consumption of *K* buildings supplied by a distribution transformer or by a distribution feeder is of interest. Hence, the normalized error is used to evaluate the accuracy of the calculated energy consumption of the heating/cooling loads of *K* buildings.

$$error^i = (\widehat{E^i}_{HVAC} - {E^i}_{HVAC})/{E^i}_{Total} \tag{3.36}$$

$$error = \frac{1}{K}\sum_{i=1}^{K}|error^i| \tag{3.37}$$

$$agg\_error = \sum_{i=1}^{K}(\widehat{E^i}_{HVAC} - {E^i}_{HVAC}) \,/\, \sum_{i=1}^{K}{E^i}_{Total} \tag{3.38}$$

where $\widehat{E^i}_{HVAC}$, ${E^i}_{HVAC}$ and ${E^i}_{Total}$ are the estimated HVAC, actual HVAC and actual total annual energy usage for building *i*. Eq. (3.36) and (3.37) measure the error at the household level and Eq. (3.38) measure the error for the aggregated load.

3.3.2 Day Type Classification Accuracy

The precision, recall, and $f_1$score of the DTC algorithm are all above 90%. An analysis is performed on the outliers. We found that among the 137 buildings, a few "no electrical heating" buildings actually have plug-in electrical heating loads (see Figure 3.10(c)). Thus, although the sub-metered HVAC loads for those buildings are zero (see Figure 3.10(d)), there are actual heating load presented in those buildings. Thus, we exclude those buildings and recalculated that precision is 0.9503, recall is 0.9274, and $f_1$score is 0.9387 for the remaining buildings in the "without

outliers" case.

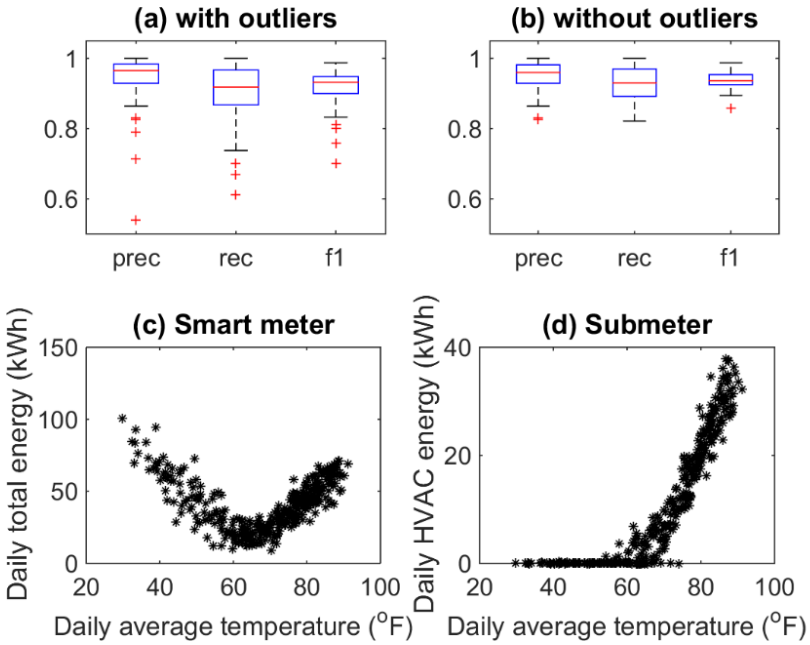


Figure 3.10: Day type classification (with and without outliers)

Note that the DTC algorithm allows the mild/cold/hot day to be accurately classified using the outdoor temperature and the smart meter data, making the SED algorithm transferrable to other location outside Texas.

3.3.3 Performance Under Different Data Resolutions

Table 3.7 shows the SED performance when 15-min, 30-min and 60-min smart meter data are used. The overall RMSE is approximately 0.18 kW, i.e. the error for the extracted cooling/heating load is about 0.18 kW. Note that the rated power of the heating and cooling loads ranges from 2 to 5 kW. The averaged error of the cooling/heating load profile estimation is consistently around 5% across three data resolutions. This shows that the SED algorithm is insensitive to the three data resolution studied. The aggregated load estimation error is about 2-3%, which is less than estimating individual loads. This is because when aggregated together, the

overestimated or underestimated loads cancelled out each other, making the aggregated error at the feeder a little smaller.

Table 3.7: The performance on different resolution dataset

	Overall RMSE calculated by Eq. (2.35)	Error calculated by Eq. (2.37)	Aggregated error calculated by Eq. (2.38)
15 min	0.1866	5.01%	2.36%
30 min	0.1812	5.16%	2.75%
60 min	0.1779	5.12%	3.01%

Figure 3.11 plots the error distributions for data sets in different granularities at 6 pivoting time. The results demonstrate that the proposed SED method has consistent performance using 15, 30 and 60 minutes smart meter data. Thus, the accuracy of SED algorithm is not influenced significantly by different data resolutions.

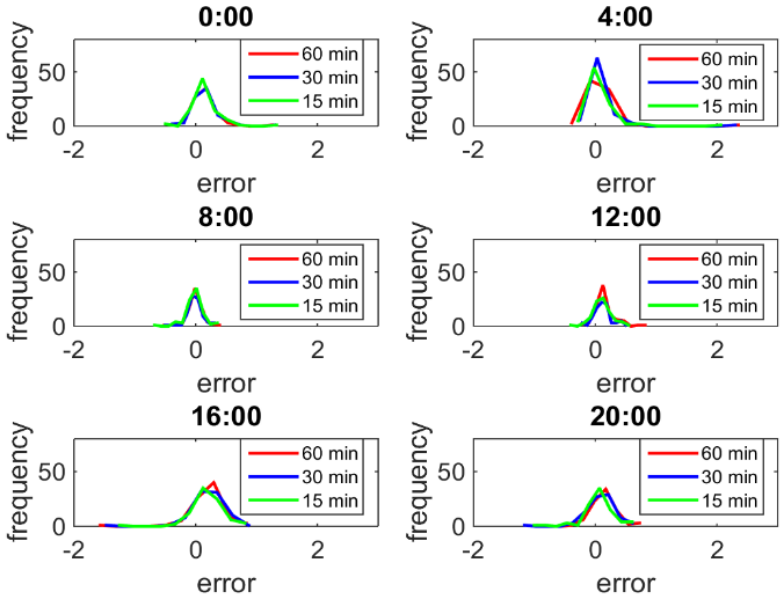


Figure 3.11: Error distributions for different granularities

We also studied the outliers in the boxplots in Figure 3.12. There are two causes for large errors (>1kW) to occur. First, some houses installed two or more heating/cooling units. Because the

owners may use one of the units more often, the calculation based on averaged behaviors can lead to large errors. Second, the number of mild days is important for the heating and cooling load extraction. Figure 3.13 shows the scatter plot of the annul HVAC kWh calculation error versus the number of mild days for all the buildings. The plot shows a strong linear trend. For buildings with 100 to 150 mild days identified, the errors are normally small. For buildings with either more or fewer mild days, the errors become bigger. This shows that when each day type group consists of sufficient number of days, infrequently-use/non-cyclic loads will be successfully excluded through the averaging process.

