SAPIENZA
UNIVERSITÀ DI ROMA

Smart water management using GNN through a semi-supervised learning approach: Design, implementation, and experimentation

Faculty: Information Engineering, Informatics, and Statistics
Department of Computer, Control, and Management Engineering (DIAG)
Course: Master's Degree in Engineering in Computer Science

Candidate
Francesco Giuseppe Crino'
ID number : 1954737

Advisor
Prof. Ioannis Chatzigiannakis

Co-Advisor
Prof. Francesca Cuomo

Academic Year 2021/2022

Thesis defended on 25/10/2022
in front of a Board of Examiners composed by:

Prof. Alessandro De Luca (chairman)
Prof. Roberto Capobianco
Prof. Ioannis Chatzigiannakis
Prof. Antonio Cianfrani
Prof. Febo Cincotti
Prof. Danilo Comminiello
Prof. Marco Console
Prof. Francesco Delli Priscoli
Prof. Giorgio Grisetti
Prof. Domenico Lembo
Prof. Manuela Petti

Smart water management using GNN through a semi-supervised learning approach: Design, implementation, and experimentation
Master's thesis. Sapienza – University of Rome

This thesis has been typeset by LaTeX and the Sapthesis class.

Author's email: crino.1954737@studenti.uniroma1.it

To my family,
who have always believed in me and have always supported me.
To my friends,
who are always by my side despite the distance.

Abstract

Today water is near to becoming a luxury resource and at the same time in Europe the 27% of drinking water fed in a Water Distribution System (WDS) is lost and in Italy we lost about 40% of drinking water during transportation. For that reason, smart water management is one of the biggest research topics and much effort is being made to avoid water loss in WDS. Monitoring a WDS is not an easy task because it is practically not possible to monitor all its nodes. This is because a WDS may have thousands of nodes and it can extend across a very large area including rural areas without an internet connection.

The main goal of this thesis is to design a GNN-based model to make predictions on the flow in the WDS and the nodal heads for all its nodes monitoring only a small subset of nodes. Also we want to define a criterion to choose the best subset of nodes to monitor to obtain the best predictions as possible.

Starting from the promising work of Lu and Sela [4], we have developed a GNN model to perform State Estimation process using a semi-supervised machine learning approach and designing the models' loss function following a physics-informed machine learning strategy.

The GNN is composed of three main blocks: encoder, processor, and decoder. The encoder formats the input and initializes the latent variables, the decoder decodes the processor output as meaningful output, i.e. heads values, and the processor is where the latent states are updated through message passing.

Since we want to find some dependencies between the performance metrics used to evaluate the predictions of the model and the centrality measures of the nodes of the WDS graph model, it is crucial to design an appropriate graph model to represent the WDS.

To model a WDS as a graph we have designed a graph model that represents not only the topology of the WDS but also its physical properties like diameter, length, and roughness of its pipes and demand and head of its nodes. Such a graph model has the weight of the edges equal to the loss coefficient of the modeled pipe and its nodes have, as features, the head and the demand of the WDS modelled nodes.

Finding these dependencies means being able to choose the nodes to monitor just by looking at the topology and the physical properties of the WDS.

In this thesis, all the used algorithmic methods and models are clearly defined and presented and the most important design choices are discussed. In the end, the strategy to find the relations between performance metrics and centrality measures is presented and the observed strength and weaknesses of the model are discussed.

Contents

1 Introduction **1**
1.1 Challenges . . . 2
1.2 Research Questions . . . 2
1.3 Outline . . . 3

2 GNN for State Estimation in a WDS **4**
2.1 WDS digitization . . . 4
2.1.1 EPANET . . . 5
2.1.2 Water Network Tool for Resilience (WNTR) . . . 7
2.2 WDS as a graph . . . 8
2.2.1 Graph Model . . . 8
2.2.2 Centrality Measures . . . 10
2.3 Data Generation . . . 12
2.4 Graph Neural Networks . . . 14
2.4.1 GNN model for State Estimation on WDS . . . 16
2.5 Training of the GNN model . . . 18
2.5.1 Semi-supervised learning technique . . . 18
2.5.2 Physics-Informed machine learning approach . . . 19
2.5.3 GNN model loss functions . . . 19
2.6 Model performance metrics . . . 21

3 Experimentation and results **23**
3.1 WDS ASnet2 and Anytown . . . 23
3.1.1 ASnet2 . . . 24
3.1.2 Anytown . . . 26
3.2 Training settings . . . 27
3.2.1 GNN parameters . . . 27
3.2.2 Training parameters and environment . . . 28
3.3 Model prediction performances with 1 observed node . . . 29
3.4 Centrality Measures - Performance metrics correlation . . . 32

4 Conclusions **40**

Bibliography **43**

Chapter 1

Introduction

Water covers 70% of our planet, and it is easy to think that it will always be plentiful. However, only 3% of the world's water is fresh water, and two-thirds of that is tucked away in frozen glaciers or otherwise unavailable for our use. As a result, some 1.1 billion people worldwide lack access to water, and a total of 2.7 billion find water scarce for at least one month of the year. Because of that, one of the most important challenges for the cities of the future is smart water management.

One of the big challenges in smart water management is to design, develop, and deploy a smart water metering system. Most of the developed smart water metering systems follow a cloud-centric paradigm where all the data are collected and processed centrally using cloud services. Very interesting is the solution developed by Amaxilatis, Chatzigiannakis at al. [1] that uses the fog computing paradigm to provide a system where the computational resources already available throughout the network infrastructure are utilized to facilitate greatly the analysis of fine-grained water consumption data collected by the smart meters, thus significantly reducing the overall load to network and cloud resources.

In the work of Zecchini et al. [2], a smart metering system based on IoT is used to monitor the water consumption of a building during the COVID-19 restrictions period. The data collected from a Smart Water Grid are utilized to examine the impact of human actions on the consumption of water and the performance of the water distribution network within a university campus.

One of the core problems of water management is the loss of water in the Water Distribution Systems (WDS). Because of that, one of the most important challenges for the cities of the future is the smart water management. More in particular one of the core problem of water management is the loss of water in the Water Distribution Systems (WDS).

The Figure 1.1, taken from the overview of the European drinking water and waste water sectors by EurEau [3], shows the percentage of drinking water lost in WDS by EU countries. In Europe, we lost about 27% of drinking water fed into a WDS while in Italy we reach 40%. Since we are talking about a loss of 40% of drinking water lost during distribution in water distribution networks this is a very huge problem that needs to be solved as soon as possible.

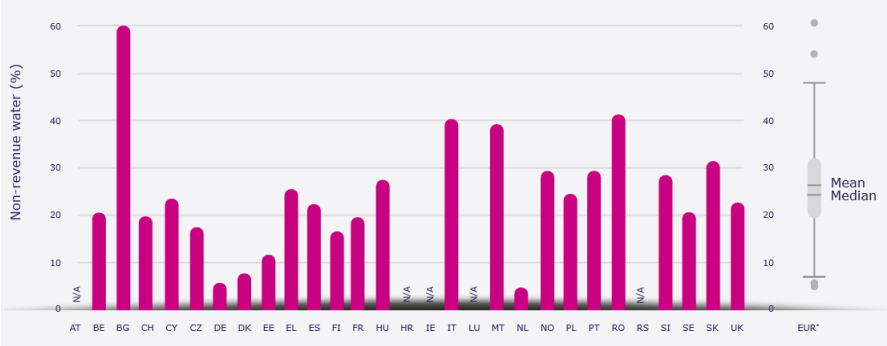

Figure 1.1. Percentage of non-revenue water of the European countries

The main goal of that thesis is to design algorithmic methods and protocols to smart manage a Water Distribution System. More in particular we will see how to model a WDS as a graph and then how to use Graph Neural Networks (GNN) models to perform State Estimation process on a WDS and we will investigate how to choose the best subset of nodes to observe in order to have the best predictions.

1.1 Challenges

To reach the final goal to use GNNs to perform State Estimation process on a WDS represented as a graph observing the best subset of nodes we can follow a clear and simple path: Modelling the Water Distribution System as a Graph, find the best nodes to observe to perform State Estimation and use a Graph Neural Network model with a semi-supervised learning approach.

Each of the three tasks of that path represents a challenge, in particular in this work we will focus on the following challenges:

- Graph Modelling: We want to model a WDS as a graph representing not only the topological aspects of the WDS but also its physical properties as roughness, length and diameter of its pipes.
- Junctions Choice: We want to choose the best subset of observed juncstions to perform State Estimation. To do that we will investigate the correlation between prediction results and centrality measures of the graphs nodes.
- GNN Model: In order to perform State Estimation we will use GNN models using a semi-supervised learning approach and a physics-informed machine learning to define the loss functions of the model.

1.2 Research Questions

From a research point of view, "smart water management" is nowadays one of the biggest research topics. In fact, it is considered a core aspect to manage the smart and green cities of the future.

As said in the previous paragraph the first thing to do is model the Water Distribution System, we want to manage, as a graph. To reach this goal we need to answer the following question:

- What is the best type of graph to model a WDS?
- Is it necessary to assign weights to nodes and edges?
- There exists a way to include the pipes' physical properties in the graph model?

Now we need to build the GNN model to perform State Estimation process using a semi-supervised learning approach and insert the hydraulic physical laws in the models loss functions answering to the following question:

- Which structure to use for the GNN model?
- How can we include physical knowledge in loss functions?
- Which are the best nodes to monitor to perform State Estimation?

In order to answer to this last question we need to answer to the following question, that is also the main question of that work:

- There exists a correlation between the State Estimation performance metrics and the centrality measures of the graphs nodes?

Finding an answer to this last question means to find a clear strategy to choose the nodes to use to perform State Estimation.

1.3 Outline

In the next chapter 2, the EPANET software will be presented as well ad the its compatible Python package WNTR. In section 2.2, we will see how to model a Water Distribution System as a graph taking into account its physical properties and in section 2.3, we will see how to generate a sufficiently large data-set from a WDS topology to train, validate and test the model. Before proceeding with the definition and the training of the model in section 2.5, we will resume the most important features of a GNN explaining its main operations and structure in section 2.4. In section 2.5 we will introduce the concept of semi-supervised learning and it will be shown in detail how to include the hydraulic domain knowledge in the models' loss functions following a physics-informed machine learning approach. The performance metrics used to evaluate the model predictions we will clearly defined in 2.6. The practical part of this work is presented and discussed in Chapter 3. At first the two used WDS, Anytown and ASnet2, are presented in section 3.1.1, and by showing their main properties. In section 3.2, we will see some statics about the training phase as well as how the training parameters are set. At the end of chapter 3, the obtained results are presented and discussed. Finally, in chapter 4 the methods and the strategies used all along this work will be discussed as well as the obtained results.

Chapter 2

GNN for State Estimation in a WDS

In this chapter, all the algorithmic methods and protocols used in this work will be presented from a theoretical point of view. It will be explained how to create a sufficiently large data set to train, evaluate and test the model starting from a WDS. We will see in detail the GNN-based model used to perform SE process and how we can use it with a semi-supervised learning approach. It will be presented the technique of physics-informed machine learning and we will see in detail how to include the hydraulic physics laws into the GNN model loss functions. Finally, the evaluation metrics used to evaluate the model will be clearly defined.

2.1 WDS digitization

Usually, treated water is conveyed to service reservoirs for distribution to consumers. In urban systems, a water transmission system may also be necessary to convey water from a treatment plant to several service reservoirs located at different convenient points in the city. In some cities, there may be many sources and water treatment plants supplying service reservoirs and water distribution systems. These distribution systems may be separate or linked [5]. Both water transmission systems and water distribution systems are networks of pipes. However, water transmission systems have a tree-like configuration, whereas water distribution systems usually have loops.

Water distribution systems contain several components. Each network is unique in source, layout, topography of the service area, pipe material, valves and meters, and consumer connections. The layout of a distribution network depends on the existing pattern of streets and highways, the topography of the service area may be flat or uneven. In uneven terrain, booster pumps may be necessary for pumping water to high areas within the network. Similarly, it may be necessary to provide pressure-reducing valves for areas with lower elevation to reduce pressure. Pipes in the distribution network may be of cast or ductile iron, mild steel, concrete and prestressed concrete, asbestos cement, polyvinyl chloride (PVC), and high-density

polyethylene. Pipes may be unlined or lined with cement mortar. Valves are provided in distribution systems to control flow, isolate pipelines during repairs, and in general manage the behaviour of the WDS.

In our case, we can think about a WDS as a set of nodes connected by links where the edges and the nodes can be of different natures and different properties.

2.1.1 EPANET

In order to work with a WDS digital model, we will use EPANET [6] that is is a computer program that performs an extended period simulation of hydraulic and water quality behavior within pressurized pipe networks. In particular, EPANET tracks the flow of water in each pipe, the pressure at each node, the height of water in each tank, and the concentration of a chemical species throughout the network during a simulation period comprised of multiple time steps.

