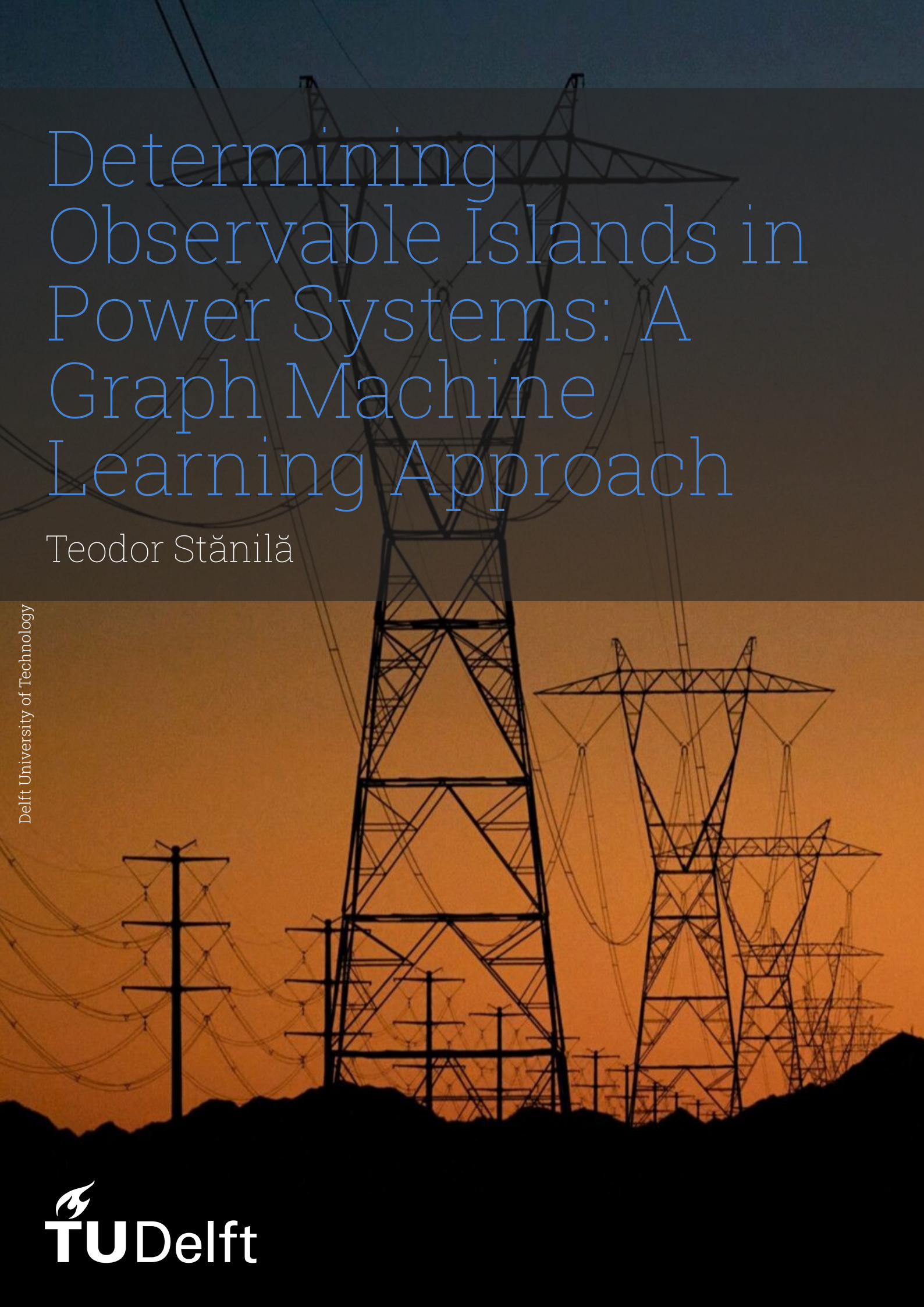




Determining Observable Islands in Power Systems: A Graph Machine Learning Approach

Teodor Stănilă

Delft University of Technology

Determining Observable Islands in Power Systems: A Graph Machine Learning Approach

by

Teodor Stănilă

University Advisor: Dr. Martin Skrodzki
External Advisors: Dr. Jerry Guo, Betül Mamudi
Thesis Committee: Dr. Martin Skrodzki, Dr. Elvin Isufi, Dr. Michael Weinmann, Dr. Jerry Guo, Betül Mamudi
Research Group: Computer Graphics & Visualization
Project Duration: February, 2026 - September, 2026
Faculty: Electrical Engineering, Mathematics and Computer Science

Abstract

State estimation (SE) is an important tool in power system management, alleviating the electrical traffic congestion problem by estimating the voltage angle and magnitude at every bus using an incomplete set of measurements.

Observability analysis represents a sub-problem in the SE pipeline. It determines if the system is fully observable. If the network is not fully observable, many algorithms will split it into observable subnetworks.

Machine learning (ML) methods have been sparsely previously used to solve observability analysis, as they tend to focus on full SE. Prior ML work solves the problem through a binary result of observable or unobservable, without dividing the grid into islands. Moreover, the data used in training those models, or the generation algorithm that created it were never made public.

We present an algorithm that randomly assigns measurements to components of a given network and computes its observable islands used for generating a training dataset. We then propose a graph machine learning architecture that can identify observable islands in a power grid from its topology and measurements. Our architecture follows the variational autoencoder (VGAE) structure, consisting of three graph convolutional layers and a graph attention layer.

The architecture was trained and evaluated on the IEEE 14, 30, 39, 57 and 118-bus cases with consistent results across all cases, alongside a general model trained on a dataset consisting in a combination of all the network sizes.

Moreover, a direct comparison with the existing literature shows that our solution produces fairly accurate results, with 5 out of 6 test cases being correctly identified as fully observable or unobservable.

Contents

Abstract	i
1 Introduction	1
2 Background and Context	3
2.1 Alternating Current Power Networks	3
2.2 State Estimation	4
2.3 Graphs	5
2.3.1 Trees and Forests	5
2.4 Network Observability and Observable Islands	6
2.4.1 Full Observability Example	8
2.4.2 Observable Islands Example	9
2.5 Neural Networks	10
2.6 Convolutional Neural Networks	12
2.7 Graph Neural Networks	12
2.7.1 Graph Convolutional Networks	12
2.7.2 Graph Attention Networks	13
2.7.3 GCN-SE	14
3 Related Work	15
3.1 Numerical Methods	15
3.1.1 Numerical Observability Analysis Examples	15
3.2 Topological methods	17
3.3 Hybrid methods	18
3.4 Machine Learning Methods	19
3.4.1 Observability Analysis Solutions	19
3.4.2 State Estimation Solutions	19
4 Methodology	20
4.1 Dataset	20
4.1.1 Generating Measured Networks	20
4.1.2 Generating Ground Truth	21
4.2 Baseline Algorithm	21
4.3 Model Architecture - Variational Graph Autoencoder	23
5 Evaluation & Results	26
5.1 Experimental Setup	26
5.2 Results	27
5.2.1 Ablation Results	27
5.2.2 Baseline Comparison - Fully Observable Networks	29
5.2.3 Baseline Comparison - Unobservable Networks	29
5.2.4 Performance across different network sizes	30
5.2.5 Baseline Comparison - Mixture of Networks	30
5.2.6 Comparison with Existing Literature	31
6 Discussion	32
6.1 Discussion about Results	32
6.2 Limitations & Future Work	33
6.3 Societal Impact	34
7 Conclusion	36

References	37
A AI Use Acknowledgement	40
A.1 Writing	40
A.2 Coding	40

1

Introduction

The modern power supply chain operates as a continuous system organized into three basic parts: generation, transmission and distribution. The majority of the energy comes from generating facilities, which usually provide electricity at lower to medium voltage levels, around 10-34kV [14]. Before routing it through the high-voltage transmission network, a system of structural towers and conductor cables that connects various demand centers across the country, this energy is transported through step-up transformers to increase voltage magnitudes. Step-down transformers process the high-voltage current once it reaches them in order to lower the voltage to operational levels. The distribution network then transmits lower-voltage current over shorter distances to industrial, commercial and residential end-users.

Traffic in the supply chain is dynamic, it continuously changes depending on the consumers' demand. More demanding regions require more electrical traffic, increasing the load on the electrical infrastructure. Every element in the power grid has a finite load limitation, and when the demand is close to reaching this physical limit, transmission becomes slower and constrains the network. When the existing infrastructure cannot handle generating or delivering enough electricity to meet demand, a phenomenon known as transmission congestion appears [16]. Under these conditions, normal electrical operations are disrupted as users queue up to wait for sufficient power to become available.

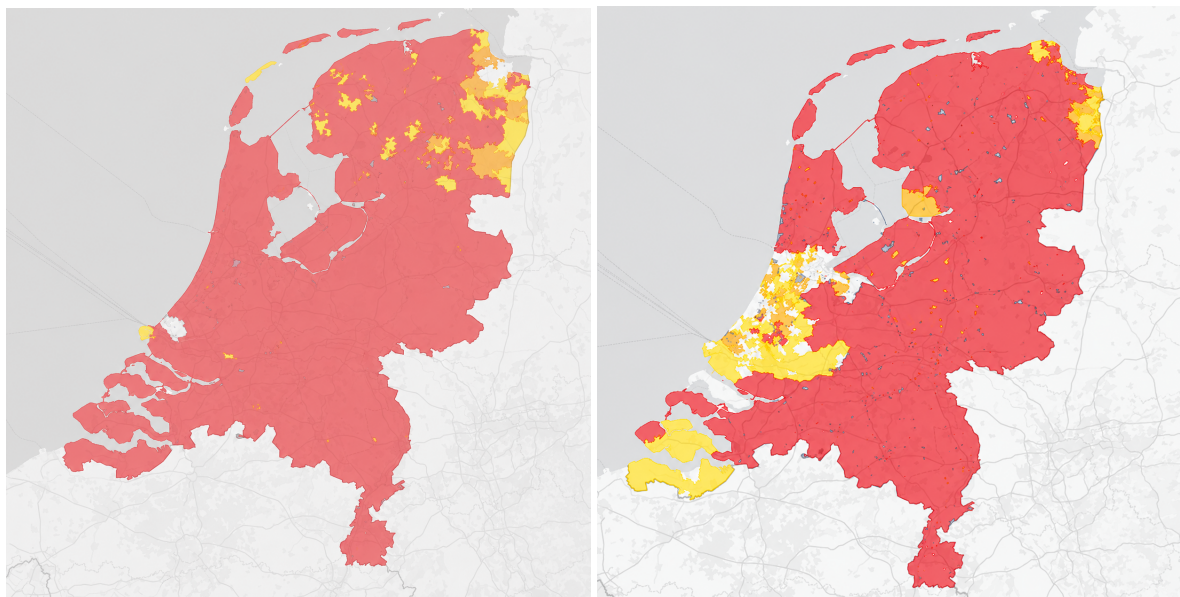


Figure 1.1: Netherlands Congestion Maps (Left = consumption | Right = feed-in), cited from <https://capaciteitskaart.netbeheernederland.nl/capaciteitskaart/totaal/afname>

One clear example of transmission congestion happening currently is the Netherlands, where approximately 12.000 requests for electrical consumption and another 8.000 for feed-in capacity are queued for grid access [30]. Feed-in requests come from new electricity producers, seeking to inject power into the network. The capacity maps published by Netbeheer Nederland [29], seen in figure 1.1, illustrate the current traffic stuck in the Dutch transmission network, showing the regions where existing infrastructure cannot handle additional demand or generation. Red and orange spots on the map denote insufficient transport capacity that require queuing, while the yellow portions do not, being generally more free.

A suitable way of mitigating transmission congestion is the use of state estimation (SE) in power grids [1]. SE provides an estimate of the state of the network by combining available measurements and state variables, including bus voltage magnitudes, line flows and voltage angles. This information enables grid operators to monitor conditions of the networks and perform corrections so that electrical flows are redistributed and congestion is reduced. However, state estimation is a complex process with more than one step, one of them deciding whether the grid has enough measurement information in order to determine its full state. This step is known as observability analysis. In its simplest form, observability analysis outputs a binary response of whether the entire network is observable or not [1].

As most practical networks are rarely fully observable due to limited measurement placement, a localized method emerged: the identification of observable islands. This method divides the network into smaller and fully observable regions, allowing state estimation and the corrective actions to be performed locally, which aids in congestion mitigation [35]. There are many methods that aim to solve this problem, mainly categorized into four categories: numerical, topological, hybrid and machine learning methods.

The first three categories have ample research behind, while the machine learning methods currently output a binary result regarding the whole network's observability without splitting it into observable islands. These solutions use as input the presence or absence of a specific measurement, never making use of the topological information inherently present in a power network. The presented models also lack deeper generalizability, being trained and evaluated only on the IEEE 5-bus and 14-bus test cases. Moreover, the data used for their training was not publicized, meaning that further model iterations would also require generating a new dataset from scratch.

With the limitations of existing machine learning solutions in mind, this research project is guided by the following research questions:

- **RQ1** How can training datasets be generated for a graph machine learning model to identify observable islands in power systems?
- **RQ2** Can a graph machine learning architecture identify observable islands in a power network based on its topology and available measurement configuration?
- **RQ3** Can a graph machine learning model generalize its performance in identifying observable islands across power networks of different sizes and topologies?

As such, for this thesis' contributions to the broader data science and artificial intelligence field, we create a dataset generation script that randomly assigns measurements to a given power network's buses, lines and transformers, alongside a stochastic algorithm that tests different possible island configurations for the ground truth to find a local maximum. Finally, we derive a graph machine learning architecture that takes a measured grid as input and identifies the observable islands present in it. For generalizability purposes, the showcased architecture is evaluated on differently sized power grids to see if the results remain consistent. A full pipeline of dataset generation, model training and evaluation is thus the end product of this project.

2

Background and Context

In this section we will go through all the necessary background knowledge for the proposed method, including alternating power networks, graph theory, state estimation, observability analysis, neural networks, convolutional neural networks and graph neural networks.

