ABSTRACT

Title of dissertation: EVENT-DRIVEN OPERATION OF DISTRIBUTED SYSTEMS WITH ARTIFICIAL INTELLIGENCE TECHNOLOGIES AND BEHAVIOR MODELING

Maria Eduarda Montezzo Coelho
Doctor of Philosophy, 2022

Dissertation directed by: Associate Professor Mark A. Austin
Department of Civil and Environmental Engineering,
and Institute for Systems Research

This dissertation aims to enhance decision making in urban settings by integrating artificial intelligence technologies with distributed behavior modeling. Today's civil engineering systems are far more heterogeneous than their predecessors and may be connected to other types of systems in completely new ways, making the task of system design, analysis and integration of multi-disciplinary concerns much more difficult than in the past. These challenges can be addressed by combining machine learning formalisms and semantic model representations of urban systems, that work side-by-side in collecting data, identifying events, and managing city operations in real-time. We exercise the proposed approach on a problem involving anomaly detection in an urban water distribution system and a metrorail system.

Keywords: Digital Twins; Urban Operations; Semantic Modeling; Behavior Modeling; Ontology; Rules; Machine Learning; Deep Learning; Graph Embeddings; Graph Autoencoders; Anomaly Detection; Water Distribution Systems; Metrolrail Systems.

Event-Driven Operation of Distributed Systems with Artificial Intelligence Technologies and Behavior Modeling

by

Maria Eduarda Montezzo Coelho

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2022

Advisory Committee:
Associate Professor Mark A. Austin, Chair/Advisor
Professor Deb A. Niemeier
Professor Gregory Baecher
Assistant Professor Allison C. Reilly
Professor Jefferey W. Hermann

Acknowledgments

I would first like to thank my dissertation advisor Dr. Mark Austin of the Department of Civil and Environmental Engineering at the University of Maryland, for his enthusiasm for this project, for his support, encouragement and patience. Working with him has been a joy and I could not have imagined having a better advisor and mentor.

I would also like to thank the Systems Engineering Research Center of the Stevens Institute of Technology for sponsoring this project and, in particular, Dr. Mark Blackburn for trusting my capabilities and providing valuable guidance throughout this journey.

Many thanks to Dr. Deb Niemeier, Dr. Allison Reilly, Dr. Greg Baecher, and Dr. Herrmann for serving on my dissertation committee, and for their valuable input. A special thank you to Dr. Deb Niemeier and Dr. Allison Reilly for allowing me to learn from their research and participate in their projects. Those experiences have contributed to this dissertation in many ways.

Thank you to previous graduate students of my research team, Parastoo Delgoshaei, Leonard Petnga, Sachraa Borjigin. This dissertation would not have been possible without your previous contributions, and your support through my studies.

Finally, I must express my very profound gratitude to my parents Rachel and Luiz, to my two brothers Mateus and Joao, to my grandparents Elio, Elza, Anna and Luiz, to my love Zabdiel, and to my dear friends for providing me with unfailing support and continuous encouragement throughout my years of study and through the process of researching and writing this dissertation. This accomplishment would not have been possible without them. Thank you.

Table of Contents

List of Figures

Glossary of Terms

This glossary provides definitions of key terms employed in this work:

Action: (Effect) is the response given to stimuli in a state transition.

Activation Function: A function (for example, ReLU or sigmoid) that takes in the weighted sum of all of the inputs from the previous neural network layer and then generates and passes an output value (typically nonlinear) to the next layer.

Anomaly Detection (AD): Anomaly detection is the process of finding outliers in a given dataset. Outliers are the data objects that stand out amongst other objects in the dataset and do not conform to the normal behavior in a dataset.

Application Program Interface (API): Set of routines, protocols, and tools for building software applications. An API specifies how software components should interact.

Association: A type of machine learning model for discovering interesting relations between variables in large databases.

Architecture: The fundamental structures of a software system and the discipline of creating such structures and systems. Each structure comprises software elements, relations among them, and properties of both elements and relations.

Artificial Intelligence: (AI) The theory and development of computer systems able to mimic the problem-solving and decision-making capabilities of human intelligence. Machine learning is a sub-field of artificial intelligence. However, in recent years, some organizations have begun using the terms artificial intelligence and machine learning interchangeably. This work uses the term machine learning as a sub-field of artificial intelligence not a substitute.

Attributes: Synonym for feature. Associated values belonging to a graph's vertices or edges. These can represent some property, like data about how the graph was constructed, the color of the vertices when the graph is plotted, or simply the weights of the edges in a weighted graph.

City Operating System: Essential hardware, software and data components that supports a city's basic functions by directing urban flows and providing shared languages towards interoperability across multiple infrastructures.

Classification: A type of machine learning model for distinguishing among two or more discrete classes.

Clustering: A type of machine learning model for grouping related examples.

Constraint: A design constraint refers to some limitation on the conditions under which a system is developed.

Data Mining: Process of extracting and discovering patterns in large data sets.

Decision Plane: The separator between classes learned by a model in a binary class or multi-class classification problems.

Deep Learning: Process of learning through deep neural networks.

Deep Neural Network: A type of neural network containing multiple hidden layers.

Depth First Search: Algorithm for traversing or searching tree or graph data structures. The algorithm starts at the root node (selecting some arbitrary node as the root node in the case of a graph) and explores as far as possible along each branch before backtracking.

Digital Twin: Virtual representation that serves as the real-time digital counterpart of a physical object or process.

District Metered Areas: small clusters of water users with a provision to individually monitor the water supplied and consumed.

Enterprise Resource Planning (ERP): Enterprise resource planning is a software system used by organizations to manage and integrate all of the processes needed to run the organization.

Event: Stimuli that may cause a transition from one system state to another system state.

First-order logic (FOL): Symbolized reasoning in which each sentence, or statement, is broken down into a subject and a predicate. The predicate modifies or defines the properties of the subject. In first-order logic, a predicate can only refer to a single subject.

Graph: Abstract data type consisting of nodes (also called vertices) that are connected by edges (also called links).

Graph Structure: The arrangement of and relations between the objects of a graph.

GPU Processing Time: How much time the computer's Graph Processing Unit took to perform calculations. GPUs were originally designed to create images for computer graphics and video game consoles, but since the early 2010's, GPUs can also be used to accelerate calculations involving massive amounts of data in high performance computing (HPC), deep learning, and more.

Hidden Layer: A synthetic layer in a neural network between the input layer (that is, the features) and the output layer (the prediction). Hidden layers typically contain an activation function for training. A deep neural network contains more than one hidden layer.

Information and Communication Technology: (ICT) Technologies that provide access to information through telecommunications.

Learning Curve: Plot of model learning performance over experience or time.

Machine Learning: A program or system that builds (trains) a predictive model from input data. The system uses the learned model to make useful predictions from new (never-before-seen) data drawn from the same distribution as the one used to train the model. Machine learning also refers to the field of study concerned with these programs or systems.

Model: A model is an approximation, representation, or idealization of selected aspects of the structure, behavior, operation, or other characteristics of a real-world process, concept, or system.

Data Model: Model that organizes data elements and standardizes how the data elements relate to one another.

Semantic Model: Model that represents the implicit meaning of the data organizing it into categories, with clearly defined relationships among specific values of data, and rules that describe behavior. They can be constructed using ontologies and reasoning mechanisms.

Neural Network: A model that, taking inspiration from the brain, is composed of layers (at least one of which is hidden) consisting of simple connected units or neurons followed by nonlinearities.

Neurons: A node in a neural network, typically taking in multiple input values and generating one output value. The neuron calculates the output value by applying an activation function (nonlinear transformation) to a weighted sum of input values.

Ontology: A model that describes what entities exist in a design domain, and how such entities are related.

Ontology Class: A placeholder for an entity in the system design. An ontology class may have some dataType or objectType properties.

Ontology Instance: An ontology instance is a specific realization of any ontology class object. An object may be varied in a number of ways. Each realized variation of that object is an instance. The creation of a realized instance is called instantiation.

Reasoner (Rule Engine): Piece of software able to infer logical consequences from a set of asserted facts or axioms.

Reasoning: To infer new statements based on set of asserted facts in the ontology.

Recovery Procedures: Process that attempts to bring a system back to a normal operating state.

Rules: Set of asserted facts or axioms governing graph transformations within a particular domain ontology.

Rule Checking: A mechanism that ensures existing data in the ontology is consistent with rules defined over the ontology. A rule engine often performs this task.

Semantic Feature Engineering: Process of using domain knowledge to extract features (characteristics, properties, attributes) from raw data for later use in machine learning algorithms.

Topology: Synonym for graph structure.

Training: The process of determining the ideal parameters comprising a model.

Trainable Parameters: A variable of a model that the machine learning system trains on its own. For example, weights are parameters whose values the machine learning system gradually learns through successive training iterations.

List of Abbreviations

AD: Anomaly Detection.

AI: Artificial Intelligence.

API: Application Program Interface

DC: District of Columbia.

DFS: Depth-first search

DNGR: Deep Neural Network for Graph Representations.

ERP: Enterprise Resource Planning.

FOL: First-Order Logic.

GAE: Graph Autoencoder.

GED: Graph Edit Distance.

GNN: Graph Neural Network.

ICT: Information and Communication Technology.

JAXB: XML Binding for Java.

ML: Machine Learning.

OWL: Web Ontology Language.

RDF: Resource Description Framework.

RDFS: Resource Description Framework Schema.

SDNE: Structural Deep Network Embedding.

SPARQL: SPARQL Protocol and RDF Query Language.

URI: Uniform Resource Identifier.

XML: Extensible Mark-up Language.

Chapter 1: **Introduction**

1.1 Problem Statement

Current urban infrastructure has been marked by outstanding advances in information Communication Technology (ICT). Such advances allow for the creation of a wealth of data that can be used to drive better decision making. However, often times, smart technology initiatives are seen as individual strategies (i.e. smart parking, smart security, smart traffic). Each domain is treated as a standalone implementation project rather than part of a comprehensive whole [37]. The resulting solutions, while smart and capable, are being implemented as silos, hampering the potential for reaching superior levels of performance in urban operation. Large-scale urban systems have a distributed, concurrent, and multidisciplinary nature. When a disruption to urban operations requires the realization of recovery procedures, it is critical that the participating domains share information to establish common knowledge, and enhance their individual ability to make decisions appropriate to their understanding of the system state and objectives.

The motivating tenet of this dissertation study is that the concept of digital twins can overcome this integration barrier. A digital twin is a cyber representation of an object, process or place that mirrors its implementation in the physical world

through real-time monitoring, and synchronization of data with events. NASA initially proposed the digital twin concept in the late 90s as a way to support the design and operation of air vehicles [26]. Since then the range of potential applications has expanded to include automotive components, manufacturing processes, power plants, design of networks of wind turbines, and smart cities [40, 43, 44, 47, 55]. Cities around the world are entertaining use of the digital twin concept as a way to transform processes for day-to-day management and long-term urban planning and design [51]. A digital twin can offer a holistic view of the changes that take place in an urban environment by generating feedback loops of interactions between human systems (i.e., government, businesses, residents), infrastructure systems, and technology systems (i.e., devices, sensors, and data analytics infrastructure).

At this time, however, many challenges remain in digital twin design and implementation. There is a lack of unified models or a generic digital twin architecture in the literature, with no consensus on how to build a digital twin system [25, 51, 52]. Multiple digital twin definitions have lead to different capability requirements. Sharma et al. reviewed the digital twin literature and found that there have been digital twin components and properties which have existed in some works and have been missing in others [52]. In the case of urban environments specifically, the lack of standards supporting cross-disciplinary data exchange and the complexity of city-wide modeling create further design difficulties. Research is needed to understand how to design the digital twin components and their relationship, so that collectively they can overcome these challenges. A first step in this direction is defining urban digital twin capability requirements which can be

generalized across domains.

1.2 Digital Twins for Urban Systems

Urban Data Hubs. The implementation of urban data hubs, a precursor to digital twins, is rapidly expanding in many cities accross the world. As a case in point, a data-driven city operations center in Rio de Janeiro, Brazil, pulls together into a single location real-time data streams from thirty agencies, including transportation, utility, emergency and security services, into a single location. The data is overseen by operators, who monitor and react to urban and environmental processes. However, the volume and rate of data generation in urban environments limits the scalability of this approach for city operation in years to come.

Strategies for bypassing human operators to manage communication between systems are also being explored: Microsoft's CityNext, IBM's Smarter City, Urbiotica's City Operating System, and PlanIT's Urban Operating. These are effectively Enterprise Resource Planning (ERP) systems designed to coordinate the activities of large organizations repurposed for cities [35]. ERP systems are capable of sharing and transferring information across all stakeholders, but many limitations of ERP systems have been reported, including the lack of real-time response to changes [50].

Transition to Urban Digital Twins. Urban systems require digital twins components that are capable of identifying anomalies (faults) in system performance, modeling the behavior of different domains and interactions among them in real-time. Since a design paradigm for building a digital twin does not exist at this time, we believe that

the best pathway forward for digital twin construction is with architectures that combine machine learning (ML) formalisms and semantic model representations that work as a team collecting and processing data, identifying events, and managing city operations in real-time.

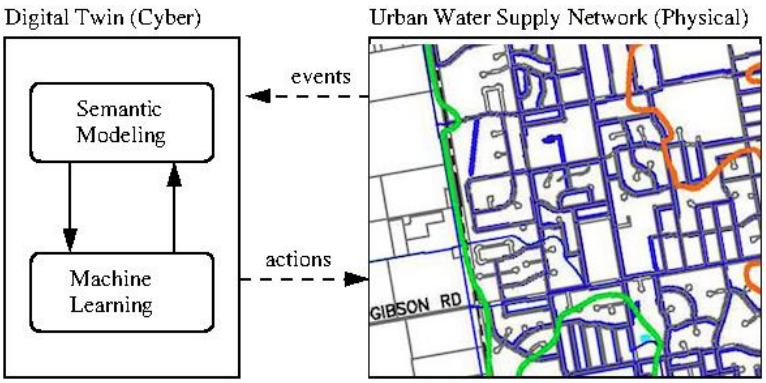


Figure 1.1: High-level representation for an urban water supply network digital twin interacting with a physical urban water supply network.

Figure 1.1 shows the application of this concept to a urban water supply system. The physical side of the problem is defined by networks of pipes, pumps, controllers, storage tanks, and schedules for end-user demand. Simulation procedures can compute time-histories of pressure and velocity throughout the water supply network versus time. In the digital twin setup, the cyber component of the water supply digital twin will mirror the physical world through real-time monitoring and synchronization of data with events. To achieve these purposes, an implementation needs to provide software and algorithms for observation, reasoning and physical systems control.

We also believe that artificial intelligence (AI) and machine learning (ML) will be deeply embedded in new software and algorithms. Within this context, AI entails knowledge representation and reasoning with ontologies and rules, construction of semantic

graphs, executable event-based processing, and multi-domain reasoning. ML entails use of modern neural networks for input-output prediction, data mining, and identification of objects and anomalies in data streams. Since AI and ML technologies are fragmented in their capabilities, this study explores design of digital twin architectures that support AI and ML formalisms working side-by-side as a team. Thus, looking forward, a key research challenge is how to design digital twin elements and their interactions to support digital twin operating system environments for urban observation, reasoning and control.

1.2.1 Machine Learning

ML techniques deviate from traditional models of computation in their ability to perform data analytics by learning patterns and hidden insights in data without being explicitly programmed for it. ML algorithms provide comprehensive support for the classification, clustering, and identification of association relationships in streams of data. Certain ML algorithms include the ability to detect anomalies in expected temporal behavior. Other ML algorithms include the ability to detect anomalies in the structure of a graph and its attributes. For our purposes, these anomalies are events that can trigger the activation of urban recovery procedures. However, urban recovery procedures are far more complex than just identifying anomalies. They require explicit reasoning using knowledge about the system, a task semantic modeling can complete with ease.

1.2.2 Semantic Modeling

Semantics is the study of meaning. Semantic models represent the implicit meaning of the data organizing it into categories, with clearly defined relationships among specific values of data, and rules that describe behavior. They can be constructed using ontologies and reasoning mechanisms [20, 21]. An ontology is the formal definition of categories, its properties, its relationship to other categories, and inference rules for some particular topic or domain [30]. Inference rules and their associated reasoning mechanisms provide a way to dynamically combine the knowledge to answer questions (backward inference) or to draw conclusions (forward inference).

1.2.3 Semantic Modeling and ML Architectural Template

Combining Semantic Modeling and ML in an urban context allows digital twins to work as city operating system, capable of monitoring and controlling urban processes, as shown in Figure 1.2. Engineering designers, infrastructure operators, disaster-relief personnel and urban planners could greatly benefit from such a system. Often times, they are put in a tough spot where quantitative decision-making regarding the adequacy of urban infrastructure is complicated by the presence of new found system interactions, incomplete knowledge of the system state, and break downs of communication among urban networks.

A similar problem, although on a different level, was encountered and solved in the latter part of the last century with computer operating systems. Operating systems

provided a consistent application program interface (API) so that all applications use hardware in the same way. This was particularly important once more than one type of computer started using the same operating system. Software written on one computer can run on other types of computers. The idea of operating systems applied to an urban environment has increasingly gained space within discussions of future of cities. In this scenario, the city is the computer, the urban systems are the attached hardware devices, and the operating system has access to data and services from any source, simultaneously allowing mixtures of centralized and decentralized control of the underlying systems.

We expect that a city operating system will make decisions based upon streams of urban data/information corralled into centralized data systems. It will provide city stakeholders (residents, businesses, disaster-relief personnel, infrastructure operators, planners and engineers) with enhanced levels of situational awareness, and decision making support for the spatio-temporal management of urban infrastructure and services. Semantic representations and ML will be at the core of the city operating system, dynamically identifying and responding to events.

Diving deeper into the semantic modeling and ML interactions, Figure 1.3 shows the proposed architectural template for a combined semantic modeling and ML approach. It builds upon our work in semantic modeling for (multi-domain) system of systems [15–17], and extends our exploration of a combined semantic and ML framework exercised in energy efficient buildings [22] and water distribution networks [14, 18].

Box 1: Multi-Domain Semantic Modeling. By their very nature, urban systems are a composition of many different types of domains and their interactions. These domains

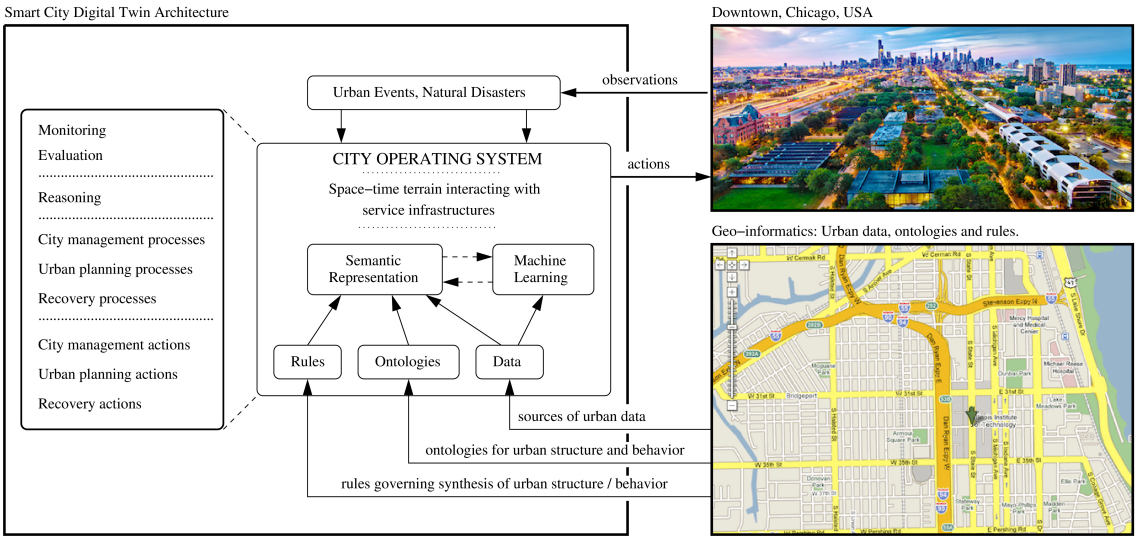


Figure 1.2: Smart city digital twin architecture and operating systems view of urban behaviors, city management, planning and recovery processes and actions [4].

are geographically dispersed and intertwined, and have behaviors that are distributed and concurrent. Box 1 presents a framework for multi-domain semantic modeling, where ontologies, rules and data models are developed concurrently, and placed on an equal footing. Instead of modeling the dynamic behavior of systems with a centralized control and one large catch-all ontology, we model systems as collections of domain-specific ontologies that dynamically evolve in response to events. This approach to semantic modeling forces cross-domain data exchange to be more homogeneous than it would otherwise be, and establishes common knowledge among domains.

Box 2: Data Mining. Box 2 shows ML for three styles of learning to gain insight into the data. Classification algorithms are predictive calculations used to assign data to preset categories by analyzing sets of training data. Clustering algorithms identify categories of data instances that belong together. Knowledge of these categories is useful for architecting ontologies and related properties in a manner that is consistent with the

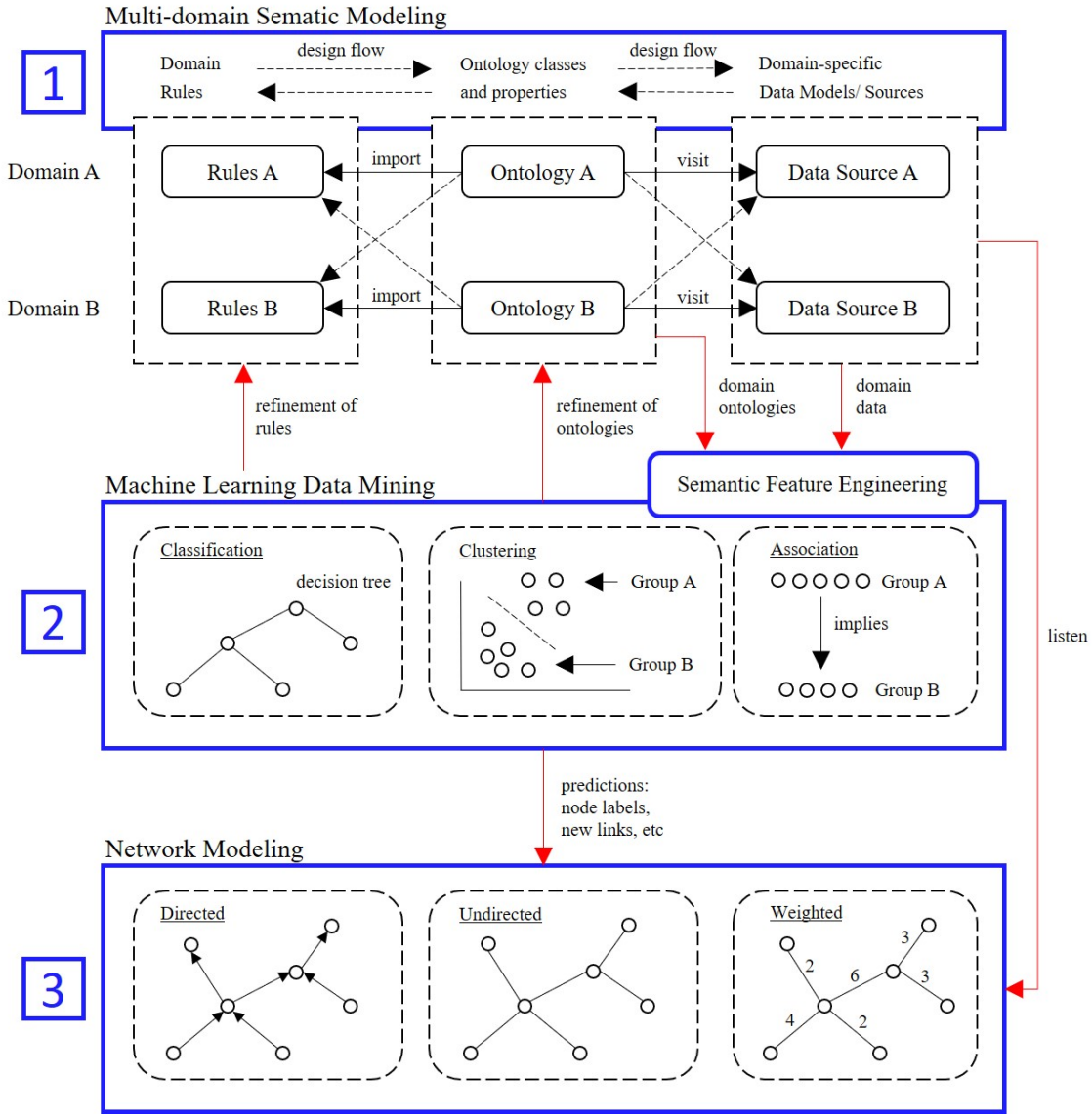


Figure 1.3: Digital twin architecture.

underlying data. Association algorithms identify rules that strongly associate different data attribute values, possibly spanning domains, and then use this knowledge to develop cross-domain rules. Just like ML can contribute to the semantic modeling effort, the semantic model can also contribute to learning through semantic feature engineering [62]. Semantic feature engineering uses domain knowledge and rules to infer features that can serve as meaningful input to ML algorithms.