EPANET uses various types of objects to model a distribution system, let's see these objects and their properties more in detail:

- **Nodes**
  - **Junctions**: Points in the network where links join together and where water enters or leaves the network.

    $$j : \{junctionID, elevation, demand, demandPattern, X - Coordinate, Y - Coordinate\}$$

  - **Reservoirs**: Reservoirs are nodes that represent an infinite external source or sink of water to the network.

    $$r : \{reservoirID, totaHead, headPattern, X - Coordinate, Y - Coordinate\}$$

  - **Tanks**: nodes with storage capacity, where the volume of stored water can vary with time during a simulation.

    $$t : \{tankID, elevation, initLevel, minLevel, maxLevel, minVolume, volumeCurve, X - Coordinate, Y - Coordinate\}$$

- **Links**
  - **Pipes**: links that convey water from one point in the network to another. EPANET assumes that all pipes are full at all times. Flow direction is from the end at higher hydraulic head to that at lower head.

    $$p : \{startNode, endNode, diameter, length, roughness, status\}$$

  - **Pumps**: Links that impart energy to a fluid thereby raising its hydraulic head.

$$pu : \{startNode, endNode, curve\}$$

  - **Valves**: links that limit the pressure or flow at a specific point in the network.

$$va : \{startNode, endNode, diameter, setting, status\}$$

- **Time pattern**: Collection of multipliers that can be applied to a quantity to allow it to vary over time. Nodal demands, reservoir heads, pump schedules, and water quality source inputs can all have time patterns associated with them.

```
[TIMES]
 Duration             168:00
 Hydraulic Timestep   0:15
 Quality Timestep     0:05
 Pattern Timestep     1:00
 Pattern Start        0:00
 Report Timestep      1:00
 Report Start         0:00
 Start ClockTime      0:00:00
 Statistic            NONE
```

Figure 2.1. Times parameters of an EPANET .inp file to have a simulation of 168 hours with report each 1 hour

- **Curves**: Curves are objects that contain data pairs representing a relationship between two quantities. For example, a Pump Curve represents the relationship between the head and flow rate that a pump can deliver at its nominal speed setting and a Volume Curve determines how storage tank volume varies as a function of water level.
- **Controls**: Controls are statements that determine how the network is operated over time. They specify the status of selected links as a function of time, tank water levels, and pressures at select points within the network.

A complete description of the EPANETs' objects is available in the paragraph 6 of the EPANET user manual [6].

2.1.2 Water Network Tool for Resilience (WNTR)

As just said, EPANET is a complete software through which it is possible to perform all kinds of hydraulic simulations on a WDS. The problem with EPANET is that it is a stand-alone software, which means if we want to use the output of a simulation we need to store them into a file and then open this file to read and manipulate the results.

The Water Network Tool for Resilience (WNTR) [7] is an EPANET compatible Python package that allows us to perform hydraulic simulations on a WDS, as well as EPANET, but inside Python software.

Using WNTR in Python software is the same as using any other Python package. At first, we need to import WNTR, and then we can start building the Water Network Model that can be created from scratch or built directly from an EPANET inp file. Once created we can perform different actions on a WaterNetworkModel: Add or remove nodes and links, modify options, time series and retrieve all the information written in the inp file. When all the WaterNetworkModel parameters are set we can perform hydraulic simulation using the WNTR EpanetSimulator. This can be used The EpanetSimulator can be used to run EPANET 2.00.12 Programmer's Toolkit or EPANET 2.2.0 Programmer's Toolkit. It is possible to perform hydraulic simulation based on Driven Demand or Pressure Driven Demand analysis.

```
import wntr

inp_file = 'wds.inp'
wn = wntr.network.WaterNetworkModel(inp_file)
results = wntr.sim.EpanetSimulator(wn).run_sim(version=2.0)
```

Figure 2.2. Snippet of code showing how to import WNTR package, create a WaterNetworkModel importing an EPANET inp file, and perform a simulation with EPANET 2.00.12 Programmer's Toolkit

The figure 2.2 shows how to import the WNTR package and build a WaterNetworkModel from an EPANET inp file and perform a simulation.

Simulation results are stored in a results object which contains:

- Timestamp when the results were created
- Network name
- Node results
- Link results

The node's and link's results are dictionaries of pandas DataFrames. The keys of the dictionaries are the simulated properties:

- Nodes: $[demand, head, pressure, quality]$
- Links: $[quality, flowrate, headloss, velocity, status, setting, friction\_factor, reaction\_rate]$

Each key has a Pandas DataFrame, as value, of shape $(time\_sim/report\_time, n\_nodes)$ for nodes and $(time\_sim/report\_timestamp, n\_links)$ for links. For example, doing a simulation of 168 hours of a network with a network composed of 51 nodes and a report timestamp of 1 hour, we can access the simulated heads by $results.node['head']$ that will be a DataFrame with shape (168, 51) with the nodes as column labels where the column n contains the 168 reported heads for the node n.

2.2 WDS as a graph

In the previous paragraph 2.1 we have seen how to create a digital model of a WDS using EPANET and how to perform hydraulic simulations in Pytho through WNTR package. The next step is to model a WDS as a graph to be able to exploit all its topology properties and compute the centrality measures. The challenge here is to model the WDS as a graph representing not only the topology aspects of the network but also its physical properties such as the nature of the nodes, roughness, diameter, and length of the pipes.

2.2.1 Graph Model

In order to create a graph model of a WDS we need at first to build the WDS WaternetworkModel, as described in figure 2.2, and exploit the WaterNetworkModel properties to create our custom graph model. We will model the WDS as a graph assigning the following weights to edges and nodes:

- nodes' properties
  - simulated head
  - simulated demand

- Links' properties: Loss coefficient c_p
  - Loss coefficient for a pipe p:

$$c_p = \frac{10.67 \times l_p}{r_{HW,p}^{1.852} \times d_p^{4.871}} \tag{1}$$

  - Loss coefficient for a pump p: $cp = 0$

In equation 1, p is the pipe we are computing its loss coefficient, l_p is the length of the pipe p, d_p is the diameter of the pipe p and $r_{HW,p}$ is the dimensionless Hazen-Williams coefficient.

The final graph model will be a graph $G(V, E)$ where :

- V is the set of vertices representing the WDS nodes where

$$\forall v \in V : v = \{head, demand\} \tag{2}$$

- E is the set of edges representing the WDS links where

$$\forall e \in E : e = \{c_e\} \tag{3}$$

In equation 3, c_p is the loss coefficient of the pipe p that corresponds to the edge e and in equation 2 head and demand are the ones simulated for the node n that corresponds with vertex v.

Now that the model is well-defined, we can use the NetworkX Python package to build the graph model. At first, it is needed to build the WaterNetworkModel of a WDS using WNTR package, then we can create an empty undirected graph, perform the hydraulic simulation using EpanetSimulator of WNTR, as shown in figure 2.2 and then iterate over the WaterNetworkModel.

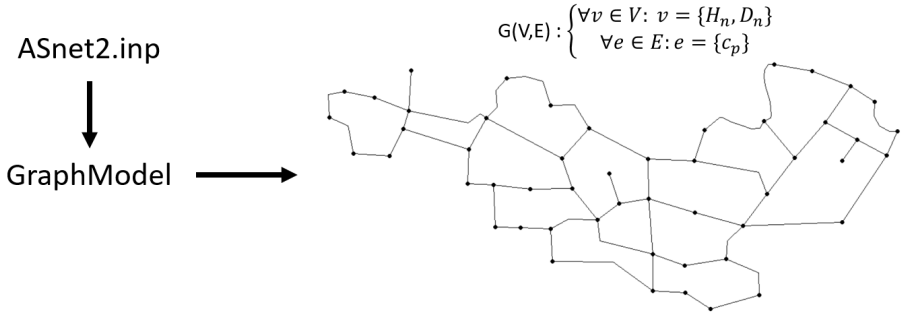

Figure 2.3. WDS Graph modelling process

Figure 2.3 shows the graph modeling process for a WDS described above. In particular, in the image we have the GraphModel which is a custom Python class that receives an EPANET inp file, builds the graph model, and computes the centrality measures.

2.2.2 Centrality Measures

As said in the previous paragraph and as shown in figure 2.3, the class GraphModel takes as input an EPNAET inp file, builds the graph model of the WDS, and then computes the following centrality measures on the graph:

- *Degree Centrality*: The degree centrality for a node v is the fraction of nodes it is connected to.
- *Closeness centrality*: Closeness centrality of a node u is the reciprocal of the average shortest path distance to u over all n-1 reachable nodes.

$$C(u) = \frac{n-1}{\sum_{v=1}^{n-1} d(u,v)} \tag{4}$$

  where $d(u,v)$ is the shortest-path distance between v and u, and n-1 is the number of nodes reachable from u.
- Eigenvector centrality: Computes the centrality for a node based on the centrality of its neighbors. The eigenvector centrality for node i is the i-th element of the vector defined by the equation $Ax = \lambda x$. Where A is the adjacency matrix of the graph G with eigenvalue λ.

- *Katz centrality*: Katz centrality computes the centrality for a node based on the centrality of its neighbors. It is a generalization of the eigenvector centrality. The Katz centrality for node i is:

$$x_i = \alpha \sum_j A_{ij} x_j + \beta \quad with \quad \alpha \leq \frac{1}{\lambda_{max}} \tag{5}$$

  Where A is the adjacency matrix of the graph G with eigenvalue λ.
- *Current flow closeness centrality*: It is variant of closeness centrality based on effective resistance between nodes in a network. This metric is also known as information centrality.

- *betweenness centrality*: betweenness centrality of a node v is the sum of the fraction of all-pairs shortest paths that pass through v:

$$c_B(v) = \sum_{s,t \in V} \frac{\sigma(s,t|v)}{\sigma(s,t)} \tag{6}$$

  where V is the set of nodes, $\sigma(s,t)$ is the number of shortest (s,t)-paths, and $\sigma(s,t|v)$ is the number of those paths passing through some node v other than s,t.

- *Current flow betweenness centrality*: It uses an electrical current model for information spreading in contrast to betweenness centrality which uses shortest paths. [8]

- *Approximate current flow betweenness centrality*: Approximates the current-flow betweenness centrality within absolute error of epsilon with high probability.

- *Communicability betweenness centrality*: It measure makes use of the number of walks connecting every pair of nodes as the basis of a betweenness centrality measure. Let $G = (V, E)$ be a simple undirected graph with n nodes and m edges, and A denote the adjacency matrix of G. Let $G(r) = (V, E(r))$ be the graph resulting from removing all edges connected to node r but not the node itself.The adjacency matrix for $G(r)$ is $A + E(r)$, where $E(r)$ has nonzeros only in row and column r. The subraph betweennes sof a node r is:

$$\omega_r = \frac{1}{C} \sum_p \sum_q \frac{G_{prq}}{G_{pq}}, \quad p \neq q, \quad q \neq r \tag{7}$$

  where $G_{prq} = (e^A_{pq} - (e^{A+E(r)})_{pq}$ is the number of walks involving node r, $G_{pq} = (e^A)_{pq}$ is the number of closed walks starting at node p and ending at node q, and $C = (n-1)^2 - (n-1)$ is a normalization factor equal to the number of terms in the sum. The resulting ω_r takes values between zero and one.

- *Load centrality*: The load centrality of a node is the fraction of all shortest paths that pass through that node.

- *Subgraph centrality*: Subgraph centrality of a node n is the sum of weighted closed walks of all lengths starting and ending at node n. The weights decrease with path length.

- *Second order centrality*: The second order centrality of a given node is the standard deviation of the return times to that node of a perpetual random walk on G. Lower values of second order centrality indicate higher centrality.

- *Harmonic centrality*: Harmonic centrality of a node u is the sum of the reciprocal of the shortest path distances from all other nodes to u.
- *Pagerank*: It computes a ranking of the nodes in the graph G based on the structure of the incoming links.

All these centrality measures are computed and returned by the get_centralities() method of the GraphModel class.

```
from models.graph_model import GraphModel

inp_file = 'ASnet2.inp'
gm = GraphModel(inp_file)
df_cent = gm.get_centralities()
```

Figure 2.4. Snippet of code of how to use GraphModel to compute and retrieve the centrality measurures

Image 2.4 shows how to use the GraphModel class to retrieve the centrality measures of the graph model built on the "ASnet2.inp" file. The centrality measures are stored in a DataFrame of shape (nNodes, 15) with the nodes' ids in the first columns and the centrality measures in the other columns.

2.3 Data Generation

In this section, it will be shown how to generate a dataset sufficiently large to train, validate and test the GNN model.

Before seeing how to generate the data, let's clearly define the structure we want for the data.