2.1. Alternating Current Power Networks

Alternating current (AC) is current that periodically changes its direction of flow, unlike direct current (DC), which flows in a single direction. Using transformers, AC has the capability of adjusting its voltage, which is why it is mostly found in households and businesses. It can be stepped up for efficient transmission over high-voltage lines and then stepped down before reaching the end user.

The periodic reversal is best described by a sinusoidal waveform, since a sine function captures a value that constantly rises to a highest point and falls back to zero, the same way voltage and current change over time in an AC circuit. Beyond the standard measurements of voltage, current intensity and power found in any power system, AC quantities are described by additional parameters:

- **Amplitude** (I_m, V_m) the peak value reached by the waveform
- **Frequency** (f) how many full cycles the function completes per second, measured in hertz (Hz)
- **Phase angle** (θ) the relative time shift between two waveforms, typically voltage and current, measured in degrees or radians

Because of the periodic shift, the instantaneous current and voltage values in an AC circuit are obtained from the following equations:

$$i(t) = I_m \sin(\omega t + \theta_i) \quad (2.1)$$

$$V(t) = V_m \sin(\omega t + \theta_v) \quad (2.2)$$

where I_m and V_m are the highest values of the current and voltage functions and ω is the angular frequency, obtained from:

$$\omega = 2\pi f \quad (2.3)$$

Lastly, θ_i and θ_v are the phase angles of the current and voltage.

Another important quantity is the root-mean-square (RMS) value, which represents the equivalent direct current (DC) voltage or current that would produce the same power dissipation in a resistive environment:

$$I_{RMS} = \frac{I_M}{\sqrt{2}} \quad (2.4)$$

$$V_{RMS} = \frac{V_M}{\sqrt{2}} \quad (2.5)$$

These RMS values are used throughout power flow analysis instead of the instantaneous ones because averaging the voltage and current waveform over a full cycle is zero, as the positives and negatives

cancel each other out, making it uninformative. The RMS value avoids this by squaring the instantaneous values, averaging them over a cycle and then taking the square root, which removes the sign cancellation problem and returns a meaningful and constant number. This way, AC power can be computed in the same form as DC power.

An AC power network is composed of buses connected by branches. The nodes of the network are busses, the points to which generation, load or other equipment connect to the system. Branches represent the physical lines and transformers that carry power between them. Generally busses are divided into three categories:

- **Slack buses** which fix the voltage magnitude and phase angle reference for the system
- **PV buses** at which voltage magnitude and active power are specified, corresponding to generators
- **PQ buses** at which active and reactive power are specified, corresponding to loads

An n-bus network is simply a power network composed of n buses, determining the dimensionality of our problem.

2.2. State Estimation

State estimation is an essential part in modern power system operation. Its objective is to determine the current state of the power network, i. e. the voltage magnitudes and phase angles at each bus, from a limited set of measurements and the network's topology. Instead of calculating the unmeasured locations, the state of the unmeasured locations can be inferred from the available data using relationships imposed by Kirchhoff's laws.

Measurements are categorized by their location in the network. Line measurements consist of power flow between the two ends of a line. Bus measurements include voltage magnitude and the net power entering or leaving it. Power itself is split into two components: active power, the actual power dissipated by the load, and reactive power, power that is alternately stored and released by reactive elements, for example capacitors and inductors.

In an AC setting, power is described by the complex formula:

$$S = P + jQ \quad (2.6)$$

where P is the active power and Q is the reactive power. This formulation captures both components, similar to a complex number.

For an N-bus system, the active and reactive power injection at bus i are given by:

$$P_i = V_i \sum_{j=1}^N V_j (G_{ij} \cos \theta_{ij} + B_{ij} \sin \theta_{ij}) \quad (2.7)$$

$$Q_i = V_i \sum_{j=1}^N V_j (G_{ij} \sin \theta_{ij} - B_{ij} \cos \theta_{ij}) \quad (2.8)$$

where $\theta_{ij} = \theta_i - \theta_j$ and G_{ij} , B_{ij} are the conductance and susceptance of the line connecting i and j , determined entirely by its physical impedance. For nodes that are not directly connected, G_{ij} and B_{ij} are zero, meaning that even though the full injection equations sum over all N buses in the network, only the terms corresponding to a bus's direct neighbours and itself have non-zero values.

The only unknowns in these equations are then the voltage magnitudes V_i , $V_j, \dots$ and phase angles θ_i , $\theta_j, \dots$ at each bus. Determining the full state of the system is then equivalent to estimating voltage magnitude and phase angle values at every bus. This relationship is captured by the system state vector:

$$x = [\theta_2, \theta_3, \dots, \theta_N, V_1, V_2, \dots, V_N] \quad (2.9)$$

with θ_1 conventionally fixed at 0 as the reference angle, since phase angles have meaning only relative to a common reference.

2.3. Graphs

Graphs are data structures that contain information about how objects are related. They consist of a set of objects, called nodes, and a set of links, called edges. Visually, the nodes are drawn as points, while the edges as the lines connecting them. Mathematically, a graph can be interpreted as:

$$G = (V, E) \quad (2.10)$$

, where G represents the full graph, V is the set of vertices and E is the set of edges.

A graph can be represented using multiple data structures, the adjacency matrix and the edge list being two of the most common ones. The adjacency matrix, denoted A is an $N \times N$ dimensional matrix, where $N = |V|$ is the number of vertices in the graph. Each of its values is filled depending on the existence of a connection between the nodes:

- $A[i][j] == 1$ if there is an edge between vertex i and vertex j
- $A[i][j] = 0$ if there is no edge between vertex i and vertex j

The edge list is a more compact representation, storing all pairs of nodes between which an edge exists. Instead of having all possible connections, the edge list purely contains an entry per each edge of the graph.

Although graph theory splits graphs into many different types, the ones we are interested in for the purposes of this study are directed and undirected graphs, and simple and multi graphs.

In an undirected graph, an edge represents the existence of a relationship between two nodes. A directed graph assigns a direction to each edge, indicating a connection only from one node to another. The figure below illustrates the comparison between these two types of graphs, where they are identical, only one is directed and the other one isn't.

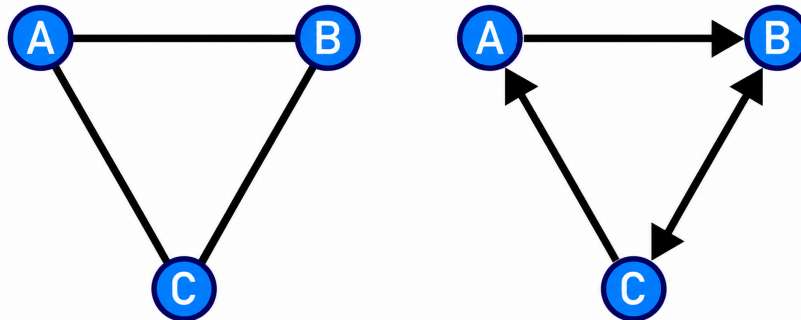


Figure 2.1: Undirected vs. Directed Graph

The other categories, simple and multi graphs refer to the number of possible edges between nodes. Simple graphs allow only one connection to exist between data points, while multi graphs do not limit the number of connections, as seen in the figure below.

Many real-world systems are naturally described by the relationships between their components, rather than static single points, so modeling them using graphs is an intuitive method. For example, in the context of power systems, a transmission network can be represented as a multi directed graph in which buses correspond to nodes, while transmission lines and transformers form the edges. This representation captures not only network topology, but also models power networks fittingly, since electric current is inherently directional and two buses can have more than one connection between them. This provides a good foundation for further analysis, the main example of the paper being determining observable islands in the electrical grid.

2.3.1. Trees and Forests

One last graph theory concept useful for understanding the algorithms described in this research paper are trees and forests. Formally, a tree is defined as a graph that does not form any loops, meaning that

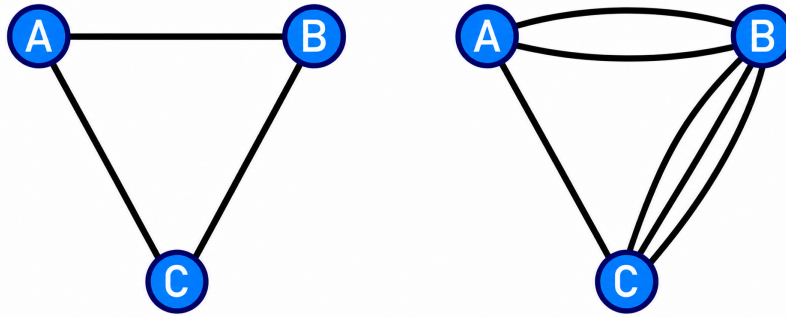


Figure 2.2: Simple vs. Multi Graph

by performing a walk on the graph from any point you can reach the starting point again without going over repeated edges. All the figures above, respectively figure 2.1 and 2.2 contain graphs with loops since, for example, performing a walk from point A, you can reach it again by going over the edges A-B, B-C and C-A.

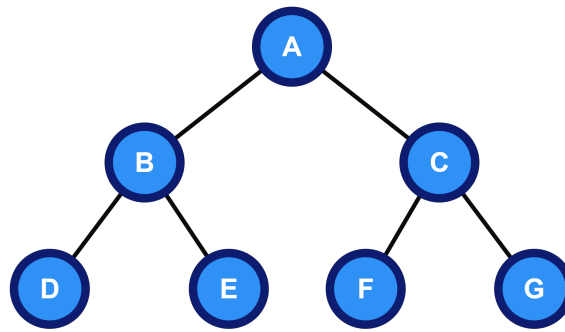


Figure 2.3: Tree

A forest is then any graph that is formed from one or more trees, while a maximal forest, or maximal forest of full rank, of a graph is the biggest possible forest, in terms of nodes, created using only its nodes and edges.

2.4. Network Observability and Observable Islands

The mathematical formulation of the state estimation problem stems from the non-linear measurement equation:

$$z = h(x) + e \quad (2.11)$$

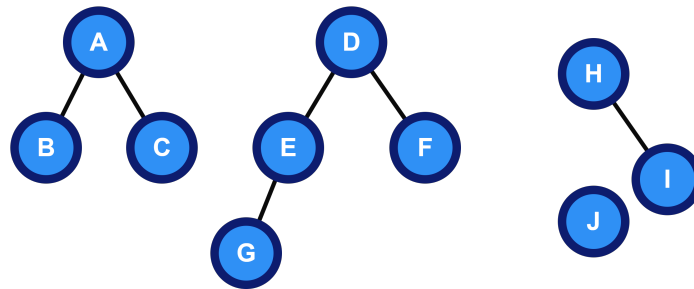


Figure 2.4: Forest

where z represents the measurement vector, containing real-time measurement information for active power (P) and reactive power (Q) line flows and bus injections. The vector x denotes the system state variables, composed of voltage magnitudes and phase angles across all network buses and the non-linear vector function $h(x)$ represents the physical relationships that map the state variables to their corresponding measurement values. e specifies the measurement error vector, which accounts for noise.

A fundamental notion for state estimation is that the network must be observable, meaning that the available set of measurements is sufficient to uniquely determine the state of the entire system. To evaluate the observability state, the system uses the measurement Jacobian matrix, written as J . This matrix contains the partial derivatives of the non-linear measurement functions $h(x)$ with respect to the state variables x . For an n -bus network where the state vector is restricted to bus voltage phase angles, the reference angle must be designated ($\theta_{\text{ref}} = 0$) because phase angles are relative quantities. Additionally, by treating the reference bus voltage magnitude as a known, fixed parameter, the number of independent state variables to be estimated in an n -bus network is reduced to $2n - 2$. The algebraic criterion for complete network observability states that the measurement Jacobian must possess full column rank:

$$\text{rank}(J) = 2n - 2 \quad (2.12)$$

If the Jacobian meets this condition, the system is globally observable and a unique state solution can be computed.

This idea has a natural grounding in Kirchhoff's circuit laws. Given a network of buses and branches, the voltages and power flows throughout the system are not independent but are contained by:

- **Kirchhoff's Current Law (KCL)** the sum of currents entering any bus must equal the sum of currents leaving it

$$\sum I_{\text{in}} = \sum I_{\text{out}} \quad (2.13)$$

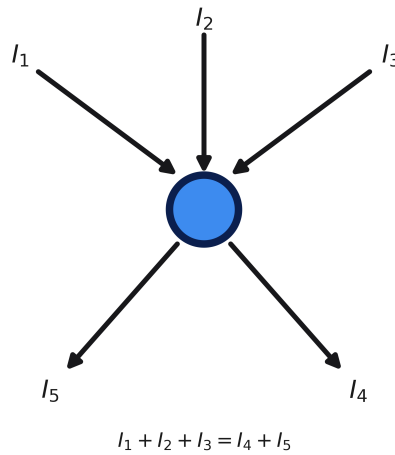


Figure 2.5: Kirchhoff's Current Law Example

- **Kirchhoff's Voltage Law (KVL)** the sum of voltage drops around a closed loop must equal zero

$$\sum V = 0 \quad (2.14)$$

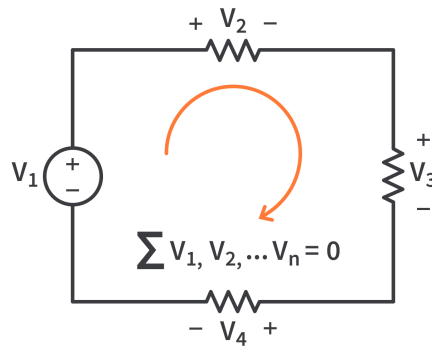


Figure 2.6: Kirchhoff's Voltage Law Example, taken from

<https://www.circuitbread.com/tutorials/how-to-solve-complicated-circuits-with-kirchhoffs-voltage-law-kvl>

These constraints mean that a well-placed subset of measurements can, in principle, allow the full system state to be inferred, if enough of them exist.