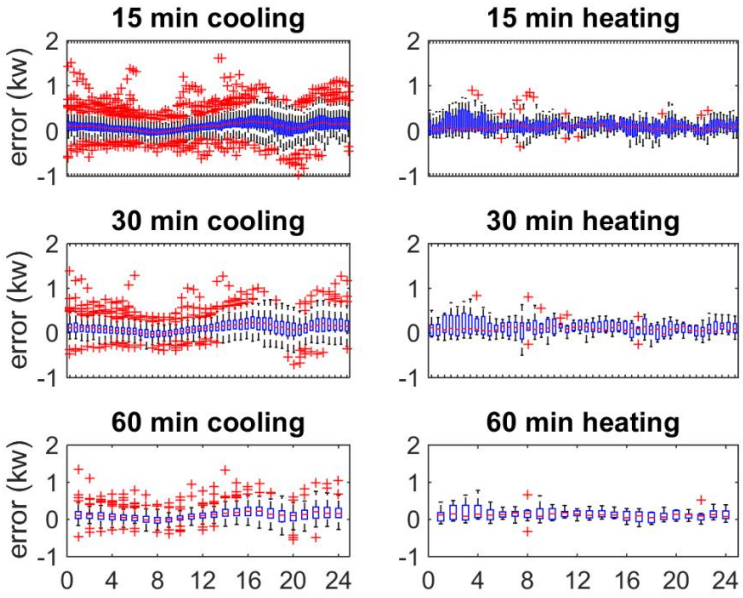


Figure 3.12: Errors in time of the day

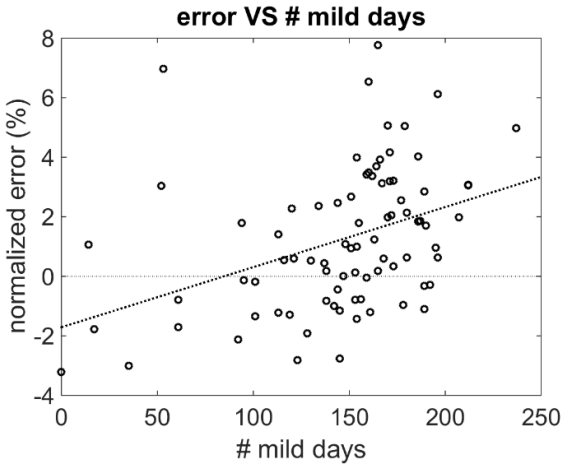


Figure 3.13: Annual HVAC kWh errors VS number of mild days

3.3.4 Comparison of the MC-AVS and D-AVS algorithms

The performance of the MC-AVS and D-AVS algorithms is compared using 30-min smart meter data. Table 3.8 shows that the MC-AVS outperforms the D-AVS on all three accuracy criteria. This is because MC-AVS extracts the cooling and heating loads by comparing the mild/hot and mild/cold daily load profile pairs while D-AVS directly subtracts the averaged values between the mild/cold and mild/hot day groups. Thus, auto-correlation and cross-correlation of daily load curves cannot be accounted for. In addition, the D-AVS method tends to be less effective when excluding other types of temperature sensitive loads (e.g. refrigerators and fans) and tends to overestimate the HVAC load, especially when estimating cooling loads in hot days, as shown in Figure 3.14.

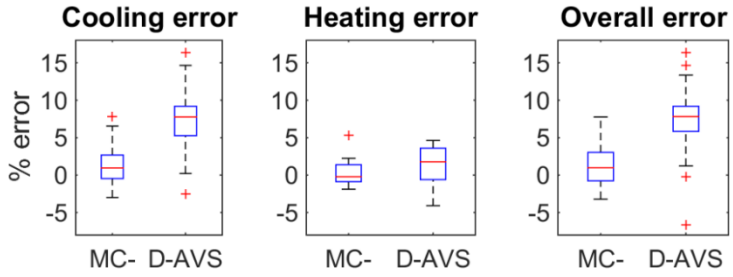


Figure 3.14: Error comparison of MC- and D-AVS results

Table 3.8: Comparison between the MC-AVS and D-AVS algorithms

	Overall RMSE calculated Eq. (2.35)	**Error calculated by Eq. (2.37)**	**Aggregated error calculated by Eq. (2.38)**
MC-AVS	0.1812	5.16%	2.75%
D-AVS	0.3512	8.13%	6.51%

3.4 Zero-Net Energy (ZNE) Case Study

In this section, based on the SED algorithm developed, we conduct case study on buildings with photovoltaic (PV) systems and electric vehicles (EVs) [40]. Among the cases, we build up load database of zero-net energy (ZNE) cases, ZNE ready cases, ZNE with EV cases. Therefore, those load data can be then used together with later generated synthetic feeders.

3.4.1 ZNE Concepts and Algorithms

A ZNE building refers to a building whose total amount of energy consumed is roughly equal to self-generated energy on an annual basis. By replacing old appliances with new energy efficient appliances, a retrofit building is called "ZNE ready". By installing rooftop photovoltaics (PV) systems together with energy storage devices, a building can be converted to "ZNE".

A. ZNE Ready Case

The HVAC load can be represented by [22],

$$P_{HVAC} = P_{th} + P_{non-th} \tag{3.39}$$

where P_{HVAC} is the disaggregated HVAC load, P_{th} is the thermal part, it can either be cooling load in the summer or heating load in the winter, and P_{non-th} is the ancillary part such as fans etc. We assume that in the mild days, the major HVAC load is the ventilation load; in hot days the major HVAC load is the cooling plus ventilation load; and in cold days the major HVAC load is the heating load plus the ventilation load.

Thus, we have

$$P_{HVAC}^{ready\_mild} = P_{non-th}(1 - R_{ven}) \tag{3.40}$$

$$P_{HVAC}^{ready\_hot} = P_{th}(1 - R_{cool}) + P_{non-th}(1 - R_{ven}) \tag{3.41}$$

$$P_{HVAC}^{ready\_cold} = P_{th}(1 - R_{heat}) + P_{non-th}(1 - R_{ven}) \tag{3.42}$$

where R_{ven}, R_{cool}, and R_{cold} are energy reduction rates for ventilation, cooling and heating sub-load parts when more energy efficient appliances are used to replace the old ones. Based on [22], the reduction rate for the ventilation load is between 13% - 17%, for the cooling load is between 7% - 10%, and for the heating load is between 25% - 30%. The reduction rates can be customized in the ZNE analysis tool we develop so that one can define the rates of energy reduction when using a different energy efficiency program.

For the water heater and the base load, the "ZNE ready" consumptions are calculated as

$$P_{water}^{ready} = P_{water}(1 - R_{water}) \tag{3.43}$$

$$P_{base}^{ready} = P_{base}(1 - R_{base}) \tag{3.44}$$

where R_{water} and R_{base} are energy reduction rates for the electric water heater load and the base loads; P_{water} and P_{water}^{ready} are the power of electric water heater before and after the energy efficiency program; P_{base} and P_{base}^{ready} are the power of baseline load before and after the energy efficiency program.