Consider a WDS with n nodes and m pipes; let $N = (1, ..., n)$ and $M = (1, ..., m)$ denote the node set and the pipe set, respectively. We want to obtain three dimensional vectors A, B, and U such as:

- A: is a three dimensional structure of dimension $(nSims \times nTime, nLinks, 3)$
  - $A[i][j][0]$ = Start node of link $j \in M$
  - $A[i][j][1]$ = End node of link $j \in M$
  - $A[i][j][2]$ = Loss coefficient C_j of link $j \in M$ (same of 2.2.1)

- B: is a three dimensional structure of dimension $(nSims \times nTime, nNodes, 4)$
  - $B[i][j][0]$ = Demand indicator of node $j \in N$
  - $B[i][j][1]$ = Demand of node $j \in N$
  - $B[i][j][2]$ = Head indicator of node $j \in N$
  - $B[i][j][2]$ = Head of node $j \in N$
- U: is a three dimensional structure of dimension $(nSims \times nTime, nNodes, 1)$
  - $U[i][j]$ = Head of node $j \in N$

The nodes' features "Demand indicator" and "head indicator" are used to say whether a node is a junction (demand indicator = 1) and to specify the nodes whose head is known (head indicator = 0), these last ones are the observed nodes.

To start generating data we need to create a WaterNetworkModel passing as input the EPANET inp file of the WDS we want to work with. Once having created the WaterNetworkModel, it is possible to start collecting data by filling in A, B, and U as described above.

To collect data into A, it is possible to iterate over all the links of the WDS retrieving the start node, the end node, and the loss coefficient.

In order to obtain the heads and demands to put into B and U, it is necessary to perform a hydraulic simulation on the WDS. We can perform the hydraulic simulation through the WNTR EpanetSimulator and then retrieve the output values of demand and head.

Since we need a large dataset that we can split into train, validation, and test set, we need to perform multiple hydraulic simulations for a long enough time. In our application, to have a large enough dataset, we have performed $nSim = 256$ simulation of 168 hours with a report timestamp of one hour. Since for each simulation we have an output of $nTimes = 168$ entries, at the end of the 256 simulation we will have $nSim \times nTimes = 43264$ entries.

Furthermore, we want that each of the 256 hydraulic simulations will not produce the same outputs. To let different simulations produce different outputs, it is possible to make some modifications to the WDS topology before starting a new simulation. The modifications we want to apply to the WDS topology are not such that the topology will be upset instead we will apply small changes. Before starting the data generation process, we will find the "removable links" of the WDS, These are links that are connected to nodes connected with more than one link, and for that reason, these links can be safely removed. The removable links are placed in an array of dimension $nSim$ in random order and, during the data generation process, before starting the i^{th} simulation the i^{th} removable link is deleted. Finally, the $nSim$ simulation will work on different WDS topologies producing different outputs.

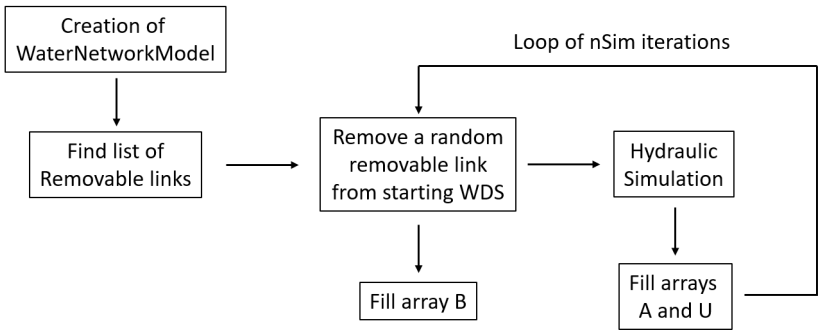

Figure 2.5. Schema of the data generation process

The schema in figure 2.5 reports all the core steps of the data generation process described above.

2.4 Graph Neural Networks

One of the core aspects of this work is to use a Graph Neural Network based model to perform State Estimation process on a WDS.

In the context of Smart Water Management and in particular, for WDS management, different neural network architectures have been proposed in the past.

Fully connected NNs were proposed to detect and identify pipe bursts from transient pressure waves [10]. A fully connected neural network consists of a series of fully connected layers. A fully connected layer is a function from R^m to R^n [11]. Each output dimension depends on each input dimension. In fully connected layers, the neuron applies a linear transformation to the input vector through a weights matrix. A non-linear transformation is then applied to the product through a non-linear activation function. A such neuron is called Perceptron.

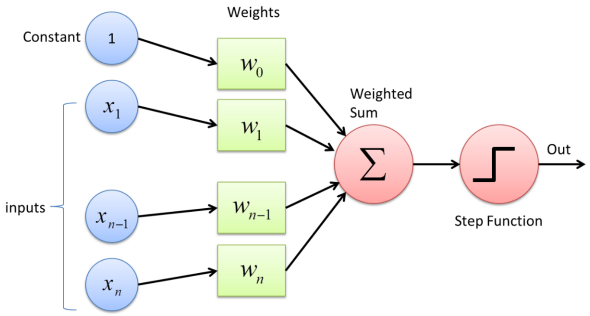

Figure 2.6. Structure of a Perceptron unit that transforms the input vectors as weight matrix, compute the sum of the weights and then applies a step function as activation function

Convolutional Neural Networks (CNN) models were trained as time-series classifiers to distinguish leaks from normal signals collected by piezoelectric accelerometers [12]. The goal of a CNN is to reduce the input vectors into a form that is easier to process, without losing features that are critical for getting a good prediction. To achieve this goal a convolutional layer performs a convolution that is effectively a sliding dot product, where the kernel shifts along the input matrix, and we take the dot product between the two as if they were vectors.

The majority of the previous works focus on anomaly detection and rely on the supervised learning approach. One of the biggest problems of performing State Estimation on a WDS is that we are not able to observe the entire set of its nodes but also a small subsets. Since we have this lack of information, we need to use other available information and in our case the topology properties of the WDS. Having a graph model of the WDS is very important because it can be used as input for a Graph Neural Network.

The main idea of a Graph Neural Network is that nodes in a graph represent objects or concepts, and edges represent their relationships [13]. Thus, we can attach a state to each node that is based on the information contained in its neighborhood. The core process of a GNN is the message propagation process that is performed by the so called Propagation Module [14]. The propagation module is used to propagate information between nodes so that the aggregated information could capture both feature and topological information. In propagation modules, the convolution operator is usually used to aggregate information from neighbors. The information exchange mechanisms can be abstracted as a forward path containing two phases: a message-passing phase, which updates the latent node states based on messages/information from neighbor nodes, and a readout phase, which decodes the latent states to the output feature space.

2.4.1 GNN model for State Estimation on WDS

We want to build a GNN-based model that will be able to make predictions on the head H at all nodes and flows Q in all pipes of a WDS of which we know network topology, pipe characteristics, nodal demands, and restricted area head measurements.

The input of the GNN model is an input interaction graph, $G_I = (V, E)$, where

- $V = (V_I)_{I \in N}$ with N sets of WDS nodes, and $V_I \in R^4$, $V_i = [di_i, d_i, hi_i, H_i]$ where:
  - di_i: Junction indicator

$$di_i = \begin{cases} 1 & if\ \ i\ \ is\ \ a\ \ junction \\ 0 & otherwise \end{cases} \tag{8}$$

  - d_i: Demand at node i if it is a junction, 0 otherwise
  - hi_i: Head indicator

$$hi_i = \begin{cases} 0 & if\ \ i\ \ is\ \ an\ \ observed\ \ node \\ 1 & otherwise \end{cases} \tag{9}$$

  - H_i: Head of node i if it is observed, 0 otherwise
- $E = (E_p)_{p \in M}$ with M sets of WDS links, and $E_p \in R^3$, $V_i = [startNode_p, endNode_p, c_p]$

The output interaction graph $G_O(H, Q)$ represents the states of the interaction graph, where $H = (H_i)_{i \in N}$ stands for the heads at all nodes, and $Q = (Q_p)_{p \in M}$ represents the flows in all pipes.

The overall structure of the GNN model is the one of figure 2.7 and consists of three blocks: Encoder, Processor, and Decoder.

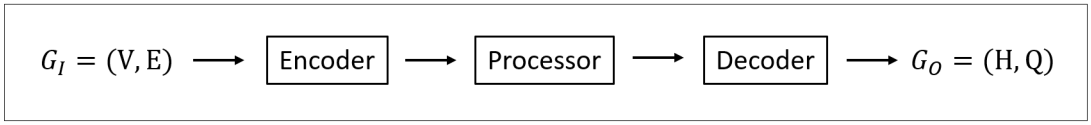

Figure 2.7. Overall structure of the GNN model

Let's see more in detail what append in the three GNN blocks:

- Encoder: Transforms input space to an abstract vector space, taking into account the WDS topology. At first the input vectors are normalized to obtain data of same order of order of magnitude and then the latent states are initialized
  - Input Normalization: $G_I = (V, E) \leftarrow G_I = (v, e)$ where:
    - $v_i = \frac{V_i - \mu_V}{\sigma_V}$ with μ_V and σ_V mean and standard deviation of V
    - $e_p = \frac{E_p - \mu_E}{\sigma_E}$ with μ_V and σ_V mean and standard deviation of E
  - Latent variables initialization: Performs input embedding in latent states, i.e. nodal heads are embedded in $X \in R^{d_x}$ with d_x dimension of latent states. All X^0 variables are initialized to zeroes vectors.
- Processor: This is where the message propagation process takes place. The processor iteratively updates the latent states at each node by passing and aggregating information to and from neighboring nodes using the input interaction graph. In the k^{th} update layer for each node i, messages from incoming ϕ^k_{in}, outgoing ϕ^k_{out} and self-loops ϕ^k_{loop} pipes are computed based on the latent state at node i, X^k_i, the latent state of its neighbours, and the properties of its connected pipes as follow:

$$\phi^k_{in} = \sum_{j \in N_{i,j}} \Phi^k_{in,\theta}(X^k_i, e_{(i,j)}, X^k_j) \tag{10}$$

$$\phi^k_{out} = \sum_{j \in N_{i,j}} \Phi^k_{out,\theta}(X^k_j, e_{(i,j)}, X^k_i) \tag{11}$$

$$\phi^k_{loop} = \Phi^k_{loop,\theta}(X^k_i, e_{(i,j)}) \tag{12}$$

Where $\Phi^k_{in,\theta}$, $\Phi^k_{out,\theta}$, and $\Phi^k_{loop,\theta}$ are Multi Layer Perceptron (MLP).

For each node i the three messages $\phi^k_{in,\theta}$, $\phi^k_{out,\theta}$, and $\phi^k_{loop,\theta}$ and the node inputs v_i are aggregated using another MLP Ψ^k_i computing the aggregated message ψ^k_i that is added to the latent variable of the previous layer to update the current layer latent variable as follow:

$$\psi^k_i = \Psi^k_i(X^k_i, v_i, \phi^k_{in,\theta}, \phi^k_{out,\theta}, \phi^k_{loop,\theta}) \tag{13}$$

$$X^{k+1}_i = X^k_i + \alpha \psi^k_i \tag{14}$$

where α is an hyperparameter of the model

- Decoder: After each processor update, latent state X^{k+1} is decoded into the meaningful output state, i.e., nodal heads H_i^{k+1} as follow:

$$H_i^{k+1} = D_\theta^k(X_i^{k+1}) \tag{15}$$

with D_θ^k MLP parametrized by θ. The output state at the last update layer $H\hat{}k$ is the final output of the model.

2.5 Training of the GNN model

Until now we have seen how to model a WDS as a graph, and how to build the dataset to train, validate, and test the GNN model. The GNN model has been presented explaining its structure, how the information is transmitted among its nodes, how the latent variables are updated, and finally how the latent variables are used to produce a meaningful output prediction.

In this section, we will see how to train the GNN neural network to produce the best predictions as possible considering the fluid dynamic physical laws and the particular properties of a WDS.

2.5.1 Semi-supervised learning technique

As already discussed, one of the core problems to face to perform WDS management is the lack of information due to the fact that we can monitor only a small subset of nodes. Due to this lack of information, as we have already discussed, it is convenient to use a GNN model through which we can exploit the topological properties of the WDS, and it is also necessary to apply a semi-supervised learning approach.

Semi-supervised learning is an approach to machine learning that combines a small amount of labeled data with a large amount of unlabeled data during training. Semi-supervised learning falls between unsupervised learning (with no labeled training data) and supervised learning (with only labeled training data). The semi-supervised learning comes from an assumption called semi-supervised smoothness assumption" which says: If two points x_1, x_2 in a high-density region are close, then so should be the corresponding outputs y_1, y_2 [15]. This assumption implies that if two points are linked by a path of high density (e.g., if they belong to the same cluster), then their outputs are likely to be close. In the literature, there are cases where using a semi-supervised learning approach the accuracy of the model is better than using a supervised approach.

To perform State Estimation on WDS using a semi-supervised learning approach is almost the only way since, we cannot observe all the nodes of the WDS and apply an unsupervised learning technique is very inefficient in this case.

To apply a semi-supervised approach in our laboratory test, we will simulate the condition to have only a small subset of observed nodes by setting the head indicator during the data generation process as described in section 2.3. If the head indicator

is set to 0, the node is considered to be observed, and its head is stored in array B. On the contrary, if the head indicator is set to 1, the node is considered to not be observed, and its head is not stored in B. Finally, we will have a small subset of monitored nodes $N_H \subset N$, with N set of nodes of WDS, whose dimension $n_H << n$, from which the model will learn and then it will be able to make predictions on the entire set of node. This is exactly the definition of the State Estimation process.