When this condition is not globally satisfied, the network is said to be unobservable. In such cases, it is still possible to identify maximal connected subnetworks within which the state can be estimated from the available measurements. These subnetworks are referred to as observable islands [40, 28]. The boundaries of the islands are determined by unobservable branches, lines for which no measurement path exists to propagate state information across them. Restoring full observability then reduces to placing additional measurements or incorporating pseudo-measurements, estimated values derived from historical data, at locations that bridge these islands for a fully observable system to be obtained [28].

2.4.1. Full Observability Example

Consider the simple network presented in figure 2.7. It is composed of three buses and two lines, having the measured components marked with red. The measured branch 1-2 gives us the relational information between bus 1 and bus 2, and with the measurement on bus 1 we can mark buses 1 and 2 as being observable. However, since no available measurements connect this island to bus 3, the whole network is deemed unobservable.

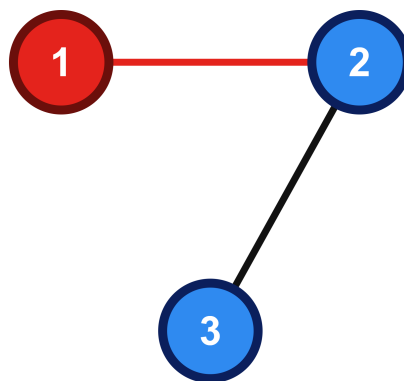


Figure 2.7: Unobservable Network

To make the network fully observable, we must either place an injection measurement on bus 3 or a flow measurement on line 2-3. This way, the observable island composed of buses 1 and 2 can be connected to bus 3, allowing the state of the entire network to be determined. Figure 2.8 shows both

of the described cases.

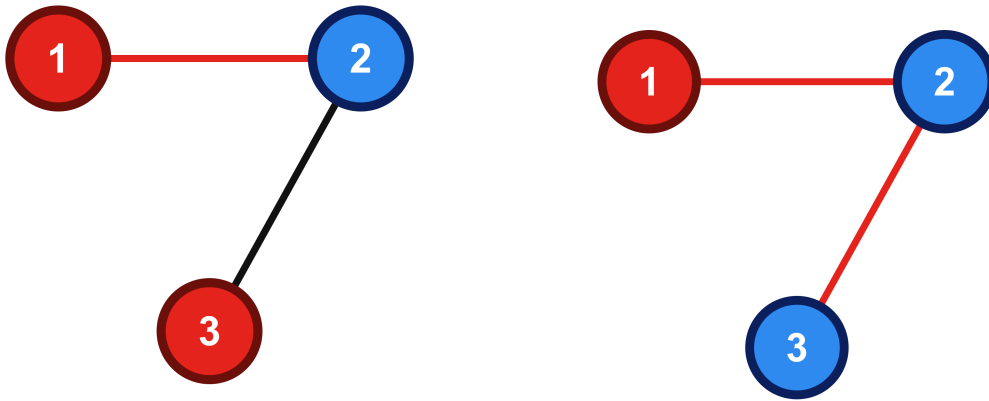


Figure 2.8: Observable Networks

It can be observed that different measurement configurations can result in the same set of observable islands. Specifically, for the presented network both measurement configurations:

- $M_1 = \{1, 1 - 2, 3\}$
- $M_2 = \{1, 1 - 2, 2 - 3\}$

will lead to discovering the same observable island, composed of all three buses.

2.4.2. Observable Islands Example

In order to showcase a set of multiple observable islands, we illustrate the network shown in Figure 2.9. Power injection measurements are placed at buses 1, 2, 4, 5 and 10, while line flow measurements are available on branches 2-3, 3-4, 4-5, 4-6, 8-9 and 9-10. We must construct the maximal forest in order to determine the observable islands.

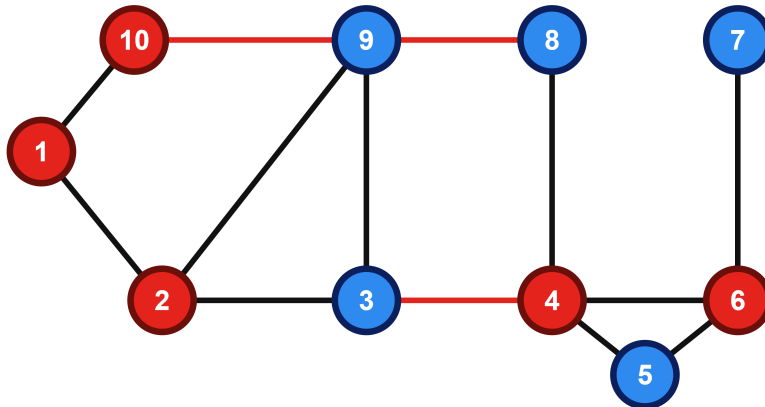


Figure 2.9: 10-bus Example

We start by merging all the buses connected by measured lines into superbuses. Thus, buses 8, 9 and 10 become superbus S8 and buses 3 and 4 become superbus S3, obtaining the configuration in figure 2.10. We tackle the measured buses inside a superbus by assigning it to one of the superbuses' neighboring branches.

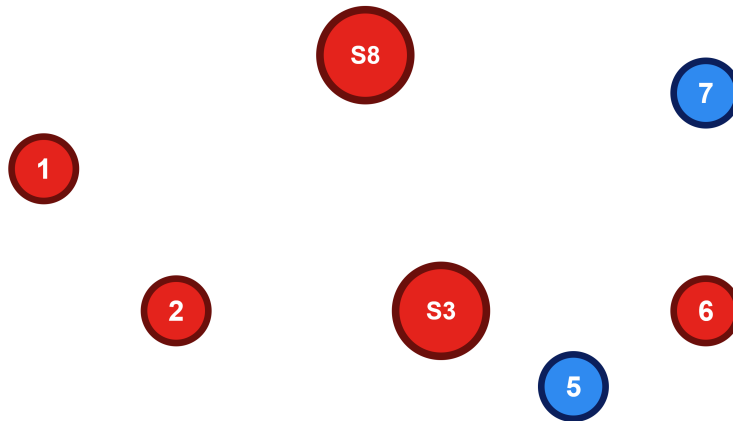


Figure 2.10: 10-bus Example after merging Flow Measurements

Next, each measured bus will assign, if possible, only one neighboring branch in the forest without forming any loops. This procedure can be computed randomly, so that no loops are created in the constructed graph. With this in mind, we make the following assignments:

- bus 1 $\rightarrow$ branch 1-2
- bus 2 $\rightarrow$ branch 2-S3
- bus 4 $\rightarrow$ branch S3-5
- bus 6 $\rightarrow$ branch 6-7
- bus 10 $\rightarrow$ branch 10-1

and obtain the maximal forest illustrated in fig 2.11. In the end, we obtain two observable islands, one composed of buses 1, 2, 3, 4, 5, 8, 9 and 10 and another smaller one containing only buses 6 and 7. This method is a simplification of the baseline algorithm further discussed in section 4.2.

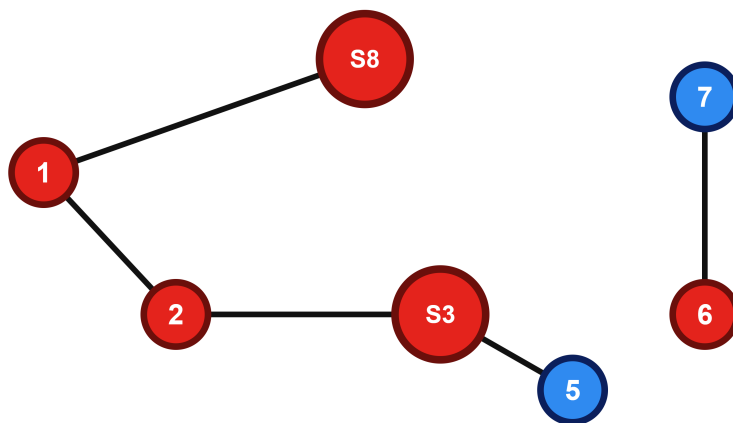


Figure 2.11: Observable Islands for 10-bus Example

2.5. Neural Networks

A neural network (NN) can be understood as a collection of simple neurons, aligned in layers, that compute and send information to one another. Each neuron receives input values from the nodes of the previous layer and combines them using a set of learnable weights and by applying a non-linear

activation function. Resulting values are then forwarded to the next layer. Stacking multiple such layers allows the network to create a complex mapping between the input and the output. Only the input and output layers are visible, while the calculations are done in the hidden layers in between.

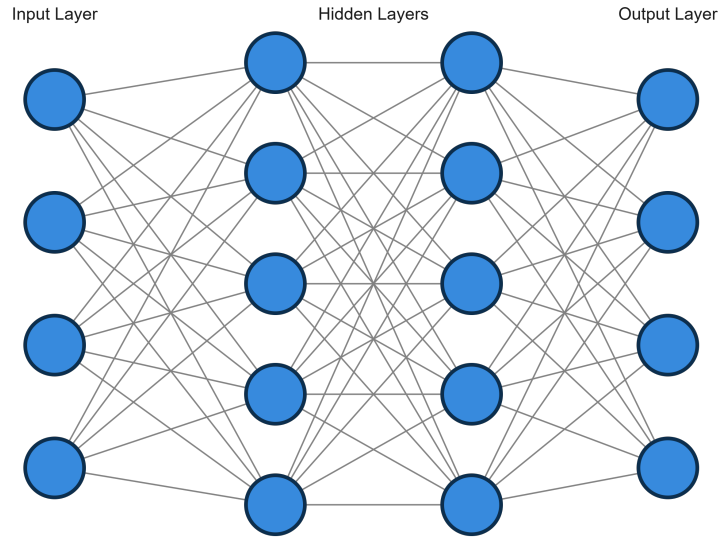


Figure 2.12: Neural Network Architecture

The objective of the network consists of a loss function, a scalar value that measures how far the network's predictions are from the desired outputs. Training an NN is done by feeding it batches of data, computing the resulting predictions and evaluating them using the loss function. Adjusting the weights changes the predictions of the network and consequently the value of the loss function. Learning the connection between the input and output becomes an optimization problem where the objective is finding the set of weights that minimizes the loss function on the training data.

This optimization is performed using gradient descent. This method computes the derivative of the loss function with respect to every weight in the network, i. e. a gradient, to determine the direction in which the loss increases. The backpropagation algorithm computes all gradients, by propagating the error from the output layer back through the network, so that the contribution of every weight can be determined. The weights are then updated by taking a step in the opposite direction of their respective gradients:

$$W \leftarrow W - \eta \nabla_W \mathcal{L} \quad (2.15)$$

where W is the weight matrix of the network, $\mathcal{L}$ is the loss function, ∇_W is the gradient with respect to W , and η is the learning rate, a hyperparameter controlling the size of each step. Repeating this process over many iterations minimizes the loss to a point where the network's predictions are closest to the desired output.

The dataset is typically partitioned into three sets, in order to test the learned behaviour generalizes beyond the training data. The training set is used to compute the loss and update the weights. A validation set is used once per epoch to see how the model performs on data it was not directly trained on. This allows users to monitor the network for overfitting, the phenomenon in which a network only learns the exact examples in the training set. If the validation loss is significantly smaller than the training loss, it is likely that the network is overfitting. Finally, the test set is only once used after training to estimate the network's performance on truly new data.

2.6. Convolutional Neural Networks

Convolutional neural networks (CNNs) are designed to exploit local relationships in the input data by representing it as a structured matrix [11]. A receptive field, also known as a kernel or filter is an area which extracts information from only a portion of the nodes. With each new computation, the field is shifted until information from all nodes is processed. By creating connections between neighboring elements, CNNs are able to identify hierarchical patterns within the data. This property makes them particularly effective for solving more complex problems such as image recognition, where the matrix structure of images naturally represents local relationships between pixels, each pixel containing specific information about the data. In addition to the input and output layers, a typical CNN is mainly composed of several other types of layers, including convolutional, activation, pooling and fully connected layers [11].

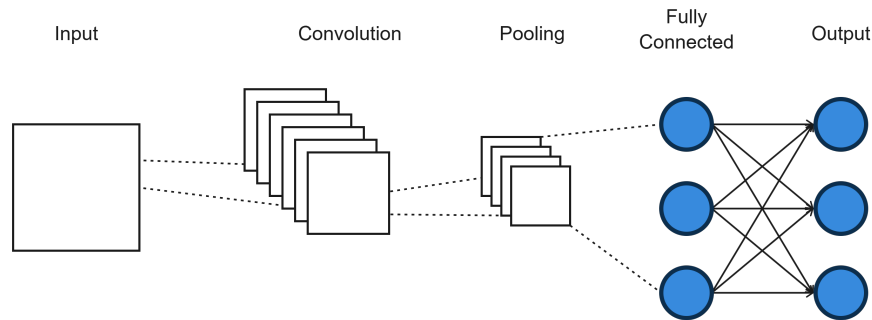


Figure 2.13: CNN Architecture

The convolutional layer extracts information from a subset of pixels by applying filters over specific regions of the image. After each filter shift, a new value is calculated in the next layer, creating a convoluted matrix. The learned features are passed to an activation layer, which applies a non-linear function to allow the model to capture more complex relationships, similar to the activation layer in the regular neural network.

The pooling layer reduces the size of the feature maps while preserving the relevant information from the previous layers, by summing, taking the maximum or averaging. By reducing the size of the convolutional matrix, pooling decreases computational cost and improves the network's ability to generalize[11]. Finally, the fully connected layer performs the final prediction. Each neuron from the previous layer is connected to each neuron in this layer, allowing the model to use the complete set of learned features.