Hence, the total load before and after the energy efficiency program can be calculated as

$$P_{total} = P_{base} + P_{HVAC} + P_{water} \tag{3.45}$$

$$P_{total}^{ready} = P_{base}^{ready} + P_{HVAC}^{ready} + P_{water}^{ready} \tag{3.46}$$

where P_{total} and P_{total}^{ready} are the total power of a building before and after the energy efficiency program, i.e., the power of baseline stage and ZNE ready stage.

B. ZNE Case

The core purpose for the ZNE simulation is to calculate the number of PV panels required to achieve the ZNE target, i.e., the net energy from the electric power gird is zero. In our study, we size the PV panels so that the annual energy consumed equals to the self-generated PV power. Thus, the ZNE balance function is expressed as

$$N \times P_{cap} \times (\sum_{i=1}^{365\times48} PV_i / P_{ref}) = \sum_{i=1}^{365\times48} P_{total\_i}^{ready} \quad (3.47)$$

where P_{ref} is the total capacity for the rooftop PV panel arrays and PV_i is the solar power generated at time i. Power output data collected from a roof-top PV system at Raleigh North Carolina area is used as reference to calculate the PV generation. N is the number of PV panels required for ZNE; P_{cap} is the capacity of PV panels to be installed, which also can be customized in the developed tool. The left side calculates the energy generated by PV panels and the right hand side calculates the energy usage in the ZNE ready stage. Since our smart meter data resolution is 30 min, there are 48 points for each day.

3.4.2 Simulation Results

Smart meter data with 30-minute data resolution collected from 5000 buildings by Duke Energy in the North Carolina area are used for conducting the study.

A. House-level Impacts

We first studied at the house and building level impacts for the ZNE-ready case and the ZNE case by comparing with the base case. Figure 3.15 illustrates the simulation outputs for a residential building with square footage of 2368 ft^2. In the simulation, reduction rates for cooling, heating, ventilation and water heater are set as 10%, 20%, 10%, and 10%. As shown in Figure 3.15, the energy efficiency program reduces annual energy by 1129.66 kWh. It requires approximately 20 PV panels to go ZNE with each 300 W. If new loads such as an electric vehicle

(EV) is added, eight more panels will be added to achieve ZNE. Note that the EV charging curve is obtained from the PECAN street data sets [35].

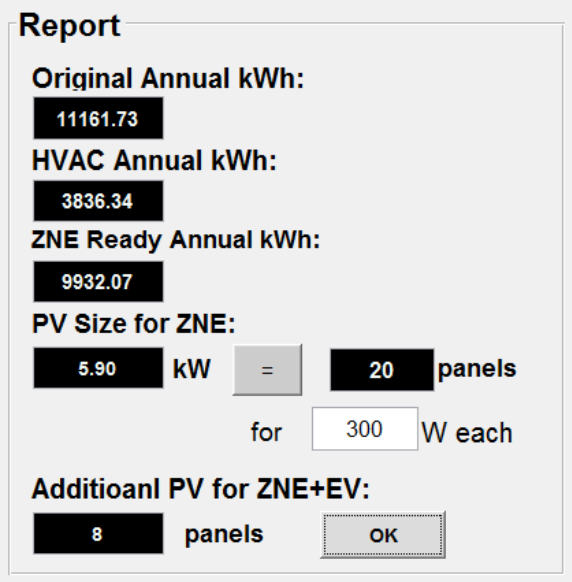


Figure 3.15: Summary of energy consumptions of a ZNE building

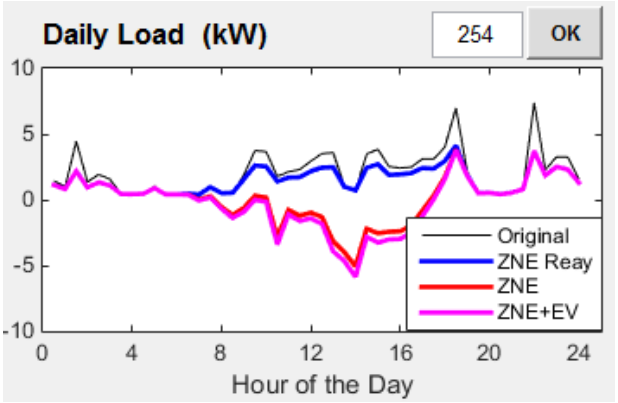


Figure 3.16: Daily load curves for base, ZNE-ready, ZNE and ZNE+EV cases

Figure 3.16 shows the daily load curves in a sunny day (day 254, Sept. 12th). Compared with the baseline, the ZNE-ready case does not significantly change the building load profile while the ZNE and ZNE+EV cases reshape the baseline load profile to resemble the duck curve. Therefore, when a building is converted to ZNE and backfeeding power is allowed, the net load will be negative most of the time during the day if it is a sunny day. In the next section, we conduct

power flow studies at the feeder level to evaluate the aggregated impact of ZNE on the distribution feeder at different penetration levels. Note that in our study, the feeder models were from the same feeder where the 5000 buildings were connected to.

B. Feeder-level Impact

To conduct the feeder level power flow study, we first generate load database for the baseline, ZNE ready, and ZNE cases for all 5000 houses. Then, we modeled the distribution feeder in OpenDSS using the method presented in [41]. Due to the page limit, we only present one case study in this dissertation. In this study, we randomly select 20% of nodes to be "ZNE" nodes and 20% of nodes as "ZNE ready" nodes. A weekly feeder head load profiles of the two cases are shown in Figure 3.17. It can be observed that although there is only 20% ZNE nodes, the net load at feeder head will decrease to approximately 0 kW at noon.

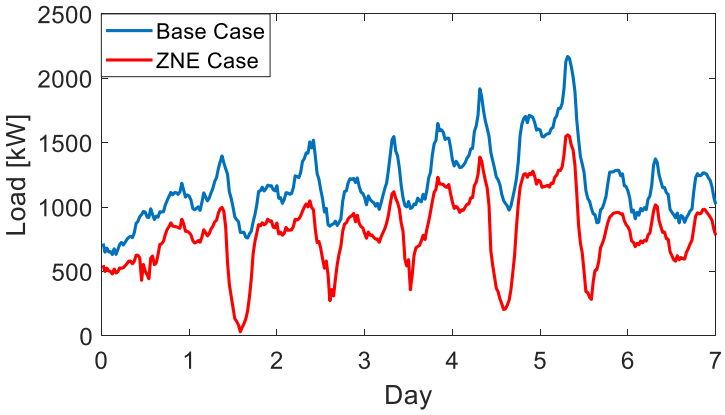


Figure 3.17: Feeder head load profile for different cases

Figure 3.18 shows the feeder voltage profile at noon. Due to the power generated by PV system, over-voltage issue appeared at most nodes of this feeder. Therefore, to supply ZNE buildings, voltage regulators or other voltage control devices are needed. Figure 3.19 shows the results when 4 voltage regulators are added in this feeder with 1 at feeder head and 3 at middle of the feeder, voltage violations can be resolved.