For the most part, these three data mining techniques were developed in the 1980s and 90s, and so the associated algorithms and software [57] are now quite mature. Data mining techniques also include use of recurrent neural network architectures to represent temporal sequences, and algorithms to detect anomalies in expected temporal behavior. For our purposes, these anomalies are events that can trigger the activation of urban recovery procedures modeled in Box 1.

Box 3: Machine Learning of Graphs. Box 3 focuses on teaching machines to learn different types of large-scale graphs. In this case, the machine learns a function to make a prediction of a defined label based on the input graph structured data, creating a network model. From an urban perspective, its is extremely important that digital twins are capable of learning on graph structured data. Remarkable advances in ML algorithms (2016-2019) include the ability of a machine to learn the structure of a graph and its attributes. These learned network models can be later used to advance various learning tasks, such as node classification, node clustering, node recommendation, link prediction, and so forth.

1.3 Research Scope and Objectives

This dissertation aims to advance knowledge in part of the large urban digital twin design problem, namely, understanding ways in which Semantic Models and ML can work together to support event- and data-driven decision making. The investigation will be divided into three parts and a total of seven research questions:

1. **Learning Graph Topology** Because urban applications are safety and life critical, learned graph representations have to be accurate and complete. This requirement sets our work apart from other graph learning procedures that can allow approximations (e.g. social networks). In addition, urban graphs come in different shapes, sizes, and can have dynamic topologies. The latter suggests a strong need for compositional approaches to the learning and assembly of network models. Therefore, a key research objective is to understand (1) what types of graph topology can ML learn, (2) what machine architectures and learning strategies can guarantee accurate graph representation, (3) what are the effects of graph size on learning performance, and (4) how can the problem of learning large-scale graphs be decomposed.

2. **Anomaly Detection** Given that the semantic side of our proposed approach is event driven (i.e., semantic graphs respond to events), one of the key roles of ML is to identify anomalies in behavior from incoming data streams. If a machine has a spatial and temporal understanding of the network structure, then it will also have the ability to identify abnormal behavior. Additional research objectives are to understand: (4) how can learned graph representations help us model anomalies in

behavior, and (5) what machine architectures and learning strategies can identify outlier measurements.

3. **Semantic Modeling + ML** To create fully operational digital twin demonstrations in the future, our current work also needs to address the connection between the Semantic Model and ML modules. Although these modules have individual purposes of their own, they work hand-in-hand in order to achieve a greater common purpose. Therefore, it is extremely important that their designs are cohesive with one another for achieving an integrated digital twin. Part of that integration is dependent on an efficient protocol for transmission of events. Hence, further research objectives are to understand (6) what are the basic mechanisms for semantic/ML interaction, and (7) what design approaches should be taken for ensuring cohesion of digital twin modules.

1.4 Research Contributions

In the past five years, our work has advanced knowledge on the semantic modeling of multi-domain system of systems and provided innovative approaches for the architecting of smart city digital twins through ten publications [4,7–9,13–18], including two best paper awards at the International Conference on Systems in 2017 and 2020.

This dissertation research continues to contribute to the architecting of smart city digital twins by:

1. Identifying machine architectures and strategies of learning for a variety of graph structures.

2. Exercising the machine architecture and strategies of learning on case study problems (i.e. a water distribution system and an urban metrorail system).

3. Exploring the effects of graph size on learning performance.

4. Identifying scalability approaches for learning of large-scale graphs.

5. Identifying events in learned network models via time-series anomaly detection.

6. Establish basic mechanisms for semantic / machine learning interaction.

Figure 1.4 is a schematic for how the contributions of this dissertation might be integrated in an urban network digital twin. We start by extracting a graph representation of the urban infrastructure and determining the initial parameters of the system. As time progresses, the digital twin monitors changes in the system. Anomaly models

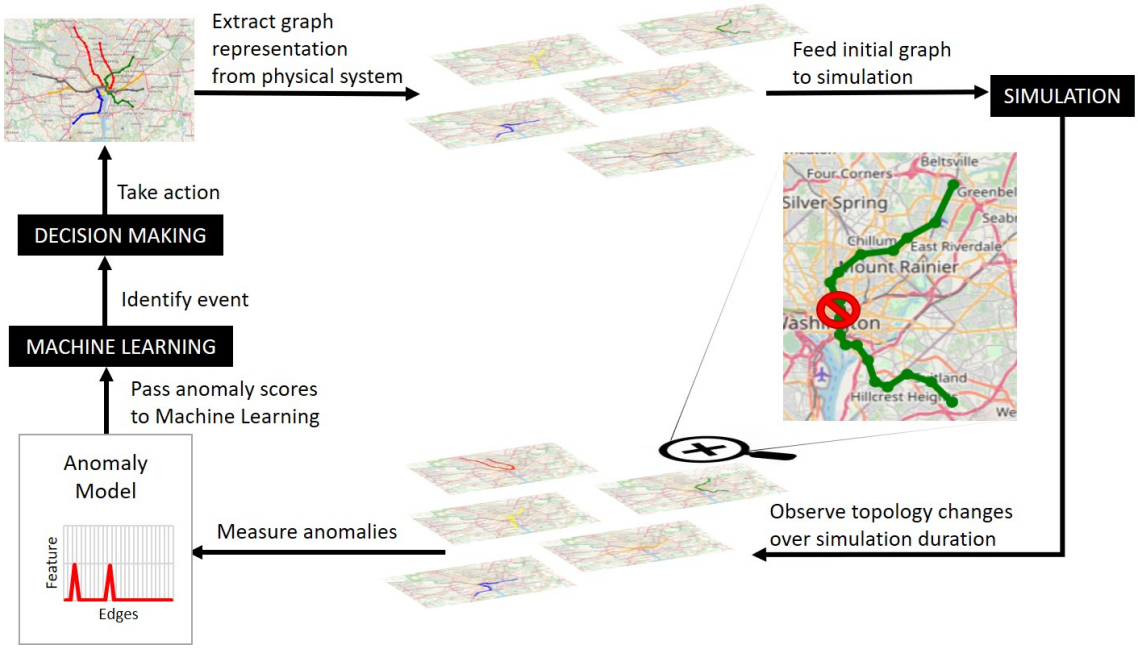


Figure 1.4: Process flowchart for training and executing machine.

are generated, and machines are trained to understand a number of salient features of acceptable and unacceptable behavior. When an unacceptable behavior is identified, urban recovery procedures are be triggered.

1.5 Dissertation Outline

The remainder of this dissertation proceeds as follows. Related work in ML algorithms for graphs and semantic modeling of graphs is covered in Chapter 2. Results in designing and exercising machine architectures on graph structures are presented in Chapter 3. The details of these exercises are reported in Appendices A, B and C. Results in designing and exercising anomaly detection procedures on learned network models are presented in Chapter 4. Results in designing and exercising event-driven decision-making with multi-domain semantic models are presented in Chapter 5. The design of basic

mechansims that support ML and semantic model modules interaction is covered in 6. Conclusions and proposed directions for future work are located in Chapter 7.

Chapter 2: **Related Work**

This chapter will discuss related work that has provided the basis for this dissertation. The first section will give an overview on graphs. Later in this chapter, we present semantic graphs and the technical infrastructure necessary for building them. Lastly, we introduce graph learning methods.

2.1 Graphs

2.1.1 Graph Theory

The interacting infrastructures found in an urban system form a network – a collection of discrete objects and relationships between them – that can usually be represented as a graph of nodes connected by edges. Typical urban networks include waterways, roadways, telephone lines, among others (see Figure 2.1). Graph theory is the mathematical study of graph structures. In mathematical terms, a graph is defined as G = (V, E), where V is a set of vertices (i.e. nodes), E = set of edges, and each edge is formed from pair of distinct vertices in V.

Studying urban systems and their interactions as networks, therefore, enables many useful applications of graph theory. The information to be preserved in the network

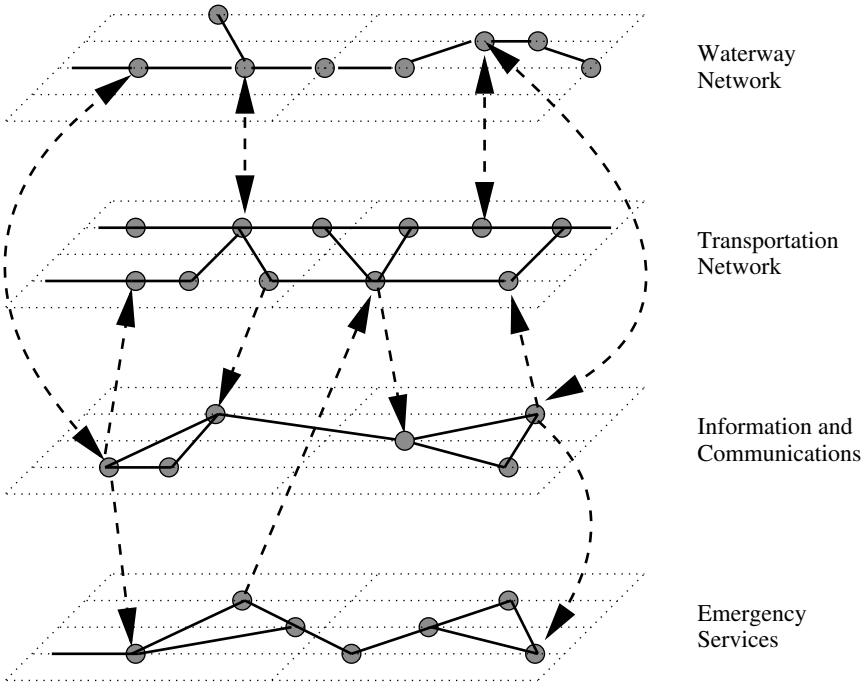


Figure 2.1: Illustration of urban systems represented as networks.

is strongly affected by the underlying characteristics of the system. Urban networks may be homogeneous, heterogeneous, and carry auxiliary information modeled as attributes. Their edges may be undirected, directed and/or weighted.

Kong and Simonovic [38] describe a few examples of how urban systems can be modeled as graphs. A street network can be represented as a graph composed of street junctions, end points and street segments. Generally, the edges are undirected, homogeneous, and the street network is fully connected. A water distribution network can be represented as graph where water works, storage facilities and pump stations are nodes with different attributes, and the water distribution pipes are directed edges, where the direction indicates the flow of water. A power grid can be represented by a graph where power plants, distribution and transmission substations are nodes with different attributes, and power lines are directed edges for the flow of electricity transmission. Information

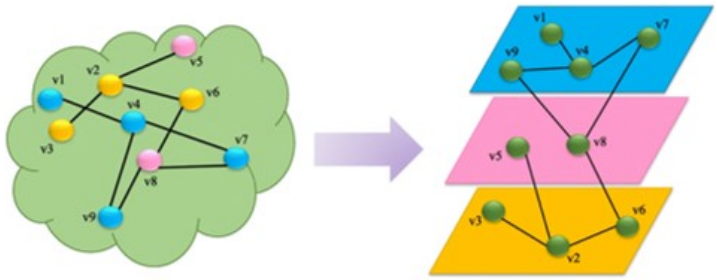
infrastructure can be represented as a graph where the Internet Service providers are nodes and cable connections are undirected edges, since these networks provide a bidirectional flow of information.

All of these city subsystems form a larger complex system as shown in Figure 2.1. Each subsystem has its own subsystems, without clear boundaries, and multiple layers of connectivity to other subsystems. As research on complex systems has matured, it has become increasingly essential to move beyond simple graphs and investigate more advanced graph structures, like multilayer networks [2, 36].

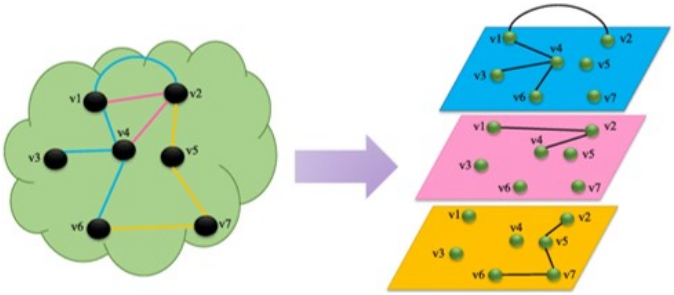
2.1.2 Graph Decomposition

Multilayer networks are decomposed graph structures capable of representing systems that consist of networks with multiple types of edges, or with other similar features. In a multilayer network a node u in layer α can be connected to any node v in any layer β. Layers represent aspects that characterize the nodes or the edges that belong to that layer. As a consequence, the set of edges can be partitioned into intra-layer edges, that is edges that connect nodes set in the same layer, and inter-layer edges, which connect nodes set in different layers. The choice of what constitutes a layer is application dependent and relies on the underlying characteristics of the network. Some examples include the type of network components, type of connections, time, or location (see Figure 2.2).

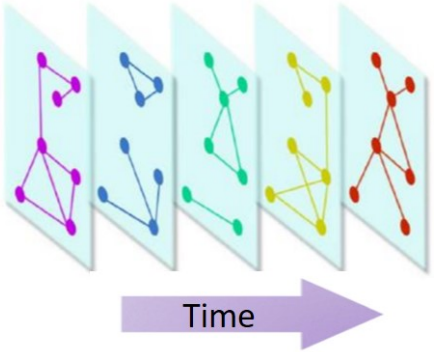
Kivelä et al. presented several types and denominations of multilayer networks such as multiplex network, multi-relational network, edge-colored network, node-colored network, multilevel network, multi-dimensional network, independent networks, networks



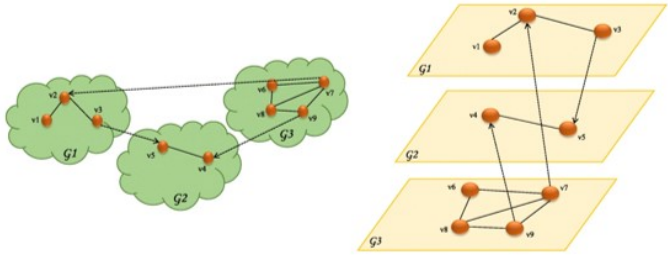
(a) Layers defined by component characteristics.



(b) Layers defined by connection characteristics.



(c) Layers defined by temporal characteristics.



(d) Layers defined by spatial characteristics.

Figure 2.2: Different layer definitions for multilayer networks [29].

of networks, temporal network and so on [36]. For instance, multiplex networks are networks where the same set of nodes is represented in every layer (although the interaction between nodes might be different in each layer), while networks of networks are formed by networks that are interlaced to each other but formed by different types of nodes.

A distinct tool for representation of a multilayer network, is the supra-adjacency matrix (i.e., "super-adjacency" matrix). The supra-adjacency matrix combines all the inter-layer and intra-layer adjacency matrices of the multilayer network into a single large matrix. It is 'filled' with all the intra-layer adjacency matrices on its diagonal, and with all the inter-layer adjacency matrices elsewhere. Figure 2.2 is a visualization of a multilayer network's supra-adjacency matrix. The supra-adjacency matrix representation of a network, allows for the storage of information pertaining to all layers of the network, and their relationships, in a single place.

Regardless of the type of graph and its purpose in an urban setting, data mining techniques can be applied to access the information contained in graphs. This information can then be used to improve the design and operation of urban systems. Techniques for graph data mining can be divided into graph analysis and graph analytics, and can be found in Section 2.3.

2.2 Semantic Graphs

2.2.1 Introduction to Semantic Graphs

As described in Chapter 1, ontologies give data values a logical structure, categories, and relationships, making it possible to express knowledge we have about its domain(s). For example, imagine we have information on artists, paintings and museums. An ontology can be created with classes that capture the unique types of elements that are in the data, in this example case artist, painting and museum. Relationships such as "painting has artist" or "painting is located at museum" allows us to represent how these categories relate to each other in a generalized way. Other properties, such as “painting has completion date,” are attributes, describing only one class, instead of connecting two classes together. These relationships and attributes might apply to any given painting, but they don’t necessarily have to apply to every painting. Therefore, ontologies are a general data model, meaning that they don’t include information about specific data values. Instead, they create a reusable framework that could be used to describe additional data values in the future. When we combine classes, relationships and attributes, we can view the ontology in a graph format as shown in Figure 2.3.

Using the ontology in Figure 2.3 as a framework, we can add in real data about individual paintings, artists, and museums to create a semantic graph as shown in Figure 2.4.

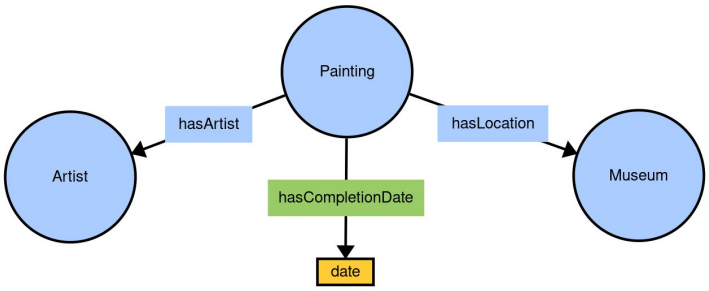


Figure 2.3: Simple example of a domain ontology.

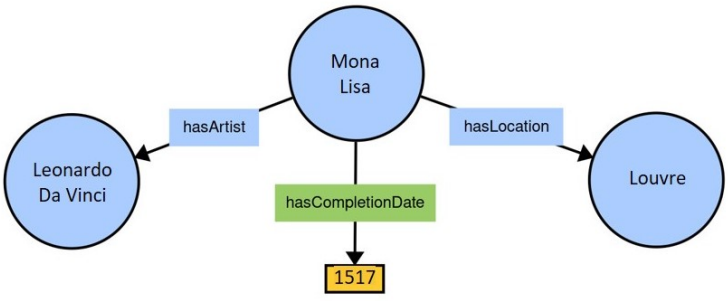


Figure 2.4: Simple example of semantic graph.

2.2.2 Urban Knowledge Representation with Semantic Graphs

The technological evolution of urban infrastructure with large amounts of data being now gathered in real-time, and the increasing interconnection between different urban systems, has lead a need to classify information in a meaningful way that can be machine-interpretable. Classification consists of organizing data according to their characteristics, a task that can be performed easily with ontology schemes. An overview on the use of ontologies in urban development projects was presented by Falquet, Metral, Teller and Tweed [23]. According to this study, ontologies have been developed for the geographic information sector, to model interconnections among urban models, and to describe urban mobility processes. Given the recent interest in Smart Cities, so called smart city ontologies have also been proposed. These ontologies contain an exhaustive list of elements you might find in a smart city, and proposals for relationships among elements.

By themselves, ontologies provide a framework for the representation of knowledge, but otherwise, cannot reason through all that information. This situation changes when ontologies are combined with domain-specific rules. Rules enable semantic graph transformations by formal reasoning and event-based input from external sources. For instance, lets suppose that our painting example ontology is coupled with a rule stating that in commemoration of a painting's 500th anniversary of completion, the painting shall visit different museums across the world during the year of anniversary, and return to its home museum for permanent display at the end of that year. Such a rule would impose

changes to the semantic graph as shown in Figure 2.4.

The use of ontologies in combination with domain-specific rules is particularly beneficial in urban applications, where logic is dynamic, and changes may be imposed on the system by external entities. A notable effort in this direction is the DogOnt ontology and rules for statechart behavior modeling of devices in home automation [10]. Ontologies and rules rely on the technical infrastructure of the Semantic Web.

2.2.3 The Semantic Web

In 1989 Tim Berners-Lee invented the World Wide Web, with the objective of providing members of the scientific community an automatic information-sharing tool [6]. Although the World Wide Web introduced the capability of linking documents on a network of machines through hyperlinks, end users still had to interpret the document's content. Years later, the World Wide Web was expanded to overcome that limitation and allow machines to interpret the semantic meaning of documents. This expansion is known today as the Semantic Web. The Semantic Web provides mechanisms that enable data elements to be parsed and structured. These elements are then assigned a "label" describing its meaning in a standardized language (i.e., markup language). Figure 2.6 illustrates the technical infrastructure that supports the Semantic Web vision, and the foundation upon which we will build our semantic models. Each layer exploits and uses capabilities of the layers below. A brief summary for each layer relevant to this dissertation is provided below.

URI and UNICODE: At the bottom of the semantic web stack, unicode provides

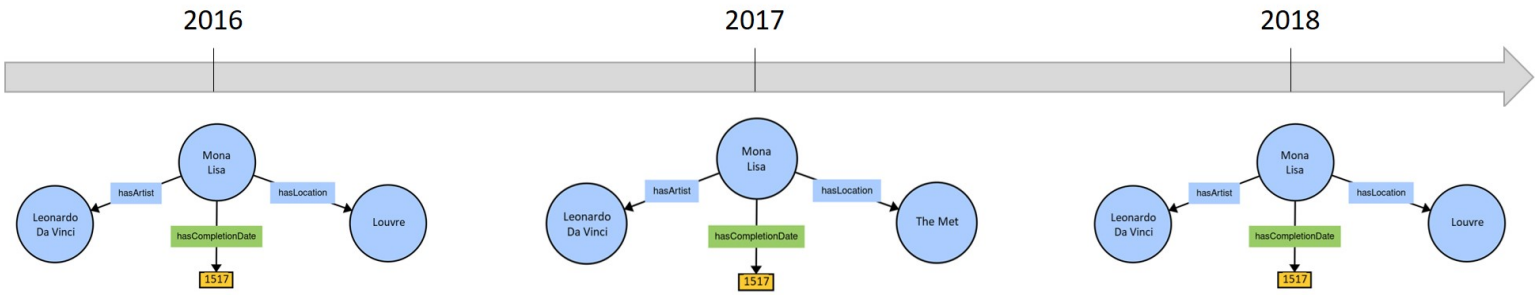


Figure 2.5: Example of simple semantic graph transformation.

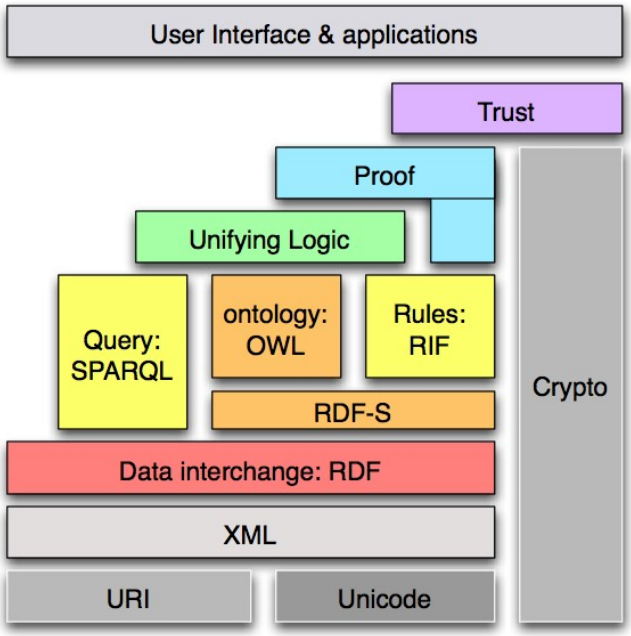


Figure 2.6: Technologies in Semantic Web Layer Cake [24].

support for multiple languages, and uniform resource identifiers (URI) allow for the unique identification of resources on the Web.