2.7. Graph Neural Networks

Graph Neural Networks (GNNs) differ from regular neural networks by requiring the input data to be in the form of a graph, where information is stored through interconnected nodes and edges [34]. Since many real-world systems can naturally be represented in this way, GNNs are a very useful tool for solving a wide range of problems involving relationships between different data points. There are mainly three types of tasks solved by these architectures:

- **Node-level tasks**, which involve predicting a category or property of a node in the graph, for example labeling pixels in an image [26]
- **Edge-level tasks** which involve predicting the existence of a connection between pairs of nodes, for example friend recommendations in a social media app [36]
- **Graph-level tasks** which involve predictions about an entire graph, for example categorizing the toxicity of a molecule by analyzing its atomic structure[9]

2.7.1. Graph Convolutional Networks

Graph Convolutional Networks (GCNs), introduced by Kipf and Welling [19], extend the concept of convolution from regular grids to irregular graph structures. In a GCN, each node aggregates feature

information from its neighbours weighted by the graph's normalized adjacency matrix, depending on the number of layers in the convolution. For our power network case, each node in the graph represents a bus in the grid and each edge denotes a line or transformer present in the network. When applying a single layer of convolution, only the 1-hop neighbours, i. e. its nearest, are taken into account. With two layers, the information from the 2-hop neighbours is also incorporated, so on and so forth, as illustrated in figure 2.14.

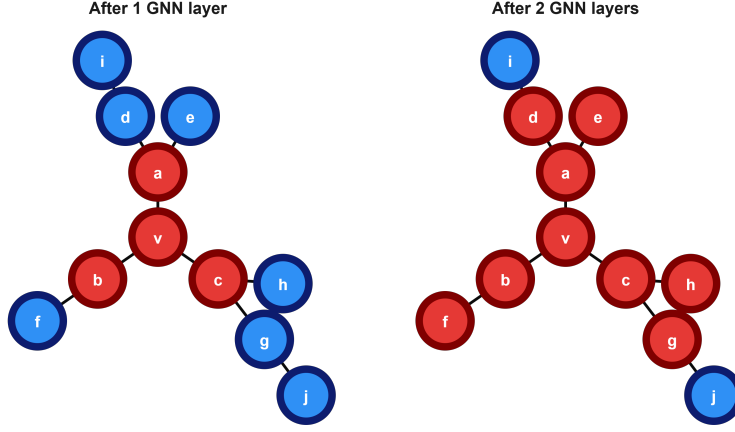


Figure 2.14: Different number of GCN Layers comparison

Formally, the update rule for a single layer is:

$$H^{(l+1)} = \sigma \left(\tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}} H^{(l)} W^{(l)} \right) \quad (2.16)$$

where $\tilde{A} = A + I$ is the adjacency matrix with added self-loops, $\tilde{D}$ is its corresponding degree matrix, $H^{(1)}$ is the matrix of node features at layer 1, $W^{(1)}$ is a learnable weight matrix, and σ denotes a non-linear activation function.

2.7.2. Graph Attention Networks

Graph Attention Networks (GATs), proposed by Veličković et al. [39], address a key limitation of GCNs by replacing the aggregation weights based on the graph's topology with learnable attention coefficients. A GAT computes a scalar attention score α_{ij} for each edge (i, j) which indicates the importance of neighbour j to node i during aggregation:

$$\alpha_{ij} = \frac{\exp(\text{LeakyReLU}(\mathbf{a}^T [W\mathbf{h}_i \| W\mathbf{h}_j]))}{\sum_{k \in \mathcal{N}(i)} \exp(\text{LeakyReLU}(\mathbf{a}^T [W\mathbf{h}_i \| W\mathbf{h}_k]))} \quad (2.17)$$

where $\mathbf{h}_i$ is a feature vector of node i , W is a shared linear transformation, $\mathbf{a}$ is a learnable attention vector and $\|$ means vector concatenation. This mechanism is topologically agnostic since every edge shares the learnable parameters W and $\mathbf{a}$. The updated node representation is then calculated as a weighted sum of the transformed neighbour features.

To stabilize training, multi-head attention is typically used, where K independent attention mechanisms are run in parallel and their outputs are concatenated or averaged. This dynamic weighting makes GATs particularly well-suited to graphs where structural connectivity alone is insufficient to determine the relevance of neighbouring nodes.

The GAT architecture is useful for developing our model since in power networks not every neighbour of a bus contributes equally for observability. For example, a measured bus contains more relevant information than an unmeasured one. By calculating different coefficients for each neighbour, the GAT learns to give a bigger weight to the more informative nodes.

2.7.3. GCN-SE

The Squeeze-and-Excitation Graph Convolutional Network (GCN-SE), proposed by Fan et al. [7] extends standard graph convolutional layers by a learned attention layer that re-weights the intermediate node representations. The network first applies a *squeeze* operation through a global average pooling over the node representations obtained from the convolutional layers, obtaining a compact vector. An *excitation* operation then passes it through a two-layer bottleneck with a sigmoid function to produce a set of scalar attention weights W^{att} :

$$\mathbf{W}^{att} = \phi(W_2 \delta(W_1 \mathbf{c})) \quad (2.18)$$

where W_1 and W_2 are learnable projection matrices, δ represents the ReLU function and ϕ represents the sigmoid activation. This mechanism captures the varying contribution of different representation channels and produces attention weights that are correlated with the true feature importance as measured by perturbation evaluation.

3

Related Work

In this chapter, we will go through all categories of the observability analysis solutions alongside their representative research papers. They are mainly divided into four: numerical, topological, hybrid and machine learning. Each method will have their upsides and downsides presented.

3.1. Numerical Methods

These approaches typically involve performing a LU/LDU factorization on the Jacobian, gain or Gram matrix and then inspecting the resulting diagonal matrix. The number of zero pivots indicates the rank deficiency in the system. If the matrix's rank satisfies the formula:

$$\text{rank}(J) = n - 1 \quad (3.1)$$

where n is the number of nodes and J the Jacobian, then the system is observable.

Early work by Monticelli and Wu [40] used the gain matrix G , while Gou [13] introduced the "test matrix", which contains only those rows of L^{-1} corresponding to the zero pivots in the diagonal matrix D to solve certain pathological cases. Building on this, Gou proposed a method that applies Gaussian elimination directly on the Jacobian H [12], which carries equivalent information, but with reduced computation time, due to H 's sparsity.

Although numerical methods are the most straight-forward to implement, one big limitation is their computational cost. The LU/LDU factorization involves $O(n^3)$ floating-point operations [3], making this solution sensitive to numerical precision and cubically expensive for larger grid sizes. This is especially a problem in modern power systems, which may consist of thousands of buses and require real-time monitoring. Benchmarking on systems from 1000 to 70,000 buses shows that numerical methods result in significantly higher execution times compared to other approaches [6].

3.1.1. Numerical Observability Analysis Examples

In this subsection, we will revisit the examples in section 2.4.1 by computing the Jacobian matrix for the unobservable network case and for the observable network that has a measurement on bus 3. Then, we will test their observability status by calculating the matrix's rank.

We will be using the general power flow and injection formulas for the DC model:

$$P_{ij} = b_{ij}(\theta_i - \theta_j) \quad (3.2)$$

$$P_i = \sum_{j \in \mathcal{N}_i} b_{ij}(\theta_i - \theta_j) \quad (3.3)$$

where P_{ij} and P_i represent the power value for line ij and bus i respectively, b_{ij} denotes the susceptance of line ij and θ_i is the voltage angle of bus i .

Additionally, we will take bus 1 as the reference:

$$\theta_1 = 0 \quad (3.4)$$

, meaning we have two independent states, θ_2 and θ_3 .

Unobservable Network Example

We are looking at figure 2.7 for this example. First, the active power injection at bus 1 is:

$$P_1 = b_{12}(\theta_1 - \theta_2) = -b_{12}\theta_2 \quad (3.5)$$

Therefore

$$\frac{\partial P_1}{\partial \theta_2} = -b_{12}, \frac{\partial P_1}{\partial \theta_3} = 0 \quad (3.6)$$

So its Jacobian row is:

$$H_{P_1} = [-b_{12} \ 0] \quad (3.7)$$

Then, the power flow from bus 1 to bus 2 is:

$$P_{12} = b_{12}(\theta_1 - \theta_2) = -b_{12}\theta_2 = P_1 \quad (3.8)$$

So its Jacobian row is the same as for P_1 :

$$H_{P_{12}} = [-b_{12} \ 0] \quad (3.9)$$

Therefore, the complete Jacobian matrix is:

$$H = \begin{bmatrix} -b_{12} & 0 \\ -b_{12} & 0 \end{bmatrix}, H \in \mathbb{R}^{2 \times 2} \quad (3.10)$$

However, its rank is:

$$\text{rank}(H) = 1 < 2 \quad (3.11)$$

where 2 is the number of independent states. In conclusion, the network is unobservable.

Observable Network Example

For this example, we will look at the leftmost network in figure 2.8. Because two measurements are identical to our previous example, we only need to compute the Jacobian row for the measurement at bus 3 in order to get the full matrix.

The power injection at bus 3 is then:

$$P_3 = b_{32}(\theta_3 - \theta_2) \quad (3.12)$$

Therefore

$$\frac{\partial P_3}{\partial \theta_2} = -b_{32}, \frac{\partial P_3}{\partial \theta_3} = b_{32} \quad (3.13)$$

So its Jacobian row is:

$$H_{P_3} = [-b_{32} \ b_{32}] \quad (3.14)$$

The complete Jacobian is then:

$$H = \begin{bmatrix} -b_{12} & 0 \\ -b_{12} & 0 \\ -b_{32} & b_{32} \end{bmatrix}, H \in \mathbb{R}^{3 \times 2} \quad (3.15)$$

Its rank is:

$$\text{rank}(H) = 2 \quad (3.16)$$

which is equal to the number of independent states. In conclusion, the network is observable.

3.2. Topological methods

Since numerical methods are solely based on matrix multiplications, in order to avoid excessive computation, topological methods are used. These solutions treat the problem purely as a graph search problem for a maximal forest of full rank. The objective is to maximize the number of assigned branches without forming loops. Generally, flow measurements are assigned to their branches, while injection measurements can be assigned to any branch. These assignment conditions are called topological rules.

Nucera & Gilles introduced a more efficient algorithm [31] based on augmenting sequences, which reassigns injections without backtracking. This type of algorithm was already discussed in section 2.4.2, applied on the topology presented in the paper. Zhang et al. [15] solved the problem by analyzing the boundary nodes. They form preliminary islands using flow measurements, then apply combinability rules to merge the islands. A general flowchart of the topological methods can be seen below:

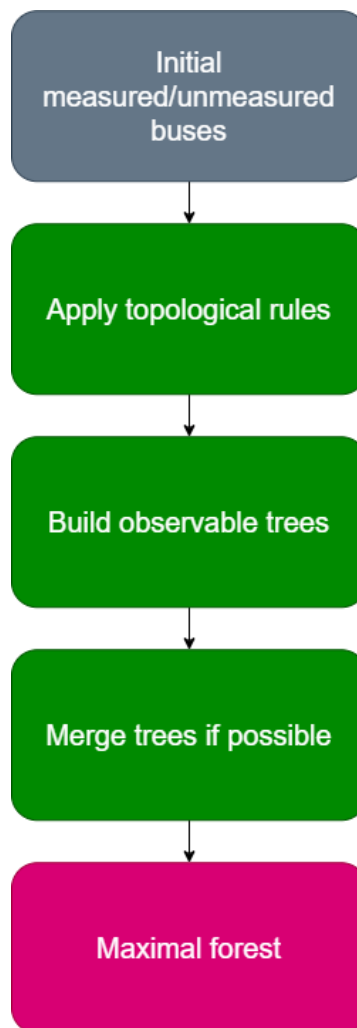


Figure 3.1: Topological Methods Flowchart

Unlike numerical methods, topological methods are numerically robust, as the observability calculations are based on logical operations rather than floating-point calculations. As a result, they offer speed advantages for bigger power networks. However, they are more complex to implement, requiring multiple combinatorial rules and special edge case handling. For example, zero-injection buses in particular do not carry load and require dedicated observability rules beyond the base algorithm [8].

3.3. Hybrid methods

They combine the strengths of both topological and numerical methods by first applying graph based searches to simplify the network to obtain flow islands and then performing matrix decomposition to determine observability. The network simplification helps overcome problems commonly found in the numerical approaches, such as round-off errors.

Contaxis and Korres [5] proposed a reduced model that merges buses into supernodes and uses orthogonal transformations on the reduced Jacobian matrix for observability and restoration. Korres and Katsikas [21] introduced a reduced-graph approach that forms supernodes from flow islands and uses echelon reduction of a test matrix for observability testing and pseudo-measurement placement. Later, Zhang and Han [41] proposed a combinability rule-based topological step that merges observable islands before the numerical analysis.

Benedito et al. [2] solved the observability problem with path graph concepts and a triangular factorization of the Jacobian. Their work identifies observable islands and critical measurement without solving algebraic equations. Lastly, Göl and Abur [10] focused on phasor measurements, providing a global angular reference. This way, the system no longer requires a slack bus for angle reference, thus a full-rank Jacobian is needed for an observable system.

Most hybrid observability methods follow the outline presented in the figure below:

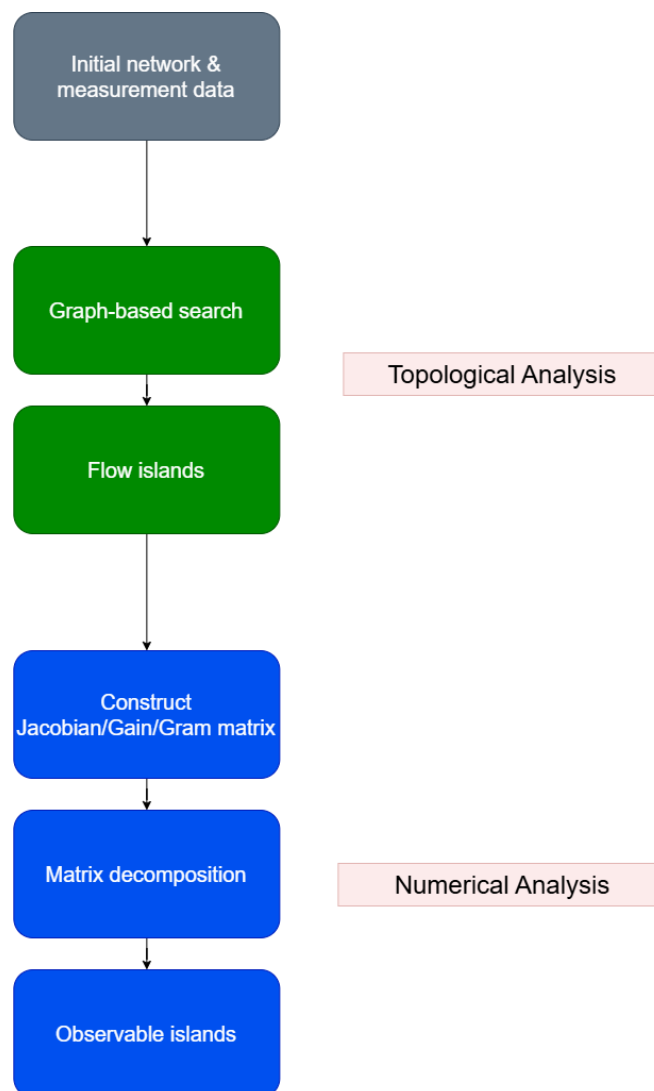


Figure 3.2: Hybrid Methods Flowchart

Hybrid methods reduce computational time by using topological preprocessing before the numerical analysis, improving scalability and reducing sensitivity to round-off errors. However, they may suffer from numerical ill-conditioning in weakly observable systems and become increasingly complex to implement for large networks [21, 41].