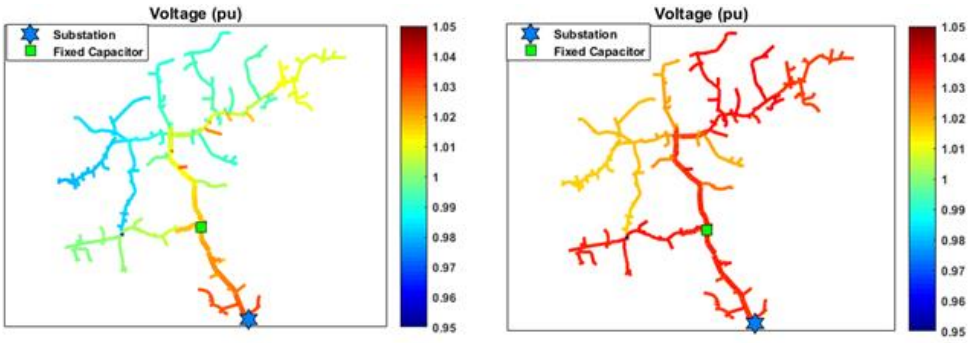


Figure 3.18: Feeder voltage profiles for base case (left) and ZNE case (right)

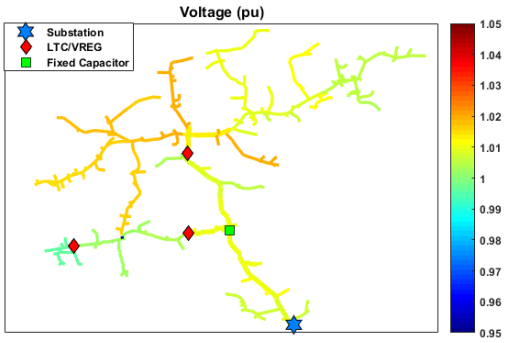


Figure 3.19: Feeder voltage profile for ZNE case with voltage regulators

C. Utility Revenue and Customer Savings

In this section, we conduct a revenue study from the utility's perspective and a cost-saving study from the customer's perspective. Due to the page limits, we only present the revenue change on residential buildings. The billing rates and policy are based on Duke Energy Progress *Net Metering for Renewable Energy Facilities Rider NM-4B* [42]. For services without ToU, feedback PV energy can compensate the building energy usage and accumulated from prior month, but the kWh usage billed will never be less than zero. For services with the ToU schedule, on-peak load can only be compensated by on-peak PV feedback. Off-peak load can be compensated by both on-/off- peak PV feedback. The feedback PV can be accumulated from prior month, but the kWh usage billed will never be less than zero. The billing rates are summarized as follows. The peak, shoulder and valley hours, are shown in Figure 3.20 and the billing rates are shown in Table 3.9.

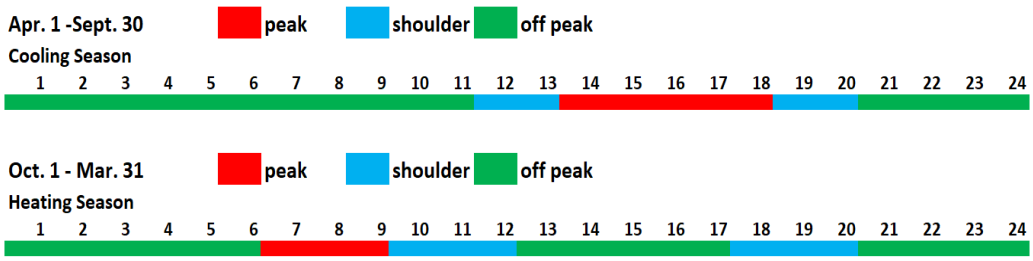


Figure 3.20: Duke Energy residential Time-of-Use rate program [42]

Table 3.9: ToU and Non-ToU billing rates [42]

	Month	Basic charge	kWh Energy charge
Non-ToU service	July – Oct.	$14.00	10.420¢ per kWh
	Nov.- June	$14.00	9.947¢ per kWh
ToU service	June-Sept.	$16.85	23.507¢ per on-peak kWh 11.996¢ per shoulder kWh 7.063¢ per off-peak kWh
	Oct. - May	$16.85	22.356¢ per on-peak kWh 11.708¢ per shoulder kWh 7.063¢ per off-peak kWh

Figure 3.21 shows the averaged revenue over the 3785 residential buildings when ZNE ready and ZNE buildings increasing from 0 to 50% out of the total buildings using Non-ToU rate.

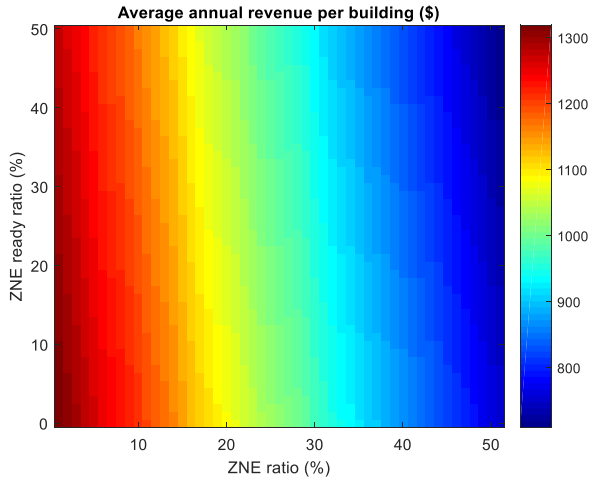


Figure 3.21: Revenue changes with increasing of ZNE and ZNE ready buildings

From the simulation results, we have the following observations:

(1) The ZNE ready buildings will reduce the revenue but the influence is rather small.

(2) When the penetration of the ZNE buildings increase from 0 to 50%, the utility's revenue will decrease from about $1320 to $710 per residential building every year.

Thus, in the rest of the studies, we only consider the penetration of ZNE buildings. Four pricing schemes are compared. Scheme 1 uses non-ToU prices and uses net meter to calculate the energy usage. Scheme 2 uses ToU price and uses net meter to calculate the energy usage. Scheme 3 uses non-ToU price but doesn't measure the feedback PV energy. Scheme 4 uses ToU price but doesn't measure the feedback PV energy. For Schemes 3 and 4, home owners will not receive payment from the utility for the backfed power. This scheme models the scenarios when utilities try to limit backfeeding power when PV penetration is high and there are no additional volt/var control devices in place to regulate voltage.

Figure 3.22 shows that when ZNE buildings increase from 0 to 50%, utility revenue will decrease almost linearly in all 4 pricing schemes. Revenue collected by the utility in the non-ToU pricing scheme will be higher than those of the ToU ones. The decreasing rate for Scheme 1 is about $11.3 per 1% ZNE buildings, i.e. if the number of the ZNE buildings increases by 1%, the utility's revenue will decrease by about $11.3 per building every year. Similarly, the decreasing rate for strategy 2, 3 and 4 are $9.4, $4.7 and $4.1 per 1% ZNE buildings, respectively. Hence, ToU prices results in less revenue per building, but they are also less sensitive to the increasing of ZNE buildings.