Extensible Markup Language (XML): XML provides the fundamental structure for representation and management of data, by organizing it into tree hierarchies. Information is wrapped in user-defined tags, that are human readable. In this way the information can be transported, and later searched and re-purposed by its receiver. Figure 2.7 shows an XML file for the painting example introduced earlier in this chapter. XML is, however, limited to storing and transporting information that can be organized within hierarchical relationships. This can be a problematic situation as a data value may or may not fit into a hierarchical (tree) model. A graph, on the other hand, can, and thus we introduce the Resource Description Framework (RDF).

```
<?xml version="1.0" encoding="UTF-8"?>
<Painting>
        <attribute text="Name" value="Mona Lisa"/>
        <attribute text="Completion Date" value="1517"/>
        <Artist>
                <attribute text="Name" value="Leonardo Da Vinci"/>
        </Artist>
        <Museum>
                <attribute text="Name" value="Louvre"/>
        </Museum>
</Painting>
```

Figure 2.7: Simple example of XML file.

Resource Description Framework (RDF): RDF allows for the representation of data as graphs. It is an assertional data model for describing the relationships within the data. An assertion is the smallest expression of useful information. RDF captures assertions made in simple sentences by connecting a subject to an object using a verb (i.e., "painting has artist"). In practice, English statements are transformed into RDF triples consisting of a subject (this is the entity the statement is about), a predicate (this is the named attribute, or property, of the subject) and an object (the value of the named attribute). Subjects are denoted by a URI. Objects are denoted by a ”string” or URI. The latter can be web resources such as requirements documents, other Web pages or, more generally, any resource that can be referenced using a URI (e.g., an application program or service program). Each predicate has a specific meaning, defines its permitted values, and the types of resources it can describe. A set of related RDF triples constitute an RDF graph.

RDF Schema (RDFS) and SPARQL: RDFS provides the basic vocabulary needed for RDF. SPARQL is a query language, and can be used to query any RDF-based data.

Web Ontology Language (OWL): OWL allows for data knowledge representation. It is based on the basic features of RDF introduced above but it strengthens it by adding structure and vocabulary for describing attributes and classes. They enable richer attribute definitions(e.g.: transitivity), class attribute restrictions(e.g.: allValuesFrom), and relationship between classes(e.g.: subClassOf). The additional capabilities allow ontological systems to infer new facts (triples) from existing ones with First-order logic (FOL).

Together, XML, RDF and OWL provide the necessary framework for the implementation of reasoning.

2.2.4 Apache Jena and Jena Rules

Apache Jena [3] is a widely used open source Java framework for constructing Semantic Web applications. It provides Java libraries for developers to build, work or analyse on RDF, RDFS, OWL and SPARQL. Jena includes a rule-based inference engine, known as Jena Rules, to perform reasoning based on OWL and RDFS ontologies. This inference engine uses facts and assertions described in OWL to infer additional facts from instance data and class descriptions. Such inferences result in structural transformations to the semantic graph model, as shown in Figure 2.5.

For the purposes of this study, Apache Jena and Jena Rules will be used to construct the multi-domain semantic modeling framework shown in Figure 2.8. In this framework, domain-specific ontologies, rules and data models are developed concurrently. This approach forces developers to think about the chain of dependency relationships between

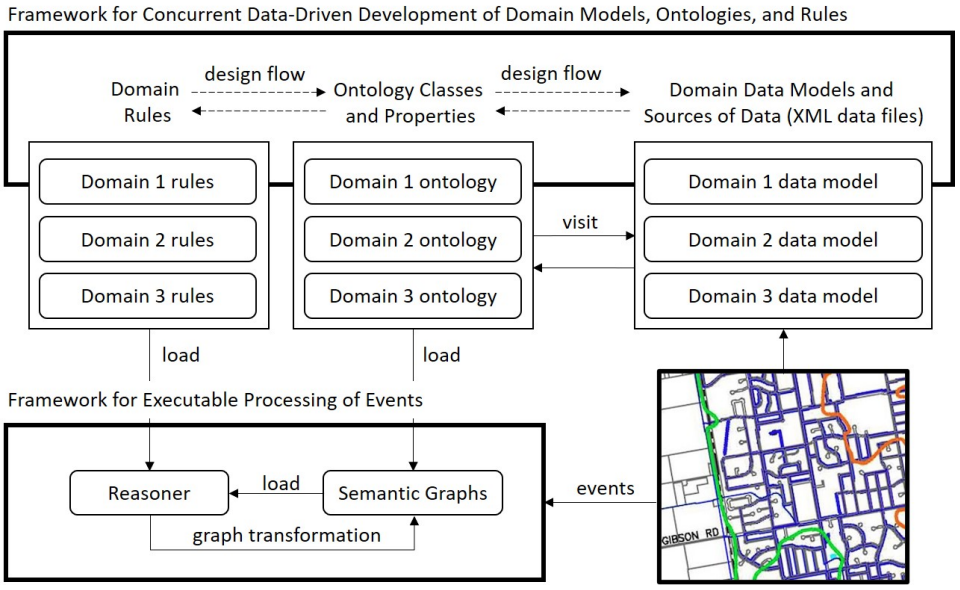


Figure 2.8: Architectural template for multi-domain semantic modeling and reasoning.

the data, ontologies and rules, and provide the data needed to support decision making [17]. Semantic graphs are populated with ontologies and data, and evolve in response to events. Jena Rules can be developed for determining necessary changes to the semantic graphs when events occur. The Jena Reasoner transforms the semantic graph by executing the changes dictated by the Jena Rules.

2.3 Graph Analysis and Analytics

2.3.1 Graph Analysis

Graph analysis tasks focus on exploring the information stored in graphs to gain insights about the properties of its topology. Network topology nearly always affects function. For example, the topology of social networks affects the spread of information and disease, and the topology of the power grid affects the robustness and stability of power transmission [54]. Therefore, it is important to model network topology.

Traditional approaches to network/graph modeling (see the upper half of Figure 2.9) employ adjacency matrices (or a simplified representation of network adjacency) to model the topology of graphs. The properties of the topology can then be extracted through graph analysis tasks such as connectivity analysis, traceability analysis, cycle detection, and shortest path identification. Traditional approaches to graph analysis do not capture the attributes of the network.

2.3.2 Graph Analytics

For problems that are high-dimensional and/or data sparse, traditional approaches to graph modeling and analysis can quickly become computationally prohibitive. In recent years, there has been a surge in ML approaches [27, 28, 31, 58, 59, 63] that automatically learn to encode graph topology and attributes into low-dimensional vector representations, known as graph embeddings. Figure 2.9 shows simplified representations of traditional

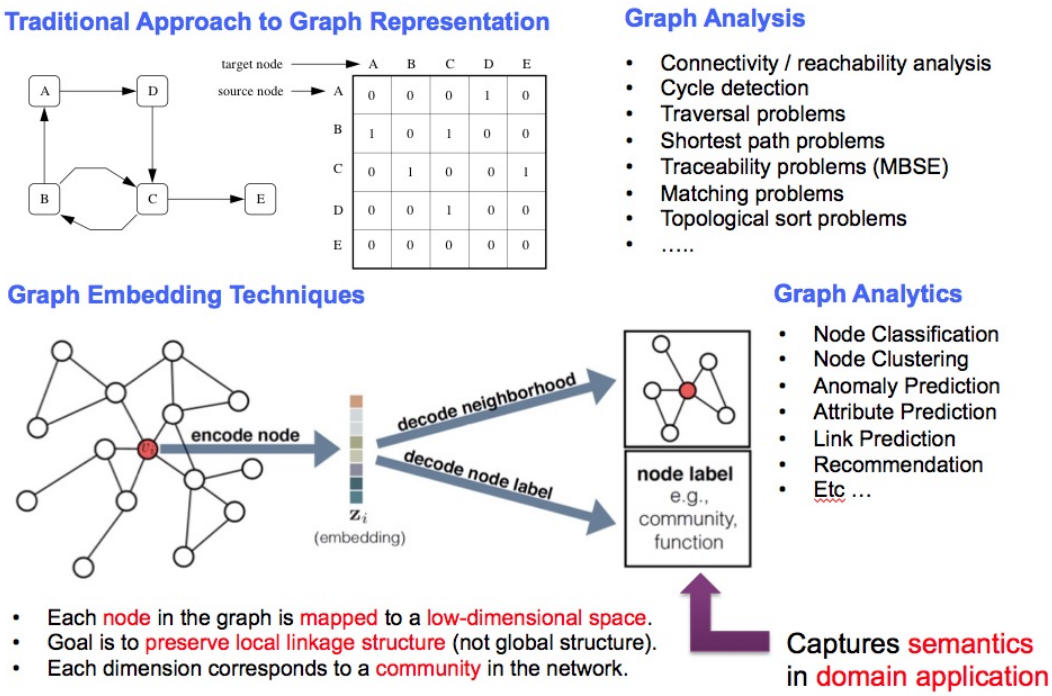


Figure 2.9: Graph analysis vs graph analytics.

and ML approaches to graph modeling.

Graph embedding methods learn a continuous vector space for the graph, assigning each node (and/or edge) in the graph to a specific position in the vector space, with the objective of preserving local linkage structure (not global structure) and/or network attributes.

The encoder maps each node to a low-dimensional vector, based on the node's position in the graph and in some cases its attributes. The decoder extracts information from the vector (i.e., node's local graph neighborhood, or a classification label associated with the node). Through several iterations the encoder and decoder are jointly optimized, allowing information about graph to be compressed into the low-dimensional embedding space.

Because information can be lost in the embedding process, the output vectors are statistical in nature and, as such, should be interpreted as graph analytics (not graph analysis). Graph analytics can extract unseen or difficult to obtain properties of the graph, either directly or by feeding the learned vector representations to a downstream inference pipeline, such as node classification, node clustering, and link prediction.

2.4 Graph Learning

2.4.1 Machine Learning on Graphs

In recent years, many embedding approaches for learning graphs have been developed. Goyal and Ferrara [27] have organized graph embedding methods into three broad categories: factorization based, random walk based, and deep learning based. Both the factorization and random walk based approaches train embedding vectors for each node independently, which results in several limitations:

- There is no sharing of trainable parameters between the nodes, which leads to computational inefficiency since the number of trainable parameters grows linearly with the number of nodes in the graph.
- These approaches are not generalizable as they are only able to generate vectors for the nodes that were present during the training phase and not for any unseen nodes.
- These approaches lack an ability to incorporate node attributes during vector generation, when node attributes can be highly informative about the node's position and role in the graph.

In an effort to mitigate these concerns, recent research efforts have led to the emergence of deep learning based methods.

2.4.2 State-of-the-Art Graph Learning Methods

Deep learning has been considered one of the most successful artificial intelligence techniques [39, 42]. Consequently, techniques that use deep learning to learn graphs have attracted lots of attention since 2015, and are widely studied and used in various fields [59]. Deep learning based approaches use a deep neural network architecture, generally referred to as Graph Neural Networks (GNNs), to generate embeddings vectors which account for both the structure and the attributes of the graph.

Wu et al. identified many types of deep learning based approaches to learning graphs, among which are graph autoencoders (GAEs) [58]. GAEs are deep neural network architectures that are trained with the objective of reconstructing their original graph input. Figure 2.10 shows a high-level GAE architecture. First, an encoder takes a graph as its input and systematically compresses it into a low-dimensional vector. The decoder then takes the vector representation and attempts to generate a reconstruction of some user-defined graph analysis tool (e.g., the adjacency matrix) of the original graph input. Encoder-decoder pairs are designed to minimize the loss of information between the input graph and the output (i.e., reconstructed) graph. These frameworks may be deterministic or probabilistic [28].

Early GAE algorithms such as Deep Neural Network for Graph Representations (DNGR) [12] and Structural Deep Network Embedding (SDNE) [56] only consider node

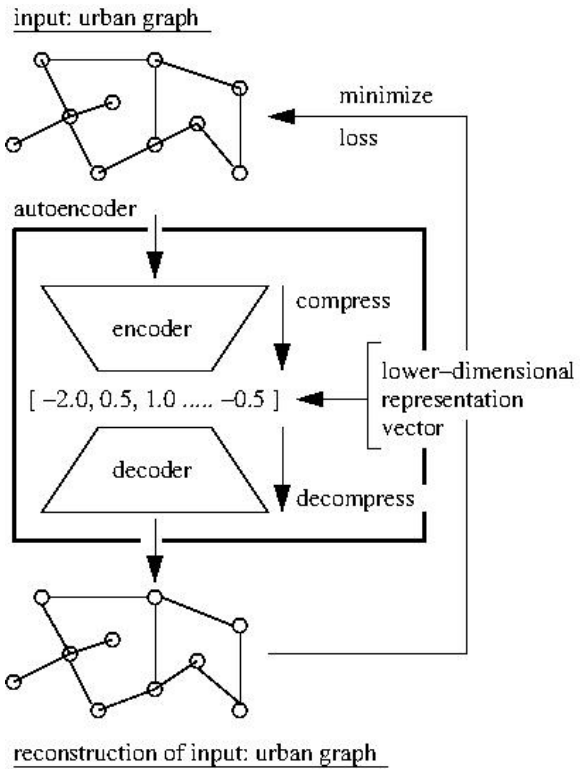


Figure 2.10: Traditional encoder-decoder approach.

structural information (i.e., connectivity between pairs of nodes), and ignore node attribute information. In 2016 Kipf et al. introduced Graph Autoencoder (GAE*) [33], an algorithm that leverages a ConvGNN [34] to encode node structural information and node feature information at the same time.

Kipf's GAE* encoder consists of two graph convolutional layers and a simple inner product decoder, which aims to decode node relational information from generated embeddings by reconstructing the graph adjacency matrix. The first convolutional layer takes as input the graph's node feature matrix X and the symmetrically normalized adjacency matrix $\tilde{A} = D^{-1/2}AD^{-1/2}$, where A is the adjacency matrix with added self connections and D is the diagonal node matrix of A. The first convolutional layer generates

a lower-dimensional feature matrix defined as:

$$\bar{X} = ReLu(\tilde{A}XW^0)$$

where W^0 is a trainable parameter matrix. The second convolutional layer takes as input the output of the previous layer and generates the node embeddings:

$$Z = \tilde{A}ReLu(\tilde{A}XW^0)W^1$$

where W^1 is also a trainable parameter matrix. The purpose of the decoder is to reconstruct the adjacency matrix A (with added self-connections) from Z. By applying the inner product on the latent variable Z and Z^T, the algorithm learns the similarity of each node inside Z. By applying the sigmoid function $\sigma(\cdot)$ the algorithm computes the probability of edges existing between the range of 0 and 1. Therefore, the reconstructed adjacency matrix is defined as:

$$A' = \sigma(ZZ^T)$$

In order to arrive at the optimal embedding matrix Z, the W^0 and W^1 parameters are systematically updated through an Adam optimization [32] of the weighted cross-entropy loss between the adjacency matrix A and the soft reconstruction A'.

Although Kipf et al. used a neural network with 2 hidden layers, 32 neurons in the first hidden layer, and 16 neurons in the second hidden layer when presenting the GAE* architecture [34], it is possible to modify these characteristics of the framework in order to adapt to specific needs.

2.5 Current Graph Learning Limitations

Modifying a GNN's architecture, in accordance with application specific needs, requires understanding the limitations of the GNN in generating a representation of the network at hand. For urban applications specifically, one important question is how to design architectures that are capable of learning large urban graphs. Our previous work [18] experimented with the reconstruction of three water distribution networks of varying sizes. We explored different GAE* architectures and their effect on reconstruction accuracy. We were interested in investigating whether modifying the number of neurons (i.e. embedding dimensions) in the GAE* algorithm could bring benefits, such as faster convergence or lower loss. Results are summarized in Figure 2.11.

In order to measure the accuracy of the reconstruction, the graph edit distance (GED) between the input graph and the reconstructed graph was used. GED is a graph similarity measure defined as the minimum cost of a sequence of node and edge edit operations transforming the reconstructed graph into the input graph. A GED value of zero shows that the reconstructed graph is isomorphic to the input graph. The higher the GED value, the higher the dissimilarity between the reconstructed graph and the input graph. In order to maintain a sense of GED importance for different graph sizes, the GED values obtained in our experiments were then normalized by the size of the input graph.

Results showed that if the neural network used to train the GAE* has an insufficient number of neurons (i.e. embedding dimensions), then the reconstruction performance is poor, which makes perfect sense. But when the same network is modeled with too many

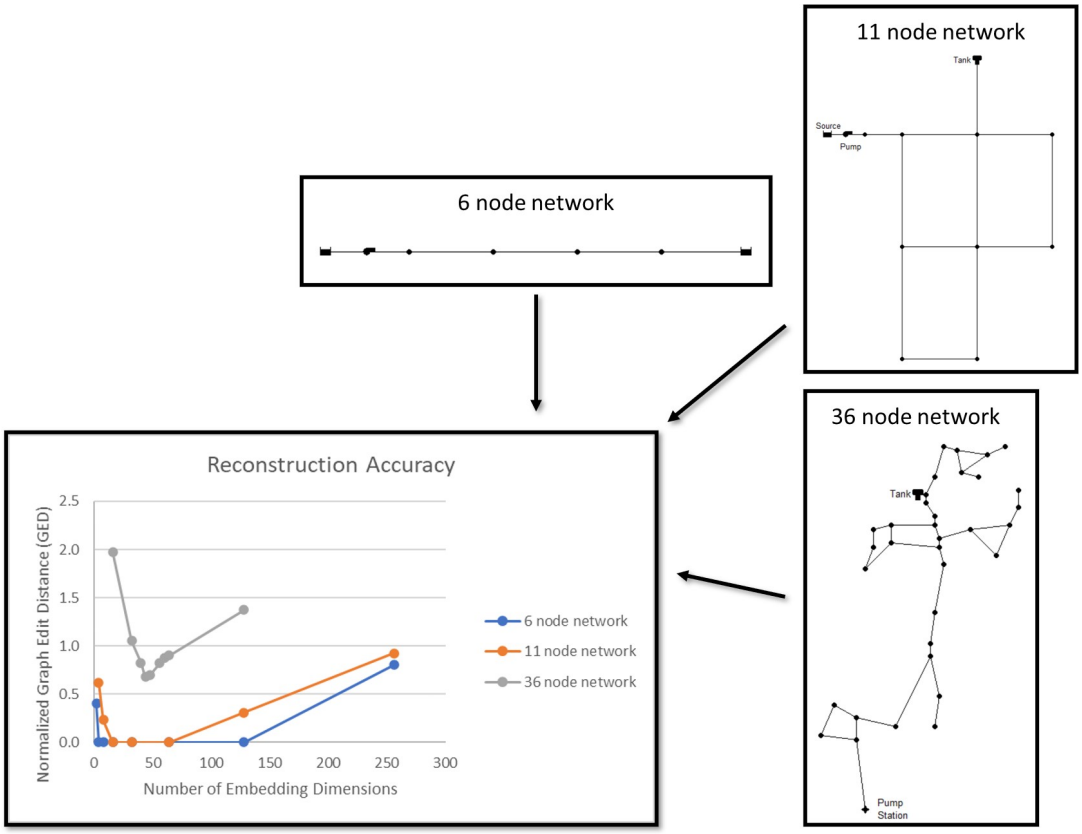


Figure 2.11: GAE* reconstruction accuracy for networks of varying sizes.

neurons, the reconstruction performance is also poor, suggesting that there is a window of convergence where good reconstruction can be achieved. To obtain a good reconstruction of the input graph, the underlying algorithm needs to work. In turn, the latter depends on:

1. Convergence of the Adam optimization
2. The encoder architecture design
3. The decoder architecture design

We suspect that the phenomena observed in Figure 2.11 may be related to limitations in these three GAE* components [7]. However, we recognize that the encoder architecture is a fundamental piece. Without an accurate encoder architecture at the start of the graph embedding process, no optimization or decodification will work further along the process.

Therefore, there is a need to better understand the role of neurons and layers in the learning. Despite their practical success, deep understanding and theoretical analysis of GNNs like GAE* is still largely lacking [19]. According to Zhou et. al GNNs are black-boxes that lack interpretability. An important research direction is to apply GNN models on real-world applications with trusted explanations [63]. Hence, in this work we take a step back and investigate interpretability of neural networks when applied to graph-structured data.

Chapter 3: **Learning Graph Topology**

This chapter will discuss how neural networks can learn a function that predicts graph topology. The first section will give an overview on neural networks and their architecture design. Later in this chapter, neural networks are applied to different graph structures, including a water distribution network and a metrorail network.

3.1 Neural Networks Design

Neural networks are a subset of ML, whose architecture is inspired by the neurons in the brain. A brain neuron receives an input and based on that input it generates an output that is used by another neuron. Similarly, neural networks are based on a collection of connected nodes called artificial neurons. Each connection, can transmit a signal to other neurons. Connections typically have a weight that adjusts as learning proceeds, increasing or decreasing the strength of the signal.

Neural networks can be created from at least three layers of neurons: the input layer, the hidden layer(s) and the output layer. The hidden layer (or layers) consist of many neurons, with connections between the layers. As the neural network learns the data, the weights of the connections between these neurons are fine-tuned, allowing the network to

reach a function that makes accurate predictions.

A first step towards achieving our objective is to understand how neural networks predict graph topology. We can divide this task into two sub-tasks: (1) understanding the role of neurons in the learning, and (2) understanding the role of layers in the learning. It was shown by Lippmann in his work "An Introduction to Computing with Neural Nets" presented in 1987 [41] that:

- Neural networks with no hidden layers are capable of solving only linearly separable problems, correctly classifying data sets where the classes can be separated by one decision plane.
- Neural networks with one hidden layer and two hidden layers can approximate any desired bounded continuous function. The neurons in the first hidden layer generate decision planes to divide the input space. Neurons in the second hidden layer form regions as intersections of these decision planes.
- Output neurons form unisons of the regions.

Lippmann's findings are summarized in Figure 3.1 and will serve as a basis for our investigations.

3.2 The XOR Problem

The topology prediction problem can be seen as a binary classification problem, we expect the system to output 1 for existing links or 0 for non-existing links. A famous binary classification problem in the domain of AI is the classical XOR problem. It has

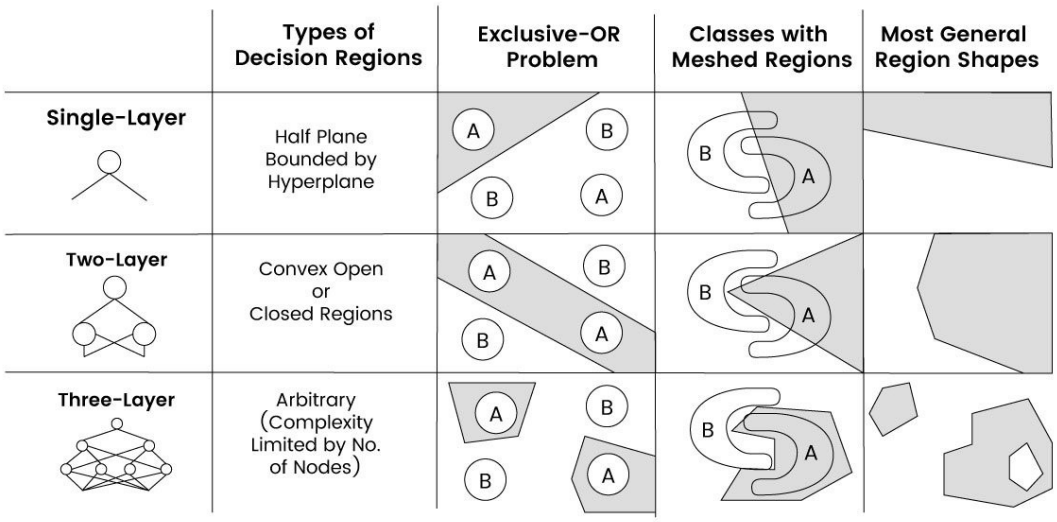
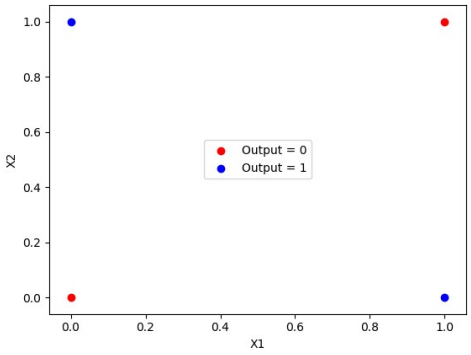


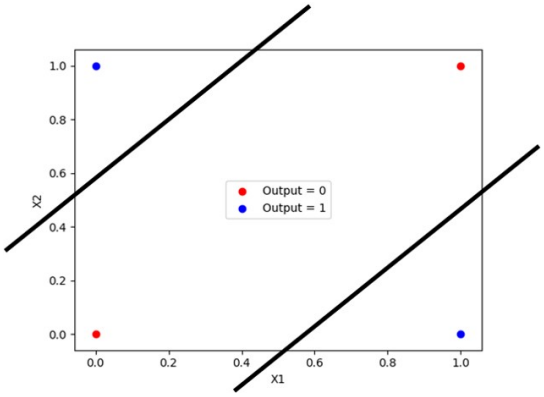
Figure 3.1: Types of decision regions that can be formed by single and multi-layer neural networks (Source: Lippmann [41]).

two input values (0 or 1), and outputs a value of 0 or 1 depending on the combination of the inputs. It can be depicted graphically as show in Figure 3.2a.