3.4. Machine Learning Methods

Because the existing literature on pure observability analysis solved with machine learning methods is very limited. This section is then separated into two subsections, the first covering the observability analysis research papers, while the second looks at machine learning solutions for the state estimation task.

3.4.1. Observability Analysis Solutions

These methods require a full pipeline of model training, inference, and evaluation. Jain et al. [18] proposed an early approach based on a feed-forward neural network with a single hidden layer, trained with a back-propagation algorithm. The network takes as input an exhaustive measurement set, consisting of an injection measurement for each bus and a flow measurement for each line, with each one encoded as 1 or 0, depending on its availability. The single output node then indicates whether the resulting configuration is observable or unobservable. Since the network architecture is tied to the size of the measurement set, a separate model had to be trained for each system, and the method was demonstrated on the 5-bus and 14-bus test cases. In both cases, the trained networks correctly classified most measurement configurations not seen during training, though with reduced confidence on the borderline cases.

Building on a similar architecture, Jain et al. [17] proposed a follow-up method that instead uses the quickprop algorithm for training, which incorporates second-order derivative information of the error function to accelerate convergence. The input and output representations remain the same, with a set of encoded injection and flow measurements and a single output node with the observability status. Tested again on the 5-node and 14-node networks, the models converged considerably faster than standard back-propagation, producing accurate classifications on most novel test patterns.

3.4.2. State Estimation Solutions

More recently, graph neural networks (GNNs) have been applied to closely related power system problems, although not directly to the topological observability classification task. Kundacina et al. [23, 22] proposed GNN models operating on a factor graph representation of the power system for state estimation. They evaluated robustness by removing measurement phasors from the test data to simulate unobservable regions. A follow-up study [24] extended the factor graph formulation by representing measurements as factor nodes that can be inserted or removed, while connecting variable nodes within a 2-hop neighbourhood to keep the graph connected even when many measurement are missing.

Across both neural network based methods and the more recent GNN approaches, the most accurate models still require training a dedicated network for each system topology and size, since input dimensions are tied to the specific configuration of buses and lines. A more generalizable solution would represent practical improvement over the current state of the art.

4

Methodology

This section will present the full methodology for the proposed solutions. We will go through the dataset generation needed to answer **RQ1**, including input and ground truth creation, the baseline algorithm to which our solution will be compared for **RQ3**, and the model’s architecture alongside its inspirations.

4.1. Dataset

In order for our graph machine learning (GML) model to properly learn to solve the observability problem, we need a reasonable sized dataset that maps working networks to their observable island configurations. So the construction of the dataset is split into two steps: creating unique measurement configurations for our network and finding sets of observable islands stemming from them, if such sets exist.

4.1.1. Generating Measured Networks

The proposed method was first trained and evaluated on the standard IEEE 14-bus test system [32] using the MATPOWER package [42]. Synthetic power grids are artificially constructed grid models that reflect statistical and topological features of real-world infrastructures without exposing sensitive data, making them a widely used tool in power systems research [27]. This specific network was chosen for its reduced size, as manually testing the observable islands is more manageable on a smaller grid.

To perform observability analysis on the grid, a dedicated measurement dataset was created. We do this by listing all candidate measurements of the network, generating a random probability of keeping the measurement for each one and then comparing them to a universal keeping probability, chosen at random between 0.2 and 1. Each iteration of the algorithm rerolls these numbers, so more varied grids are output. The elements that potentially have measurements are buses, lines and transformers and the potential measurements are voltage and coupled power injections on the buses and coupled active/reactive power flow measurements on the lines and transformers. The values were derived from working state estimation runs performed directly on the grid. These ranges were selected to expose the model to both densely and sparsely measured grids, to promote better generalization across real-world cases where coverage can vary significantly.

We would also want the dataset to be balanced between observable and unobservable networks, so that the model doesn’t implicitly assume that either of these cases is the default. For that, we chose a dataset balance of: $N_{obs} \div N_{unobs} = 0.25$. In order to verify each network for observability, a numerical observability algorithm had to be implemented. It is a simple checker that calculates the system’s Jacobian and computes its rank. If it has full rank, the network is deemed observable, otherwise it is classified as unobservable.

If it doesn’t, the rank deficiency shows how many independent ambiguities exist in the system, i. e. cases where two different network states would produce identical measurements. To find them the algorithm looks in the null space of the Jacobian, where each vector represents an ambiguity. It picks

Algorithm 1 Observable Island Identification and Sampling

```

UF  $\leftarrow$  InitializeUnionFind(net.buses)
snapshots  $\leftarrow$  []
for each flow measurement on edge (u, v) do
    UF.Union(u, v)
    snapshots.append(Copy(UF))
end for
for each injection measurement on bus u do
    v  $\leftarrow$  RandomUnmergedNeighbour(u)
    if v exists and UF.Union(u, v) then
        snapshots.append(Copy(UF))
    end if
end for
candidates  $\leftarrow$  UniqueIslands(snapshots)
sort candidates descending by size
for each island  $\in$  candidates do
    if CheckObservability(island) is True then
        SaveAndPlot(island)
        continue
    end if
    Eunmeas  $\leftarrow$  UnmeasuredEdges(island)
    attempts  $\leftarrow$  0
    while attempts < max_attempts do
        attempts  $\leftarrow$  attempts + 1
        subisland  $\leftarrow$  DropRandomEdges(island, Eunmeas)
        if CheckObservability(subisland) is True then
            SaveAndPlot(subisland)
        end if
    end while
end for

```

▷ Phase 1: Union-Find merging

▷ Phase 2: Island sampling

Once such a forest is obtained, it directly corresponds to the observable islands configuration in the network.

The method begins with the full set of nodes and no branches assigned. It assumes that active and reactive measurements are always available in pairs, as with our own model. It then processes all available measurements, assigning as many branches as possible without creating a loop, according to the following rules:

- **Flow measurements** can only be assigned to the corresponding measurement branch
- **Injection measurements** can be assigned to exactly one of the incident branches of the measured bus

Because a flow measurement has only one possible assignment branch, it is processed separately from injection measurements and more efficiently. All buses linked by an assigned flow measurement are merged into a single superbus before injection measurements are considered. In the end, the algorithm undoes this merge for the observability islands configuration.

Augmenting sequences tackle are essential when an injection measurement cannot be assigned to any of its incident branches without forming a loop. The solution is to simply assign it to an available branch, even though it creates a loop. The algorithm then removes an edge in the loop and attempts to reassign that injection to one of its remaining candidate branches. If this produces a loop, the algorithm repeats. The resulting chain of assignments and removals is called an augmenting sequence. It ends successfully once an injection can be placed without creating a loop, growing the forest by one branch, or unsuccessfully once every candidate branch has been checked, in which case the original injection is marked redundant and the forest remains unchanged.

After processing all measurements and building a maximal forest, the algorithm performs a final pass to decide which injection measurements are truly usable. The rule is that if an injection is assigned to a bus that doesn't have all its incident branches in the span of the forest, it is unusable. Because removing an edge can disqualify further injections, it will create a cascading effect where this check is repeated until no more unusable injections remain. The resulting forest is the observability islands configuration.

4.3. Model Architecture - Variational Graph Autoencoder

Our model follows a Variational Graph Autoencoder (VGAE) structure, originally introduced by Kipf and Welling [20], adapted with the attention layers inspired by the GCN-SE architecture [7]. A graph autoencoder is composed of two separate models: an encoder and a decoder. The encoder's purpose is to embed the input graph into a lower-dimensional latent vector, capturing its structural features. Then, the decoder must learn to reconstruct the original graph using the latent representations. However, the decoder can also be trained to predict a different target derived from the input. This way, both components are optimized to minimize the difference between the reconstruction and the ground truth. The loss function can be written in its simplest form as:

$$\mathcal{L} = ||X - \hat{X}|| \quad (4.1)$$

where X represents the ground truth and $\hat{X}$ is the reconstruction.

The "variational" part comes in for the encoder component. Instead of directly outputting the latent vectors, a variational graph autoencoder outputs the parameters of a Gaussian distribution in the latent space, mainly the mean μ and the standard deviation σ . The latent vector is then sampled using these two parameters and is then fed to the decoder for reconstruction.

Our encoder takes as input the edge list E , a matrix of node features X containing measurement value, measurement type (voltage, power and relative power measurements), node degree and a boolean value indicating whether or not that node was measured. It also uses Y , the edge feature matrix which has the same fields as the node feature matrix, with the polarity instead of node degree.

The model first applies three GCN layers, in order to get information about its 3-hop neighbourhoods. We experimented with a range of GCN layers, from one to four and got the best accuracy and generalization capabilities with only three layers. This output is then passed to two separate GAT layers in parallel, which produce μ_z and $\log \sigma_z^2$ respectively, which parametrize the latent Gaussian distribution from which the latent embedding z is sampled.

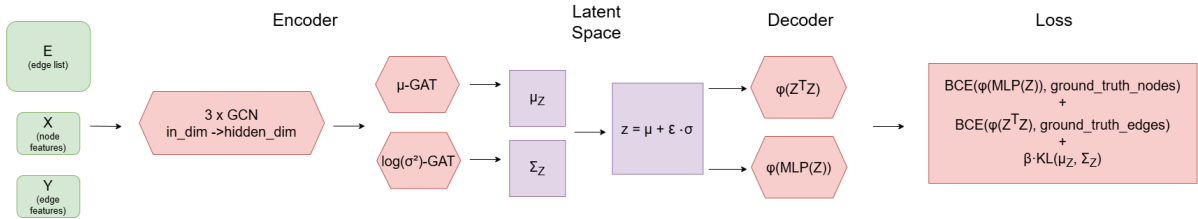


Figure 4.2: Model Architecture

At first, we fed the encoder the adjacency matrix as input directly and trained the model to reconstruct it. The resulting latent embeddings were then processed using a clustering algorithm in order to obtain the observable islands. However, two major problems arose from this design choice:

- **Increased computational time.** Training the model to reconstruct the full adjacency matrix required more than twice the computational time compared to the alternative method, with virtually no changes to accuracy.
- **Clustering doesn't inherently translate to observable islands.** Although the reconstruction loss decreased by over 50% during training, performing a clustering algorithm on the embeddings did not produce the expected observable islands configurations. Algorithms like KMeans [25] ask the user to specify the number of clusters, which defeats the purpose of automatically detecting

the islands. Other methods, such as MeanShift [4], can estimate the number of clusters directly but still tended to produce equally-sized clusters, which are not representative of the varying structure and topology of the observable islands.

As such, we decided to change the model's learning objective. Instead of reconstructing the full adjacency matrix, the target now becomes the observable islands configuration. As described in section 4.1.2, we determined for each network its own set of observable islands. We push the model to learn two different objectives: which nodes are present in the ground truth and which nodes are connected in the same observable island.

The decoder then reconstructs the observability islands configuration graph through two separate functions on the latent embeddings z . Edge reconstruction is performed with an inner product between latent node representations, $\hat{A} = ZZ^T$, which calculates edge existence scores for each pair of nodes. Node reconstruction is handled by a separate two-layer Multi-Layer Perceptron (MLP) head [33] with ReLU nonlinearity, which maps a node's latent embedding to a score that indicates its observability status.

Together with the KL-divergence loss of the VGAE architecture, the learning objective becomes three-fold:

$$\mathcal{L} = \mathcal{L}_{\text{node}} + \mathcal{L}_{\text{edge}} + \beta \cdot \text{KL}(\mu_z, \sigma_z) = \mathcal{L}_{\text{node}} + \mathcal{L}_{\text{edge}} - \frac{\beta}{2} \mathbb{E} [1 + \log \sigma_z^2 - \sigma_z^2 - \mu_z^2] \quad (4.2)$$

It consists of an edge loss term, a node loss term and a KL-divergence regularization term. The term $\text{KL}(\mu_z, \sigma_z)$ acts as a regularizer on the latent space, penalizing deviations of μ_z and σ_z from a standard Gaussian prior. The expectation $\mathbb{E}[\cdot]$ denotes averaging over latent dimensions and produces a scalar loss, while the coefficient β controls the trade-off between the expressiveness of the latent space, which supports both node observability and island membership, and the regularization towards the Gaussian prior.

Node Prediction

We perform this task by employing a simple two-layer MLP head with the ReLU activation function that takes the node embeddings from the latent space and maps them to the observability logits determined in the ground truth calculation.

Most observable islands configurations contain fewer nodes than the full network, so we observed the model taking a shortcut and predicting absent nodes far more than present ones, since it didn't affect the loss in a substantial way. To counteract this, we added a positive class weight, so that the model is encouraged to prioritize identifying present nodes over absent ones. This weight is computed as the ratio between the total negative instances over the total positive instances across the ground truth data, offsetting the class imbalance caused by the absent nodes.