2.5.2 Physics-Informed machine learning approach

Traditional data-driven learning approach ignores valuable domain knowledge, thus failing to satisfy the underlying physical laws and potentially yielding inconsistent solutions. To avoid physic inconsistent predictions, it is needed to integrate physical laws and domain knowledge in the model training process, which can, in turn, provide physically consistent solutions and avoid overfitting.

This technique to build a machine learning model including physics knowledge is known as physics-informed machine learning.

The leading motivation for developing these algorithms is that such prior knowledge or constraints can yield more interpretable ML methods that remain robust in the presence of imperfect data, such as missing or noisy values or outliers, and can provide accurate and physically consistent predictions, even for generalization tasks [16].

In the GNN model proposed here, a physics-informed machine learning technique has been applied inserting domain knowledge into the training phase by customizing the loss functions. In particular, the loss functions of the GNN model are built such as to penalize differences between heads and flow simulated using EPANET and the predicted ones, and to penalize the violation of the hydraulic physics laws.

2.5.3 GNN model loss functions

Now that we know what physics-informed machine learning means, let's see how to define the GNN model loss function to include physics knowledge in the model minimizing the violation of physical laws.

At each update layer k, a loss function $l^k(G_I, G_o(H^k))$ is evaluated. The loss l^k is composed of two parts:

$$l^k = \beta l_m^k + (1 - \beta) l_v^k \tag{16}$$

In equation 16, the fist part is the loss of the sum of the squared errors between model prediction and measured values, and it is defined as follow:

$$l_m^k = I^m ||H^k - H^*||_2 \tag{17}$$

In the previous equation 17, I^m is a vector that indicates the measurements locations.

The second part of the equation 16 is where the loss l_v^k, representing the physical violation, is computed. To define the loss l_v^k we start from the Hazen-Williams model that defines the head loss h_p for a pipe p in a WDS in steady-state conditions.

$$h_p = H_j - H_i = c_p Q_p^{1.852} \tag{18}$$

Equation 18 defines the head loss h_p of the pipe p as difference of the heads H_j, H_i of its connected nodes i and j. Exploiting this equation we can express the flow Q_p in the pipe p as:

$$Q_p = \left(\frac{1}{c_p} (H_j - H_i) \right)^{\frac{1}{1.852}} \tag{19}$$

Now we can exploit this definition of Q_p to define a kind of mass conservation equation where, for each node i, the sum of the flow of the outgoing and incoming pipes is equal to its demand d_i. So iterating over all the neighbors j of i we have the following equation:

$$\sum_{j \in N_i} \left(\frac{1}{c_p} (H_j - H_i) \right)^{0.54} \tag{20}$$

Where $i \in N$ with N set of nodes of WDS and N_i set of neighbors of i.

Another important steady-state condition of a WDS is the head boundary conditions at the source.

$$H_i = H_{s,i} \quad \forall i \in N \tag{21}$$

The equation 22 says that, in a steady-state, each node of the WDS i has a head H_i that is equal to the head $H_{s,i}$ that is the head of the source connected with i. Now we can combine the two equations 20 and 22 to define the loss l_v^k as follow:

$$l_v^k = \sum_{i \in N} \left(\sum_{j \in N_i} \left(\frac{1}{c_p} (H_j^k - H_i^k) \right)^{0.54} - q_i \right)^2 + \sum_{i \in N} (H_i^k - H_{s,i})^2 \tag{22}$$

The first part of the loss l_v^k measures the difference between the simulated flow and the simulated ones at each node while the second part measures the difference between the predicted head at node i H_i^k and the head at the source. It may append that $H_i^k = H_j^k$, which made the first term of the loss not differentiable which is a problem since during the training the loss function will be minimized. To avoid that, in the case of $H_i^k = H_j^k$, the term $(H_j^k - H_i^k)$ is replaced with a very small number.

Finally, we know all the terms of equation 16 bu β that is a coefficient used to prioritize the first or the second term of the loss definition.

2.6 Model performance metrics

Now that we have clearly defined the input data structure, the GNN model, and the training design, we need to define the metrics to evaluate the predictions of the model.

To evaluate the predictions of the model we will compare them with the values simulated by EPANET. The performance metrics used to evaluate the prediction of the model are the following:

- Correlation on the Heads measures $corr_H$:

$$corr_H = \frac{\sum_{i=1}^{n} \sum_{j=1}^{T_e} (H_{i,j}^k - \mu(H^k))(H_{i,j}^* - \mu(H^*))}{\sqrt{\sum_{i=1}^{n} \sum_{j=1}^{T_e} (H_{i,j}^k - \mu(H^k))^2 (H_{i,j}^* - \mu(H^*))^2}} \tag{23}$$

  Where $H_{i,j}^k$ and $H_{i,j}^*$ are the head at the i^{th} node and at the j^{th} test sample, with T_e dimension of the test set, respectively computed by the GNN model and by EPANET. The values $\mu(H^k)$ and $\mu(H^*)$ are the mean values of all the heads of the test set computed by the GNN and simulated by EPANET.

- Correlation on the Flow measures $corr_Q$

$$corr_H = \frac{\sum_{i=1}^{n} \sum_{j=1}^{T_e} (Q_{i,j}^k - \mu(Q^k))(Q_{i,j}^* - \mu(Q^*))}{\sqrt{\sum_{i=1}^{n} \sum_{j=1}^{T_e} (Q_{i,j}^k - \mu(Q^k))^2 (Q_{i,j}^* - \mu(Q^*))^2}} \tag{24}$$

Where $Q^k_{i,j}$ and $Q^*_{i,j}$ are the flow values at the i^{th} node and at the j^{th} test sample, with T_e dimension of the test set, respectively computed by the GNN model and by EPANET. The values $\mu(Q^k)$ and $\mu(Q^*)$ are the mean values of all the flow values of the test set computed by the GNN and simulated by EPANET.

- Root mean squared error of the Heads measures $rmse_H$

$$rmse_H = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(H^k_{i,j} - H^*_{i,j})} \tag{25}$$

This is the root mean squared error between the prediction of the GNN model of the nodal heads and the simulated ones.

- Root mean squared error of the flow measures $rmse_Q$

$$rmse_Q = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(Q^k_{i,j} - Q^*_{i,j})} \tag{25}$$

This is the root mean squared error between the prediction of the GNN model of the flow and the simulated one.

- Mean absolute error of the Heads measures mae_H

$$mae_H = \frac{1}{n}\sum_{i=1}^{n}|H^k_{i,j} - H^*_{i,j}| \tag{26}$$

This is how the mean average error between the prediction of the GNN model of the nodal heads and the simulated one.

- Mean absolute error of the flow measures mae_Q

$$mae_Q = \frac{1}{n}\sum_{i=1}^{n}|Q^k_{i,j} - Q^*_{i,j}| \tag{27}$$

This is how the mean average error between the prediction of the GNN model of the flow and the simulated one.

Chapter 3

Experimentation and results

In chapter 2 we have seen how to generate the input data for our model performing multiple hydraulic simulations and adding noise to the WDS topology. The graph model used in this work has been properly defined as well as all the centrality we will measure and compare with the predictions performance metrics. At the end of chapter 2, the two machine learning techniques semi-supervised learning and physics-informed machine learning have been introduced and we have seen in detail how to include the hydraulic laws into the GNN model loss functions.

In this chapter, the two investigated Water Distribution Systems ASnet2 and Anytown2 will be presented talking about their topology and their nodes and links properties. Some considerations about the computational time and consequently about the hardware to use to train the model will be done. We will see how to set the GNN models' parameters to start the training phase. The obtained State Estimation predictions will be evaluated using the performance metrics described in section 2.6, the centrality measures computed from the graph models of the two WDS will be presented and then it will be presented the adopted strategy to find some relation between performance metrics and centrality measures. Finally, all the obtained results will be discussed.

3.1 WDS ASnet2 and Anytown

To apply the algorithms and the models described in chapter 2 we have worked with two benchmarks WDS, the first one called ASnet2 and the second one called Anytown. ASnet2 and Anytown are two WDS widely used in other state of the art works and that could be helpful to obtain further analytic information about those WDS and also to better understand our results by comparing them with those of the other works.

3.1.1 ASnet2

ASnet2 is a WDS composed of 51 junctions, 65 pipes, and 1 reservoir. Its topology is reported in figure 3.1.

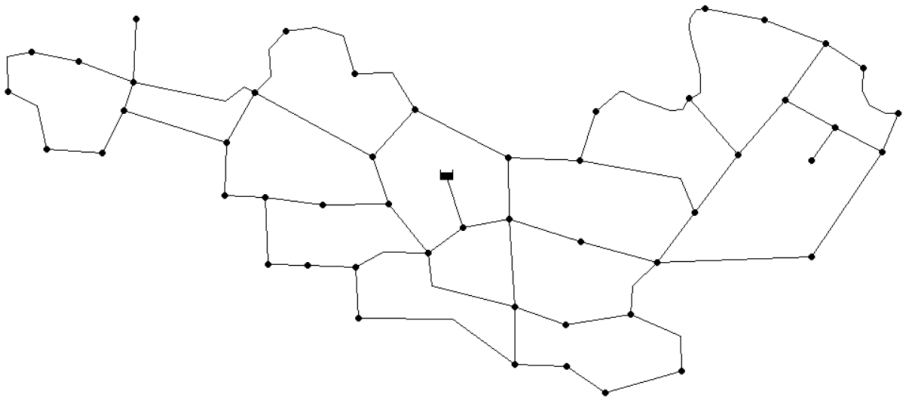

Figure 3.1. Topology of the WDS Asnet2 where the dots are junctions, the lines are pipes, and the square symbol represents the reservoir

As we can see from figure 3.1, the nodes of ASnet2 are well distributed and the reservoir is at the center of the network.

Figure 3.2 shows the table of the properties of the junctions of ASnet2. The demand values are between 0 and 115, and very interesting we have 5 different patterns. The patterns have the prefix name "DMA", an acronym used in other state of the works that stays for District Metered Areas. A DMA is a portion of the WDS where the nodes have similar properties or similar behaviour, so it seems that the nodes of ASnet2 can be divided into 5 clusters.

The 65 junctions of ASnet2 have length values between 20 and 1285 meters and they all have fixed diameter of 508 millimeters and same roughness of 130 millimeters.

Detailed information about ASnet2 WDS can be found in the work "Design of water distribution system" by Alperovits and Shamir [17]

JUNCTIONS	ELEVATION [m]	DEMAND [m^3/h]	PATTERN
1	370.2	25	DMA1_pat
2	350	37	DMA2_pat
3	335.7	80	DMA3_pat
4	340	115	DMA1_pat
5	357	27	DMA2_pat
6	345.4	85	DMA4_pat
7	318.8	29	DMA1_pat
8	317.4	55	DMA2_pat
9	315.1	32	DMA3_pat
10	309.1	43	DMA1_pat
11	306	34	DMA4_pat
12	309	70	DMA2_pat
13	360	29	DMA3_pat
14	360	24	DMA1_pat
15	348.2	56	DMA3_pat
16	360	39	DMA4_pat
17	343.5	67	DMA1_pat
18	331	45	DMA4_pat
19	311	53	DMA1_pat
20	306.3	31	DMA3_pat
21	343.7	33	DMA5_pat
22	315.7	88	DMA2_pat
23	308.8	0	DMA5_pat
24	366.7	16	DMA1_pat
25	341.9	45	DMA5_pat
26	326.8	7	DMA2_pat
27	313.4	36	DMA3_pat
28	305.7	58	DMA1_pat
29	311.9	23	DMA5_pat
30	321.8	30	DMA2_pat
31	327.2	31	DMA3_pat
32	320	17	DMA2_pat
33	340	16	DMA4_pat
34	350	26	DMA1_pat
35	355.6	65	DMA3_pat
36	361	16	DMA5_pat
37	370	6	DMA2_pat
38	358.1	30	DMA4_pat
39	353.2	29	DMA3_pat
40	345	26	DMA1_pat
41	359.3	24	DMA4_pat
42	348.2	35	DMA5_pat
43	362.2	23	DMA3_pat
44	358.4	29	DMA5_pat
45	370.6	3	DMA4_pat
46	360	13	DMA1_pat
47	315.8	17	DMA3_pat
48	360	0	-
49	345	0	-
50	345	0	-

Figure 3.2. Properties of the junctions of ASnet2

3.1.2 Anytown

Anytown is a small benchmark WDS composed of 19 junction, 3 reservoir, 40 pipes and 1 pump.