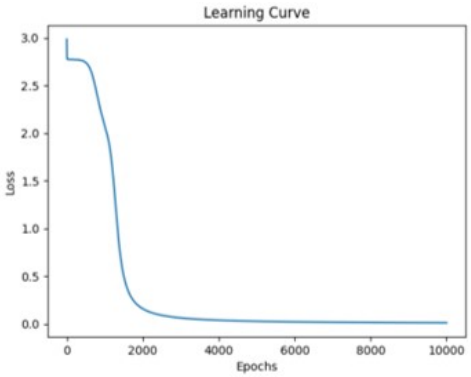
In order to solve the XOR problem, a neural network needs to produce two different decision planes to separate the input data based on the output classification (see Figure 3.2b). Therefore, the XOR problem can be solved with one hidden layer, two neurons (since two decision planes are needed), and one output layer with one neuron. It is important to note the need to apply a sigmoidal activation function to the outputs of the hidden layers to solve this problem. This function allows us to obtain outputs that makes more sense with regards to classification by fitting them within a range of 0 to 1, which can be interpreted directly as a probability of a link existing or not.



(a) The XOR problem.



(b) XOR problem solution.



(c) Learning curve.

Figure 3.2: Linearly separable problem solved with 1 hidden layer and 2 neurons.

3.3 Convergence

Learning involves adjusting the weights of the neural network to improve the accuracy of the result (i.e. minimizing the observed errors). Practically this is done by defining a loss function that is evaluated periodically during learning. As long as its output continues to decline through the iterations (i.e. epochs), learning continues. Figure 3.2c shows the learning process for the XOR problem. The loss function is evaluated at each iteration, and converges to nearly zero after 10,000 iterations. Even after learning, the loss function typically does not reach 0, as the actual outputs are real values (i.e. 0 or 1) and the outputs predicted by the model are probability values. If after learning, the loss value is too high, the neural network typically must be redesigned.

Most learning models can be viewed as an application of optimization theory. In this work we use a Gradient Descent optimization approach to find the optimal weight values that minimize the loss function. Gradient Descent optimization finds the direction to move the weight values in order to get a lower loss on the next iteration(i.e. gradient). Knowing which direction is downhill, allows the model to update the weight values until reaching a global minimum.

The learning rate defines the magnitude of weight updates in order to reduce the loss value in each iteration. If the learning rate is set too low, then learning will progress very slowly as we are making very small updates to the weights values. However, if the learning rate is set too high, then it can cause undesirable divergent behavior in the loss function (i.e., the gradient of the weight oscillates back and forth, and it is difficult to make

the loss reach the global minimum).

3.4 Measuring Performance

In order to understand how well the model predicted the topology, it is important to define an evaluation metric. Given a model output, the evaluation metric will measure how good or bad is the prediction. Later this information can be used for optimizing the model. In our case, the model outputs a probability between 0 and 1 of a link existing, therefore it is considered a binary classification model. A commonly used evaluation metric for binary classification models is the binary cross-entropy loss. Binary cross-entropy compares each of the predicted probabilities to the actual output (i.e. either 0 or 1). It then calculates the score that penalizes the probabilities based on the distance from the actual value. Mathematically, the loss function can be defined as:

$$L = -\frac{1}{M}\sum_{i=1}^{M} -y_i \log\left(p(y_i)\right) + (1 - y_i) \log\left(1 - p(y_i)\right)$$

where M is the number of classes (i.e. 2 for binary classification problems), y_i is the actual class value (i.e. 0 or 1 for binary classification problems), and $p(y_i)$is the predicted class value. Figure 3.2c shows the binary cross-entropy loss as a function of the learning iterations.

3.5 Urban Topologies

With a better understanding of the role of neurons and layers in the neural network architecture, we can design neural network architectures that predict system topologies common to urban settings (see Figure 3.3).

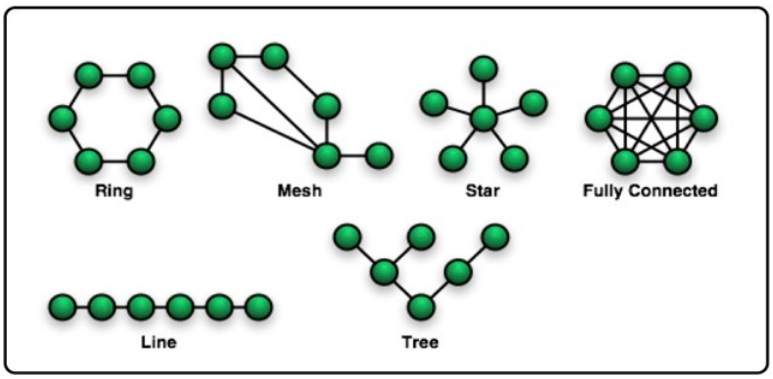
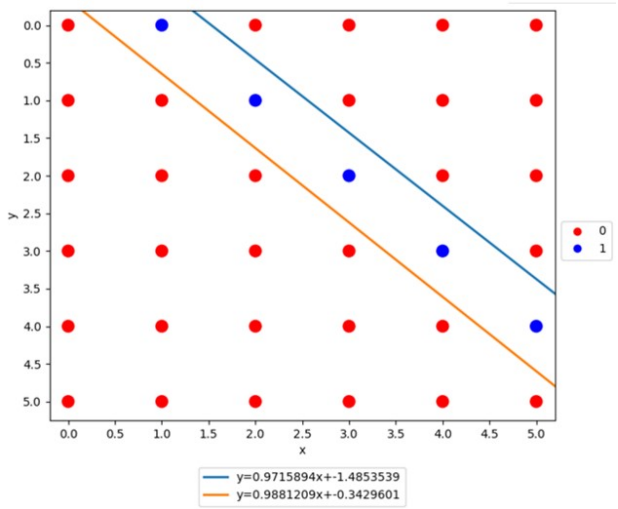


Figure 3.3: Common system topologies.

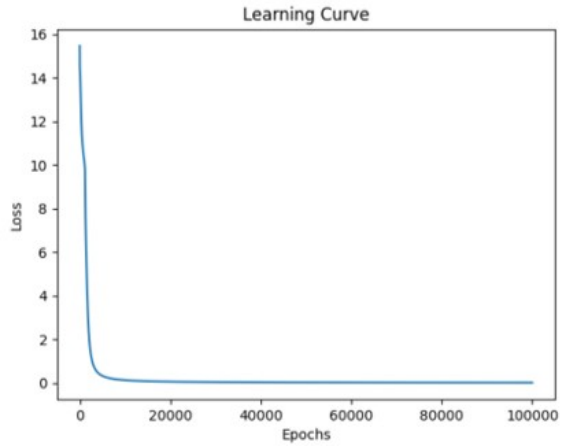
Learning a urban network topology can be framed similarly to the XOR problem. Imagine a directed line network with six nodes, potentially modeling a bus route, an electric transmission line, or a natural gas pipeline. Its topology can be represented by an adjacency matrix:

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

The adjacency matrix can be graphically represented as shown in Figure 3.4a. The two decision planes were derived through the same neural network architecture as the XOR problem, one hidden layer with two neurons. Hence, the number of neurons in a



(a) Decision boundaries.



(b) Learning curve.

Figure 3.4: Six node directed line topology problem.

neural network hidden layer is determined by the number of decision planes required to separate the classes.

Note that in the examples discussed so far, the class being isolated forms one region. If a same class forms more than one region, more hidden layers will be required to solve the problem. For instance, imagine now a undirected line with six nodes, potentially modeling an interstate highway, internet cable connection, or an airport taxiway. Its topology can be represented by an adjacency matrix:

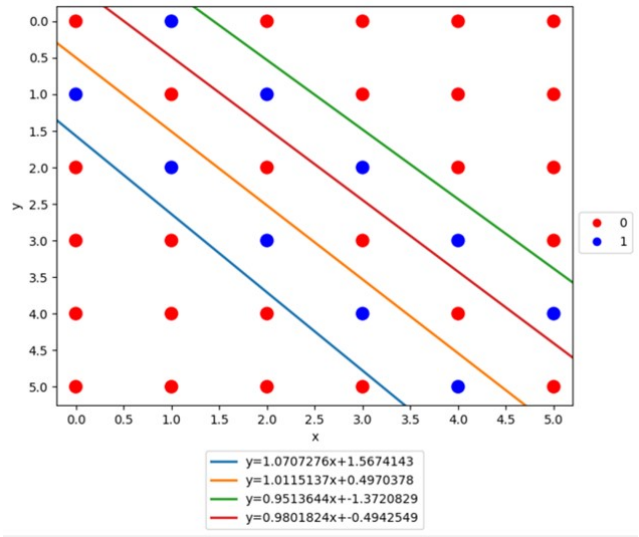
$$A = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

The adjacency matrix can be graphically represented as shown in Figure 3.5a. In this case, because there are two regions of the class of ones, we will need two hidden layers to solve the problem. The first hidden layer will contain four neurons (one for each decision plane), and the second layer will contain two neurons (one for each region).

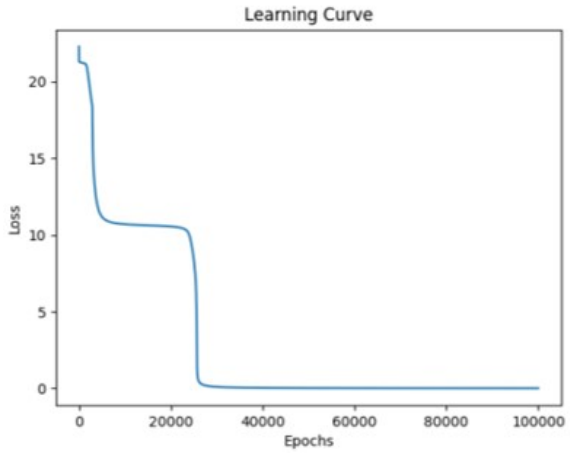
Another interesting case study is the mesh topology. The mesh topology can be represented by an adjacency matrix:

$$A = \begin{bmatrix} 0 & 1 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

By looking at the adjacency matrix of the mesh topology, we can see it is visually



(a) Decision boundaries.



(b) Learning curve.

Figure 3.5: Six node undirected line topology problem.

hard to determine the required neural network architecture. However, there may exists a better ordering of vertices that could still represent the same topology (i.e. an isomorphic graph) and group similar elements close together to form regions (or clusters) that can be visually identified. Such technique is called matrix reordering and will be investigated in the next section. Classification results for the remaining topologies shown in Figure 3.3 can be found in Appendix A.

3.6 Matrix Reordering

The problem of finding a good ordering for the vertices of an adjacency matrix is known as matrix reordering. In a survey of "Matrix Reordering Methods for Tables and Network Visualization" Behrisch et al. described available reordering algorithms and analyzed their differences [5]. According to this survey, when the objective is to reorder a matrix so that similar rows (and respective columns) are arranged close and dissimilar rows (and respective columns) are placed farther apart, Robinsonian methods are the most adequate. Within Robinsonian methods, hierarchical clustering showed visually promising results when the possibility for clustering existed. Applying the hierarchical clustering to the mesh topology we obtain the reordering shown in Figure 3.6.

As it can be seen in the reordered topology, the number of clusters is visually easy to spot (i.e. 2 clusters of ones). Hence the number of decision planes to learn this reordered topology will be two. It can be solved with a neural network containing one hidden layer with two neurons, and an output layer. The classification results are shown in Figure 3.8a.

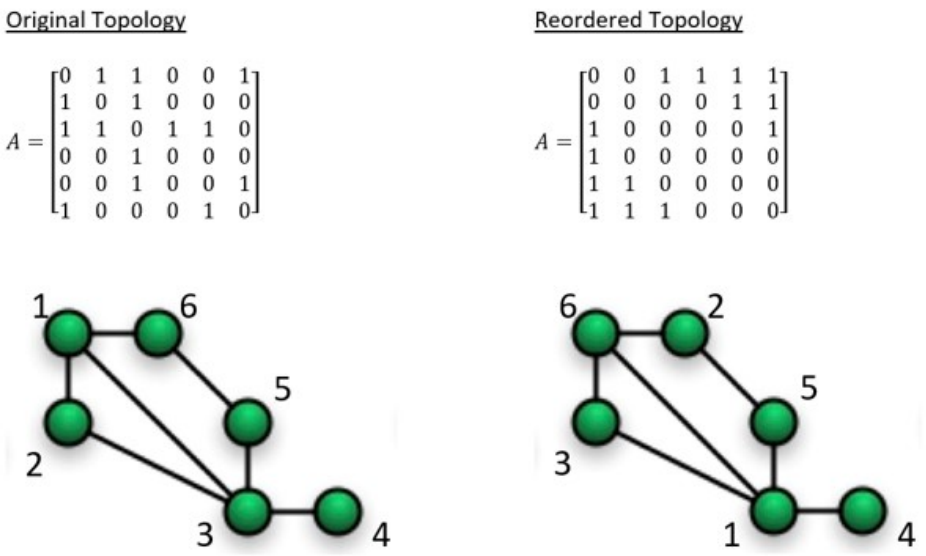


Figure 3.6: Matrix reordering for mesh topology using hierarchical clustering.

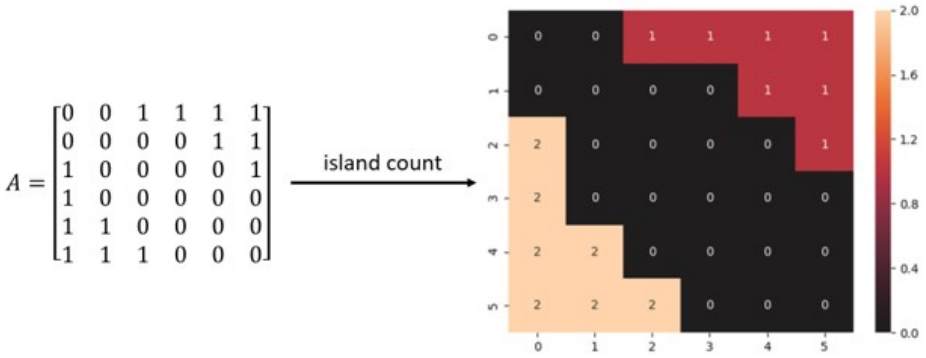
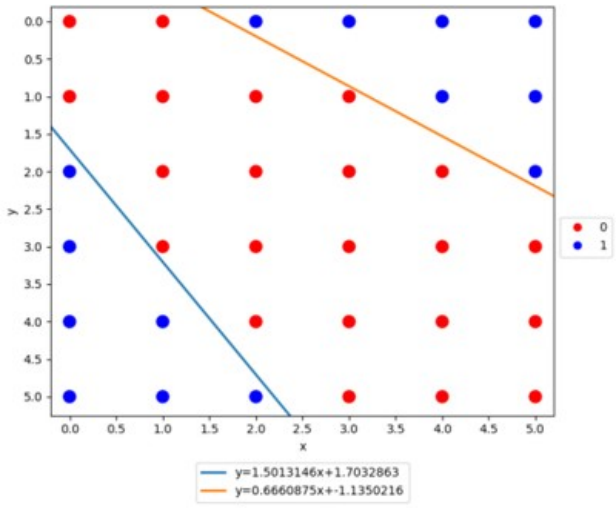
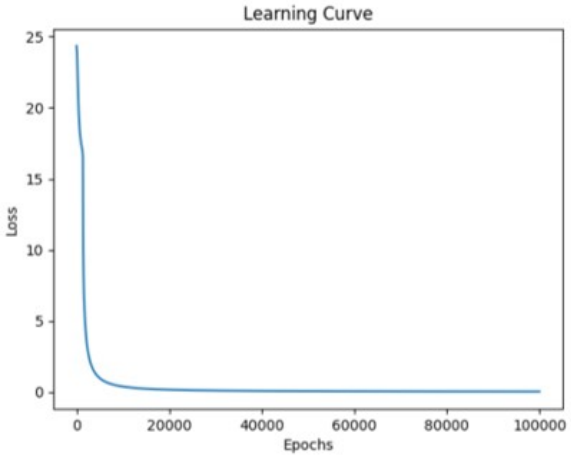


Figure 3.7: Automatic region detection for the mesh topology.

In this case, the number of clusters present in the reordered topology was easy to visually identify. However, the approach is not scalable. Knowing the number of clusters present in the reordered adjacency is important because it will determine the number of decision planes required to classify the data, and hence determine the number of neurons/layers required to build a good neural network.



(a) Decision boundaries for mesh topology.



(b) Learning curve.

Figure 3.8: Mesh topology problem.

3.7 Automating Neural Network Architecture Design

Automating the count of regions is similar to the "Number of Islands Problem". Given a boolean 2D matrix, a group of connected 1s forms an island. The idea is to start a Depth-first search (DFS) from each unprocessed cell and increment the island count. A cell in the matrix can be connected to 8 neighbours, therefore the problem can be solved by recursively calling for the 8 neighbours and keeping track of the visited 1s so that they are not visited again. The number of DFS traversals gives the number of islands. Figure 3.7 shows the number of islands found for the reordered mesh adjacency matrix.

Figure 3.9: Water distribution network topology retrieved from EPANET's "Example Network 2".

3.8 Case Study 1: Learning a Water Distribution Network Topology

In order to test the scalability of our approach to larger systems, we will exercise it in a water distribution network composed by 36 nodes, whose topology was retrieved from an example network provided by the hydraulic simulation software EPANET [49], and is shown in Figure 3.9.

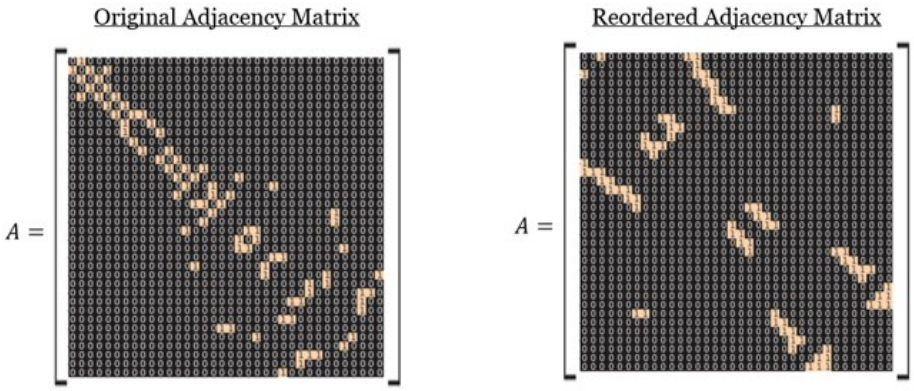


Figure 3.10: Matrix reordering for water network topology using traveling salesman.

As the original adjacency matrix in the left of Figure 3.10 shows, there may exists a better ordering of vertices that could lead to the formation of identifiable regions. After applying several matrix reordering algorithms suggested by Behrisch et al. [5], we found that the Traveling Salesman Algorithm was the best fit for this topology. The reordering obtained is shown in the right of Figure 3.10.

Figure 3.11 shows the 13 islands found for the reordered water network adjacency matrix using DFS. The number of planes needed to contain all regions is 74. Hence, two hidden layers will be needed to learn this reordered topology, with 74 neurons in the first hidden layer, and 13 neurons in the second hidden layer. The classification results are

shown in Figure 3.12a. As it can be seen in the learning curve, the learning does converge to nearly zero loss. However, in order to arrive to that loss value 2935.4 seconds of GPU time were required. When comparing this GPU time to the 82.1 seconds required for the mesh topology, it becomes evident that larger topologies will become computationally prohibitive.

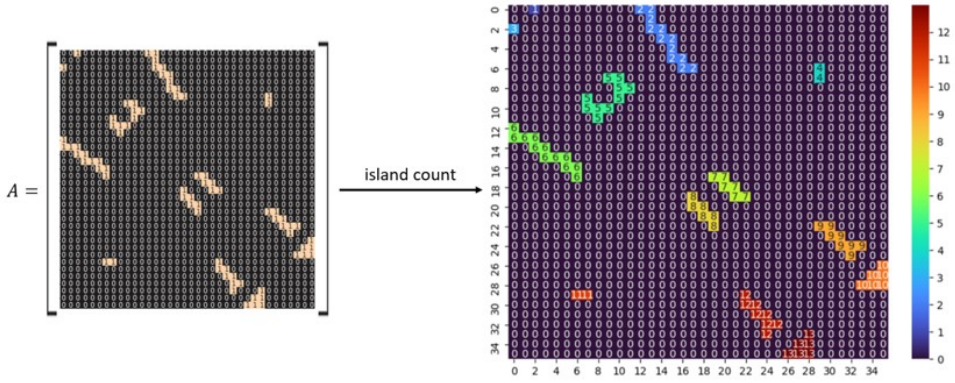
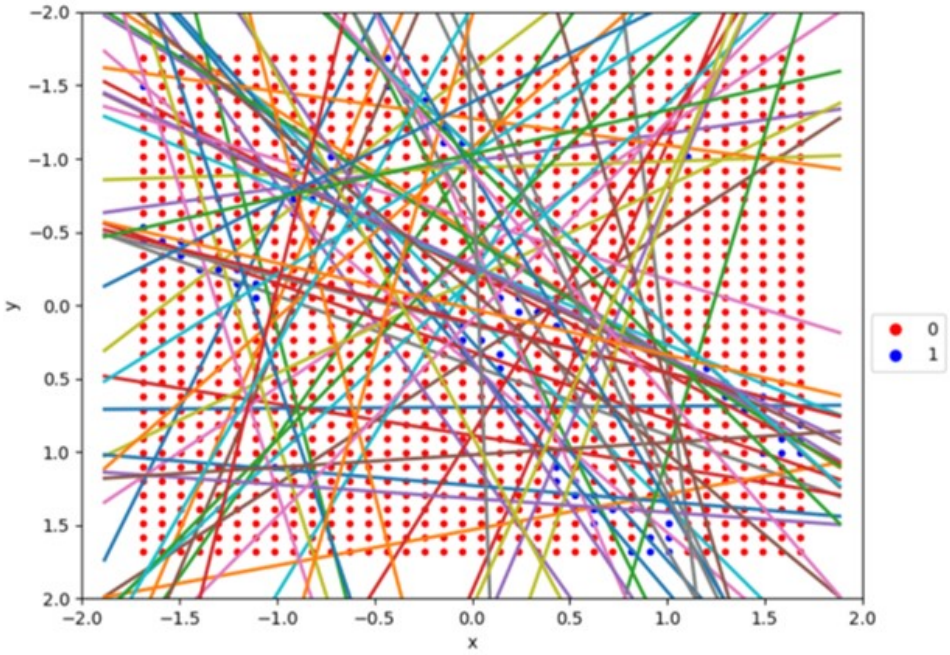


Figure 3.11: Automatic region detection for the water distribution topology.

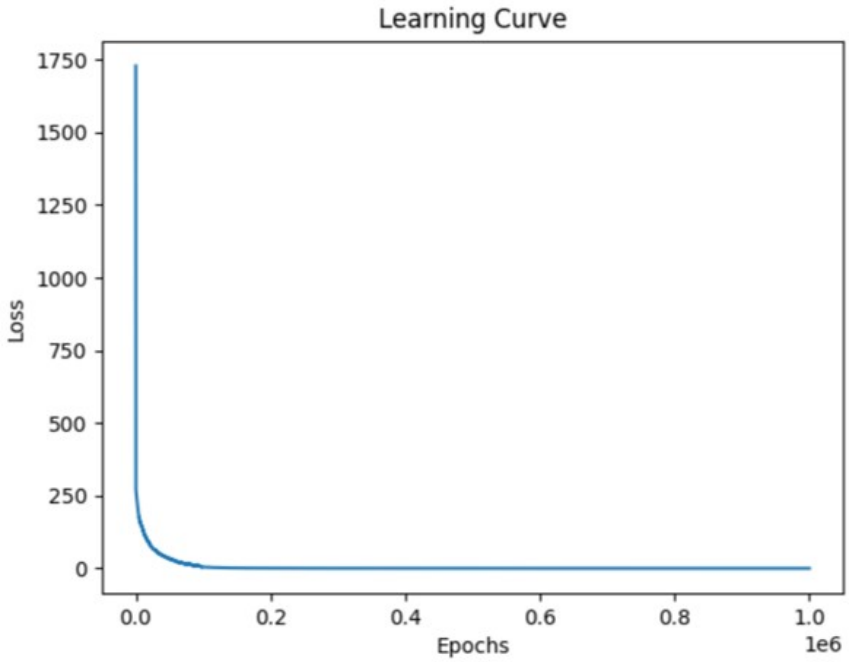
There is a clear need to approach this problem from a modular perspective, decomposing the network and learning the topology by parts. In addition to improving GPU processing times, a modular approach allows for easier adaptation to constant urban networks changes. New streets, pipes, fiber-optic cables are added/removed frequently as cities expand and adapt to its residents' needs. Once a change occurs, modeling can be facilitated if only part of the topology has to be "re-learned".