The loss function for node prediction then becomes:

$$\mathcal{L}_{\text{node}} = -\frac{1}{N} \sum_{i=1}^N \left[w_{\text{pos}} y_i^{\text{obs}} \log \varphi(n_i) + (1 - y_i^{\text{obs}}) \log (1 - \varphi(n_i)) \right] \quad (4.3)$$

where N is the number of nodes, w_{pos} is the positive weight, n_i is the predicted node logit, y_i^{obs} is the ground truth node label and φ is the sigmoid function.

Edge Prediction

This step is less computationally demanding for the decoder, as it simply performs a dot product between the latent embedding vector and its transpose, producing a score for each node pair that indicates whether the two nodes belong to the same island. Similar to the node prediction, we want to encourage the model to predict positives so we calculate a corresponding positive class weight.

However, since the number of positive edges is substantially outnumbered by the other possible pairs, we also want to address this by making use of negative sampling [20]. Rather than training against every possible negative pair, we sample a subset of them at a fixed ratio relative to the positives. This keeps the negative class large enough for the model to learn to distinguish between negatives and positives, without letting the negatives overwhelm the loss. We sample the negatives at a ratio of 1.5 per positive edge.

Taking all of this into account, the loss for edge prediction is:

$$\mathcal{L}_{edge} = -\frac{1}{|\mathcal{E}_{pos}| + |\mathcal{E}_{neg}|} \sum_{(i,j) \in \mathcal{E}_{pos} \cup \mathcal{E}_{neg}} \left[w_{pos} y_{ij}^{edge} \log \varphi(e_{ij}) + (1 - y_{ij}^{edge}) \log (1 - \varphi(e_{ij})) \right] \quad (4.4)$$

where $e_{ij} = z_i \cdot z_j$ is the edge logit, $\mathcal{E}_{pos}$ is the set of positive pairs, $\mathcal{E}_{neg}$ is the set of sampled negative pairs, y_{ij}^{edge} is the ground truth label of the pair, same-island membership, w_{pos} is the positive weight and φ is the sigmoid function.

5

Evaluation & Results

This chapter presents the evaluation setup for both our architecture and the baseline algorithm. It goes through the performance metrics, how they are calculated and then presents the results on both a fully observable portion of the dataset and an unobservable one. It also presents the ablation results in order to see which attributes makes the model perform better.

5.1. Experimental Setup

In order to test and compare the proposed model against the baseline algorithm, we evaluate their predictions by computing the accuracy, i. e. counting the number of predicted islands for each network and then checking how many of them actually are observable.

The observability check is performed using the Jacobian rank deficiency function mentioned in 4.1.1. Initially, we relied on pandapower's estimate function [38] to verify every island. However, we found that running a full Weighted Least Squares state estimation for every candidate is considerably slower than a Jacobian rank deficiency check. Moreover, the built-in estimator can be too forgiving, sometimes applying pseudo-measurements to force a solution. This behavior contradicts the goal of evaluating the network strictly on its existing measurements.

Accuracy alone is not sufficient to evaluate an observable islands generator. Its goal is not only to identify observable islands, but also to recover as much of the original network as possible. For example, in the case of a 100-node network, predicting five observable islands covering all its nodes is generally more informative than identifying a single 90-node island, while leaving the remaining ten nodes unobservable. For this reason, in addition to the accuracy, we report three additional metrics: the average island size, the average network coverage and the average rank deficiency, since our observability checker uses the rank difference of the Jacobian.

The average island size measures the mean number of nodes contained in a predicted island, while the average network coverage measures the ratio of nodes covered by the predicted islands across all evaluated networks and the average rank deficiency quantifies the extent to which the observability condition is not satisfied. They are computed as:

$$\text{Average Rank Deficiency} = \frac{1}{I} \sum_{i=1}^I \text{rank deficiency}_i \quad (5.1)$$

$$\text{Average Island Size} = \frac{1}{I} \sum_{i=1}^I \text{number of predicted nodes}_i \quad (5.2)$$

$$\text{Average Network Coverage} = \frac{1}{N} \sum_{i=1}^N \frac{\text{number of predicted nodes}_i}{\text{number of total nodes}} \quad (5.3)$$

where I represents the total number of predicted islands and N denotes the total number of evaluated networks.

As for the dataset used in this evaluation step, we have generated 1500 different measurement configurations for the IEEE 14-bus network, with 85% of them being used for training evaluation, leaving us with the rest of 15% unseen networks used for the final evaluation.

5.2. Results

This subsection first showcases the ablation results obtained from testing different training techniques and optimizations, alongside the baseline and existing literature comparisons needed for **RQ2** and the model's performance on networks of varying sizes needed to answer **RQ3**.

5.2.1. Ablation Results

Before the head-to-head comparison between the model and the baseline algorithm, it is useful to examine how the different training techniques and architectural changes affect its accuracy. All results in this section reflect the base 14-bus model's architectural and hyperparameter changes. This accuracy was computed on datasets consisting of a mix of fully observable and unobservable power grids, fact which bears mentioning since the evaluation was performed on separate datasets.

The first fully trained model iteration used a single GCN layer, no positive sample weights and a negative sample ratio of 1 on the edge prediction task. This initial version achieved an accuracy of 40.8%. After establishing the base model, we started tweaking its hyperparameters and architecture, in order to improve performance. The final version obtained an accuracy value of 63.3% on a dataset consisting of both fully observable and unobservable networks.

The effect of adding positive sample weights during training can be observed in the first figure below:

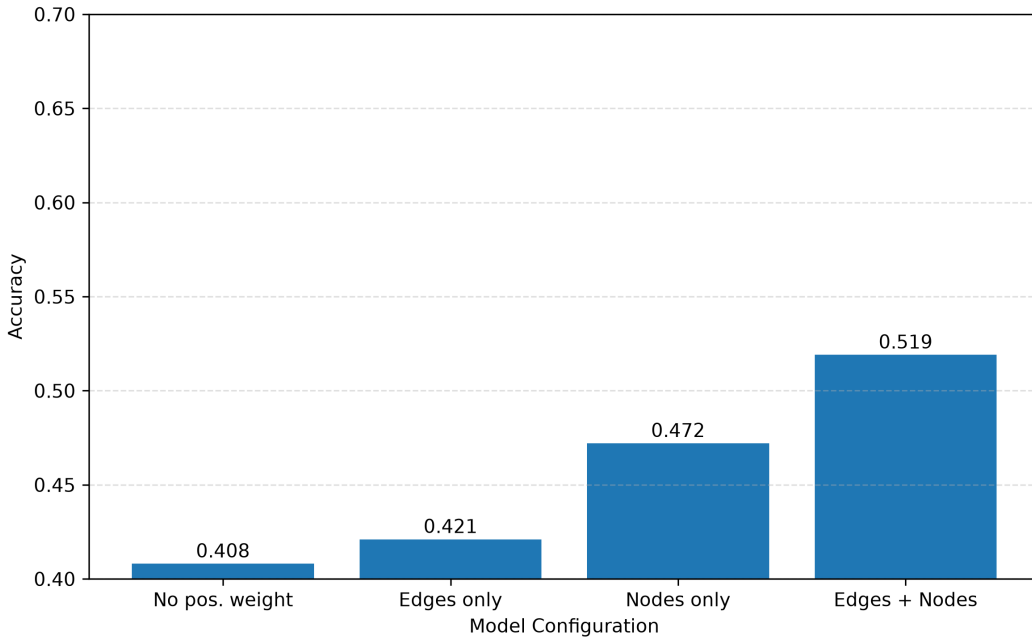


Figure 5.1: Effect of positive sample weights on accuracy

The addition of positive sample weights on both nodes and edges resulted in an increase of approximately 10% in accuracy. The weights applied to the node observability prediction task had the larger impact, which is expected since what constitutes an observable island lies mainly in its node configuration. Therefore, pushing the model to correctly learn observable nodes is more important than focusing solely on the relationship of nodes within an island.

The impact of the negative samples ratio can be seen in Figure 5.2. Starting from an initial ratio of 1, several other values were tested to see how they affect performance. Although the improvements were relatively small compared to the addition of positive weights, the choice of ratio still affected the final accuracy. A noticeable decrease can be seen between a ratio of 1.5, which produced the best results in our experiment, and a ratio of 2.

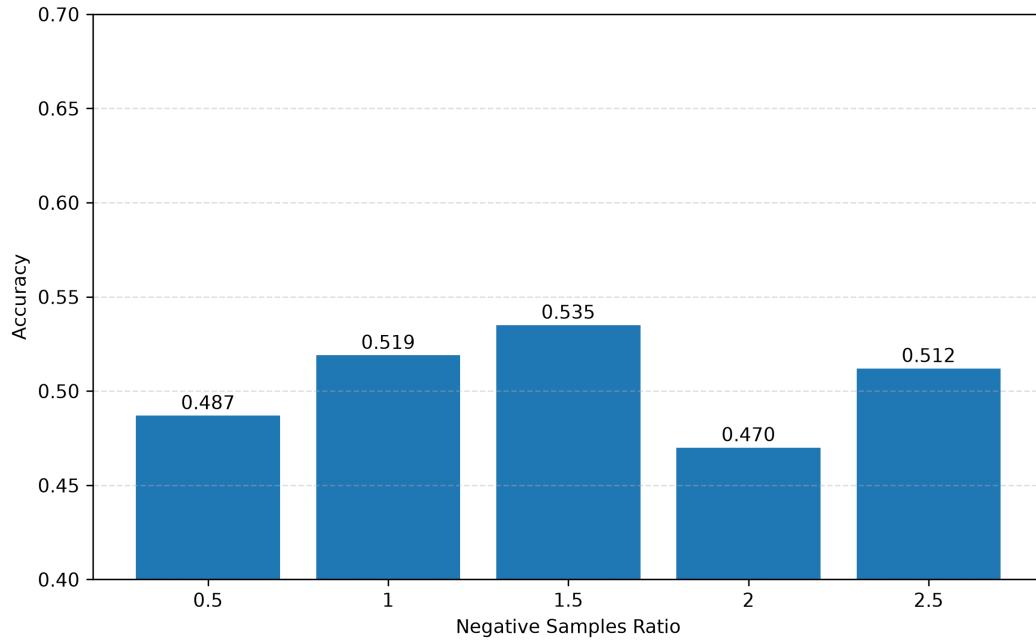


Figure 5.2: Effect of negative samples ratio on accuracy

Finally, Figure 5.3 shows the effect of changing the number of GCN layers in the model architecture.

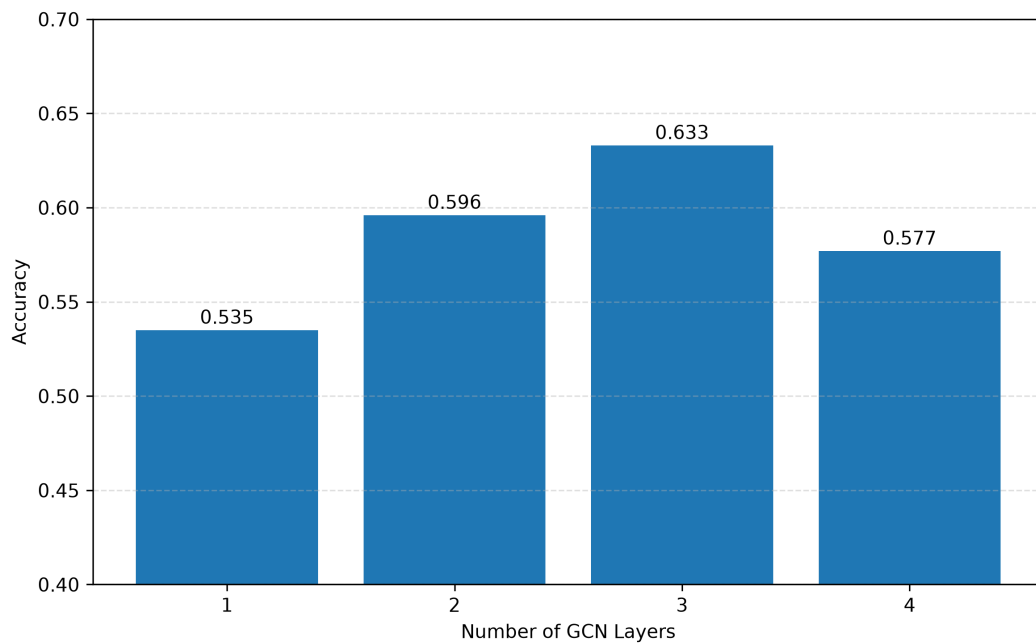


Figure 5.3: Effects of number of GCN layers on accuracy

As expected, this architectural change had a considerable impact on the model's accuracy. Increasing the number of layers allows the model to gather information from a larger neighbourhood, with three GCN layers allowing it to capture structural information from a 3-hop neighbourhood, instead of only a 1-hop neighbourhood as in the base model. The additional context improved the model's predictions, with three layers increasing its accuracy by approximately 10%. When using four GCN layers the accuracy decreased noticeably, although the model still performed better than the initial version with a single layer.

We will evaluate both our solution and the baseline on a split version of the dataset: one part containing only fully observable networks and another containing only unobservable networks. This way we can see if there exists a gap in performance based on the grid's overall observability or if the algorithms perform similarly. Moreover, the additional metrics besides accuracy, mainly the average network coverage, would give us a more in-depth look into why this potential gap exists.

5.2.2. Baseline Comparison - Fully Observable Networks

The comparison between the best-performing version of the proposed model and the baseline algorithm on the fully observable subset is presented in the table below.

Metric	Proposed Model	Baseline Algorithm
Accuracy (%)	91.64	100
Average Island Size	11.13	13.96
Average Network Coverage (%)	85.6	99.7
Average Rank Deficiency	0.05	0

Table 5.1: Comparison between our model and the baseline on fully observable networks

Both methods perform extraordinarily well on fully observable networks, since a subgraph of an observable grid is likely observable too. The baseline achieves a perfect score, every one of its predicted islands being observable, while the proposed solution lags behind a bit with a 91.64% accuracy.