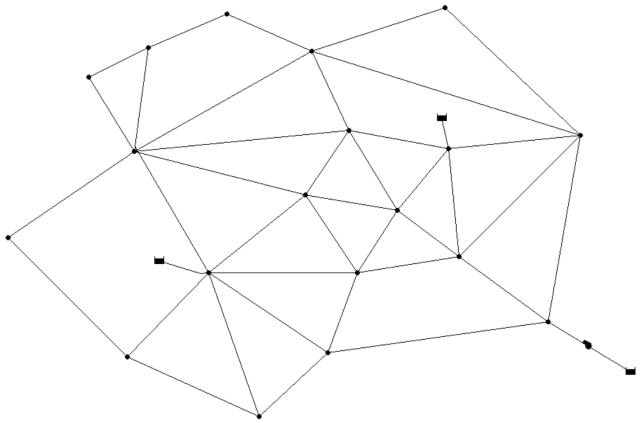

Figure 3.3. Topology of the WDS Anytown where the dots are junctions, the lines are pipes, the square symbols represents the reservoir, and the square crossed to circle symbol represents a pump

Even if Anytown is smaller than ASnet2, considering the number of junctions, it more complex from a hydraulic point of view since it has 3 reservoirs and a pump.

JUNCTIONS	ELEVATION [m]	DEMAND [m^3/h]
20	6	113
30	15	45
40	15	45
50	15	45
55	24	0
60	15	113
70	15	113
75	24	0
80	15	113
90	15	227
100	15	113
110	15	113
115	24	0
120	36	45
130	36	45
140	24	45
150	36	45
160	36	181
170	36	45

a) Anytowns' junctions properties

RESERVOIRS	HEAD [m]
16	3
65	65.5
165	65.5

b) Anytowns' reservoirs properties

DIAMETER [m]	LENGTH [m]
0.203	8778
0.254	2560
0.304	20178
0.406	3658

c) Anytowns' pipes properties

Figure 3.4. Tables of properties of Anytown. The table a contains the properties of the junctions of Anytown, table contains the reservoirs' properties and table c summarizes the Anytown's pipes properties

Figure 3.4 shows three tables each showing different properties of Anytown WDS. Table a) shows the junctions properties, elevation in meters and demand in m^3/h, in this case, any pattern is specified. Table b) shows the heads, measured in meters, associated to Anytown's reservoirs and table c) resumes the properties of the pipes connecting the diameter, measured in meters, with the correspondent pipe's length, measured in meters.

Unlike ASnet2, Anytown's pipes have different diameters and roughness where the two physical properties are dependent on each other as shown in table c) of figure 3.4.

Further information on the Anytown WDS can be found in the description paper edited by Thomas M. Walski [18]

3.2 Training settings

In this section, we will see how the GNN model and training parameters have been set. It will be presented also the hardware used for the training and the training statistics.

3.2.1 GNN parameters

Before starting the training of the model we need to set the parameters of the GNN model. The GNN parameters are the following:

- Iteration: 70000
- Batch size: 50
- correction updates k: 20
- Latent Dimension: 20
- α: 0.01
- Discount factor γ: 0.9
- Learning rate l_r for the Adam optimizer: 0.001
- MLP parameters:
  - Hidden layers: 2
  - Activation function: Leaky-ReLU

3.2.2 Training parameters and environment

In this work, we are interested in testing the GNN model described in section **??** adopting a semi-supervised approach, as described in section 2.5.2, but we also want to define a criterion to choose the best nodes to consider for the State Estimation process. To reach that last goal, we need to perform n times, where n is the number of junctions of the considered WDS, State Estimation considering as observed only the n^{th} junction. In this way, we will have n State Estimation evaluations, one for each junction of the WDS, and we can finally compare the performance metrics of the n^{th} junction with its centrality measures to find some relations.

Performing n State Estimation processes means training the model n times. At first, we have trying to train the model using a GPU Nvidia GeForce GTX 1050Ti, with 4 GB memory, 768 CUDA cores, and 2.138 TFLOPS, and with this configuration, it took 6 hours and 30 minutes per training. Since we need to perform 50 training for ASnet2 and 19 training for Anytown, it would have taken respectively $50 \times 6.5 = 325$ hours and $19 \times 6.5 = 123.5$ hours, that in both cases is too much.

To speed up things we have looked for a much more powerful setting and we have found an Amazon Elastic Compute Cloud (EC2) with 4 GPU Nvidia Tesla T4, with 16 GB memory, 2560 CUDA cores, and 8.141 TFLOPS. With this GPU a training needs 2 hours and 30 minutes to complete if one training per GPU or 3 hours and 30 minutes running two training on the same GPU. That means we can launch 8 training at the same time, each of 3 hours and 30 minutes, and since they run in parallel, 8 training are completed in 3 hours and 30 minutes.

$$T_{tr} = \left(\frac{n - nmod8}{8} + 1 \right) \times 3.5 \tag{28}$$

The equation 28 is the general to estimate the time needed to conclude all the training for a WDS whit n junctions. In our case, we need $T_{tr,asnet2}$ to perform all the training for ASnet2 (50 junctions), and $T_{tr,anyt}$ to complete all the training for Anytown (19 junctions).

$$T_{tr,asnet2} = \frac{48}{8} \times 3.5 + 3.5 = 24.5 \quad hours \tag{29}$$

$$T_{tr,anyt} = \frac{16}{8} \times 3.5 + 3.5 = 10.5 \quad hours \tag{30}$$

By the two equations 29 and 30 we have that we need 24.5 hours for the training phases of ASnet2 and 10.5 hours for training the model on the junctions of Anytown.

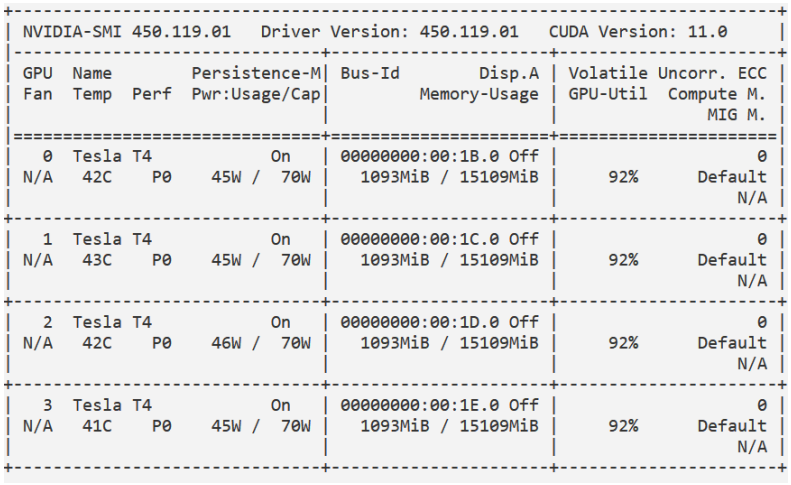

Figure 3.5. Statistics of the 4 Tesla T4 GPU running 8 training phases at the same time, 2 per GPU

As shown in image 3.5, running two training phases per GPU, the 92% capability of each GPU is used and the GPUs are not too stressed since the temperatures are quite low, between 40 and 50 degrees. In practice, all the training for ASnet2 and Anytown have respected the estimated times to conclude.

To perform the training using GPU computation we have used the TensorFlow structure and sessions. Very important is to set a constant random seed for all the training TensorFlow sessions, otherwise, the training on the different junctions will be affected by the different random seeds.

3.3 Model prediction performances with 1 observed node

As already said, we want to investigate the impact of each junction of the considered WDS on the State Estimation prediction performances. After finishing all the training, we test the model on the test set, we compute all the performance metrics described in section 2.6 and we store them in a csv file, to make them rapidly available for the future.

The table in figure 3.6, contains all the statistics of the predictions of the State Estimation for each junction that has been considered to be observed. The results are surprising, in fact, for the 11 junctions 18, 22, 19, 17, 9, 8, 24, 7, 16, 4, and 10 we have very high levels of $corr_H$ and $corr_Q$, higher than 0.9. For the 5 nodes 18, 22, 19, 17 and 9 the order of order of magnitude of $rmse_h$ and mae_H is of 10^{-2} and the order of magnitude of $rmse_Q$ and mae_Q is of 10^{-3}

NODE	CORRH	CORRQ	RMSEH	RMSEQ	NRMSEH	NRMSEQ	MAEH	MAEQ	NMAEH	NMAEQ
1	0.8226	0.9291	0.3334	0.0193	0.1407	0.0305	0.2366	0.0146	0.0998	0.0231
2	0.8759	0.9496	0.2809	0.0164	0.1185	0.0258	0.1966	0.0124	0.0829	0.0195
3	0.8647	0.9433	0.2810	0.0173	0.1186	0.0272	0.1970	0.0131	0.0831	0.0206
4	0.9243	0.9671	0.2283	0.0141	0.0963	0.0222	0.1549	0.0106	0.0653	0.0167
5	0.7906	0.9251	0.3418	0.0199	0.1442	0.0315	0.2446	0.0151	0.1032	0.0239
6	0.8703	0.9493	0.2945	0.0168	0.1242	0.0265	0.2030	0.0127	0.0857	0.0201
7	0.9515	0.9756	0.1750	0.0118	0.0738	0.0185	0.1120	0.0085	0.0472	0.0134
8	0.9788	0.9853	0.1037	0.0090	0.0437	0.0142	0.0682	0.0064	0.0288	0.0102
9	0.9805	0.9862	0.0981	0.0092	0.0414	0.0145	0.0638	0.0066	0.0269	0.0104
10	0.9001	0.9325	0.2126	0.0200	0.0897	0.0316	0.1280	0.0131	0.0540	0.0207
11	0.7666	0.9146	0.3214	0.0229	0.1356	0.0362	0.2208	0.0154	0.0932	0.0243
12	0.7609	0.9151	0.3234	0.0228	0.1364	0.0360	0.2244	0.0154	0.0947	0.0243
13	0.8580	0.9458	0.2903	0.0172	0.1225	0.0271	0.1996	0.0131	0.0842	0.0206
14	0.7636	0.9076	0.3859	0.0230	0.1628	0.0362	0.2749	0.0175	0.1160	0.0276
15	0.8383	0.9336	0.3197	0.0195	0.1349	0.0308	0.2187	0.0149	0.0923	0.0235
16	0.9376	0.9708	0.1862	0.0134	0.0785	0.0211	0.1239	0.0099	0.0523	0.0156
17	0.9829	0.9781	0.0978	0.0109	0.0413	0.0171	0.0625	0.0073	0.0263	0.0115
18	0.9913	0.9926	0.0667	0.0065	0.0281	0.0103	0.0423	0.0047	0.0179	0.0073
19	0.9830	0.9868	0.0936	0.0091	0.0395	0.0144	0.0633	0.0067	0.0267	0.0105
20	0.7664	0.9150	0.3158	0.0227	0.1332	0.0359	0.2131	0.0154	0.0899	0.0243
21	0.8332	0.9295	0.2860	0.0217	0.1207	0.0343	0.1644	0.0145	0.0693	0.0229
22	0.9858	0.9861	0.0884	0.0091	0.0373	0.0144	0.0519	0.0061	0.0219	0.0097
23	0.8324	0.9299	0.2761	0.0206	0.1165	0.0326	0.1536	0.0130	0.0648	0.0206
24	0.9754	0.9815	0.1152	0.0105	0.0486	0.0165	0.0692	0.0073	0.0292	0.0115
25	0.7999	0.9337	0.3089	0.0206	0.1303	0.0326	0.1781	0.0143	0.0751	0.0226
26	0.8044	0.9377	0.3096	0.0207	0.1306	0.0326	0.1706	0.0142	0.0720	0.0225
27	0.7838	0.9331	0.3258	0.0216	0.1375	0.0341	0.1839	0.0150	0.0776	0.0237
28	0.7986	0.9320	0.3104	0.0215	0.1310	0.0339	0.1752	0.0146	0.0739	0.0231
29	0.7782	0.9341	0.3316	0.0214	0.1399	0.0337	0.1861	0.0146	0.0785	0.0230
30	0.8466	0.9359	0.2594	0.0190	0.1094	0.0301	0.1526	0.0132	0.0644	0.0209
31	0.8101	0.9356	0.2997	0.0201	0.1264	0.0317	0.1741	0.0140	0.0735	0.0221
32	0.8180	0.9351	0.2827	0.0197	0.1193	0.0311	0.1586	0.0135	0.0669	0.0214
33	0.8111	0.9327	0.2864	0.0198	0.1208	0.0313	0.1728	0.0143	0.0729	0.0226
34	0.8008	0.9322	0.2900	0.0192	0.1223	0.0303	0.1934	0.0141	0.0816	0.0223
35	0.8225	0.9366	0.2759	0.0186	0.1164	0.0294	0.1824	0.0136	0.0769	0.0215
36	0.8038	0.9340	0.2890	0.0195	0.1219	0.0308	0.1804	0.0139	0.0761	0.0220
37	0.8064	0.9366	0.2873	0.0191	0.1212	0.0301	0.1785	0.0136	0.0753	0.0215
38	0.7643	0.9417	0.3245	0.0175	0.1369	0.0276	0.2239	0.0127	0.0945	0.0201
39	0.7728	0.9404	0.3150	0.0178	0.1329	0.0281	0.2181	0.0132	0.0920	0.0208
40	0.7412	0.9388	0.3539	0.0180	0.1493	0.0284	0.2533	0.0135	0.1068	0.0212
41	0.7070	0.9351	0.3829	0.0187	0.1615	0.0295	0.2799	0.0141	0.1181	0.0223
42	0.6483	0.9270	0.4310	0.0203	0.1818	0.0320	0.3228	0.0153	0.1362	0.0242
43	0.6952	0.9340	0.3925	0.0189	0.1656	0.0299	0.2894	0.0145	0.1221	0.0228
44	0.5742	0.9163	0.4647	0.0225	0.1960	0.0355	0.3489	0.0168	0.1472	0.0266
45	0.7899	0.9357	0.3054	0.0203	0.1288	0.0320	0.1802	0.0134	0.0760	0.0211
46	0.7833	0.9076	0.3463	0.0259	0.1461	0.0409	0.2159	0.0185	0.0911	0.0291
47	0.7586	0.9300	0.3729	0.0232	0.1573	0.0367	0.2133	0.0162	0.0900	0.0256
48	0.7845	0.9454	0.3085	0.0170	0.1301	0.0268	0.2083	0.0124	0.0879	0.0195
49	0.8493	0.9453	0.3203	0.0176	0.1351	0.0277	0.2243	0.0133	0.0946	0.0210
50	0.8439	0.9412	0.3205	0.0180	0.1352	0.0285	0.2226	0.0137	0.0939	0.0216

Figure 3.6. Tables containing the performance metrics of the SE for each node of ASnet2

Concerning Anytown there were some problems at the beginning due to the physical properties of its pipes.