3.8.1 Water Distribution Network Decomposition

As mentioned in Chapter 2, network decomposition can be performed by modeling the network from a multi-layer perspective. An important step is to decide what



(a) Decision boundaries for water distribution topology.



(b) Learning curve.

Figure 3.12: Water distribution topology problem.

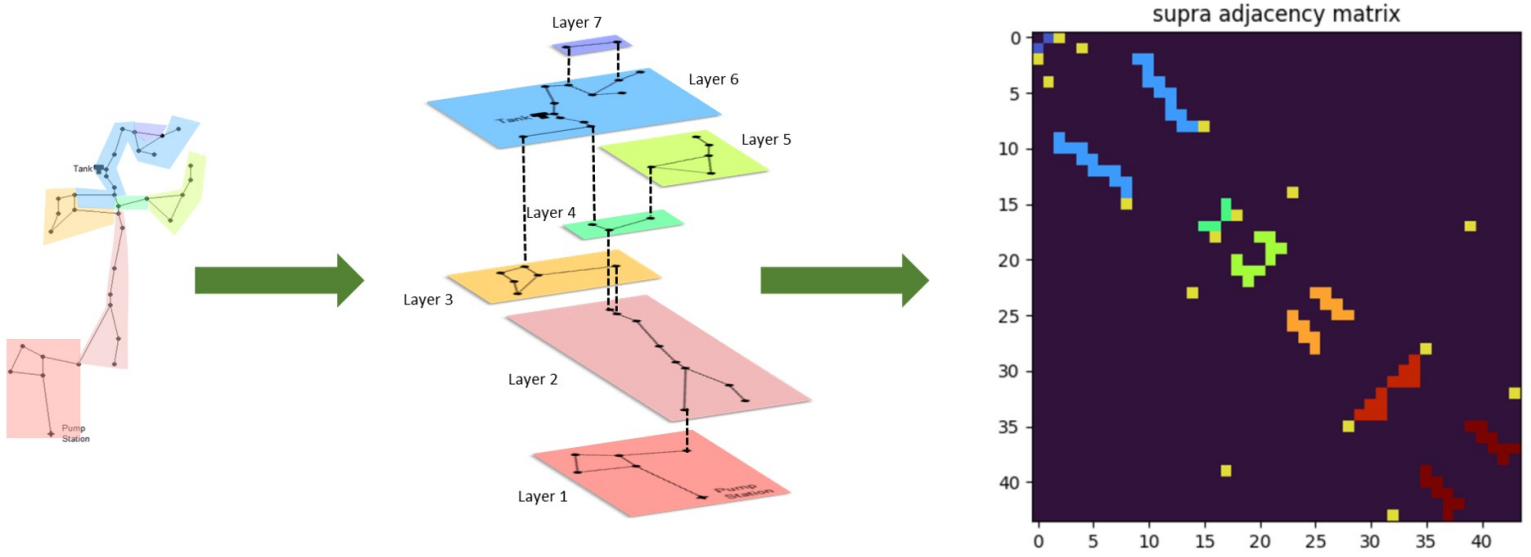


Figure 3.13: Water distribution network decomposition into a multi-layer network.

characteristic will define the layers. In the case of water distribution networks one possible defining characteristic is space. Water distribution networks are spatially distributed and usually divided into independently controlled subnetworks called district metered areas (DMAs). Because the topology used in this case study is a simulated topology retrieved from EPANET, it has no real defined spatial districts. Therefore, we will use the clustering obtained from the Traveling Salesman matrix reordering as spatial subdivisions of the network. Figure 3.13 shows the decomposition of the network into layers and its corresponding supra matrix. By learning each layer individually, the GPU time is reduced to 553.9 seconds (see Figure 3.14). Classification results for each layer can be found in Appendix B. Although the decomposition approach learns the layers individually by isolating intra-layer adjacency matrices, it is important to note that interconnections between layers can also be learned by isolating inter-layer adjacency matrices.

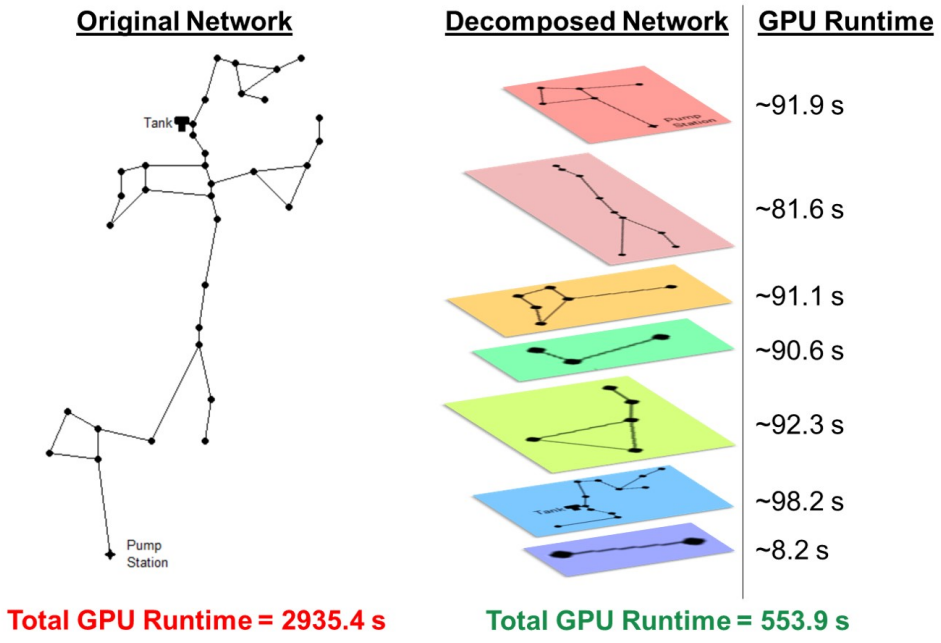


Figure 3.14: GPU runtime improvements when decomposing water distribution network into a multi-layer network.

3.9 Case Study 2: Learning a Metro Network Topology

Further expanding the scalability of our approach to even larger systems, we will exercise it in a real world network, the Washington DC metro system, composed by 91 stations and 6 lines. Its simplified topology is shown in Figure 3.15, where nodes represent metro stations and edges represent connections between stations.

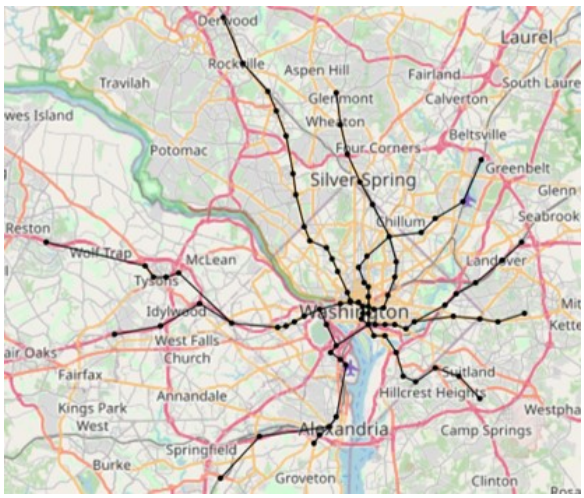


Figure 3.15: Metro network topology.

The Washington DC metro network is an interesting case study as it does not require matrix reordering. Because the system is composed of clearly defined lines, its adjacency matrix is inherently in a good format to visually determine the required neural network architecture. When matrix reordering is applied, there is no improvement of the ordering of vertices that could lead to the formation of identifiable regions (see Figure 3.16).

Figure 3.17 shows the 34 islands found for the original metro network adjacency matrix using DFS. The number of planes needed to contain all 34 regions is 98. Hence,

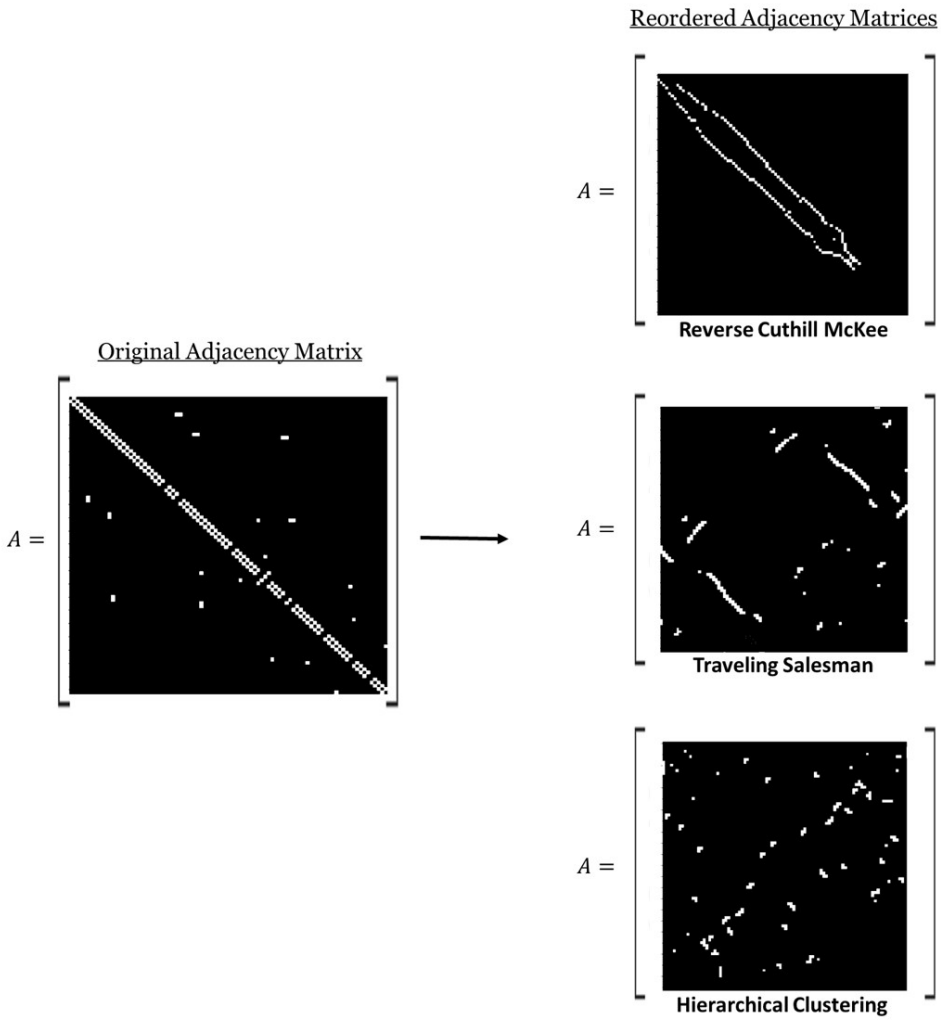


Figure 3.16: Matrix reordering for metro network topology using Reverse Cuthill McKee, Traveling Salesman, and Hierarchical Clustering.

two hidden layers will be needed to learn this reordered topology, with 98 neurons in the first hidden layer, and 34 neurons in the second hidden layer. The classification results are shown in Figure 3.18a. As it can be seen in the learning curve, the learning does converge to nearly zero loss. However, in order to arrive to that loss value 25582.1 seconds of GPU time were required. Just like in case study 1, network decomposition will be needed to achieve computational efficiency.

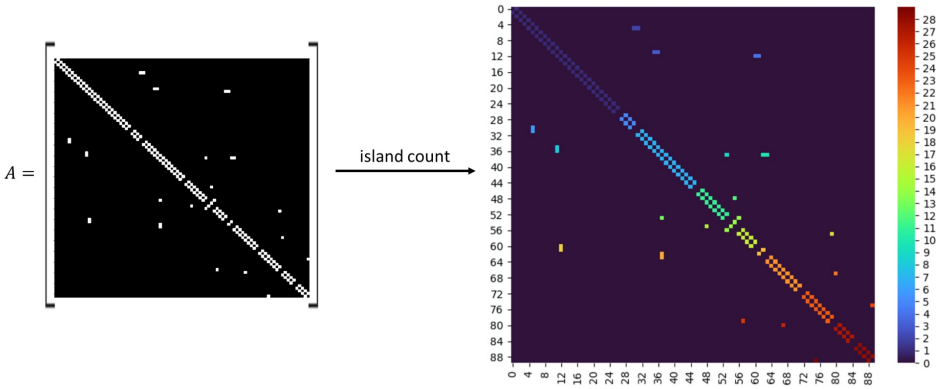
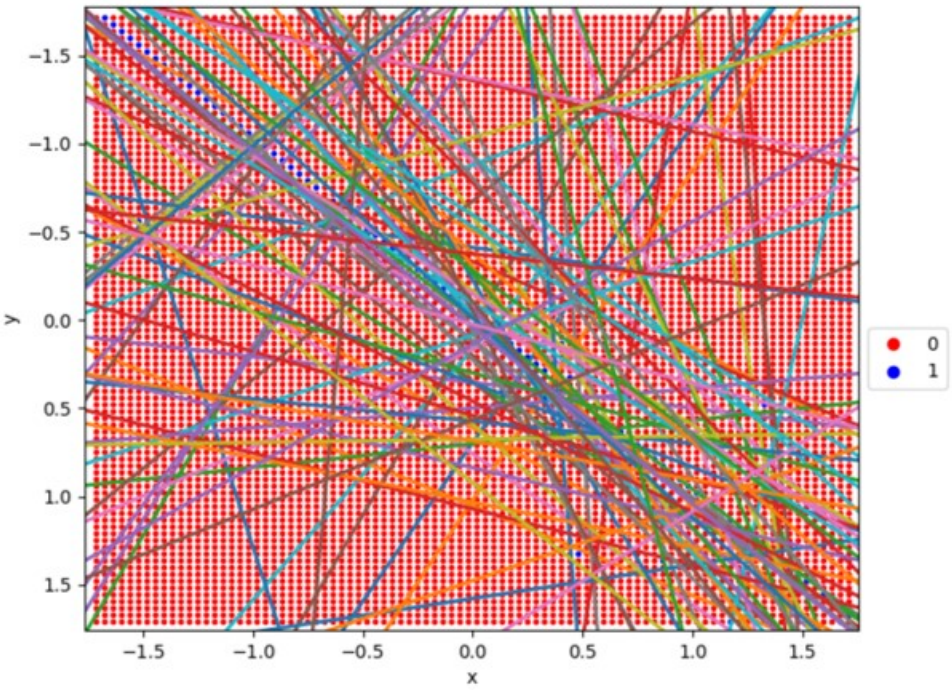


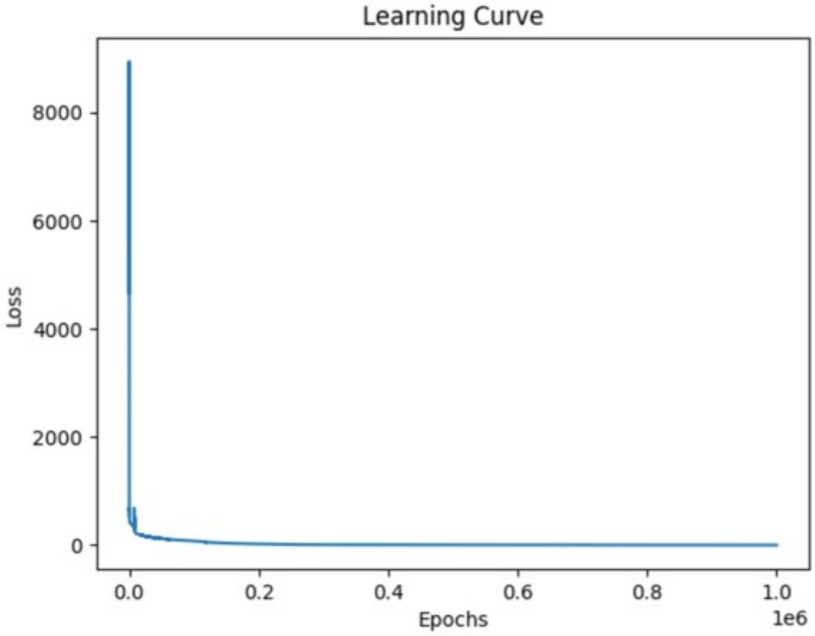
Figure 3.17: Automatic region detection for the metro topology.

3.9.1 Metro Network Decomposition

Similarly to case study 1, metro network decomposition can be performed by modeling the network from a multi-layer perspective. In the case of metro network one possible defining characteristic is the line colors. The Washington, DC metro network consist of multiple lines, each servicing a specific route. Some lines share track with each other for a portion of their route (e.g. blue and yellow lines) while others operate solely on their own right-of-way (e.g. red line). Figure 3.19 shows the decomposition of the network into layers and its corresponding supra matrix. By learning each layer individually, the GPU time is reduced to 617.55 seconds (see Figure 3.20). Classification results for each layer can be found in Appendix C.



(a) Decision boundaries for metro topology.



(b) Learning curve.

Figure 3.18: Metro topology problem.

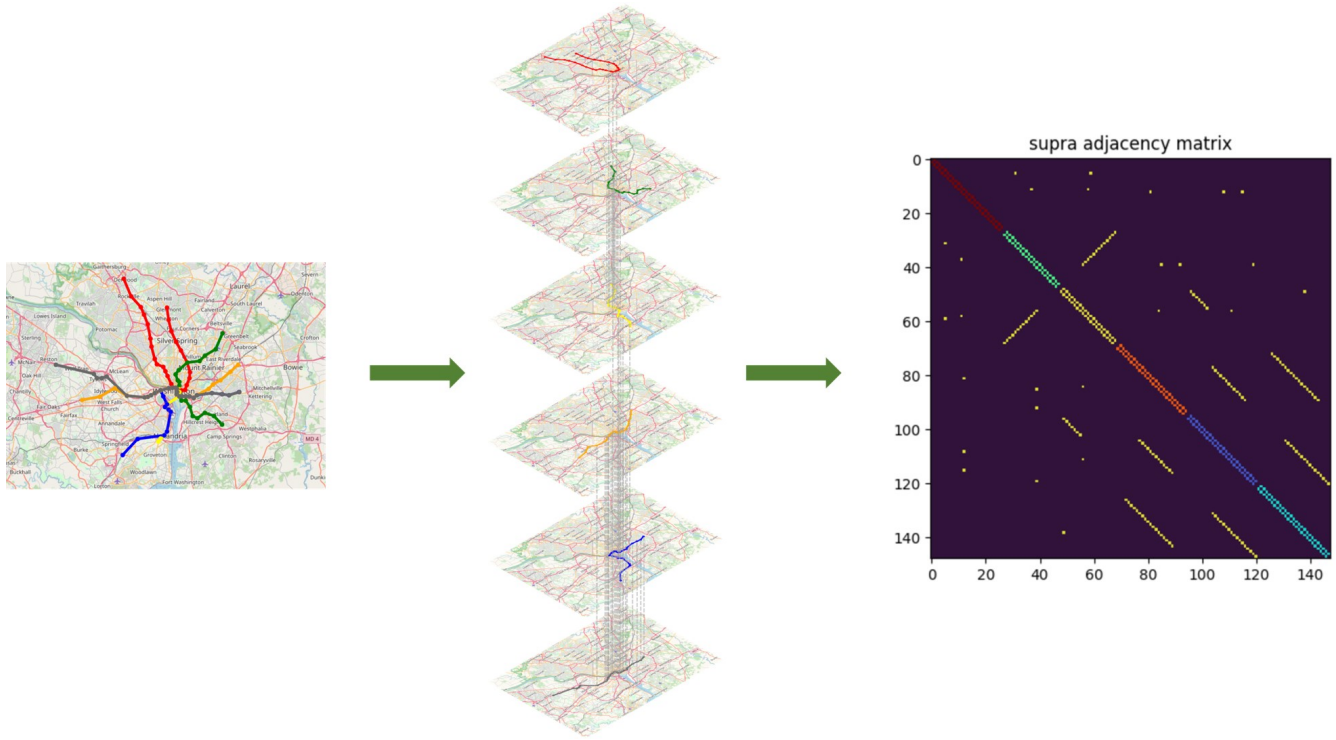


Figure 3.19: Metro network decomposition into a multi-layer network.

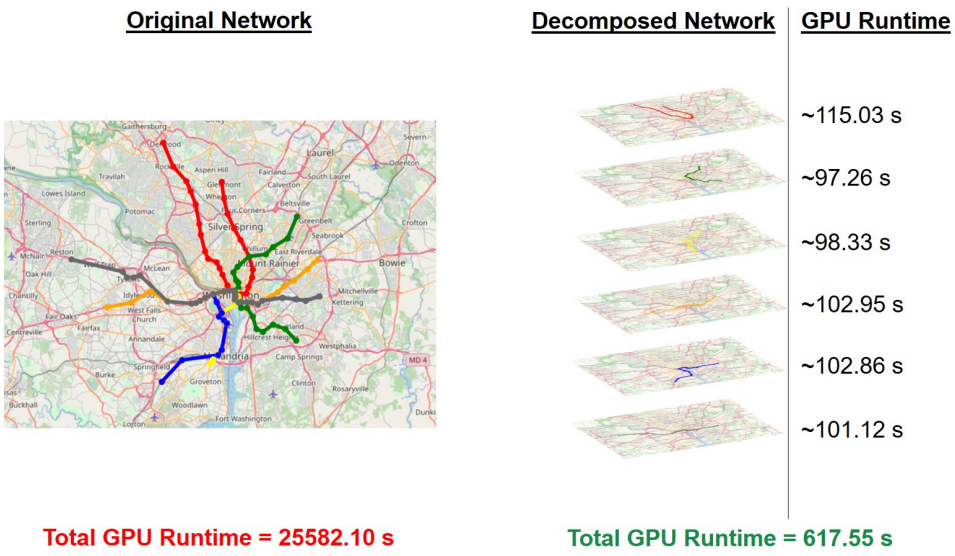


Figure 3.20: GPU runtime improvements when decomposing metro network into a multi-layer network.

Chapter 4: **Anomaly Detection**

This chapter will discuss how graph autoencoders can be used for Anomaly Detection (AD). The first section will give an overview on AD approaches. Later in this chapter, autoencoders are used to identify anomalies in graph structures, including a water distribution network and a metrorail network.

4.1 Anomaly Detection Approaches

AD is the task of finding data points or patterns in data that do not conform to a notion of normal behavior given previous observations. The capability to detect anomalies can provide highly useful insights across an urban environment. Large scale modern cities have a large number of constituent behaviors, and multiple dimensions of interdependency among physical, cyber and geographic systems [48]. These facts are what makes cities "system of systems". When part of a system fails, there exists a possibility that the failure will cascade across other correlative infrastructures, and sometimes even back to the originated source. Therefore, flagging unusual observations and enacting a planned response when they occur can save time, money and prevent cascading failures across systems.

Most AD approaches first model normal behavior, and then exploit this knowledge to identify anomalies [53]. The first step involves building a model of normal behavior using available data. Based on this model, an anomaly score is then assigned to each data point that represents a measure of deviation from normal behavior. These detection systems are either manually built by experts setting thresholds on data, or constructed automatically by learning from the available data through ML. Building an AD system by hand requires domain knowledge and foresight. In addition, systems created using this approach cannot adapt, or are costly and untimely to adapt. For an ecosystem where the data changes over time, like cities, manually created AD systems cannot be a good solution.