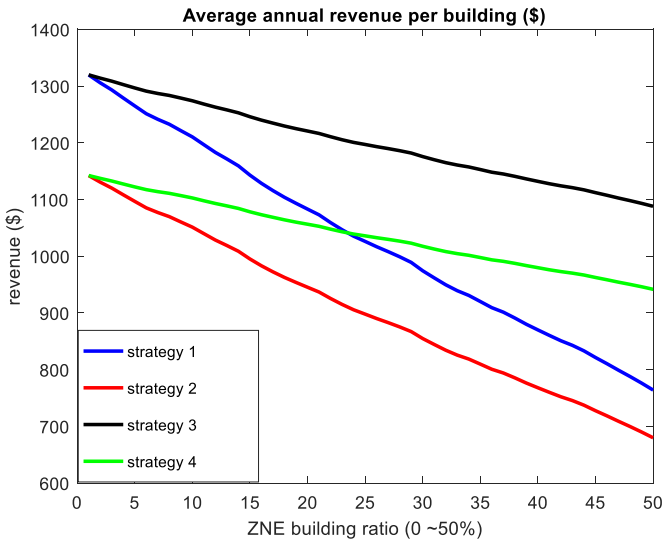


Figure 3.22: Revenue changes for different strategies

From customers' perspective, a cost-benefit study is conducted using the net present value (NPV) method under the non-ToU rate with net meter policy. The results are shown in Table 3.10.

Table 3.10: Cost-benefit study for ZNE buildings

Annual Pre-bill	Annual Post-bill	Annual Bill saving	Payback period
$1320	$139	$1181	12
A/C cost	PV cost	Total cost	
$2000	$15000	$17000	

All values are averaged over the 3785 residential buildings. The annual bill before and after ZNE is considered at $1320 and $139, respectively. Although ZNE buildings consume Zero-Net-Energy from the grid in a whole year, the annual energy bill will not be zero because of the base charge listed in Table 3.9. The annual bill saving is about $1181. The cost for HVAC and water heater loads are $2000. On average, 5kW PV panels are required for achieving ZNE based on the previous case study. The cost for the PV panel is $3/W, so the total cost for purchasing the PV

system is approximately $15000. The discount rate in NPV is 5%. The results show that payback period is approximately 12 years.

3.5 Conclusion

3.5.1 Methodology Summary

In this chapter, we developed a sequential energy disaggregation (SED) method to identify the HVAC heating and cooling energy consumptions, electric water heater and baseline load using low-resolution smart meter data. We proposed a step-by-step load disaggregation approach so that the large infrequently used loads can be removed first from the total energy consumptions first. Then, we used a curve fitting method to detect the breakpoints of heating and cooling degree days so that the yearly data can be clustered into hot, cold and mild day groups. To exclude temperature-correlated appliances such as electric fans when extracting the HVAC cooling and heating loads, a Monte Carlo average value subtracting method is applied. The validation of the algorithm performance using actual meter data showed that: 1) disaggregation errors are within 5% for 98% buildings, 2) algorithm performance is consistent for 15-, 30-, and 60- minute smart meter data, and 3) because the SED algorithm will not detect the appliance ON/OFF status, customer privacy is protected.

Based on the SED algorithm, we presented a customer oriented, data-driven planning method for quantifying the impact of ZNE buildings on feeder operation, customer savings, and utility revenue. We use sequential energy disaggregation algorithm to extract the HVAC load, the water heater load, and the baseload from the lo- resolution smart meter data. Then, we build “ZNE ready” and “ZNE” load profile databases to conduct feeder level continuous power flow study on an annual basis. We also study the utility revenue and customer savings based on Duke Energy rates. The results show that retrofitting a building to ZNE will reduce the customer energy bill and

the average payback period is approximately 12 years. Based on the revenue study, high penetration of ZNE buildings will significantly reduce the utility revenue and cause increasing voltage regulation issues. Thus, different rate structures might be required to redesign the basic charge, encourage self-consumption, and encourage smart inverter-based reactive power regulation.

3.5.2 Summary of Synthetic Load Database

Using the SED algorithm developed, we can not only extract the synthetic load data of HVAC load, electrical water heater load and the baseline load, we also build up a database of ZNE case, ZNE ready case and ZNE plus EV cases. These load data facilitate the study of distribution system planning, especially with the consideration of future high PV and EV penetration scenarios. Figure 3.23 shows the SED and ZNE tool we developed to generate the synthetic load data. Figure 3.24 illustrates an example of the synthetic load data for different cases for a real residential building in North Carolina Area.

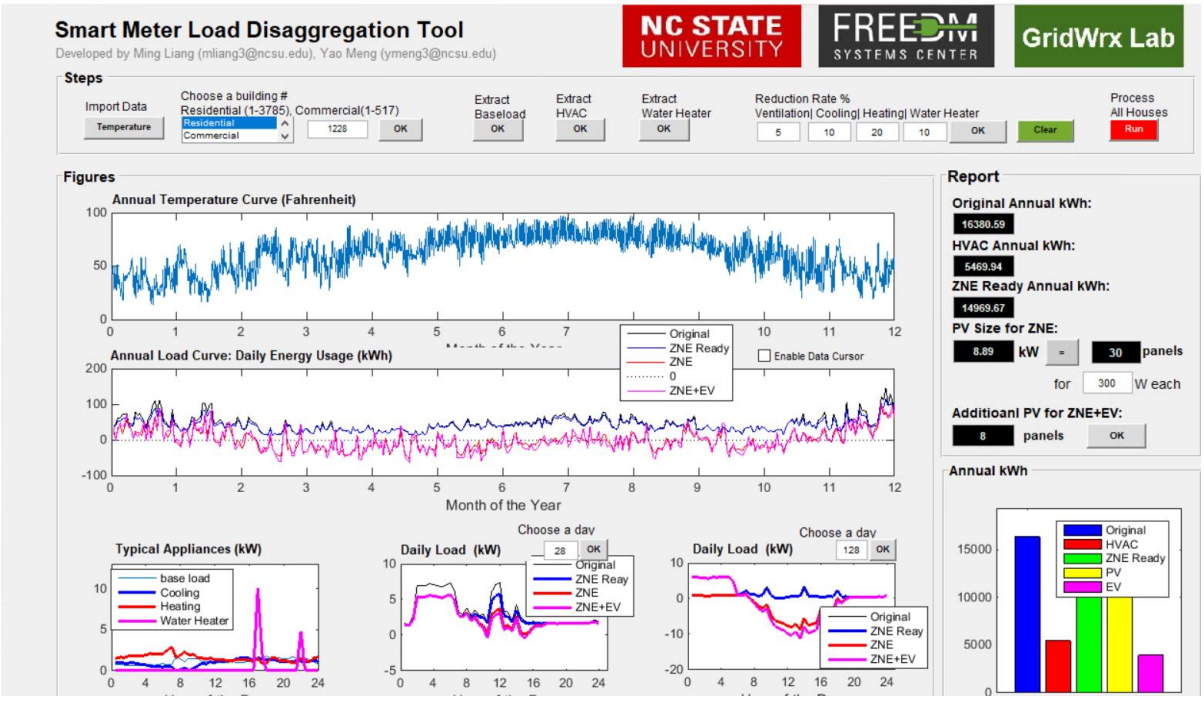


Figure 3.23: SED and ZNE Matlab based tool

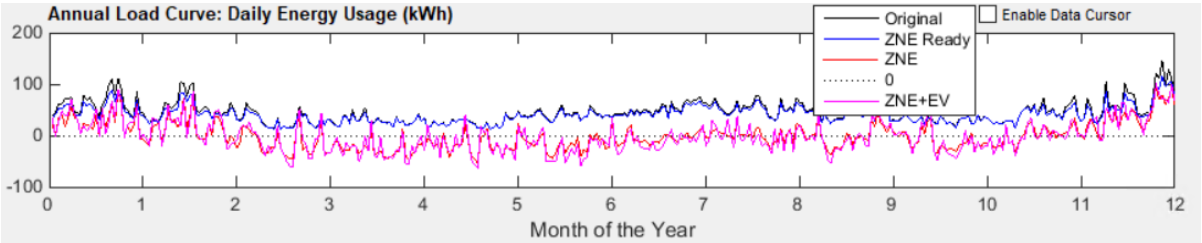


Figure 3.24: Synthetic load data for ZNE cases

Finally, we enclose the synthetic load database with the generated synthetic feeders from Chapter 2. A time series power flow case study is conduct as shown in Figure 3.25. With the FeederGAN tool and the load generation method, we can automatically generate as many synthetic feeders as we want, and use them to test and validate different algorithms and strategies.