In fact, the first attempts to apply our model on Anytown were not successful, the model was struggling during the entire training phase, giving very high loss at each update iteration, and the output values were very bad. Playing with EPANET, we experienced the warning "Negative Pressure" during the hydraulic simulation of Anytown. This warning means that the system is unable to meet the given demand and one of the suggestions to solve this problem was to Reduce head loss en route to the critical nodes by increasing pipe diameters. We have then increased the diameters and consequently the roughness of the pipes. After these modifications, the warning is solved and the model performs much better giving the following results.

NODE	CORRH	CORRQ	RMSEH	RMSEQ	NRMSEH	NRMSEQ	MAEH	MAEQ	NMAEH	NMAEQ
20	0.3221	0.6201	790.1856	3.8059	2.9561	0.1809	666.1628	2.2058	2.4921	0.1049
30	0.4673	0.6414	56.0354	2.9510	0.2096	0.1403	14.3407	0.6860	0.0536	0.0326
40	0.4672	0.7132	55.7898	2.8961	0.2087	0.1377	14.2275	0.6908	0.0532	0.0328
50	0.4670	0.7610	55.8315	2.8989	0.2089	0.1378	13.7089	0.6565	0.0513	0.0312
55	0.4683	0.5141	56.3084	2.9564	0.2107	0.1405	17.5103	0.7321	0.0655	0.0348
60	0.4678	0.8575	55.3983	2.7891	0.2072	0.1326	14.7093	0.6889	0.0550	0.0327
70	0.4672	0.7029	56.2207	2.9563	0.2103	0.1405	14.3821	0.6641	0.0538	0.0316
75	0.4688	0.6778	55.7679	2.8543	0.2086	0.1357	16.6138	0.7123	0.0622	0.0339
80	0.4672	0.8221	55.7149	2.8565	0.2084	0.1358	14.1193	0.6682	0.0528	0.0318
90	0.4674	0.6201	56.3052	2.9661	0.2106	0.1410	15.2012	0.7026	0.0569	0.0334
100	0.4690	0.8501	55.4458	2.8086	0.2074	0.1335	14.5572	0.6889	0.0545	0.0327
110	0.4687	0.7315	55.8984	2.9280	0.2091	0.1392	13.4284	0.6712	0.0502	0.0319
115	0.4705	0.7192	55.5426	2.8076	0.2078	0.1335	16.8177	0.7063	0.0629	0.0336
120	0.4682	0.6485	55.9589	2.9449	0.2093	0.1400	13.7578	0.6947	0.0515	0.0330
130	0.4689	0.7631	55.5927	2.8619	0.2080	0.1360	13.5972	0.6839	0.0509	0.0325
140	0.4683	0.6787	56.0302	2.9367	0.2096	0.1396	14.6919	0.6993	0.0550	0.0332
150	0.4671	0.6934	56.0824	2.9388	0.2098	0.1397	14.5016	0.6849	0.0543	0.0326
160	0.4687	0.8330	55.5365	2.8422	0.2078	0.1351	13.6727	0.6728	0.0511	0.0320
170	0.4683	0.5644	56.0844	2.9540	0.2098	0.1404	15.4146	0.7284	0.0577	0.0346

Figure 3.7. Tables containing the performance metrics of the SE for each node of Anytown

The table in figure 3.7 contains the performance metrics of the SE performed on each node of Anytown. The results of the SE process on Anytown are not as good as the ones of ASnet2. In fact, all the junctions have $corr_H$ around 0.46 but junction 20 that has $corr_H = 0.32$. For all the junctions the values of $corr_q$ are between 0.51 and 0.85 while the $rmse_H$ is around 55, and the mae_H is between 13.4 and 17.5. The mae_Q is of order of magnitude 10^{-1}.

The very exception here is junction 20 which has the worst performances. In fact, it has a $rmse_H = 790.18$ and a $mae_H = 666.16$ that are very high values. Also, its mae_Q is 10 times bigger than the ones of the other junctions. Very interesting is that junction 20 is the only one connected with a pump, this may be a very interesting indication that must be thorough to understand if this is a particular exception or effectively can exclude to consider to observe junctions connected to a pump to perform SE.

3.4 Centrality Measures - Performance metrics correlation

Now that we have the State Estimation predictions performances we want to find a dependency between them and the centrality measures of each junction of the two WDS.

To compute the centrality measures described in section 2.2.2, at first we build the graph model of the WDS as described in 2.2.1 and then we compute the centrality measures of each nodes of the two graphs using the Python library NetwokX.

The two tables in figure 3.8 and 3.9 show the centrality measures of the node of the ASnet2 graph model. The centrality measures of ASnet2 are in general low and there seem to be no outliers nodes. Particularly low are the values of the Eigenvector centrality that reach an order of magnitude of 10^{-7}.

Looking at the centrality measures of the nodes of the Anytown graph model in figure 3.10 and figure 3.11, the first things to notice are the values of the degree centrality. In fact, for Anytown the degree centrality is of an order of magnitude 10^{-1} while the degree centrality of ASnet2 has order of magnitude of 10^{-2}. What stands out is the behaviour of the current flow closeness centrality that is 10^{-12} for all the nodes as well as the second order centrality that is $1.0670 \times 10^{+6}$ for all the nodes.

Also in this case node 20 acts as an outlier. In fact, it is the only one to have current flow closeness centrality and second order centrality different than 10^{-12} and $1.0670 \times 10^{+6}$. In particular, the current flow closeness centrality of node 20 is 100 times smaller than all the others and the second order centrality is 10 times smaller. Furthermore, node 20 is the only one to have betweenness centrality, current flow betweenness centrality, Approximate current flow betweenness centrality and load centrality equal to 0.

NODE	DEGREE CENT	CLOSE CENT	EIGENVECTOR CENT	KATZ CENT	CFC CENT	BET CENT	CFB CENT
1	0.04	0.1401	3.2103e-04	0.1386	0.0003	0.0163	6.0728e-02
2	0.04	0.1471	6.3771e-04	0.1383	0.0003	0.0457	5.8302e-02
3	0.06	0.1672	1.5838e-02	0.1386	0.0003	0.1445	1.1768e-01
4	0.04	0.1873	3.4072e-04	0.1376	0.0003	0.1633	9.8594e-02
5	0.06	0.1511	5.2535e-03	0.1392	0.0003	0.0261	1.1842e-01
6	0.06	0.2212	3.0301e-04	0.1392	0.0004	0.1298	2.3265e-01
7	0.08	0.2165	2.1624e-05	0.1397	0.0005	0.2335	3.3055e-01
8	0.06	0.2092	1.5899e-06	0.1389	0.0004	0.0906	2.2638e-01
9	0.06	0.1953	6.5799e-03	0.1390	0.0004	0.0890	2.0327e-01
10	0.04	0.1773	2.8620e-07	0.1380	0.0003	0.0457	9.2135e-02
11	0.04	0.1695	2.4701e-07	0.1381	0.0003	0.0139	8.8269e-02
12	0.04	0.1453	1.4392e-04	0.1380	0.0003	0.0057	8.3112e-02
13	0.04	0.1613	1.2987e-05	0.1385	0.0003	0.0261	9.1216e-02
14	0.06	0.1859	1.6650e-04	0.1388	0.0004	0.0620	1.5473e-01
15	0.04	0.2016	4.0810e-06	0.1381	0.0004	0.0767	1.2813e-01
16	0.04	0.2183	4.3399e-07	0.1380	0.0004	0.0645	1.2293e-01
17	0.04	0.1773	3.2202e-07	0.1381	0.0004	0.0065	9.6979e-02
18	0.08	0.2475	1.4524e-06	0.1397	0.0005	0.2898	2.6798e-01
19	0.08	0.2381	9.1396e-07	0.1393	0.0005	0.2531	2.1963e-01
20	0.06	0.2008	6.2555e-07	0.1390	0.0004	0.0653	1.5796e-01
21	0.06	0.2294	1.3638e-06	0.1397	0.0005	0.1992	2.4044e-01
22	0.06	0.2404	6.1960e-07	0.1386	0.0005	0.3551	2.0067e-01
23	0.06	0.2415	4.3525e-07	0.1385	0.0004	0.2727	2.2089e-01
24	0.04	0.1608	1.0910e-06	0.1386	0.0003	0.0678	7.6560e-02
25	0.08	0.2155	9.2646e-07	0.1393	0.0004	0.3159	2.5442e-01
26	0.04	0.1718	7.6050e-07	0.1388	0.0004	0.0155	1.8856e-01
27	0.06	0.1779	9.6883e-07	0.1400	0.0004	0.0416	2.2830e-01
28	0.04	0.1953	7.3886e-07	0.1385	0.0004	0.0522	1.1136e-01
29	0.04	0.1656	6.6820e-07	0.1386	0.0004	0.0204	1.3324e-01
30	0.06	0.1887	8.6934e-07	0.1391	0.0004	0.1363	2.2087e-01
31	0.04	0.1931	2.9063e-07	0.1379	0.0003	0.0090	1.0148e-01
32	0.08	0.1873	3.1624e-06	0.1422	0.0003	0.1404	1.7945e-01
33	0.06	0.1667	1.6803e-06	0.1395	0.0003	0.0792	1.5586e-01
34	0.04	0.1453	5.1290e-07	0.1381	0.0003	0.0449	5.9517e-02
35	0.04	0.2058	2.1765e-07	0.1378	0.0003	0.0245	1.0143e-01
36	0.04	0.1289	2.2433e-07	0.1378	0.0002	0.0098	4.2498e-02
37	0.04	0.1408	4.8740e-07	0.1385	0.0002	0.0351	5.4909e-02
38	0.04	0.2262	9.2125e-07	0.1378	0.0004	0.1657	1.2229e-01
39	0.04	0.1269	2.8078e-07	0.1381	0.0002	0.0057	4.2014e-02
40	0.06	0.2262	2.9721e-06	0.1415	0.0005	0.0400	1.9194e-01
41	0.08	0.2336	2.0852e-06	0.1405	0.0005	0.2196	3.2314e-01
42	0.06	0.2538	8.8022e-07	0.1389	0.0005	0.3518	2.8566e-01
43	0.02	0.1582	2.5047e-06	0.1395	0.0003	0.0000	8.7005e-17
44	0.06	0.1592	7.0706e-01	0.1660	0.0004	0.0400	9.8721e-02
45	0.04	0.1845	6.2678e-07	0.1383	0.0004	0.0114	1.0953e-01
46	0.06	0.1984	1.2579e-06	0.1400	0.0005	0.0482	1.9035e-01
47	0.04	0.1754	8.1232e-07	0.1389	0.0004	0.0245	1.3678e-01
48	0.06	0.2370	5.1988e-06	0.1385	0.0004	0.2033	2.2354e-01
49	0.06	0.1724	1.9454e-01	0.1448	0.0004	0.0449	1.6138e-01
50	0.02	0.1852	2.3532e-06	0.1395	0.0004	0.0000	4.6983e-16
51	0.02	0.1377	6.7963e-01	0.1605	0.0004	0.0000	9.6648e-16