Machine learning, on the other hand, better suits the need to create an urban AD system that is adaptive, on time, and able to handle high-dimensional datasets. In recent years, a large number of deep learning approaches to AD have been introduced. In a review of deep learning methods for AD, Pang et. al offered several advantages for using such approaches [45]. For instance, these deep learning models are designed to work with multivariate and high dimensional data, eliminating challenges associated with individually modeling anomalies for each variable and aggregating the results. In addition, they offer the opportunity to model complex, nonlinear relationships within data, jointly modeling the interactions between multiple variables with respect to a given task.

Within deep learning approaches, one particularly interesting approach is the use of autoencoders for AD, as they eliminate the need for labeled abnormal data to train their learning models. Although there is a myriad of normal urban events happening everyday

and everywhere, the abnormal events rarely occur and are usually not recorded [61]. Hence, most of urban data are produced by normal events, and a very small part produced by abnormal events. In addition, with the surge in climate and weather related disasters in the last fifty years [1], there is an increasing concern in detecting historically unseen abnormal urban events. The extreme imbalance of dataset and nonexistence of historical datasets for catastrophic situations creates potential for an autoencoder implementation.

4.2 Anomaly Detection with Autoencoders

As introduced in Chapter 2, autoencoders are neural networks designed to learn a low-dimensional representation, given some input data. They consist of two components, encoder and decoder. The encoder learns to map input data to a low-dimensional representation, and the decoder learns to map this low-dimensional representation back to the original input data. The model is trained by minimizing the reconstruction error, which is the difference between the original input and the reconstructed output produced by the decoder.

An autoencoder-based AD follows the general principle of first modeling normal behavior and then generating an anomaly score for each new data sample. The autoencoder is first trained on normal data samples, to allow the model to learn a mapping function that successfully reconstructs normal data samples with a very small reconstruction error. At test time, the reconstruction error will be small for normal data samples, and large for abnormal data samples. To identify anomalies, the reconstruction error is be used

as an anomaly score, and samples with reconstruction errors above a given threshold are flagged.

The binary classifiers presented in Chapter 3 can be framed as autoencoders, as they learn a mapping function that successfully reconstructs input data samples with very small reconstruction error. See Figure 4.1 for a parallel comparison of the two mechanisms.

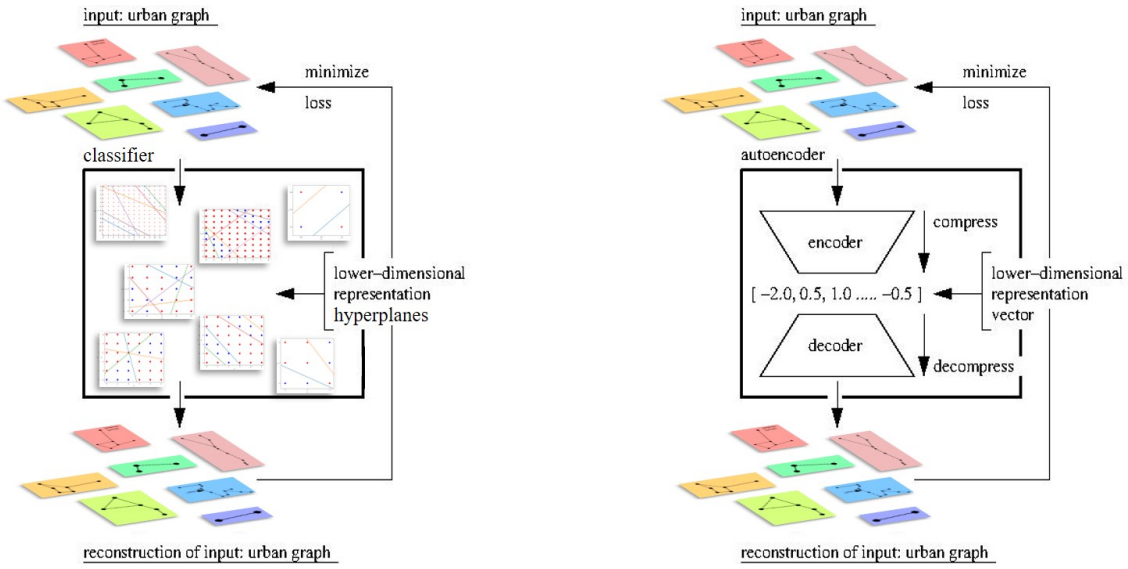


Figure 4.1: Parallel comparison between binary classifier (left) and autoencoder mechanisms(right).

4.3 Learning Reconstruction Error

As mentioned previously, manually setting thresholds on data is an approach that does not apply well to large and dynamic datasets. Therefore, flagging samples that exceed set reconstruction error threshholds is a task that can be more efficiently performed by ML. By reframing the problem as a classification task, machines can learn to accurately predict whether a given example is an anomaly or not. For the case studies presented later

in this section a random forest classifier is trained to output a value of "1" for abnormal cases and a value of "0" for normal cases. There are a number of good reasons to choose this pathway of analysis. First, the random forest classifier is composed of a number of decision trees. This problem solving strategy is well known to be accurate and robust. In addition, it does not suffer from the overfitting problem often encountered in other ML algorithms, since it takes the average of all the predictions of decision trees, and cancel out the biases. Random forests can also handle missing values, a problem encountered very often in urban data, by using median values to replace continuous variables, or computing the proximity-weighted average of missing values. Finally, random forest provides a measurement of relative feature importance, which helps in the selection of contributing features for the classifier [11].

4.4 Measuring Performance

In order to estimate the random forest model's ability to detect anomalies, this work will use a binary cross-entropy loss. As described in Section 4.4, the binary cross-entropy loss is a commonly used evaluation metric for binary classification models. In this case, the random forest model outputs a probability between 0 and 1 of a case being an instance of behavior anomaly, therefore it is considered a binary classification model. Recall that binary cross-entropy compares each of the predicted probabilities to the actual output (i.e. either 0 or 1). It then calculates the score that penalizes the probabilities based

on the distance from the actual value. The loss function is defined as:

$$L = -\frac{1}{M}\sum_{i=1}^{M} -y_i \log\left(p(y_i)\right) + (1-y_i)\log\left(1-p(y_i)\right)$$

where M is the number of classes (i.e. 2 for binary classification problems), y_i is the actual class value (i.e. 0 or 1 for binary classification problems), and $p(y_i)$is the predicted class value.

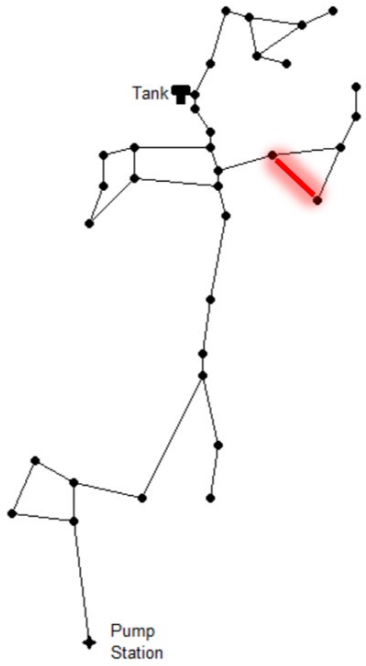


Figure 4.2: Case Study 3 failure edge.

4.5 Case Study 3: Identifying Anomalies in a Water Distribution Network Topology

In order to test our suggested approach to AD, we will exercise it in the water distribution network presented in Section 3.8. In this case study we will simulate a failure

in one of the edges of the network as shown in Figure 4.2, by removing the edge from the network representation (i.e. adjacency matrix).

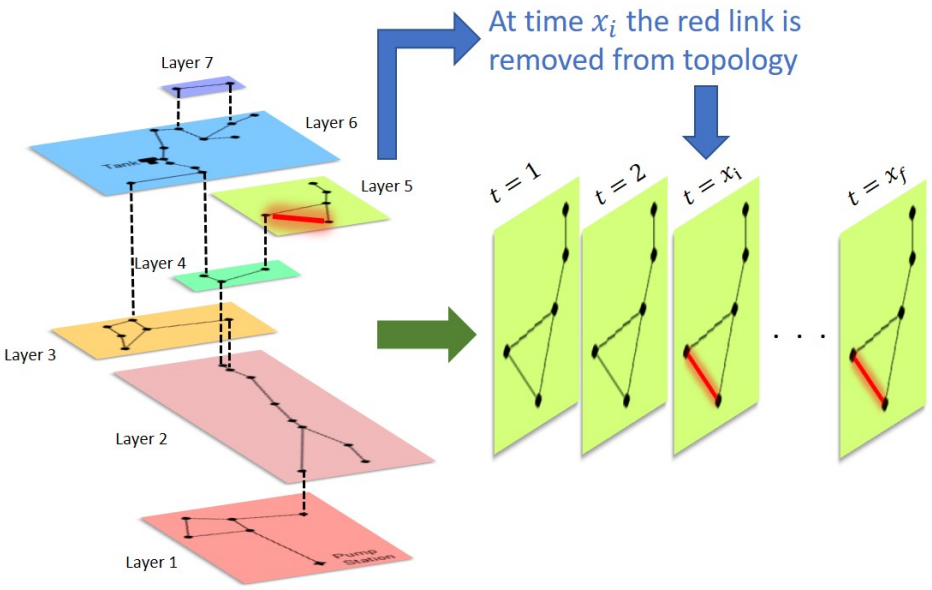


Figure 4.3: Case Study 3 simulation mechanics.

4.5.1 Simulation Mechanics

After training a binary classifier to learn normal graph topology using the machine architectures an learning strategies presented in Section 3.8, we construct an adjacency matrix of the subsystem where the failure occurs (i.e. layer 5 of the decomposed network) with the failure edge being removed. We then simulate failure by feeding normal or abnormal adjacency matrices, at different points of the simulation duration, to the binary classifier previously trained to reconstruct normal topology (See Figure 4.3). The model attempts to reconstruct the inputs, yielding small reconstruction errors when the input was the normal adjacency matrix, and large reconstruction errors when the input was the abnormal adjacency matrix. Figures 4.4 and 4.5 show the reconstruction errors for normal

and abnormal cases respectively.

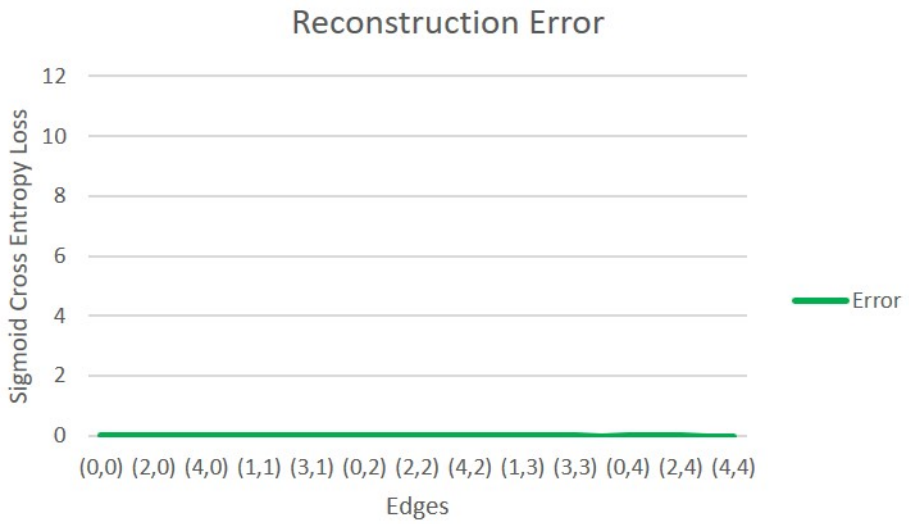


Figure 4.4: Case Study 3 reconstruction error for normal behavior.

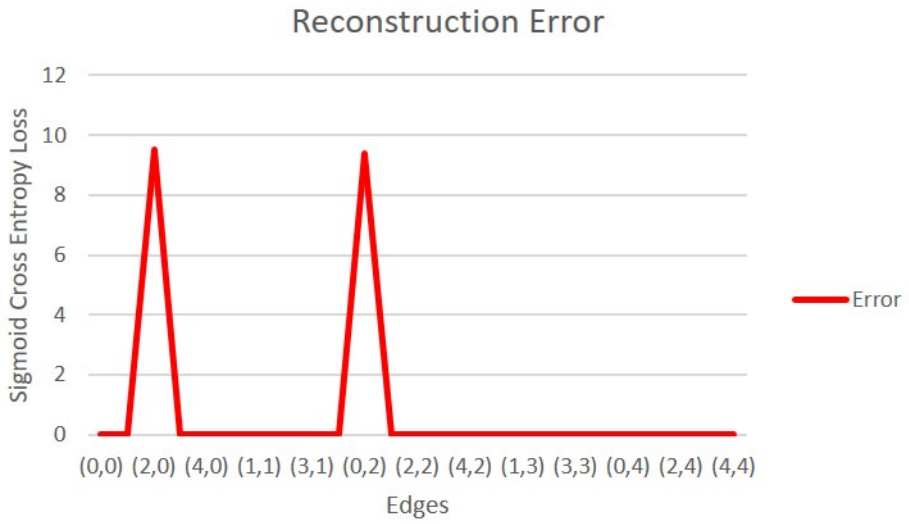


Figure 4.5: Case Study 3 reconstruction error for abnormal behavior.

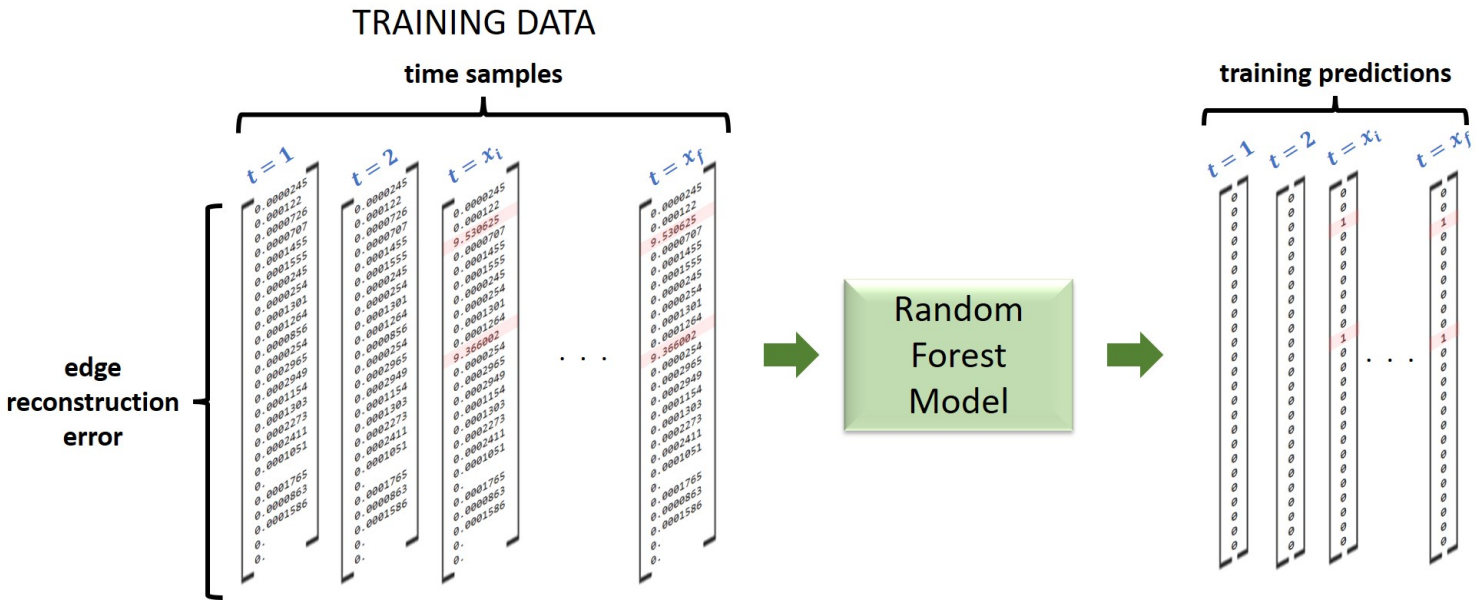


Figure 4.6: Case Study 3 AD classifier training.

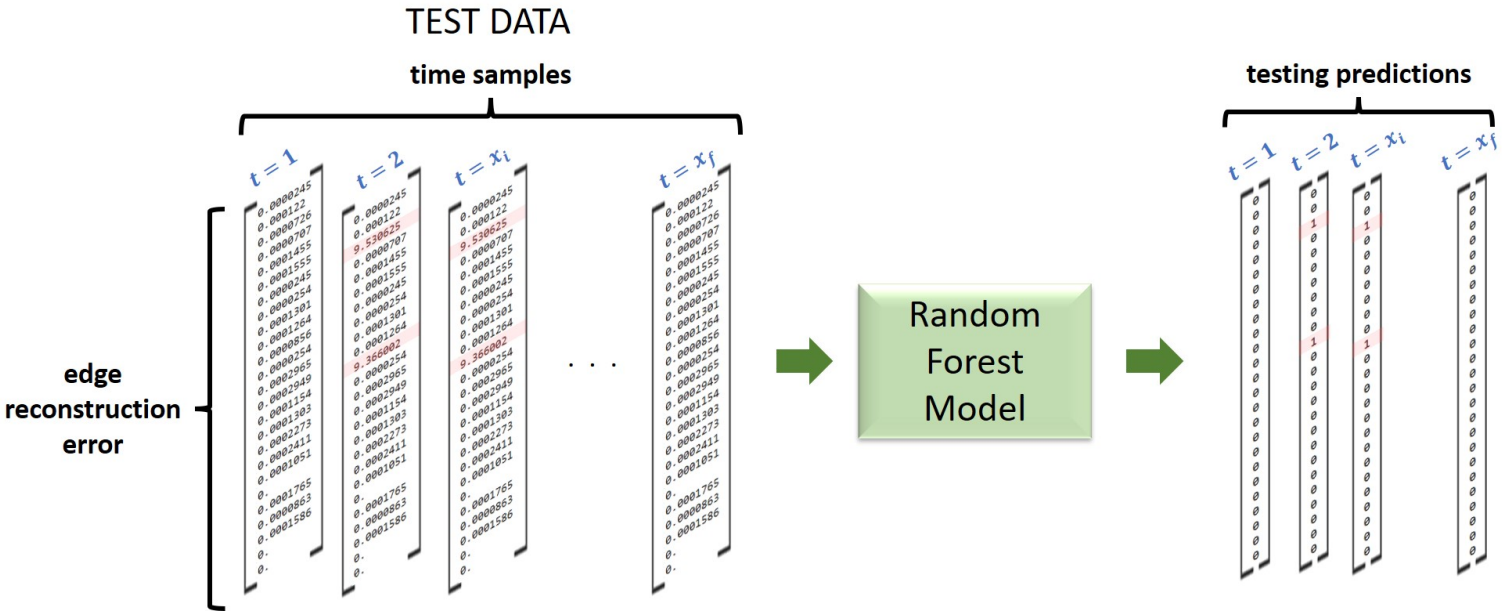


Figure 4.7: Case Study 3 AD classifier testing.

4.5.2 Identifying Abnormal Behavior

Using the reconstruction error values obtained for each edge at different times during the simulation as inputs, we train a random forest classifier as shown in Figure 4.6. Then, by changing the times at which failure occurs we test the learned random forest model as shown in Figure 4.7. The outputs of the model show a value of "1" at edge indexes and time stamps where a failure is predicted to have occurred, and "0" otherwise. Results show the trained model is capable of predicting with a very high accuracy the time and location where anomalies happened.

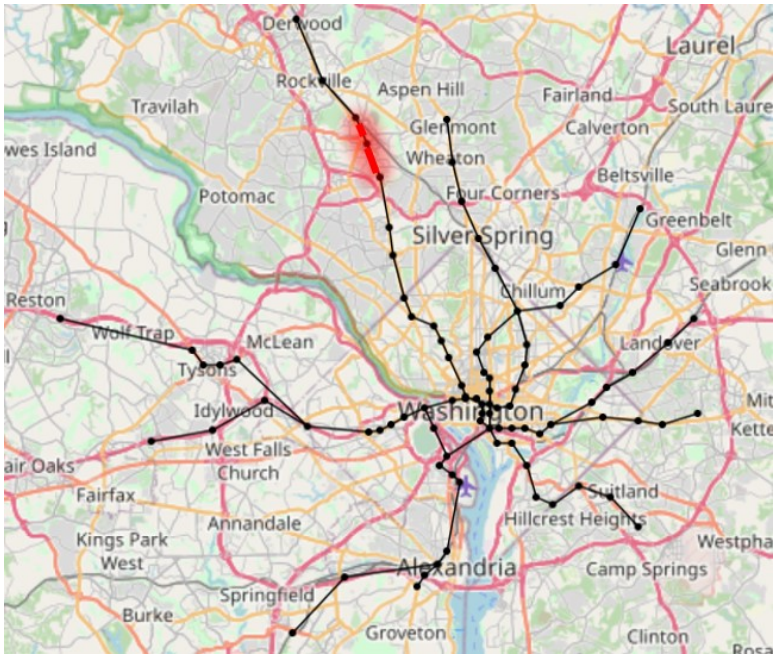


Figure 4.8: Case Study 4 failure edges.

4.6 Case Study 4: Identifying Anomalies in a Metro Network Topology

Further expanding the scalability of our approach to even larger systems, we will exercise it in a real world network, the Washington DC metro system presented in Section 3.9. In this case study we will simulate a failure in two of the edges of the network, as shown in Figures 4.8 and 4.9, by removing the edges from the network representation (i.e. adjacency matrix).

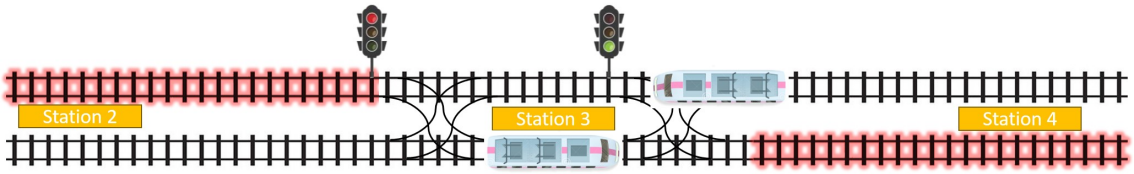


Figure 4.9: Case Study 4 failure scenario (red lines are not operational).

4.6.1 Simulation Mechanics

After training a binary classifier to learn normal graph topology using the machine architectures an learning strategies presented in Section 3.9, we construct an adjacency matrix of the subsystem where the failure occurs (i.e. red line layer of the decomposed network) with the failure edge being removed. Then, similarly to Section 4.5, we simulate failure by feeding normal or abnormal adjacency matrices, at different points of the simulation duration, to the binary classifier previously trained to reconstruct normal topology (See Figure 4.10). The model attempts to reconstruct the inputs, yielding small reconstruction errors when the input was the normal adjacency matrix, and large reconstruction errors when the input was the abnormal adjacency matrix. Figures 4.11 and 4.12 show

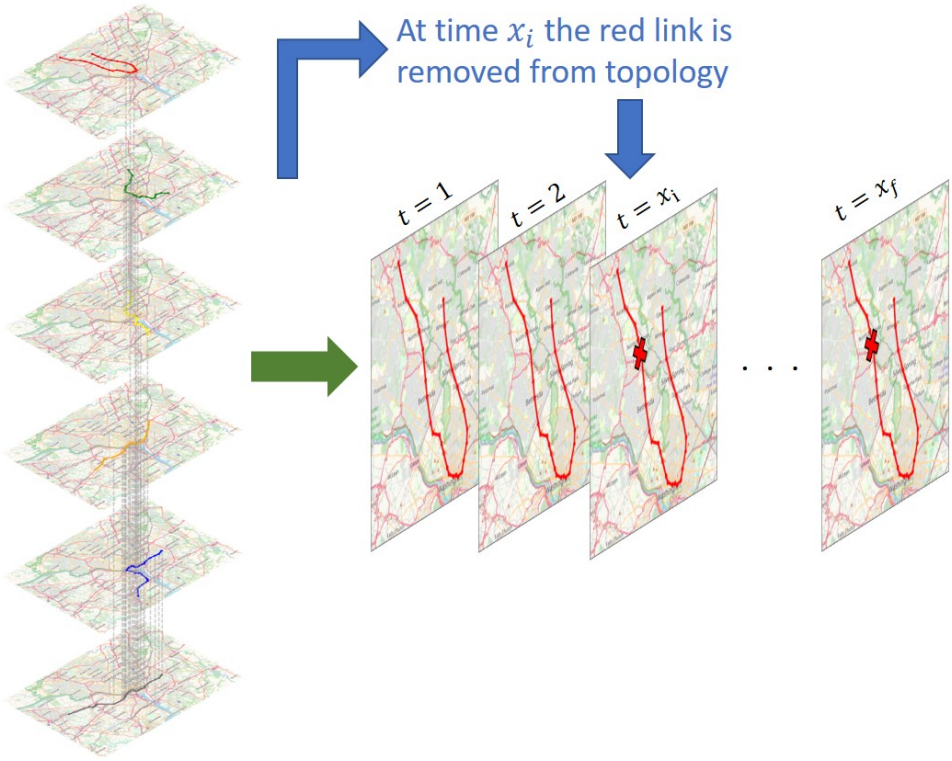


Figure 4.10: Case Study 4 simulation mechanics.

the reconstruction errors for normal and abnormal cases respectively.