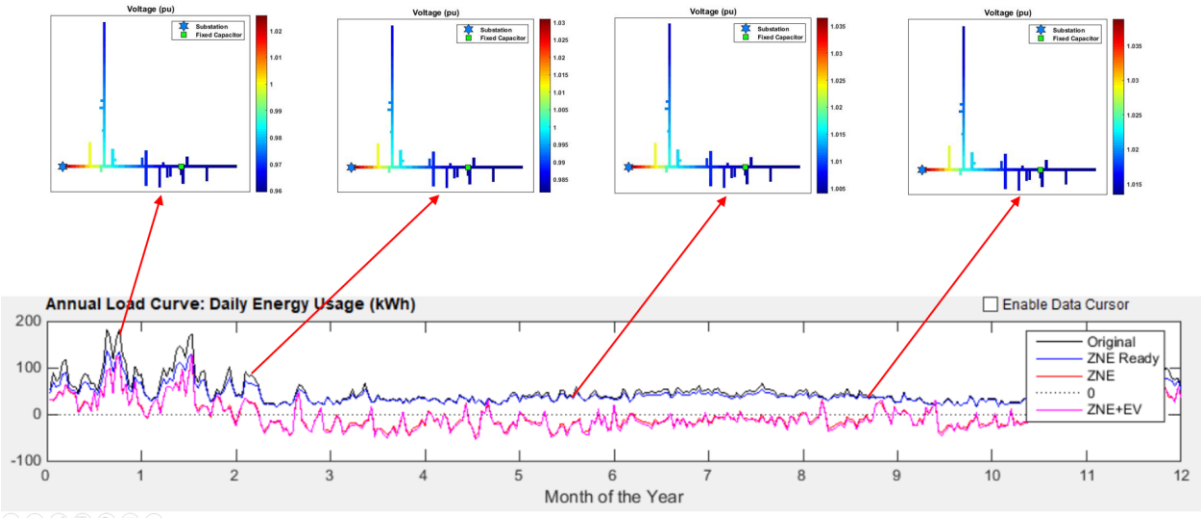


Figure 3.25: Combined time series case study with a generated feeder

CHAPTER 4 SUMMARY AND FUTURE WORK

Faced with the great demand for high fidelity, realistic synthetic testing systems, we propose a novel framework to generate synthetic feeder topology, line segment attributes and synthetic load data. The framework can ingest real distribution feeders and generate a large amount of synthetic feeders automatically. At the meanwhile, load disaggregation method is used to generate synthetic load data in different zero-net energy (ZNE) cases. Hence, a large amount of comprehensive synthetic distribution feeders are generated for distribution research and study. In this chapter, we briefly summarize the related work and the future steps.

4.1 Summary of Previous and Current Research

4.1.1 Smart Meter Load Disaggregation

Traditional non-intrusive load monitoring (NILM) methods are effective for load disaggregation using high resolution smart meter data, e.g. data collected by power quality meters of several hertz or thousand hertz. These methods either directly capture the signal changing values, e.g. signal-processing-based methods, or indirectly encode the transition states, e.g. Markov based methods, which are very sensitive to the data resolution/granularity. However, smart meter data collected and stored by utilities are normally 15-, 30- or 60-minute in granularity, making most NILM methods ineffective. Therefore, in this research work, a novel sequential energy disaggregation (SED) algorithm is proposed to extract heating, ventilation, and air conditioning (HVAC) energy consumptions from residential and small commercial building loads using 30 min smart meter data. Large, infrequently used loads are first detected and removed from the total building energy consumption. Then, base energy consumption curves, defined as the energy consumption without heating and cooling loads, are identified using the mild-day method. After that, the heating and cooling loads are extracted using an average value subtracting method.

Simulation results with real data show that the proposed SED method is computationally efficient, simple to implement and robust in performance across different types of buildings.

Based on the load disaggregation results, case studies are conducted to generate synthetic load data in ZNE case, ZNE ready case and ZNE plus EV case. Hence, the generate synthetic feeder can be used not only to conduct research under current normal conditions, but also can be used to evaluate the future scenarios, such as high PV and EV penetration situations.

4.1.2 Synthetic Feeder Generation

In order to satisfy the research demand for realistic synthetic feeders, many previous works have been done to either manually or automatically generate synthetic distribution feeders. For instance, there are works aimed at varying the network topology, adjusting the placement and parameters of apparatus (e.g., adding new devices, removing old ones, retrofitting, etc.), and changing user patterns in an actual utility feeder model. More recent works are using automatic algorithms to generate synthetic feeders based on road maps and population survey data.

Instead of using the geography-based method, this research presents a deep-learning-based method (named as FeederGAN) to generate feeder topology and line segment attributes without the knowledge of geographical information, such as road maps or global positioning system (GPS) data. First, real distribution feeders are modeled as directed graphs, where the devices (such as a cable) is represented as a node in the graph, and the directed edges are only used as an indicator for the graph direction from the feeder head to each load node. Then, feeder connectivity is modeled by graph adjacency matrix and device characteristics (such as phase connection, current ratings) are modeled as node attributes. Next, we use a GAN-based framework together with GCN to train the model and generate adjacency matrix and attribute matrix. Graph theory-based method is used to solve the node ordering issue. Thus, the generated adjacency and attribute matrix can be

used to reconstruct the feeder topology and attributes in its actual values. Finally, the quality of the generated feeder is evaluated by the empirical statistics of real feeders in aspects of topology and line segment attributes. The result turns out that the generated feeders resemble the actual feeder in both topology and attributes verified by visual inspection and by empirical statistics.

4.2 Vision and Plan of Future Work

Since the synthetic feeder generation framework includes two parts of feeder and data generation. The next steps are aimed to extend the current research towards generating synthetic time series load data and refine the feeder generation framework, such as device automatic placement.

4.2.1 Time Series Load Generation

The synthetic load database is obtained via load disaggregation, which requires that there should be enough load data at hand. Sometimes, researchers may not have much real load data, just like the situation where they don't have access to realistic feeders. Inspired by the GAN framework to generate synthetic feeder topology and attributes. We can then use a GAN-based framework to generate synthetic load data. Hence, once the model trained, researchers can directly use the model to generate lots of load data for their diversified research purpose. Since the load data in most situations are time series data, the goal of the GAN is to generate time series. Hence, recurrent neural network (RNN) based GAN can be used to generate synthetic load data that resemble implicit load patterns.