Figure 3.8. Centrality measures of the nodes of ASnet2 graph model 1/2

NODE	ACFB CENT	CB CENT	LOAD CENT	SUBGRAPH CENT	SO CENT	HARMONIC CENT	PAGERANK
1	6.0728e-02	0.0105	0.0163	2.3406	596.9807	10.4430	0.0157
2	5.8302e-02	0.0237	0.0457	2.3405	607.5208	10.7845	0.0134
3	1.1768e-01	0.0943	0.1445	3.1418	562.2983	12.6072	0.0145
4	9.8594e-02	0.1156	0.1633	2.4514	582.7529	13.0477	0.0086
5	1.1842e-01	0.0417	0.0261	3.1448	559.3827	11.7842	0.0180
6	2.3265e-01	0.2032	0.1298	3.3444	403.0688	15.1044	0.0203
7	3.3055e-01	0.2265	0.2335	4.1326	375.3281	15.6103	0.0274
8	2.2638e-01	0.1096	0.0906	3.2053	410.0798	14.3353	0.0215
9	2.0327e-01	0.1514	0.0890	3.2845	447.6396	14.0810	0.0159
10	9.2135e-02	0.0306	0.0457	2.3378	545.0123	11.8961	0.0142
11	8.8269e-02	0.0172	0.0139	2.2835	562.4243	11.2402	0.0160
12	8.3112e-02	0.0119	0.0057	2.3406	597.0225	10.5382	0.0120
13	9.1216e-02	0.0340	0.0261	2.3410	560.3750	11.1318	0.0167
14	1.5473e-01	0.0839	0.0620	3.1652	502.1631	13.0405	0.0192
15	1.2813e-01	0.0803	0.0767	2.4148	502.3461	12.9044	0.0135
16	1.2293e-01	0.0736	0.0645	2.4665	453.4907	14.0103	0.0129
17	9.6979e-02	0.0292	0.0065	2.3383	517.9192	11.8179	0.0154
18	2.6798e-01	0.2273	0.2898	4.4002	326.9385	16.5524	0.0244
19	2.1963e-01	0.2176	0.2531	4.4809	361.2575	16.4238	0.0234
20	1.5796e-01	0.0805	0.0653	3.2327	413.6191	13.9222	0.0214
21	2.4044e-01	0.1971	0.1992	3.2746	351.1848	15.4675	0.0234
22	2.0067e-01	0.3216	0.3551	3.3493	391.4001	15.7631	0.0168
23	2.2089e-01	0.2663	0.2727	3.2261	404.9890	15.2619	0.0177
24	7.6560e-02	0.0697	0.0678	2.3991	648.6465	11.4628	0.0170
25	2.5442e-01	0.3094	0.3159	4.3999	432.8611	15.4810	0.0228
26	1.8856e-01	0.0387	0.0155	2.3972	450.1796	12.1287	0.0174
27	2.2830e-01	0.0540	0.0416	3.0813	420.4945	12.7211	0.0272
28	1.1136e-01	0.0488	0.0522	2.3942	433.3079	12.9909	0.0147
29	1.3324e-01	0.0231	0.0204	2.3359	458.1196	11.4529	0.0159
30	2.2087e-01	0.1077	0.1363	3.3912	446.0390	13.6671	0.0204
31	1.0148e-01	0.0326	0.0090	2.4254	542.7694	12.6810	0.0127
32	1.7945e-01	0.1931	0.1404	4.2586	562.4086	14.1171	0.0410
33	1.5586e-01	0.0986	0.0792	3.3854	560.7068	12.7033	0.0227
34	5.9517e-02	0.0476	0.0449	2.3430	664.5533	10.6751	0.0142
35	1.0143e-01	0.0410	0.0245	2.3624	543.0492	12.8155	0.0125
36	4.2498e-02	0.0130	0.0098	2.2819	742.2144	9.5333	0.0125
37	5.4909e-02	0.0325	0.0351	2.2836	708.6529	10.0763	0.0184
38	1.2229e-01	0.1116	0.1657	2.5251	457.1886	14.4770	0.0102
39	4.2014e-02	0.0058	0.0057	2.2801	744.5723	9.3012	0.0154
40	1.9194e-01	0.0920	0.0400	3.4052	364.5239	14.8238	0.0326
41	3.2314e-01	0.2437	0.2196	4.4722	323.2681	16.3071	0.0281
42	2.8566e-01	0.3258	0.3518	3.3298	349.7327	16.0190	0.0199
43	8.7005e-17	0.0033	0.0000	1.7044	604.4586	10.4628	0.0202
44	9.8721e-02	0.0558	0.0400	3.1452	520.4928	12.2584	0.0514
45	1.0953e-01	0.0224	0.0114	2.4182	454.9499	12.4405	0.0136
46	1.9035e-01	0.0835	0.0482	3.2326	392.7661	13.8921	0.0264
47	1.3678e-01	0.0352	0.0245	2.3400	442.1985	11.8933	0.0178
48	2.2354e-01	0.2491	0.2033	3.2303	426.3186	15.1786	0.0171
49	1.6138e-01	0.1063	0.0449	3.2610	509.1612	13.1568	0.0205
50	4.6983e-16	0.0020	0.0000	1.6519	426.5494	11.1889	0.0188
51	9.6648e-16	0.0014	0.0000	1.6448	526.7400	9.3930	0.0363

Figure 3.9. Centrality measures of the nodes of ASnet2 graph model 2/2

NODE	DEGREE CENT	CLOSE CENT	EIGENVECTOR CENT	KATZ CENT	CFC CENT	BET CENT	CFB CENT
20	0.0476	0.2957	6.1462e-15	0.2094	4.7619e-14	0.0000	0
30	0.2380	0.4468	4.3477e-02	0.2126	1e-12	0.2023	1.9309e-01
40	0.1428	0.3684	2.0600e-03	0.2097	1e-12	0.0047	8.0070e-02
50	0.2380	0.4772	1.0435e-01	0.2145	1e-12	0.0809	2.9499e-01
55	0.2380	0.4772	1.7275e-01	0.2145	1e-12	0.0404	2.3454e-01
60	0.2380	0.4200	2.5214e-01	0.2196	1e-12	0.1071	2.7932e-01
65	0.3333	0.5000	6.3480e-01	0.2216	1e-12	0.1753	3.9939e-01
70	0.1904	0.4117	1.9597e-03	0.2096	1e-12	0.2317	1.0672e-01
75	0.3333	0.5250	1.3603e-01	0.2128	1e-12	0.3515	2.9781e-01
80	0.0476	0.3000	2.2507e-01	0.2157	1e-12	0	1.1926e-15
90	0.0952	0.3442	3.9536e-04	0.2096	1e-12	0	4.4635e-02
100	0.2380	0.4773	2.4515e-02	0.2126	1e-12	0.2361	2.1206e-01
110	0.0952	0.3750	1.0115e-02	0.2114	1e-12	0	5.2278e-02
115	0.1428	0.3559	1.1688e-01	0.2125	1e-12	0.0047	1.2532e-01
120	0.1904	0.4285	1.3947e-01	0.2126	1e-12	0.1246	1.6384e-01
130	0.0952	0.3629	2.0543e-03	0.2096	1e-12	0.0000	3.7468e-02
140	0.2380	0.4565	1.1602e-01	0.2145	1e-12	0.0226	2.1042e-01
150	0.2380	0.4375	8.0525e-02	0.2126	1e-12	0.0880	1.7034e-01
160	0.2380	0.4565	1.7011e-01	0.21457	1e-12	0.0166	2.6071e-01
170	0.1428	0.3750	1.3256e-01	0.2125	1e-12	0.0111	1.1508e-01

Figure 3.10. Centrality measures of the nodes of Anytown graph model 1/2

NODE	ACFB CENT	CB CENT	LOAD CENT	SUBGRAPH CENT	SO CENT	HARMONIC CENT	PAGERANK
20	0	0.0052	0.0000	1.7517	2.2406e+07	7.1833	0.0068
30	1.9309e-01	0.2048	0.2023	9.3694	1.0669e+06	11.5000	0.0482
40	8.0070e-02	0.0692	0.0047	4.4411	1.0669e+06	9.4500	0.0197
50	2.9499e-01	0.2634	0.0809	12.5157	1.0669e+06	11.9166	0.0608
55	2.3454e-01	0.2719	0.0404	14.0504	1.0669e+06	11.9166	0.0575
60	2.7932e-01	0.2165	0.1059	10.4069	1.0669e+06	11.0833	0.0993
65	3.9939e-01	0.3780	0.1755	15.7287	1.0669e+06	12.9166	0.1231
70	1.0672e-01	0.1649	0.2313	6.0740	1.0669e+06	10.5833	0.0143
75	2.9781e-01	0.4196	0.3519	14.9023	1.0669e+06	13.2500	0.0485
80	1.1926e-15	0.0062	0.0000	1.8971	1.0669e+06	7.3500	0.0575
90	4.4635e-02	0.0294	0.0000	2.8466	1.0669e+06	8.5833	0.0137
100	2.1206e-01	0.2383	0.2371	8.8201	1.0669e+06	11.8333	0.0511
110	5.2278e-02	0.0350	0.0000	3.4727	1.0669e+06	9.0833	0.0337
115	1.2533e-01	0.0638	0.0047	4.8046	1.0669e+06	9.2833	0.0422
120	1.6384e-01	0.1586	0.1242	7.6779	1.0669e+06	10.8333	0.0402
130	3.7468e-02	0.0374	0.0000	3.4722	1.0669e+06	8.9500	0.0136
140	2.1042e-01	0.2174	0.0236	13.2146	1.0669e+06	11.5000	0.0557
150	1.7034e-01	0.2141	0.0880	11.4235	1.0669e+06	11.4166	0.0393
160	2.6071e-01	0.2369	0.0158	12.5971	1.0669e+06	11.5833	0.0567
170	1.1508e-01	0.0718	0.0109	5.6267	1.0669e+06	9.6166	0.0387

Figure 3.11. Centrality measures of the nodes of Anytown graph model 2/2

To find some dependencies between the performance metrics and the centrality measures we have decided to compute the correlation between them.

Practically we have stored both the performance metrics values and the centrality measures as columns of the same Pandas DataFrame and then we have applied the method *corr()*. The method *corr()* applied to a DataFrame, computes pairwise correlation of columns. If the DataFrame has n number of columns, *corr()* will return a DataFrame D of dimension $n \times n$ where:

$$D : \begin{cases} D_{i,i} = 1 \\ D_{i,j} = \frac{cov(i,j)}{\sigma(i)\sigma(j)} \quad with \;\; i \neq j \end{cases} \tag{31}$$

In the equation 31, i and j are two different column of the DataFrame where we are applying the method *corr()*, $D_{i,i} = 1$ since correlation of a column with itself is 1 and $D_{i,j}$ is the correlation between column i and column j computed using the Pearson standard correlation coefficient that exploit $cov(i, j)$ that is the covariance between column i and column j.

From D we have removed the entries $D_{i,j}$ where both the columns i and j are performance metrics or centrality measures. In this way, we have a DataFrame with R rows representing the centrality measures, C columns representing the performance metrics, and $D_{r,c}$ being the correlation between the r^{th} centrality measure and the c^{th} performance metrics.

In order to filter the correlations between performance metrics and centrality measures, we have set a threshold of 0.35. We use this threshold to take only the performance metrics with at least a correlation with a centrality measure higher than 0.35 and in this case, the performance metric is inserted as a column c in a DataFrame D_{th} and the centrality measure is inserted as a row r in D_{th} and $D_{th}(r, c)$ is the correlation between these measures

```
interesting_corr = dict()
threshold = 0.35

for ts in train_stats:
    interesting = dict()
    perf_metrics = dict(df_corr.loc[ts])
    perf_metrics_keys = list(perf_metrics.keys())
    for i, corr in enumerate(list(perf_metrics.values())):
        if abs(corr) > threshold:
            interesting[perf_metrics_keys[i]] = corr
    interesting_corr[ts] = interesting

df_inter_corr = pd.DataFrame(interesting_corr)
```

Figure 3.12. Python code used to filter the correlations higher than the fixed threshold

In figure 3.12 there is the code used to filter the correlations setting a threshold and take only the couples with correlation higher than the threshold. In this snippet of code code *df_corr* is the DataFrame D, described earlier, and *df_interr_corr* is the DataFrame D_{th} described earlier.

The table in figure 3.13 contains the correlation values of the couples of performance metric and centrality measure with a correlation greater than 0.35. As shown in the table, mae_H is the performance metric with a higher correlation with the 9 filtered centrality measures. In particular, mae_H has a correlation of -0.71 with close centrality, which could be considered a significant value. A negative correlation means that the absolute values of the two vectors are dependent on each other for the 71% but, for example in this case, if we want to find the nodes with the smallest mae_H we need to take the ones with higher close centrality.

The table in figure 3.14 contains the correlation values for the pairs' performance metrics and centrality values with correlations higher than 3.5 for Anytown. In this case, all the performance metrics are very connected with current flow closeness centrality and second order centrality. In fact, all the performance metrics have a correlation higher than 0.99 or lower than -0.99 with current flow closeness centrality and second order centrality. This is not surprising since, as already discussed in 3.3 and earlier in this section, all the Anytowns' performance metrics do not vary a lot and second order centrality and current flow closeness centrality are constant for all the junctions of Anytown but for the junctions 20.