The baseline almost always predicts the full island, with an average network coverage of 99.7%, which is in line with its perfect accuracy score. Very few measurements are lost due to absent nodes or lines, so the grid's inherent observability remains intact. The model, however, has a lower network coverage of 85.6% which, although still very high, can produce results with missing measurement information, thus leading to an unobservable prediction. Looking at its average rank deficiency of 0.05, we can confidently assume that the failed predictions fall short by only a couple measurements, spread across the subset.

5.2.3. Baseline Comparison - Unobservable Networks

Metric	Proposed Model	Baseline Algorithm
Accuracy (%)	54.27	18.09
Average Island Size	7.07	12.3
Average Network Coverage (%)	60.23	96.76
Average Rank Deficiency	0.46	1.35

Table 5.2: Comparison between our model and the baseline on unobservable networks

With fewer measured elements available, both solutions perform worse, which is to be expected. In this case, the baseline still tends to predict almost full, its network coverage being a high 96.76%. This tendency proves to be a weakness because the data it's trained on is inherently unobservable, so bigger predictions are more likely to fail by default, leading to an accuracy of 18.09%. This observation is further backed by the fact that the average rank deficiency is 1.35, significantly higher than on the fully observable dataset.

The model, on the other hand, has an accuracy of 54.27%, meaning that the majority of its predictions end up being observable. Indeed, it doesn't span as much of the grid as the baseline with a more

modest coverage of 60.23%. In the end, our model still recovers on average more than half of the unobservable networks' nodes.

Even though the baseline algorithm has better results on the fully observable portion of the dataset, the real difference lies in the unobservable part. The proposed model's accuracy is three times higher than the baseline, which is arguably the more important result since in real-world scenarios such an algorithm would be deployed on unobservable power grids for state estimation.

5.2.4. Performance across different network sizes

We would like our model's architecture to produce good, consistent results on grids of varying sizes, and training and evaluating it on a single network does not fulfill this generalization scope. For this, we have also trained the model on IEEE's power system test cases with 30, 39, 57 and 118 buses respectively, to see how the evaluated metrics change across grid dimensions. We used 5000 measurement configurations for these other networks, a more than threefold increase. This increase in dataset size is justified because with each new element added, the number of potential configurations increases exponentially.

Metric	30-bus Model		39-bus Model		57-bus Model		118-bus Model	
	Obs.	Unobs.	Obs.	Unobs.	Obs.	Unobs.	Obs.	Unobs.
Accuracy (%)	94.2	56.0	93.5	49.0	85.0	54.6	91.3	53.6
Average Island Size	17.7	9.4	20.6	12.1	34.7	11.1	42.5	8.5
Average Network Coverage (%)	71.0	35.9	65.4	40.6	88.9	45.7	79.0	27.7
Average Rank Deficiency	0.04	0.70	0.05	0.97	0.15	0.86	0.06	0.67

Table 5.3: Model comparison for observable and unobservable cases across networks of different sizes

Just like for the first trained model, there is a significant gap between results on fully observable networks versus unobservable networks. The accuracy on observable networks ranges from 85% to 94.2%, further confirming that algorithms generally perform better on power grids with a high number of measurements. The average network coverage is also consistent, with a spike of 88.9% for the 57-bus model. This correlates with the lowest accuracy of 85% which supports the idea that more conservative predictions lead to better observability results. Earlier versions of the models were trained on datasets of 1500 networks each. After retraining on 5000 grids, there was no significant difference in performance.

On the unobservable side, the model's accuracy ranges from 49% to 56%, a difference of only 7 percentage points. These results are in line with the first iteration of the model, which had an accuracy of 54.27%, proving that the proposed architecture is able to produce consistent, replicable results across networks of different sizes. It is important to note that the first batch run of the models, trained on 1500 data points did not achieve such high accuracy. Their values never surpassed 40% and would only exceed it after being fed more data. In order to achieve this accuracy, it seems that the average network coverage is smaller than the 14-bus model, with the highest value of 45.7%, meaning that the bigger the networks become, the smaller the recovered subnetwork. Overall, none of the metrics here show a clear dependence on network size, except perhaps the average coverage, showing that the model's behaviour remains consistent as the network becomes larger.

5.2.5. Baseline Comparison - Mixture of Networks

Another way of testing the generalizability of our architecture and answering **RQ3** is to train and evaluate it on a mixture of topologies of different magnitudes, instead of doing each one separately. For this reason, we created a mixed dataset by adding 20% of the entries for each of the five network sizes into a single one. This new dataset includes both fully observable and unobservable grids. The results of the general model compared against the baseline algorithm can be observed in table 5.4.

Metric	General Model	Baseline Algorithm
Accuracy (%)	56.08	34.64
Average Island Size	13.03	53.78
Average Network Coverage (%)	52.64	98.88
Average Rank Deficiency	0.96	3.11

Table 5.4: Comparison between our general model and the baseline on both fully observable and unobservable networks

Considering the fact that both our model and the baseline perform significantly better on fully observable networks and that both types of power grids are present in this mixed dataset, it is expected for their accuracies to lie between the results obtained in section 5.2.2 and 5.2.3. The trends observed in previous comparisons continue, with the baseline predicting noticeably larger islands and having a lower accuracy than our model.

5.2.6. Comparison with Existing Literature

In order to strengthen our findings for **RQ2**, we will compare our architecture’s performance, more specifically the general model from section 5.2.5, against the current machine learning solutions from [17, 18]. As stated before, these models do not output islands, instead opting for a binary response: 1 for observable and 0 for unobservable. For this comparison to make sense, we will only count predictions encompassing the whole power grid as a fully observable prediction, all others being labeled as unobservable.

Existing literature evaluates the models on a custom power system consisting of 14 buses, with a different topology from the IEEE 14-bus case. Depending on the measurement configuration, the system may be fully observable or unobservable. Since the papers’ results are perfectly in line with the ground truth, we will not write it. The comparison can be observed in table 5.5.

	Obs. 1	Obs. 2	Obs. 3	Unobs. 1	Unobs. 2	Unobs. 3
Existing Models	1	1	1	0	0	0
Proposed Model	0	1	1	0	0	0

Table 5.5: Comparison of the proposed and existing models

The only difference in output lies in the first observable case, where our model did not output the entire network. Notably, it predicted an island made up of 10 buses out of the 14 total, showing a fairly close result to the ground truth. Aside from that case, the proposed architecture’s results are identical to the existing literature, proving that it is consistent with current solutions, while providing a more informative representation of observability.

6

Discussion

In this section, we will draw conclusions based on our solution’s results, strengths and limitations along with future work and societal and ethical implications.

6.1. Discussion about Results

The primary objective of this study was to answer **RQ2: Can a graph machine learning architecture identify observable islands in a power network based on its topology and available measurement configuration?** Based on the obtained results, the answer seems to be yes. The model demonstrates that observable islands are learnable property of a power network when both its structure and available measurements are provided as input.

Although the final average accuracy of 63.3% is not sufficient for real-world deployment, it provides a strong groundwork for further improvement. As shown in figures 5.1, 5.2 and 5.3, relatively small changes to the training procedure and model architecture can have a significant effect on output quality. In particular, the addition of positive sample weights and increasing the number of GCN layers each improved the accuracy by approximately 10%. These results show that the model has not yet reached its full potential and that further hyperparameter tuning, perhaps optimizing the β in the KL-divergence loss, could yield additional improvements. It may also be worthwhile to investigate task specific weighting factors for the node and edge prediction losses.

Comparing our solution to the baseline algorithm further cements these improvements. The first iteration of the model performed nearly as well as the baseline algorithm, subsequent changes further increasing the accuracy difference. The proposed model also produced smaller islands on average, with an average island size of 7.7 compared to the baseline’s 12.5.

On the evaluation dataset, this proved to be an advantage, as the majority of networks are not fully observable and thus require smaller observable subnetworks to be found. Perhaps running the same comparison on a different dataset, where a larger portion of the networks are observable or entail bigger observable islands would have led to a closer accuracy gap between the methods.

Another important scope of our project, **RQ3** was to gauge the extent to which the proposed architecture generalizes to different power system sizes and topologies. For this, we conducted two separate experiments that tackled this subject from different angles. We first looked at how the model performs when separately trained and evaluated on multiple different IEEE test cases, then trained a final model on a mixture of all the datasets. Across both experiments, the accuracy remained in a reasonable range as the baseline comparisons, 91.64% for the observable case and 54.27% for the unobservable one. Average network coverages showed greater variability, the average island size reflecting the same trend.

It is also important to note that the way we trained the models and calculated accuracy does not reflect the true observable islands. The reported accuracies and average island sizes measure different aspects of the predictions and do not fully capture the practical objective of identifying the most informative

islands. This distinction is reflected in the average island size and average network metrics reported in tables 5.1, 5.2, 5.3 and 5.4. Incorporating network coverage directly into the training objective could then help in that direction.

The comparison with existing solutions shows that our architecture can yield relatively similar results, while also having the benefit of actually producing the observable islands instead of a binary result. To further support these claims, additional tests on a larger evaluation set would need to be conducted, as the six cases discussed in this study represent a relatively limited sample size.

Regarding **RQ1**, it is clear that generating datasets consisting of one or more topologies with measurement configurations and ground truth values proved to be effective for model training. All created datasets provided an important foundation for identifying observable islands reasonably accurately, since the same algorithm was used for each of our models' data. It is possible that further work on building the ground truth values while maximizing the island size could improve the network coverage and average island sizes we obtained.

6.2. Limitations & Future Work

The proposed graph machine learning pipeline for determining observable islands is currently limited by the fact that the model's output is, in the case where observable islands can actually be found, a single configuration. In practice, real-world power networks can exhibit structural ambiguity, where multiple island configurations produce a full rank Jacobian, allowing the system to be solved. An example of

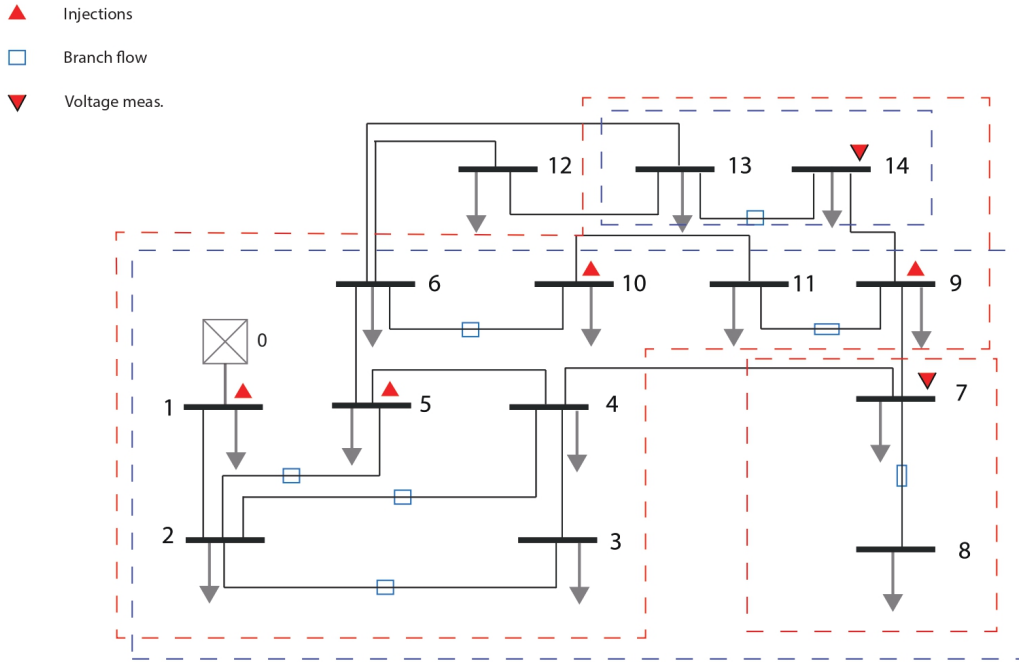


Figure 6.1: Ambiguity Case

such a case can be seen in Figure 6.1, where bus 9 represents the cut-off point separating two possible sets of islands. As a measured bus with a degree of 3, only one of its incident branches (branch 9-11) is measured. Thus, it is safe to assume that nodes 9 and 11 are guaranteed to be in the same island. However, because edges 9-7 and 9-14 lack measurements, only one of them can be grouped with bus 9, producing two valid configurations:

- $C_1 = \{\{1, 2, 3, 4, 5, 6, 9, 10, 11, 13, 14\}, \{7, 8\}\}$
- $C_2 = \{\{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11\}, \{13, 14\}\}$

In order to resolve such edge cases, future models could output multiple island configurations alongside confidence intervals, rating the best performing configuration with the highest confidence. Implementing this approach requires the creation of a specialized dataset with diverging outputs, allowing the model to compare ambiguous configurations against predefined evaluation metrics.

An additional limitation of the current methodology is the absence of real-world measurement errors in the dataset. The existing measurement values are based on a fully estimated power network with known values. In operational distribution networks, measurements can be subject to significant errors, which a robust estimator should be able to identify and bypass.

6.3. Societal Impact

The dataset generation script created for training data could prove a valuable tool for future research in power networks and artificial intelligence, not only for observability analysis, but for plenty of other problems, such as optimal sensor placement, contingency analysis or network restoration. Moreover, the proposed architecture can serve as a proof of concept for power maintenance companies to further develop a stronger model that aids faster state estimation, to alleviate the congestion problem. With better grid monitoring, the existing infrastructure could accommodate additional electricity demands or generation, which is particularly relevant as today's society relies more and more on electrical systems.