4.2.2 Automated Device Placement

The current version of FeederGAN only generates line segments, i.e. cables or overhead lines. To refine the feeders generated, we need to add devices such as sectionalizer, reclosers and voltage management equipment, such as regulator and capacitor banks. Hence, the next step is

going to use graph theory and learning-based method to automatically place these devices such that the circuit satisfies the reliability requirement and voltage along the circuit are within the security range, such as the ANSI standard.

REFERENCES

[1] W. H. Kersting, “Radial distribution test feeders,” in *2001 IEEE Power Engineering Society Winter Meeting. Conference Proceedings (Cat. No.01CH37194)*, Jan. 2001, vol. 2, pp. 908–912 vol.2, doi: 10.1109/PESW.2001.916993.

[2] “Resources | PES Test Feeder.” https://site.ieee.org/pes-testfeeders/resources/ (accessed Mar. 17, 2020).

[3] “Feeder Taxonomy - GridLAB-D Wiki.” http://gridlab-d.shoutwiki.com/wiki/Feeder_Taxonomy (accessed Mar. 17, 2020).

[4] “Electric Grid Test Cases.” https://electricgrids.engr.tamu.edu/electric-grid-test-cases/ (accessed Mar. 17, 2020).

[5] “EPRI | Smart Grid Resource Center > Simulation Tool – OpenDSS.” https://smartgrid.epri.com/SimulationTool.aspx (accessed Mar. 07, 2020).

[6] E.-E. E. Portal, “The basics of primary distribution circuits (substation branches, feeders...),” *EEP - Electrical Engineering Portal*, Aug. 28, 2017. https://electrical-engineering-portal.com/primary-distribution-circuits (accessed Mar. 17, 2020).

[7] C. McEntee, D. Mulcahy, J. Wang, X. Zhu, and N. Lu, “A VSM-Based DER Dispatch MINLP for Volt-VAR Control in Unbalanced Power Distribution Systems,” in *2019 IEEE Power Energy Society General Meeting (PESGM)*, Aug. 2019, pp. 1–5, doi: 10.1109/PESGM40551.2019.8973721.

[8] N. Samaan *et al.*, “Combined Transmission and Distribution Test System to Study High Penetration of Distributed Solar Generation,” in *2018 IEEE/PES Transmission and Distribution Conference and Exposition (T D)*, Apr. 2018, pp. 1–9, doi: 10.1109/TDC.2018.8440238.

[9] Y. Meng, M. Liang, J. F. DeCarolis, and N. Lu, “Design of Energy Storage Friendly Regulation Signals using Empirical Mode Decomposition,” in *2019 IEEE Power Energy Society General Meeting (PESGM)*, Aug. 2019, pp. 1–5, doi: 10.1109/PESGM40551.2019.8973757.

[10] Y. Meng, M. Liang, and N. Lu, “A Cost Benefit Study of using Energy Storage to Provide Regulation Services,” in *2019 IEEE Power Energy Society Innovative Smart Grid Technologies Conference (ISGT)*, Feb. 2019, pp. 1–5, doi: 10.1109/ISGT.2019.8791664.

[11] M. Liang, W. Li, J. Yu, and L. Shi, “Kernel-based electric vehicle charging load modeling with improved latin hypercube sampling,” in *2015 IEEE Power Energy Society General Meeting*, Jul. 2015, pp. 1–5, doi: 10.1109/PESGM.2015.7285758.

[12] M. Zhang, L. Wang, and C. Guo, “Studying the influence of coordinated ev charging on power system operating risk,” in *2017 IEEE Innovative Smart Grid Technologies - Asia (ISGT-Asia)*, Dec. 2017, pp. 1–8, doi: 10.1109/ISGT-Asia.2017.8378351.

[13] E. Sala, K. Kampouropoulos, M. D. Prieto, and L. Romeral, “Disaggregation of HVAC load profiles for the monitoring of individual equipment,” in *2016 IEEE 21st International Conference on Emerging Technologies and Factory Automation (ETFA)*, Sep. 2016, pp. 1–6, doi: 10.1109/ETFA.2016.7733568.

[14] Y.-H. Lin and M.-S. Tsai, “Development of an Improved Time–Frequency Analysis-Based Nonintrusive Load Monitor for Load Demand Identification,” *IEEE Trans. Instrum. Meas.*, vol. 63, no. 6, pp. 1470–1483, Jun. 2014, doi: 10.1109/TIM.2013.2289700.

[15] T. D. Huang, W. Wang, and K. Lian, “A New Power Signature for Nonintrusive Appliance Load Monitoring,” *IEEE Trans. Smart Grid*, vol. 6, no. 4, pp. 1994–1995, Jul. 2015, doi: 10.1109/TSG.2015.2415456.

[16] S. McLaughlin, P. McDaniel, and W. Aiello, “Protecting Consumer Privacy from Electric Load Monitoring,” in *Proceedings of the 18th ACM Conference on Computer and Communications Security*, New York, NY, USA, 2011, pp. 87–98, doi: 10.1145/2046707.2046720.

[17] A. Albert and R. Rajagopal, “Thermal profiling of residential energy use,” in *2015 IEEE Power Energy Society General Meeting*, Jul. 2015, pp. 1–1, doi: 10.1109/PESGM.2015.7285888.

[18] Department of Technology Management for Innovation, Graduate School of Engineering, The University of Tokyo, 214-3,7-3-1 Hongo, Bunkyo, Tokyo, Japan, 113-8656, H. Chin, K. Tanaka, and A. Rikiya, “Household electricity load disaggregation based on low-resolution smart meter data,” *Int. J. Smart Grid Clean Energy*, 2016, doi: 10.12720/sgce.5.3.188-195.

[19] Y. Ji, P. Xu, and Y. Ye, “HVAC terminal hourly end-use disaggregation in commercial buildings with Fourier series model,” *Energy Build.*, vol. 97, pp. 33–46, Jun. 2015, doi: 10.1016/j.enbuild.2015.03.048.

[20] "Monthly Energy Review – December 2017," U.S. Energy Information Administration, p. 246, 2017.

[21] R. Morell, "Grid Integration of Zero Net Energy Communities," EPRI, p. 200, Sept. 2016.

[22] W. Wu, H. M. Skye, and P. A. Domanski, "Selecting HVAC systems to achieve comfortable and cost-effective residential net-zero energy buildings," *Appl. Energy*, vol. 212, pp. 577–591, Feb. 2018, doi: 10.1016/j.apenergy.2017.12.046.

[23] K. P. Schneider, Y. Chen, D. Engle, and D. Chassin, "A Taxonomy of North American radial distribution feeders," in *2009 IEEE Power Energy Society General Meeting*, Jul. 2009, pp. 1–6, doi: 10.1109/PES.2009.5275900.

[24] "SMART-DS: Synthetic Models for Advanced, Realistic Testing: Distribution Systems and Scenarios | Grid Modernization | NREL." https://www.nrel.gov/grid/smart-ds.html (accessed Mar. 07, 2020).