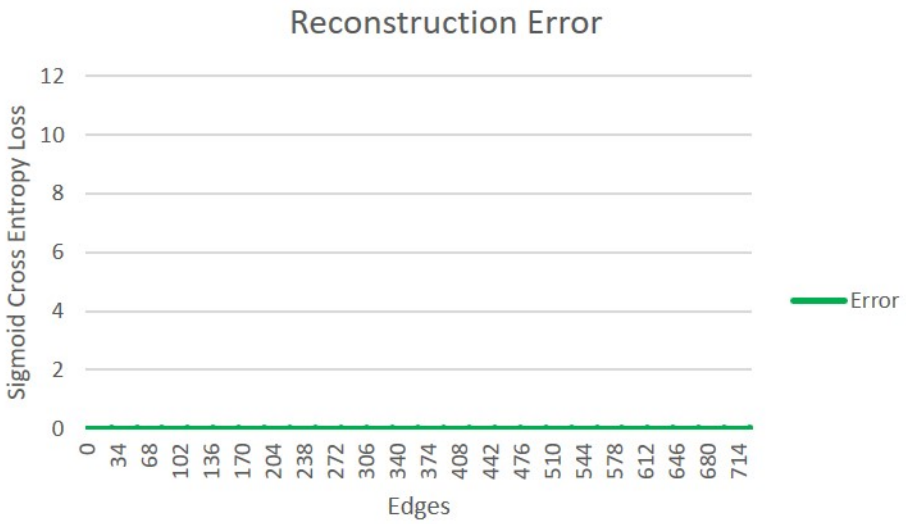


Figure 4.11: Case Study 4 reconstruction error for normal behavior.

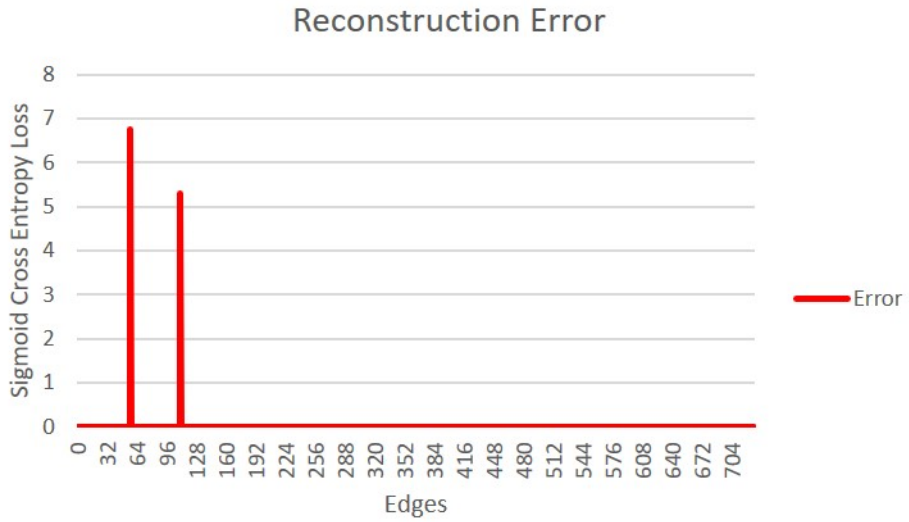


Figure 4.12: Case Study 4 reconstruction error for abnormal behavior.

4.6.2 Identifying Abnormal Behavior

Using the reconstruction error values obtained for each edge at different times during the simulation as inputs, we train a random forest classifier as shown in Figure 4.13. Then, by changing the times at which failure occurs we test the learned random forest model as shown in Figure 4.14. The outputs of the model show a value of "1" at edge indexes and time stamps where a failure is predicted to have occurred, and "0" otherwise. Results show the trained model is capable of predicting with a very high accuracy the time and location where anomalies happened.

4.7 Machine Learning Discussion

This dissertation has up to this point described the role of the ML module in the larger digital twin framework proposed in Chapter 1. It was shown in Chapters 3

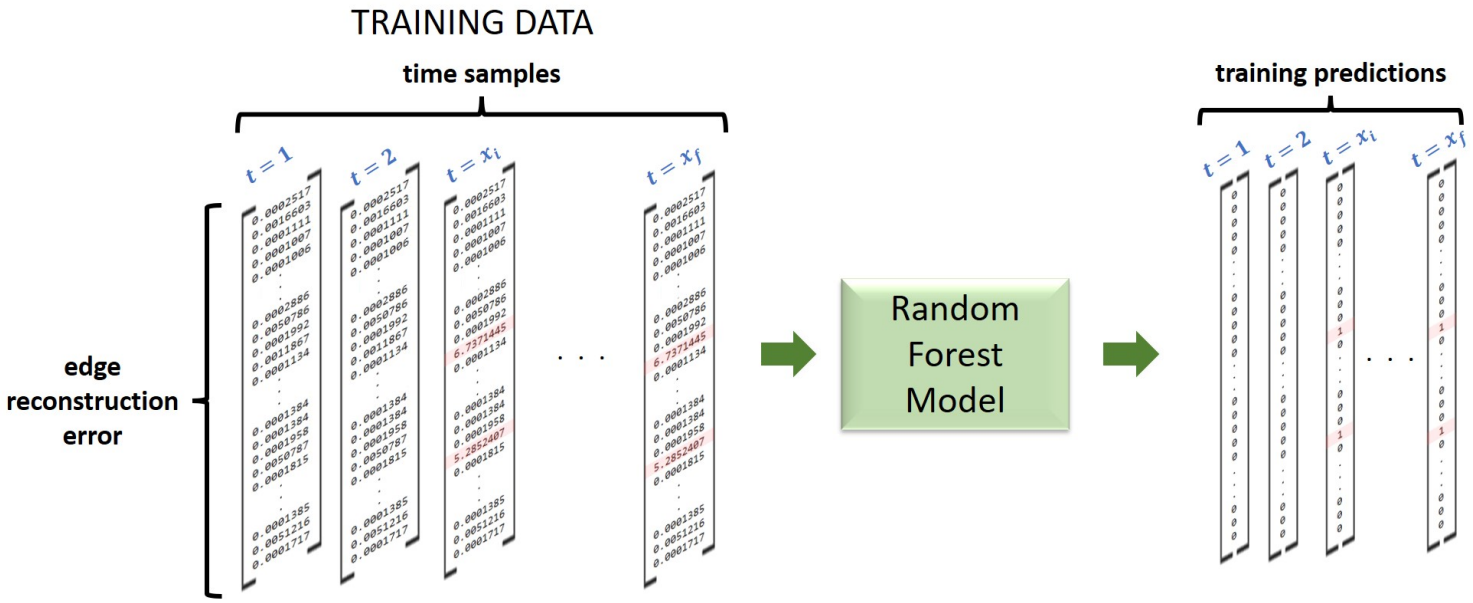


Figure 4.13: Case Study 4 AD classifier training.

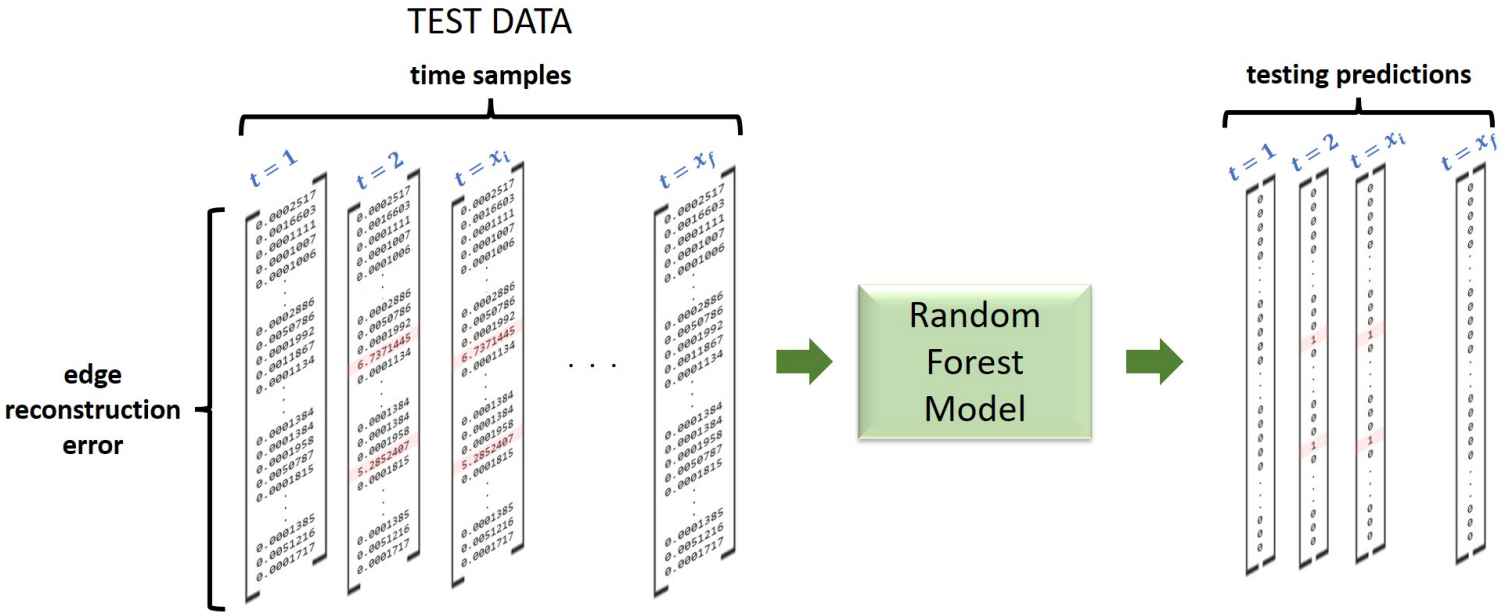


Figure 4.14: Case Study 4 AD classifier testing.

and 4 that ML enables end-to-end optimization of the whole AD pipeline, starting from learning graph topology to building AD models. However, once an event is identified, ML is not capable of performing reasoning, making decisions, and taking actions. Figure 4.15 shows the simplified milestones necessary for completing the objectives of this dissertation. Chapter 5 will discuss the missing piece, decision making.

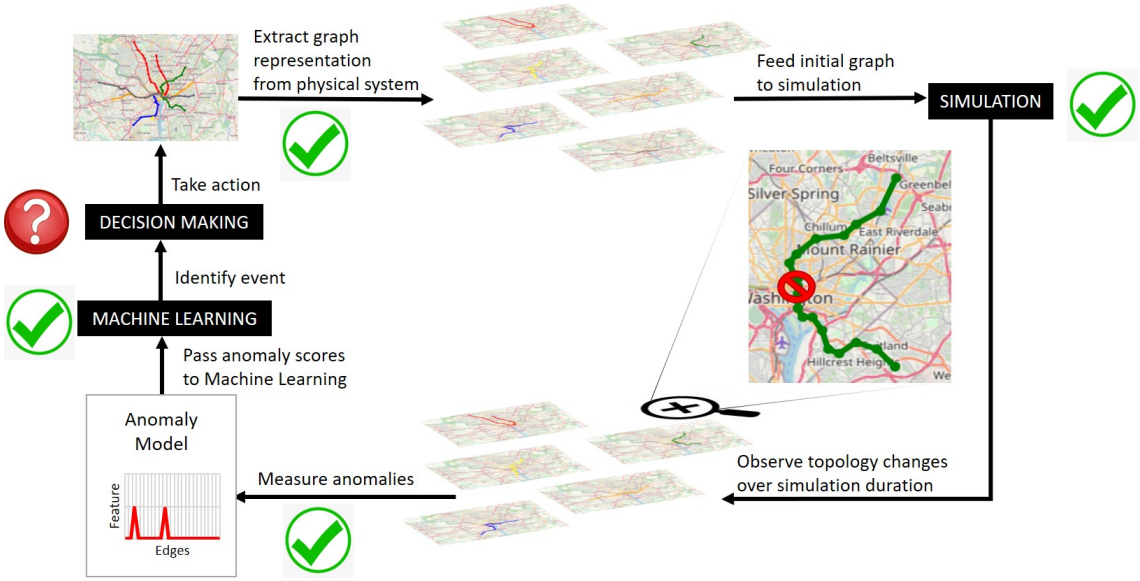


Figure 4.15: Simplified digital twin simulation milestones.

Chapter 5: **Reasoning with Semantic Models**

This chapter describes the software architecture for the generation and execution of semantic models and distributed system behaviors. The first section will give an overview on the generation of semantic graphs and their transformations due to event-based input from external sources. Later in this chapter, we apply the proposed architecture to different urban infrastructures, including a water distribution network and a metrorail network.

5.1 Data-Driven Generation of Semantic Models

Semantic models generation requires a pathway from the specification of ontologies and rules to population of the semantic graphs with instances of the ontology classes. The latter are called individuals. Our previous work on multi-domain semantic modeling [17] suggests that the construction of semantic models should be accomplished with concurrent data-driven development of domain data models, ontologies and rules. As shown along the left-hand side of Figure 5.1, the process begins with development of software for an abstract ontology model (i.e., AbstractOntologyModel). This abstract model contains software needed for the domain-neutral specification and handling of ontologies and rules. Jena Semantic Models are an extension of the abstract model, and

contain software for specification and handling of domain-specific ontologies and rules. They are capable of systematically assembling semantic graphs, transforming the graph structure with rules, and querying the graph structure.

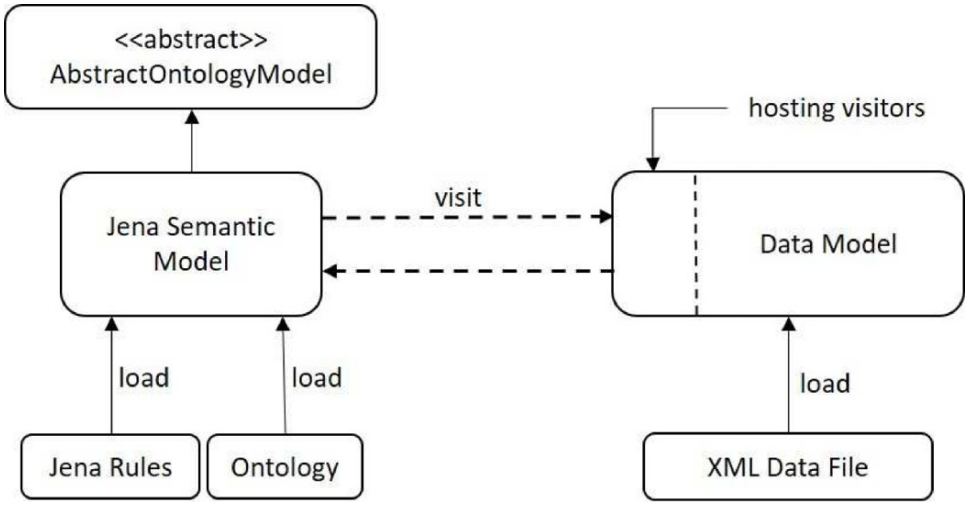


Figure 5.1: Data-driven approach to generation of semantic graphs [13].

On the right-hand side of Figure 5.1, data is imported into data models using JAXB, the XML binding architecture for Java. It is important to note that while Figure 5.1 implies a one-to-one association relationship between semantic graphs and data, in practice a semantic model might visit multiple data models to gather individuals.

After the ontologies and rules have been loaded into the Jena Semantic Model, the semantic model creates instances of the relevant OWL ontologies by visiting the urban data models and gathering information on the individuals within a particular domain. Once the data has been transferred to the Jena Semantic Model and used to create an ontology instance, the rules are applied, and semantic graph transformations can take place.

5.2 Distributed Behavior Modeling

Although urban systems have decentralized system structures, our previous work suggests that instead of modeling the dynamic behavior of systems with centralized control and one large catch-all ontology, systems should be modeled as collections of discipline-specific (or community) ontologies [13, 15, 16]. In this framework, individual urban domains operate as concurrent processes each having their own thread of execution, and respond to streams of incoming data from external domains. Each domain has a semantic graph that evolves according to a set of domain-specific rules, and subject to satisfaction of constraints. Domains interact when needed in order to achieve system-level objectives. If goals are in conflict, or resources are insufficient, then negotiation will need to take place.

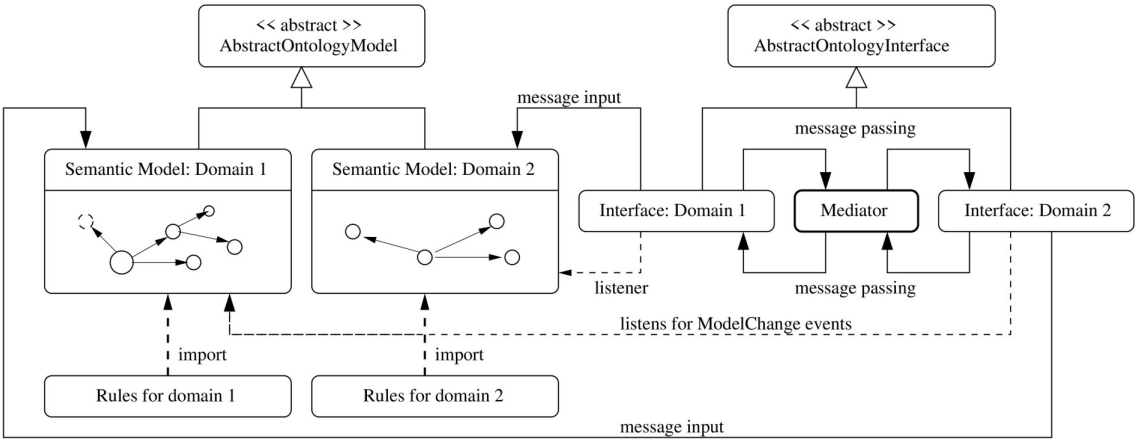


Figure 5.2: System architecture for distributed system behavior modeling with ontologies, rules, mediators and message passing mechanisms [13].

Figure 5.2 shows the software architecture for distributed system behavior modeling for collections of semantic models that have dynamic behavior. The abstract ontology

model class contains concepts common to all ontologies (e.g., the ability to receive message input). Domain-specific ontologies are extensions of the abstract ontology classes. They build the ontology classes, relationships among classes, and attributes of classes. When domain-specific rules are imported into the model, semantic graph transformations are enabled by formal reasoning and event-based input from external sources.

5.3 Communication Mechanism

Distributed behavior modeling involves multiple semantic models, and require mechanisms of communication among semantic models. We provide this functionality in our distributed behavior model by loosely coupling each semantic model to a semantic interface. Each semantic interface listens for changes to it's domain semantic graph, and when required, forwards the essential details of the change to other domains (interfaces) that have registered interest in receiving notification of such changes. They also listen for incoming messages from external semantic models. Since changes to the semantic graph structure are triggered by events (e.g., the addition of an individual; an update to an individual's attribute value; a new relationship among individuals), a central challenge is design of the rules and ontology structure so that the interfaces will always be notified when exchanges of information need to occur. Messages are defined by their subject (e.g., edge repair ticket), a source, a destination, and a reference to the value of the data being exchanged. The receiving interface will forward incoming messages to the semantic model, which, in turn, may trigger an update to the semantic graph. Since end-points of the basic message passing infrastructure are common to all semantic model interfaces, an

abstract ontology interface model is defined.

When the number of participating applications domains is very small, point-to-point channel communication between interfaces is practical. However, as the number of participating domain increases, an efficient way of handling domain communication is by delegating the task of sending and receiving specific requests to a central object. This object is responsible for the overall communication of the system. From here on out we will refer to this object as the mediator object, stemming from software engineering's mediator design pattern, where direct communications between software components is restricted by a mediator object. Components of the system send messages to the mediator rather than to the other components; likewise, they rely on the mediator to send change notifications to them. The implementation of this pattern in our previous work greatly simplified the other components in the system, as they were allowed to be more generic since they no longer have to contain logic to manage communication with other components [13, 15, 16]. Because other components remain generic, the mediator has to be application specific in order to encapsulate application-specific behavior. One can reuse all other components for other applications, and only need to rewrite the mediator object for the new application.

5.4 Case Study 5: Responding to a Water Distribution Network Event

This case study revisits the water distribution network presented in Sections 3.8 and 4.5. Here the objective is to use semantic models to reason and make decisions after an event has been identified. Suppose an edge failure as shown in Figure 4.2 demand the closing of shut-off valves for water main repairs. Shut-off valves quickly stop the

flow of water if there's a leak or other problem to prevent extensive damage. We create a distributed semantic model with the framework presented in Sections 5.1, 5.2, and 5.3, and use the anomaly detection results of Section 4.5 as an external input to the model, that triggers water main repairs.

5.4.1 Ontology Models

We start our semantic models by creating ontologies for a water system domain and a system operation domain. The water system ontology contains classes representing different components found in our water system case study, such as "layers", "valves", "edges". The ontology defines relationships between classes, such as "layer has valve" or "layer has edge". Ontology classes have attributes such as "edgeNumber" or "isOperational". Similarly, the system operation ontology contains classes representing different operation elements found in our water system case study, such as "operator" and "repair". The ontology defines relationships between classes, such as "repair is performed by operator". Ontology classes have attributes such as "repairsEdge" or "isComplete".

We employ the Web Ontology Language (OWL) to define ontologies. Visual descriptions of the OWL files for the water system and system operation models are presented in Figures 5.3 and 5.4 respectively.

5.4.2 Data Models

In this problem setup, the information needed to populate the semantic graphs is stored as key/value pairs in XML datafiles. The key (e.g., "operatorName", "edgeNumber",

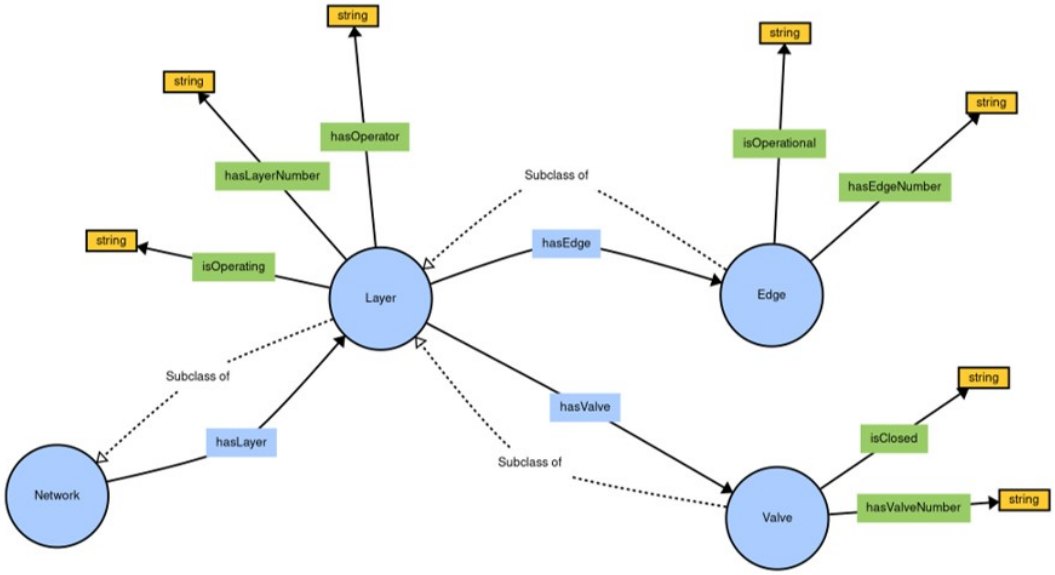


Figure 5.3: Water system ontology diagram with classes, attributes, and relationships among them.