	CORRH	CORRQ	RMSEH	NRMSEH	MAEH	MAEQ	NMAEH	NMAEQ
CLOSE CENTRALITY	0.5320	0.3743	-0.5910	-0.5910	-0.7076	-0.3949	-0.7076	-0.3949
CFC CENTRALITY	0.4703	-	-0.4695	-0.4695	-0.6498	-	-0.6498	-
BETWEENNESS CENTRALITY	0.3579	-	-0.3805	-0.3805	-0.4249	-	-0.4249	-
CFB CENTRALITY	0.3717	-	-0.3964	-0.3964	-0.5152	-	-0.5152	-
ACFB CENTRALITY	0.3717	-	-0.3964	-0.3964	-0.5152	-	-0.5152	-
COMM BET CENTRALITY	0.3778	-	-0.4034	-0.4034	-0.4558	-	-0.4558	-
LOAD CENTRALITY	0.3579	-	-0.3805	-0.3805	-0.4249	-	-0.4249	-
SO CENTRALITY	-0.5173	-	0.5029	0.5029	0.6825	-	0.6825	-
HARMONIC CENTRALITY	0.5268	0.3832	-0.5596	-0.5596	-0.6684	-0.3893	-0.6684	-0.3893

Figure 3.13. Table containing correlations value between performance metrics and centrality measures of ASnet2

	CORRH	RMSEH	RMSEQ	NRMSEH	NRMSEQ	MAEH	MAEQ	NMAEH	NMAEQ
DEGREE CENTRALITY	0.4128	-0.4087	-0.3811	-0.4087	-0.3811	-0.4092	-0.4102	-0.4092	-0.4102
CLOSE CENTRALITY	0.4423	-0.4359	-0.4194	-0.4359	-0.4194	-0.4361	-0.4327	-0.4361	-0.4327
CFC CENTRALITY	0.9996	-0.9999	-0.9638	-0.9999	-0.9638	-0.9999	-0.9983	-0.9999	-0.9983
CFB CENTRALITY	0.3783	-0.3712	-0.3605	-0.3712	-0.3605	-0.3710	-0.3708	-0.3710	-0.3708
ACFB CENTRALITY	0.3783	-0.3712	-0.3605	-0.3712	-0.3605	-0.3710	-0.3708	-0.3710	-0.3708
SO CENTRALITY	-0.9996	0.9999	0.9638	0.9999	0.9638	0.9999	0.9983	0.9999	0.9983
HARMONIC CENTRALITY	0.4677	-0.4628	-0.4353	-0.4628	-0.4353	0.4633	-0.4620	-0.4633	-0.4620
PAGERANK	0.3877	-0.3833	-	-0.3833	-	-0.3820	-0.3829	-0.3820	-0.3829

Figure 3.14. Table containing correlations value between performance metrics and centrality measures of Anytown

Finally, in both WDS Anytown and ASnet2 correlation tables, we found second order centrality and current flow closeness centrality. While in the case of Anytown, these two centrality measures have the same behaviour and they are really dependent on all the performance metrics, in the case of ASnet2, these are the second and the third with higher absolute correlations. In both cases, we found in the set of centrality measures with correlation higher than 3.5 close centrality, current flow closeness centrality, current flow betweenness centrality, approximate current flow betweenness centrality, second order centrality, and harmonic centrality. Interesting is the fact that in both cases we have current flow betweenness centrality and approximate current flow betweenness centrality but there is no betweenness centrality. This could be due to the fact that current flow betweenness centrality uses an electrical current model for information spreading in contrast to betweenness centrality which uses shortest paths.

Chapter 4

Conclusions

This thesis, developed at the end of the Master's course in Engineer in Computer Science at Sapienza University of Rome, addresses the issue of smart water management and in particular the problem of water loss in Water Distribution Systems.

The biggest difficulties in smart managing a WDS are due to the fact that it is very expensive deploy existing sensing technologies on each node of the WDS and it is complicated to keep an eye on the entire network. In fact, a WDS can have thousands of nodes that can be placed underground or in rural places where it is difficult to establish an internet connection. These do not make the monitoring of the entire WDS impossible but very expensive, and often the water supplier companies cannot invest that much to digitize the entire network.

From this comes the need to make the most of the available resources and to design sustainable solutions for managing in a smart way a WDS.

In this thesis, we start from the idea of Lu and Sela [4] to find a relation between the State Estimation performances and the physical and topological properties of the WDS. In fact, the main goal of that thesis is to find a criterion to choose the best nodes to monitor to have the best State Estimation performance by looking at the topological and physical properties of a WDS.

This work starts presenting EPANET which is a very powerful software to create a WDS digital model taking as input a file containing all the properties of the WDS and to perform hydraulic simulations on a WDS. We have seen the most important objects that make up a WDS, and the properties of each different object. Then we have talked about the need to include hydraulic simulations in a Python software and how we can do that using the Python package WNTR which is EPANET compatible and through which we can create a digital water network model and perform hydraulic simulations through the EPANET engine.

A very important task in this work is to model the WDS as a graph that can represent not only the topological aspects of the WDS but also its physical properties, such as the diameter, length, and roughness of its pipes. Such a graph is a graph where the edges are weighted taking into account the length, diameter, and roughness

of the correspondent pipes and the nodes are weighted based on the demand and the head of the correspondent nodes. Once having the graph model we can compute some centrality measures that we will use to find some relations with the state estimation process performances.

Then it has been shown how to generate the data to train, validate and test the model by performing multiple hydraulic simulations and adding some noise to the WDS structure. The structure of the generated data has been clearly presented.

At this point the GNN-based model has been presented, we have seen the input of the model, the output, and its structure. The GNN model is composed of three blocks, encoder, decoder, and processor. The encoder formats the input and initializes the latent variables, the decoder decodes the processor output as meaningful output, i.e. heads values, and the processor is where the latent states are updated through message passing. Particular attention has been made in explaining how the processor works and how the messages are propagated considering the topological aspect of the WDS.

Semi-supervised learning and physics-informed machine learning have been presented and it was explained why it is a good choice to use them in our case. In particular semi-supervised learning fits with the problem of partial observation of the nodes and physics-informed machine learning includes in the loss functions of the model the hydraulic physical laws making the predictions of the model more accurate.

At this point, all the models and algorithmic methods have been presented and we start looking at the practical part of this work. At first, the two WDS used in this work, ASnet2 and Anytown, have been presented showing their main properties. Then we discussed the choice to use an Amazon Elastic Compute Cloud to train our model, strictly related to the training computational time, and how to set all the training parameters.

Since we are interested in finding a relation between SE performances and centrality measures, we have performed N SE processes, with N number of the junctions of the WDS, considering for the n^{th} SE process to monitor only the n^{th} junction. The SE performances in the two cases are quite different. With ASnet2 we have good performances (figure 3.6), high correlation for both flow and head and low root mean squared errors and average mean errors. With regard to Anytown the correlation values are low and the error indicators are quite high (figure 3.7). Furthermore, in the results on Anytown there is an outlier, junction number 20, whose performance metrics are much higher than the other. interesting is the fact that junction 20 is the only one connected with the pump that may be the cause of the bad SE performance and may represent a criterion to discard a junction from the eligible to be observed in advance.

The centrality measures of the two graph models of Anytown and Asnet2 have been computed (Figures: 3.8, 3.8, 3.10, and 3.11)) and we have computed the correlations between them and the SE performance metrics of the correspondent WDS.

We have set a threshold to retrieve only the most correlated couples of performance metrics and centrality measures. Also in this case the obtained results are different for the two WDS. The absolute correlations for ASnet2 are quite low (Figure 3.13), under 0.7, and the best correlated performance metric is the mean average error on the head with close centrality. With regard to the correlation values of ASnet2 3.14), we have that all the performance metrics are strictly correlated with current flow closeness centrality and second order centrality with absolute values between 0.96 and 0.99. This is coherent with the fact that all the performance metrics of Anytown, if we do not take into account junction 20, do not vary a lot but in a small range and both current flow closeness centrality and second order centrality are constant. In both cases, current flow closeness centrality and second order centrality are in the set of centrality measures with higher correlation with the performance metrics.

At the end of this thesis, we have learned a lot about how the GNN model handles a WDS discovering that is very dependent of the goodness on the WDS and of its hydraulic behaviour. We have seen that a pump can cause a huge decrease in performance and it may need to be modeled in a different way. With regard to the dependency between prediction performances and centrality measures, the two WDS give very different results, in both cases, the used protocol seems to give coherent results.

All along this work we went through all the challenges described at the beginning of this work, in section 1.1. Analyzing the obtained results, we can conclude that the two challenges of modeling a WDS as an appropriate graph model and design a GNN-based model to perform SE on a WDS have been solved. The third challenge of find a criterion to choose the best junctions to perform SE has not been solved yet, however, the protocol proposed in this thesis to find such criterion is a promising one and deserves to be deepened by testing it on other WDS to confirm strengths and weaknesses observed during this work.

Bibliography

[1] Amaxilatis Dimitrios, Ioannis Chatzigiannakis, Christos Tselios, Nikolaos Tsironis, Nikos Niakas, and Simos Papadogeorgos. (2020). "A Smart Water Metering Deployment Based on the Fog Computing Paradigm". Applied Sciences 10, no. 6: 1965. doi: 10.3390/app10061965

[2] M. Zecchini, A. A. Griesi, I. Chatzigiannakis, D. Amaxilatis and O. Akrivopoulos. (2021). "Identifying Water Consumption Patterns in Education Buildings Before, During and After COVID-19 Lockdown Periods". 2021 IEEE International Conference on Smart Computing (SMARTCOMP), pp. 335-340. doi: 10.1109/SMARTCOMP52413.2021.00069.

[3] EurEau. (2021) "Europe's Water in Figures: An overview of the European drinking water and waste water sectors." `https://www.eureau.org/resources/publications/eureau-publications/5824-europe-s-water-in-figures-2021/file`

[4] Lu Xing, Lina Sela. (2022) "Graph Neural Networks for State Estimation in Water Distribution Systems: Application of Supervised and Semisupervised Learning", Journal of Water Resources Planning and Management.

[5] World Health Organization. (2014) "Water safety in distribution systems". World Health Organization. `https://apps.who.int/iris/handle/10665/204422`

[6] Rossman, L., Woo, H., Tryby M., Shang, F., Janke, R., and Haxton, T. (2020) "EPANET 2.2 User Manual". U.S. Environmental Protection Agency, Washington, DC, EPA/600/R-20/133.

[7] Klise, K.A., Murray, R., Haxton, T. (2018) An overview of the Water Network Tool for Resilience (WNTR), In Proceedings of the 1st International WDSA/CCWI Joint Conference, Kingston, Ontario, Canada, July 23-25, 075, 8p.

[8] M. E. J. Newman. (2005) "A measure of betweenness centrality based on random walks, Social Networks", 27, 39-54.

[9] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang and P. S. Yu. (2021) "A Comprehensive Survey on Graph Neural Networks," in IEEE Transactions on Neural Networks and Learning Systems, vol. 32, no. 1, pp. 4-24, doi: 10.1109/TNNLS.2020.2978386.

[10] Stephen R. Mounce and John Machell. (2006) "Burst detection using hydraulic data from water distribution systems with artificial neural networks", Urban Water Journal, 3:1, 21-31, DOI: 10.1080/15730620600578538

[11] Ramsundar, B. and Zadeh, R.B. (2018) "TensorFlow for Deep Learning: From Linear Regression to Reinforcement Learning", chapter 4, ISBN: 9781491980453

[12] J. Kang, Y. -J. Park, J. Lee, S. -H. Wang and D. -S. Eom. (2018) "Novel Leakage Detection by Ensemble CNN-SVM and Graph-Based Localization in Water Distribution Systems," in IEEE Transactions on Industrial Electronics, vol. 65, no. 5, pp. 4279-4289, doi: 10.1109/TIE.2017.2764861.

[13] F. Scarselli, M. Gori, A.C.T., M. Hagenbuchner, and G. Monfardini. (2009) "The graph neural network model," IEEE Trans. Neural Netw., vol. 20, no. 1, pp. 61–80.

[14] Zhou, J.,G.Cui,S.Hu, Z.Zhang,C.Yang, Z.Liu, L.Wang,C.Li, andM.Sun. (2020) "Graph neural networks: A review of methods and applications". AI Open 1 (Jan): 57–81. doi: 10.1016/j.aiopen.2021.01.001.

[15] Olivier Chapelle, Bernhard Schölkopf, Alexander Zien (2006) "Semi-Supervised Learning", Semi-Supervised Learning, chapter 1, The MIT Press, https://www.molgen.mpg.de/3659531/MITPress–SemiSupervised-Learning.pdf

[16] Karniadakis, G.E., Kevrekidis, I.G., Lu, L. et al. (2021) "Physics-informed machine learning.", Nat Rev Phys 3, 422–440. https://doi.org/10.1038/s42254-021-00314-5

[17] Alperovits, E., and Shamir, U. (1977), Design of optimal water distribution systems, Water Resour. Res., 13(6), 885– 900, doi:10.1029/WR013i006p00885.

[18] Walski, Thomas M., "01 Anytown" (2016). Battle of the Water Network Models. 1. https://uknowledge.uky.edu/wdst_models/1