[25] S. S. Saha, E. Schweitzer, A. Scaglione, and N. G. Johnson, "A Framework for Generating Synthetic Distribution Feeders using OpenStreetMap," *ArXiv191007673 Cs Eess*, Oct. 2019, Accessed: Mar. 06, 2020. [Online]. Available: http://arxiv.org/abs/1910.07673.

[26] C. Mateo Domingo, T. Gomez San Roman, A. Sanchez-Miralles, J. P. Peco Gonzalez, and A. Candela Martinez, "A Reference Network Model for Large-Scale Distribution Planning With Automatic Street Map Generation," *IEEE Trans. Power Syst.*, vol. 26, no. 1, pp. 190–197, Feb. 2011, doi: 10.1109/TPWRS.2010.2052077.

[27] A. B. Birchfield, T. Xu, K. M. Gegner, K. S. Shetye, and T. J. Overbye, "Grid Structural Characteristics as Validation Criteria for Synthetic Networks," *IEEE Trans. Power Syst.*, vol. 32, no. 4, pp. 3258–3265, Jul. 2017, doi: 10.1109/TPWRS.2016.2616385.

[28] A. B. Birchfield *et al.*, "A Metric-Based Validation Process to Assess the Realism of Synthetic Power Grids," *Energies*, vol. 10, no. 8, p. 1233, Aug. 2017, doi: 10.3390/en10081233.

[29] E. Schweitzer, A. Scaglione, A. Monti, and G. A. Pagani, "Automated Generation Algorithm for Synthetic Medium Voltage Radial Distribution Systems," *IEEE J. Emerg. Sel. Top. Circuits Syst.*, vol. 7, no. 2, pp. 271–284, Jun. 2017, doi: 10.1109/JETCAS.2017.2682934.

[30] T. N. Kipf and M. Welling, “Semi-Supervised Classification with Graph Convolutional Networks,” *ArXiv160902907 Cs Stat*, Feb. 2017, Accessed: Feb. 11, 2020. [Online]. Available: http://arxiv.org/abs/1609.02907.

[31] S. Fan and B. Huang, “Labeled Graph Generative Adversarial Networks,” *ArXiv190603220 Cs Stat*, Jun. 2019, Accessed: Nov. 28, 2019. [Online]. Available: http://arxiv.org/abs/1906.03220.

[32] J. You, R. Ying, X. Ren, W. L. Hamilton, and J. Leskovec, “GraphRNN: Generating Realistic Graphs with Deep Auto-regressive Models,” *ArXiv180208773 Cs*, Jun. 2018, Accessed: Mar. 07, 2020. [Online]. Available: http://arxiv.org/abs/1802.08773.

[33] M. Liang, Y. Meng, N. Lu, D. Lubkeman, and A. Kling, “HVAC load Disaggregation using Low-resolution Smart Meter Data,” in *2019 IEEE Power Energy Society Innovative Smart Grid Technologies Conference (ISGT)*, Feb. 2019, pp. 1–5, doi: 10.1109/ISGT.2019.8791578.

[34] K. Basu, V. Debusschere, S. Bacha, U. Maulik, and S. Bondyopadhyay, “Nonintrusive Load Monitoring: A Temporal Multilabel Classification Approach,” *IEEE Trans. Ind. Inform.*, vol. 11, no. 1, pp. 262–270, Feb. 2015, doi: 10.1109/TII.2014.2361288.

[35] “PECAN STREET,” *Pecan Street Inc.* https://www.pecanstreet.org/ (accessed Apr. 15, 2020).

[36] “Data Tools | Climate Data Online (CDO) | National Climatic Data Center (NCDC).” https://www.ncdc.noaa.gov/cdo-web/datatools. (accessed Apr. 15, 2020).

[37] “Energy in buildings,” *OpenLearn*. https://www.open.edu/openlearn/nature-environment/energy-buildings/content-section-0 (accessed Apr. 15, 2020).

[38] “Standard 55 – Thermal Environmental Conditions for Human Occupancy.” https://www.ashrae.org/technical-resources/bookstore/standard-55-thermal-environmental-conditions-for-human-occupancy (accessed Apr. 15, 2020).

[39] S. Makonin and F. Popowich, “Nonintrusive load monitoring (NILM) performance evaluation,” *Energy Effic.*, vol. 8, no. 4, pp. 809–814, Jul. 2015, doi: 10.1007/s12053-014-9306-2.

[40] M. Liang, J. Wang, Y. Meng, N. Lu, D. Lubkeman, and A. Kling, “Impacts of Zero Net Energy Buildings on Customer Energy Savings and Distribution System Planning,” in *2019 IEEE Innovative Smart Grid Technologies - Asia (ISGT Asia)*, May 2019, pp. 2561–2565, doi: 10.1109/ISGT-Asia.2019.8881425.

[41] J. Wang *et al.*, "A Two-Step Load Disaggregation Algorithm for Quasi-static Time-series Analysis on Actual Distribution Feeders," in *2018 IEEE Power Energy Society General Meeting (PESGM)*, Aug. 2018, pp. 1–5, doi: 10.1109/PESGM.2018.8586131.

[42] "Rate Information," *Duke Energy*. https://www.duke-energy.com/Home/Billing/Rates (accessed Apr. 15, 2020).

[43] M. Liang, Y. Meng, J. Wang, D. Lubkeman, and N. Lu, "FeederGAN: Synthetic Feeder Generation via Deep Graph Adversarial Nets," *ArXiv200401407 Cs Eess*, Apr. 2020, Accessed: Apr. 15, 2020. [Online]. Available: http://arxiv.org/abs/2004.01407.

[44] B. Xu, N. Wang, T. Chen, and M. Li, "Empirical Evaluation of Rectified Activations in Convolutional Network," *ArXiv150500853 Cs Stat*, Nov. 2015, Accessed: Mar. 07, 2020. [Online]. Available: http://arxiv.org/abs/1505.00853.

[45] Z. Qin and D. Kim, "Rethinking Softmax with Cross-Entropy: Neural Network Classifier as Mutual Information Estimator," *ArXiv191110688 Cs Stat*, Jan. 2020, Accessed: Mar. 07, 2020. [Online]. Available: http://arxiv.org/abs/1911.10688.

[46] I. Goodfellow *et al.*, "Generative Adversarial Nets," in *Advances in Neural Information Processing Systems 27*, Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2014, pp. 2672–2680.

[47] M. Arjovsky, S. Chintala, and L. Bottou, "Wasserstein GAN," *ArXiv170107875 Cs Stat*, Dec. 2017, Accessed: Mar. 10, 2020. [Online]. Available: http://arxiv.org/abs/1701.07875.

[48] S. Ruder, "An overview of gradient descent optimization algorithms," *ArXiv160904747 Cs*, Jun. 2017, Accessed: Mar. 07, 2020. [Online]. Available: http://arxiv.org/abs/1609.04747.

[49] H. Thanh-Tung and T. Tran, "On Catastrophic Forgetting and Mode Collapse in Generative Adversarial Networks," *ArXiv180704015 Cs Stat*, Oct. 2019, Accessed: Mar. 22, 2020. [Online]. Available: http://arxiv.org/abs/1807.04015.