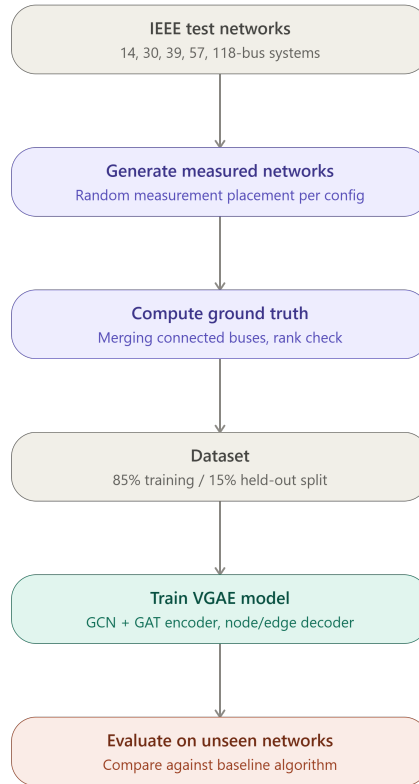


Figure 6.2: General Outline for the Proposed Method

Figure 6.2 shows a general flowchart for our GML solution to the observability analysis problem. This process yields a fully trained model that could be deployed as an additional step in the full state estimation pipeline. For power network maintenance work, the obtained observable islands could provide

additional insight into an estimation failure case, potentially speeding up the fixing process. For the end consumer, the resulting improvements could contribute to a more reliable electricity supply. Although the proposed architecture demonstrates the feasibility of addressing observability analysis using GML, further validation on operational data and testing on fully integrated pipelines would be required before direct deployment.

As for the direct misuse of this research, its potential is limited since the model does not directly operate the power network. It performs a fairly narrow technical task which gives users more information, without letting them actively affect the grid. Nevertheless, the model concerns critical infrastructure, so errors or inappropriate deployment could potentially harm distribution systems in the short run.

7

Conclusion

The purpose of this study was to provide a proof of concept for a graph machine learning model capable of identifying observable islands in a power network from its topology and available measurement configuration. Existing approaches to this problem are primarily numerical, topological and hybrid. The only previous machine learning methods, the works by Jain et al. [18, 17] use feed-forward neural networks that consider only the presence or absence of measurements and output a binary response indicating whether the whole network is observable or not. They do not exploit the graph structure of the network or incorporate richer measurement information.

The solution proposed in this thesis project addresses these limitations by introducing a graph neural network that predicts the actual set of observable islands instead of a binary response. When the electrical grid is fully observable, the model's output will reflect that by predicting the whole network as an observable island. This architecture was trained and evaluated on multiple grid sizes, showcasing its ability to generalize.

Moreover, this study presents a complete pipeline for generating a sufficiently large dataset of observable islands for training such a model. To the best of our knowledge, no such dataset or generation algorithm existed in the literature up until this point. Together, these contributions lay the foundation for future research, whether through architectural improvements or through further refinement in the dataset generation process.

The proposed solution also demonstrates an improvement over the baseline algorithm by Nucera & Gilles [31] in its observability accuracy. Although the baseline predicts larger islands with better average network coverage, the model more consistently outputs actual observable subnetworks, resulting in higher overall accuracy. At the same time, both methods produce small average deficiencies, which indicate that their predictions remain close to satisfying the observability requirement. Regarding the comparison with existing literature by Jain et al. [18, 17], our method proves to be reasonably effective in determining the full observability status, the only failed case predicting 10 out of 14 nodes for that network.

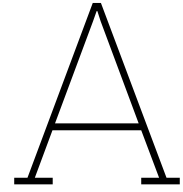
Overall, the results demonstrate that graph machine learning is a viable approach to the observable islands identification problem. While the proposed model is not yet suitable for real-world deployment, it successfully shows that this problem is solvable from the inherent graph data structure of the grid and its measurement configuration. This opens the possibility of developing more accurate and scalable graph solutions that can complement traditional observability analysis methods.

References

- [1] Ali Abur and Antonio Gómez-Expósito. *Power System State Estimation: Theory and Implementation*. CRC Press, 2004.
- [2] R.A.S. Benedito, J.B.A. London, and N.G. Bretas. “A unified algorithm for observability and redundancy analysis”. In: *2009 IEEE Bucharest PowerTech*. 2009, pp. 1–7. DOI: 10.1109/PTC.2009.5282246.
- [3] Stephen Boyd and Lieven Vandenbergh. *Convex Optimization*. Appendix C, §C.3.1. Cambridge, UK: Cambridge University Press, 2004. ISBN: 978-0-521-83378-3.
- [4] Yizong Cheng. “Mean Shift, Mode Seeking, and Clustering”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 17.8 (1995), pp. 790–799.
- [5] G.C. Contaxis and G.N. Korres. “A reduced model for power system observability: analysis and restoration”. In: *IEEE Transactions on Power Systems* 3.4 (1988), pp. 1411–1417. DOI: 10.1109/59.192947.
- [6] Mirsad Cosovic et al. “Observability Analysis for Large-Scale Power Systems Using Factor Graphs”. In: *IEEE Transactions on Power Systems* 36.5 (2021), pp. 4791–4799. DOI: 10.1109/TPWRS.2021.3057136.
- [7] Yucai Fan, Yuhang Yao, and Carlee Joe-Wong. “GCN-SE: Attention as Explainability for Node Classification in Dynamic Graphs”. In: *IEEE International Conference on Data Mining (ICDM)*. 2021.
- [8] P. K. Ghosh, S. Chatterjee, and A. Tahabilder. “Graph-Theory Based Optimal PMU Allocation Considering ZIB Effects”. In: *Instrumentation Measure Métrologie* 19.4 (2020), pp. 297–300. DOI: 10.18280/i2m.190407. URL: <https://doi.org/10.18280/i2m.190407>.
- [9] Justin Gilmer et al. “Neural Message Passing for Quantum Chemistry”. In: *Proceedings of the 34th International Conference on Machine Learning*. PMLR, 2017, pp. 1263–1272.
- [10] Murat Göl and Ali Abur. “Observability and Criticality Analyses for Power Systems Measured by Phasor Measurements”. In: *IEEE Transactions on Power Systems* 28.3 (2013), pp. 3319–3326. DOI: 10.1109/TPWRS.2012.2236367.
- [11] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. Chapter 6, §6.2.2.2: Sigmoid Units for Bernoulli Output Distributions. Cambridge, MA: MIT Press, 2016. URL: <https://www.deeplearningbook.org>.
- [12] B. Gou. “Jacobian matrix-based observability analysis for state estimation”. In: *IEEE Transactions on Power Systems* 21.1 (2006), pp. 348–356. DOI: 10.1109/TPWRS.2005.860934.
- [13] Bei Gou and Ali Abur. “A Direct Numerical Method for Observability Analysis”. In: *IEEE Transactions on Power Systems* 15.2 (May 2000), pp. 625–630. DOI: 10.1109/59.867146.
- [14] John J. Grainger and William D. Stevenson. *Power System Analysis*. McGraw-Hill, 1994. ISBN: 978-0070612938.
- [15] Zhang Haibo et al. “Observability analysis in power system state estimation based on the solvable condition of power flow”. In: *Proceedings. International Conference on Power System Technology*. Vol. 1. 2002, 234–240 vol.1. DOI: 10.1109/ICPST.2002.1053541.
- [16] S. H. Hosseini et al. “Transmission management for congested power system: A review of concepts, technical challenges and development of a new methodology”. In: *Renewable and Sustainable Energy Reviews* 38 (2014), pp. 572–580. DOI: 10.1016/j.rser.2014.05.089.
- [17] A. Jain et al. “Network observability: a solution technique using neural networks”. In: *TENCON 2003. Conference on Convergent Technologies for Asia-Pacific Region*. Vol. 3. 2003, 1007–1011 Vol.3. DOI: 10.1109/TENCON.2003.1273398.

- [18] Amit Jain, Jaeho Choi, and Joonki Min. "Power system network observability determination using feedforward neural networks". In: *Proceedings. International Conference on Power System Technology*. Vol. 4. 2002, 2086–2090 vol.4. DOI: 10.1109/ICPST.2002.1047149.
- [19] Thomas N. Kipf and Max Welling. "Semi-Supervised Classification with Graph Convolutional Networks". In: *International Conference on Learning Representations (ICLR)*. 2017.
- [20] Thomas N. Kipf and Max Welling. "Variational Graph Auto-Encoders". In: *arXiv preprint arXiv:1611.07308* (2016).
- [21] G.N. Korres and P.J. Katsikas. "A hybrid method for observability analysis using a reduced network graph theory". In: *IEEE Transactions on Power Systems* 18.1 (2003), pp. 295–304. DOI: 10.1109/TPWRS.2002.807072.
- [22] Ognjen Kundacina, Mirsad Cosovic, and Dejan Vukobratovic. "Robust and Fast Data-Driven Power System State Estimator Using Graph Neural Networks". In: *CoRR abs/2206.02731* (2022). DOI: 10.48550/arXiv.2206.02731. arXiv: 2206.02731 [cs.LG].
- [23] Ognjen Kundacina, Mirsad Cosovic, and Dejan Vukobratovic. "State Estimation in Electric Power Systems Leveraging Graph Neural Networks". In: *2022 17th International Conference on Probabilistic Methods Applied to Power Systems (PMAPS)*. IEEE, 2022. DOI: 10.1109/PMAPS53380.2022.9810559.
- [24] Ognjen Kundacina et al. "Scalability and Sample Efficiency Analysis of Graph Neural Networks for Power System State Estimation". In: *2023 International Balkan Conference on Communications and Networking (BalkanCom)*. IEEE, 2023. DOI: 10.1109/BalkanCom58402.2023.10167975.
- [25] Stuart Lloyd. "Least Squares Quantization in PCM". In: *IEEE Transactions on Information Theory* 28.2 (1982), pp. 129–137.
- [26] Yi Lu et al. "Graph-FCN for Image Semantic Segmentation". In: *arXiv preprint arXiv:2001.00335* (2020).
- [27] M. Hossain Mohammadi and Khaled Saleh. "Synthetic Benchmarks for Power Systems". In: *IEEE Access* 9 (2021), pp. 162706–162730. DOI: 10.1109/ACCESS.2021.3124477.
- [28] A. Monticelli and F. F. Wu. "Network Observability: Identification of Observable Islands and Measurement Placement". In: *IEEE Transactions on Power Apparatus and Systems* 104.5 (1985), pp. 1042–1048.
- [29] Netbeheer Nederland. *Capaciteitskaart elektriciteitsnet Nederland*. Accessed: 2026-07-21. 2026. URL: <https://www.netbeheernederland.nl/netcapaciteit-en-flexibiliteit/capaciteitskaart>.
- [30] Netbeheer Nederland. *Uitbreidingen van het net worden goed zichtbaar, wachtrijen blijven bestaan*. Netbeheer Nederland. Mar. 2025. URL: <https://www.netbeheernederland.nl/artikelen/nieuws/uitbreidingen-van-het-net-worden-goed-zichtbaar-wachtrijen-blijven-bestaan> (visited on 07/21/2026).
- [31] R.R. Nucera and M.L. Gilles. "Observability analysis: a new topological algorithm". In: *IEEE Transactions on Power Systems* 6.2 (1991), pp. 466–475. DOI: 10.1109/59.76688.
- [32] Power Systems Test Case Archive. *IEEE 14-Bus Power Flow Test Case*. https://labs.ece.uw.edu/pstca/pf14/pg_tca14bus.htm.
- [33] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. "Learning representations by back-propagating errors". In: *Nature* 323 (1986), pp. 533–536.
- [34] Franco Scarselli et al. "The Graph Neural Network Model". In: *IEEE Transactions on Neural Networks* 20.1 (2009), pp. 61–80. DOI: 10.1109/TNN.2008.2005605.
- [35] S. N. Simovic and G. S. Christensen. "Power System Observability and State Estimation". In: *IEEE Transactions on Power Systems* (1989).
- [36] Xiran Song et al. "Friend Recommendations with Self-Rescaling Graph Neural Networks". In: *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. ACM, 2022, pp. 3487–3495.

- [37] Robert Endre Tarjan. “Efficiency of a Good But Not Linear Set Union Algorithm”. In: *Journal of the ACM* 22.2 (1975), pp. 215–225.
- [38] Leon Thurner et al. “pandapower—An Open-Source Python Tool for Convenient Modeling, Analysis, and Optimization of Electric Power Systems”. In: *IEEE Transactions on Power Systems* 33.6 (2018), pp. 6510–6521.
- [39] Petar Veličković et al. “Graph Attention Networks”. In: *International Conference on Learning Representations (ICLR)*. 2018.
- [40] Felix F. Wu and A. Monticelli. “Network Observability: Theory”. In: *IEEE Transactions on Power Apparatus and Systems* PAS-104.5 (May 1985), pp. 1042–1048. DOI: 10.1109/TPAS.1985.318677.
- [41] Haibo Zhang and Kexin Han. “A Hybrid Observability Analysis Method for Power System State Estimation”. In: *IEEE Access* 8 (2020), pp. 73388–73397. DOI: 10.1109/ACCESS.2020.2987358.
- [42] Ray D. Zimmerman, Carlos E. Murillo-Sánchez, and Robert J. Thomas. “MATPOWER: Steady-State Operations, Planning, and Analysis Tools for Power Systems Research and Education”. In: *IEEE Transactions on Power Systems* 26.1 (2011), pp. 12–19. DOI: 10.1109/TPWRS.2010.2051168.



AI Use Acknowledgement

For transparency reasons, this appendix covers the use of generative artificial intelligence (AI) tools for the writing and programming parts of this thesis project, in accordance with the guidelines provided by Delft University of Technology. All ideas, both writing and code wise, analysis, discussions and conclusions are my own.

A.1. Writing

AI tools were used during the writing of this document, mainly for finding synonyms to avoid repetition and making the sentences flow better for the reader. The document's structure, the order and names of its chapters and sections, along with the sentences' core were thought of and written by me. If such tools were used, their output was never simply copied and pasted in the document, it was critically analysed and rewritten by me to fit the writing style of the project. AI tools were never used to generate talking points, to draw conclusions or to gain insight from the presented data.

A.2. Coding

In the coding part of the project, AI tools were used to speed up simple programming tasks such as finding the correct function names for the APIs used, suggesting bug fixes, producing a general template for the presented plots or sequencing two separate algorithms. All the generated code was read, tested and subsequently changed by me before being used in the experiments of this project.