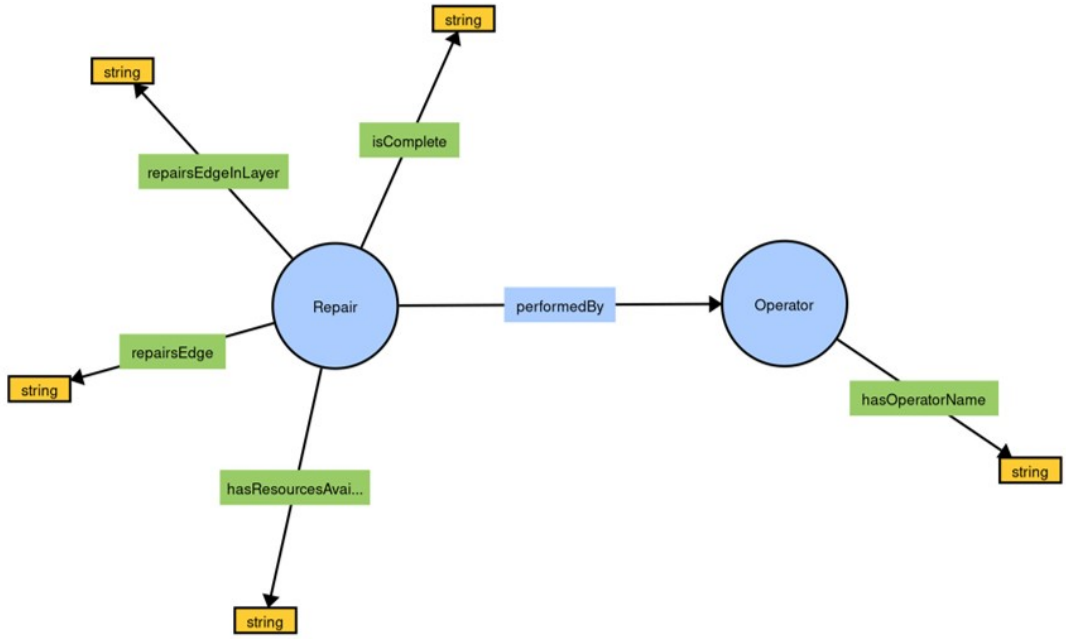


Figure 5.4: System operation ontology diagram with classes, attributes, and relationships among them.

etc.) identifies, and is used to retrieve the values (e.g., “Maria Coelho”, “Edge (0,2)”, etc.). The content stored in the XML key/value pairs is extracted and used to create ontology class instances in the data model. This implementation employs JAXB technology for the XML content extraction and creation of elements in the data models. We then systematically visit each element of the data model and create instances of the ontology classes, laden with the data from XML files. Complete descriptions of the XML data for the water system are found in Figure 5.5.

When building the XML data files, we made the following simplying assumptions about the system:

- Each layer has one shut-off valve
- Valve is always open at the start of simulation

It is important to note that the XML datafiles follow a certain standard when defining edge numbering. Such standard has to be cohesive with the standard used by the ML module, especially considering that matrix reordering approaches may have been applied to the network representation as a learning strategy, see Section 3.6. In addition, edge operation status values are retrieved from the ML module’s anomaly detection exercises, emphasizing the need for cohesion among modules. More details on design approaches taken for ensuring full integration of digital twin modules are presented in Chapter 6.


```
<?xml version="1.0" encoding="UTF-8"?>
<WaterModel author="Maria Coelho" date="2022" source="UMD">

<Layer>
        <attribute text="LayerNumber" value="Layer 5"/>
        <attribute text="OperatorName" value="Maria Coelho"/>
        <Valve>
                <attribute text="ValveNumber" value="Valve 5"/>
                <attribute text="OperationStatus" value="Open"/>
        </Valve>
        <Edge>
                <attribute text="EdgeNumber" value="Edge (1,4)"/>
                <attribute text="OperationStatus" value="0"/>
        </Edge>
        <Edge>
                <attribute text="EdgeNumber" value="Edge (1,3)"/>
                <attribute text="OperationStatus" value="0"/>
        </Edge>
        <Edge>
                <attribute text="EdgeNumber" value="Edge (0,2)"/>
                <attribute text="OperationStatus" value="1"/>
        </Edge>
        <Edge>
                <attribute text="EdgeNumber" value="Edge (0,3)"/>
                <attribute text="OperationStatus" value="0"/>
        </Edge>
        <Edge>
                <attribute text="EdgeNumber" value="Edge (2,3)"/>
                <attribute text="OperationStatus" value="0"/>
        </Edge>
</Layer>
</WaterModel>
```

Figure 5.5: Water system data stored in XML format.

5.4.3 Jena Rules

Ontologies by themselves provide a framework for the representation of knowledge, but otherwise cannot model the dynamic evolution of class instances, attributes and relationships. Consider the water system ontology, some of the data remains constant over time (e.g., edge numbers), while other data is dynamic (e.g., edge operational status). However, when combined with a set of domain-specific rules, ontological representations enable semantic graph transformations by formal reasoning and event-based input from external sources. In our application, we use Jena Rules to define domain-specific rules. Complete descriptions of the rules for the water system and system operation models can be found in Figures 5.6 and 5.7 respectively.

The two rules for the water system model state the following:

```
// Rule 01: Close a valve ....

[ Valve01: (?l rdf:type af:Layer) (?v rdf:type af:Valve) (?l af:hasValve ?v) (?v af:isClosed "false")
           (?e rdf:type af:Edge) (?l af:hasEdge ?e) (?e af:isOperational "false")
           -> drop(3) drop(6) (?v af:isClosed "true") (?l af:isOperating "false") ]

// Rule 02: Open a valve ....

[ Valve02:  (?l rdf:type af:Layer) (?l af:isOperating "true") (?v rdf:type af:Valve) (?l af:hasValve ?v) (?v af:isClosed "true")
            -> drop(4) (?v af:isClosed "false")]
```

Figure 5.6: List of Jena rules for transformation of the water system semantic graph.

```
// Rule 01: Perform a repair ....

[ Repair01: (?r rdf:type af:Repair) (?r af:isComplete "false") (?r af:hasResourcesAvailable "true") -> drop(1) (?r af:isComplete "true") ]
```

Figure 5.7: List of Jena rules for transformation of the system operation semantic graph.

1. If edge **e** in layer **l** is not operational, close valve **v** in layer **l**.

2. If all edges in layer **l** are operational, open valve **v** in layer **l**.

The rule for the system operation model states that if a repair **r** is not complete and repair **r** resources are available, complete repair **r**. Graph transformations can occur due to input. For example, the water system graph changes because a an edge is no longer operational, or the operation model graph changes because a repair has now the resources available to be completed.

5.4.4 Simulation Model

The step-by-step procedure for assembly of this case study's simulation model is as follows:

1. **Create empty semantic graph models, then load ontologies.**

2. **Connect water system and system operation models.** Each component of the system (i.e., water system and system operation) is paired with an interface for com-

munication and information exchange with other components. Every component also registers with the mediator as message listeners.

3. **Create visitor object models.** The result of the previous step is a network of connected semantic graphs that are populated with ontologies, but do not contain individuals. The data for the water system and system operation individuals is contained in XML files. We populate the semantic models with data on individual water system components and system operation by creating visitor objects models that will be given permission to visit and extract information from the water system and system operation data models.

4. **Get data from XML files.** This step creates data models for the water system and system operation ontologies, and populates them with data imported from XML datafiles.

5. **Populate semantic models with individuals.** The water system and system operation semantic graphs are populated with individuals by visiting the water system and system operation data models, respectively.

6. **Add rules and execute them.** Finally, we import rules into each of the semantic models, and trigger transformations to the graph models by executing them.

5.4.5 Simulation Results

In this case study, interactions between the water system and system operation models occur in response to data-driven events. For example, the water system rules

define that a layer's shut-off valve should be closed if an edge is not operational. When the retrieved data shows an edge's is not operational, the boolean property "isOperational" will be set to "False". Rules will be triggered so that the corresponding shut-off valve property "isClosed" will be set to "True". The water system's semantic model interface will identify the corresponding update to the semantic graph, and in response, send an edge repair request to the system operator in the form of a message, containing relevant information such the edge number.

The mediator will match the message destination (i.e. the failed edge operator) with that system's operation semantic model interface, and forward the message. The system operation semantic model interface will identify the message type (i.e. edge repair request), and trigger changes to the semantic model graph by adding a new repair to the list of pending repairs.

The system operation semantic model interface is listening for changes to the semantic model graph. Once a repair is added, and resources are available to complete the repair, rules are triggered so that the repair property "isComplete" will be set to "True". The system operation semantic model interface will identify the corresponding update to the semantic graph, and in response, send an edge repair confirmation to the water system in the form of a message.

One more time, the mediator will match the message destination (i.e. the water system to which the repaired edge belongs) with that water system's semantic model interface, and forward the message. The water system semantic model interface will identify the message type (i.e. edge repair confirmation), and trigger changes to the

semantic model graph by setting the repaired edge's boolean property "isOperational" to "True". Rules will be triggered so that the corresponding shut-off valve property "isClosed" will be set to "False".

A visualization of the semantic graph transformations through the simulation can be seem in Figure 5.8.

5.5 Case Study 6: Responding to a Metro Network Event

This case study revisit the metro network presented in Sections 3.9 and 4.6 to test the scalability of our semantic modeling approach. Similar to Section 5.5, the objective here is to use semantic models to reason and make decisions after an event has been identified. Suppose edge failures as shown in Figures 4.8 and 4.9 demand single tracking service on the metro's red line. When rail disruptions prevent trains from operating in certain track directions, single-tracking service is used to allow for continuous operation. We create a distributed semantic model with the framework presented in Sections 5.1, 5.2, and 5.3, and use the anomaly detection results of Section 4.6 as an external input to the model, that triggers single tracking service.

5.5.1 Ontology Models

We start our semantic models by creating ontologies for a metro system domain and a system operation domain. The metro system ontology contains classes representing different components found in our metro system case study, such as "layers", "signals",

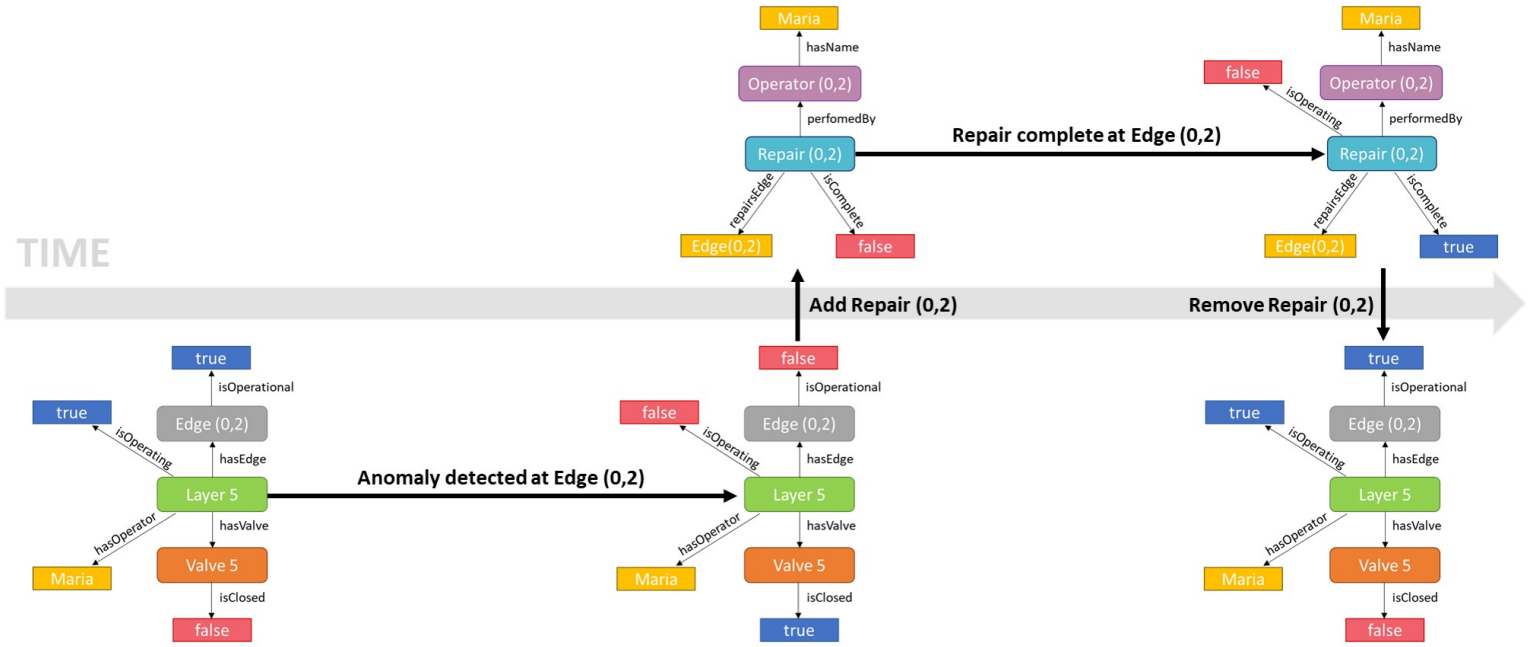


Figure 5.8: Visual representation of semantic graph transformations in Case Study 5.

"edges". The ontology defines relationships between classes, such as "edge has signal" or "layer has edge". Ontology classes have attributes such as "edgeNumber" or "isOperational". Similarly, the system operation ontology contains classes representing different operation elements found in our metro system case study, such as "operator" and "repair". The ontology defines relationships between classes, such as "repair is performed by operator". Ontology classes have attributes such as "repairsEdge" or "isComplete".

Note that the system operation ontology presented here is identical to the one presented in Section 5.5. This is because we built our operation ontology in a generic way so that it could be reused in different applications. Hence, only the system ontology has to be application specific in order to encapsulate application-specific behavior.

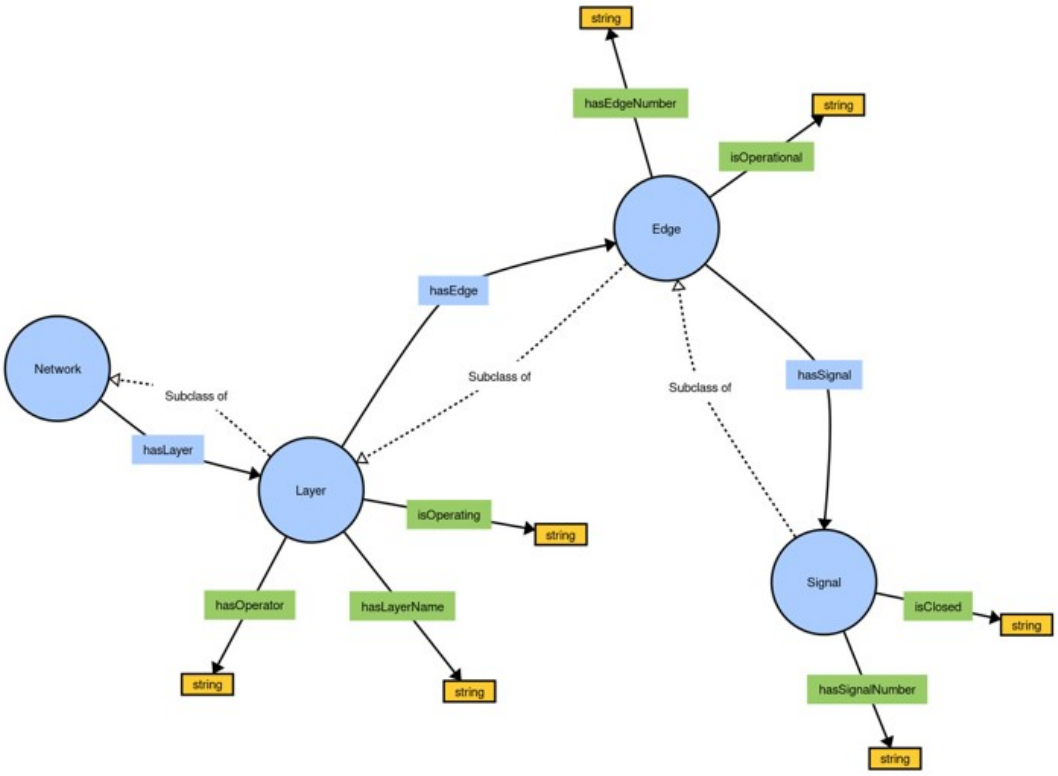


Figure 5.9: Metro system ontology diagram with classes, attributes, and relationships among them.

We employ the Web Ontology Language (OWL) to define ontologies. Visual descriptions of the OWL files for the metro system model is presented in Figure 5.9.

5.5.2 Data Models

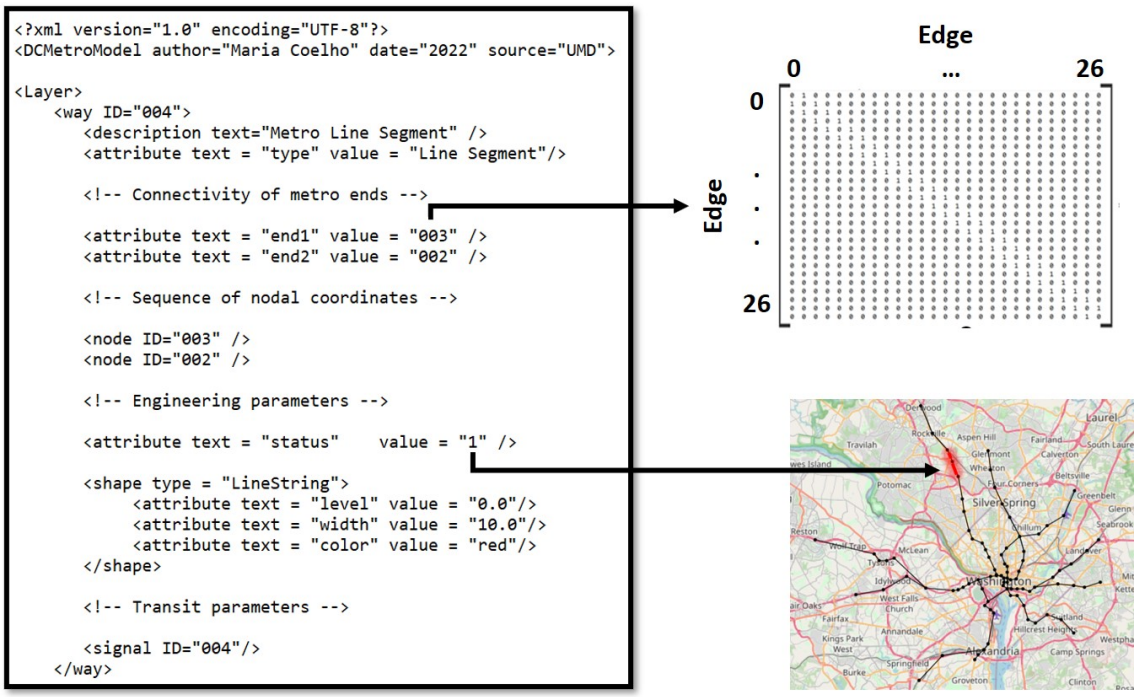
Similar to the case study presented in Section 5.5, the information needed to populate the semantic graphs is stored as key/value pairs in XML datafiles. The key identifies, and is used to retrieve the values. The content stored in the XML key/value pairs is extracted and used to create ontology class instances in the data model. The XML content extraction and creation of elements in the data models is done by employing JAXB technology. We then systematically visit each element of the data model and create instances of the ontology classes, laden with the data from XML files. Abbreviated descriptions of the XML data for the metro system are found in Figure 5.10.

When building the XML data files, we made the following simplying assumptions about the system:

- Each edge direction has one traffic signal
- Lines are always operational at the start of simulation

5.5.3 Jena Rules

Combining ontologies with a set of domain-specific rules enables semantic graph transformations by formal reasoning and event-based input from external sources. In our application, we use Jena Rules to define domain-specific rules. Complete descriptions of




```
<?xml version="1.0" encoding="UTF-8"?>
<DCMetroModel author="Maria Coelho" date="2022" source="UMD">

<Layer>
    <way ID="004">
       <description text="Metro Line Segment" />
       <attribute text = "type" value = "Line Segment"/>

       <!-- Connectivity of metro ends -->

       <attribute text = "end1" value = "003" />
       <attribute text = "end2" value = "002" />

       <!-- Sequence of nodal coordinates -->

       <node ID="003" />
       <node ID="002" />

       <!-- Engineering parameters -->

       <attribute text = "status"    value = "1" />

       <shape type = "LineString">
            <attribute text = "level" value = "0.0"/>
            <attribute text = "width" value = "10.0"/>
            <attribute text = "color" value = "red"/>
       </shape>

       <!-- Transit parameters -->

       <signal ID="004"/>
    </way>
```

Figure 5.10: Metro system data stored in XML format.

the rules for the metro system and system operation models can be found in Figures 5.11 and 5.7 respectively.

```
// Rule 01: Close a signal ....

[ Signal01: (?e rdf:type af:Edge) (?s rdf:type af:Signal) (?e af:hasSignal ?s) (?e af:isOperational "false")
           -> drop(3) (?s af:isClosed "true")]

// Rule 02: Open a signal ....

[ Signal02:  (?e rdf:type af:Edge) (?s rdf:type af:Signal) (?e af:hasSignal ?s) (?s af:isClosed "true") (?e af:isOperational "true")
            -> drop(3) drop(4) (?s af:isClosed "false") ]
```

Figure 5.11: List of Jena rules for transformation of the metro system semantic graph.

The two rules for the metro system model state the following:

1. If edge **e** is not operational, close signal **s** in edge **e**.

2. If edge **e** is operational, open signal **s** in edge **e**.

The rule for the system operation model states that if a repair **r** is not complete and repair **r** resources are available, complete repair **r**. Graph transformations can occur

due to input. For example, the metro system graph changes because a an edge is no longer operational, or the operation model graph changes because a repair has now the resources available to be completed.

5.5.4 Simulation Model

The step-by-step procedure for assembly of this case study's simulation model is as follows:

1. **Create empty semantic graph models, then load ontologies.**

2. **Connect metro system and system operation models.** Each component of the system (i.e., metro system and system operation) is paired with an interface for communication and information exchange with other components. Every component also registers with the mediator as message listeners.

3. **Create visitor object models.** The result of the previous step is a network of connected semantic graphs that are populated with ontologies, but do not contain individuals. The data for the metro system and system operation individuals is contained in XML files. We populate the semantic models with data on individual metro system components and metro system operation by creating visitor objects models that will be given permission to visit and extract information from the metro system and system operation data models.

4. **Get data from XML files.** This step creates data models for the metro system and system operation ontologies, and populates them with data imported from XML

datafiles.

5. **Populate semantic models with individuals.** The metro system and system operation semantic graphs are populated with individuals by visiting the metro system and system operation data models, respectively.

6. **Add rules and execute them.** Finally, we import rules into each of the semantic models, and trigger transformations to the graph models by executing them.

5.5.5 Simulation Results

In this case study, interactions between the metro system and system operation models occur in response to data-driven events. For example, the metro system rules define that an edge's traffic signal should be closed if an edge is not operational. When the retrieved data shows an edge's is not operational, the boolean property "isOperational" will be set to "False". Rules will be triggered so that the corresponding traffic signal property "isClosed" will be set to "True". The metro system's semantic model interface will identify the corresponding update to the semantic graph, and in response, send an edge repair request to the system operator in the form of a message, containing relevant information such the edge number.

The mediator will match the message destination (i.e. the failed edge operator) with that system's operation semantic model interface, and forward the message. The system operation semantic model interface will identify the message type (i.e. edge repair request), and trigger changes to the semantic model graph by adding a new repair to the

list of pending repairs.

The system operation semantic model interface is listening for changes to the semantic model graph. Once a repair is added, and resources are available to complete the repair, rules are triggered so that the repair property "isComplete" will be set to "True". The system operation semantic model interface will identify the corresponding update to the semantic graph, and in response, send an edge repair confirmation to the metro system in the form of a message.

One more time, the mediator will match the message destination (i.e. the metro system to which the repaired edge belongs) with that metro system's semantic model interface, and forward the message. The metro system semantic model interface will identify the message type (i.e. edge repair confirmation), and trigger changes to the semantic model graph by setting the repaired edge's boolean property "isOperational" to "True". Rules will be triggered so that the corresponding traffic signal property "isClosed" will be set to "False".

A visualization of the semantic graph transformations through the simulation can be seem in Figure 5.12.

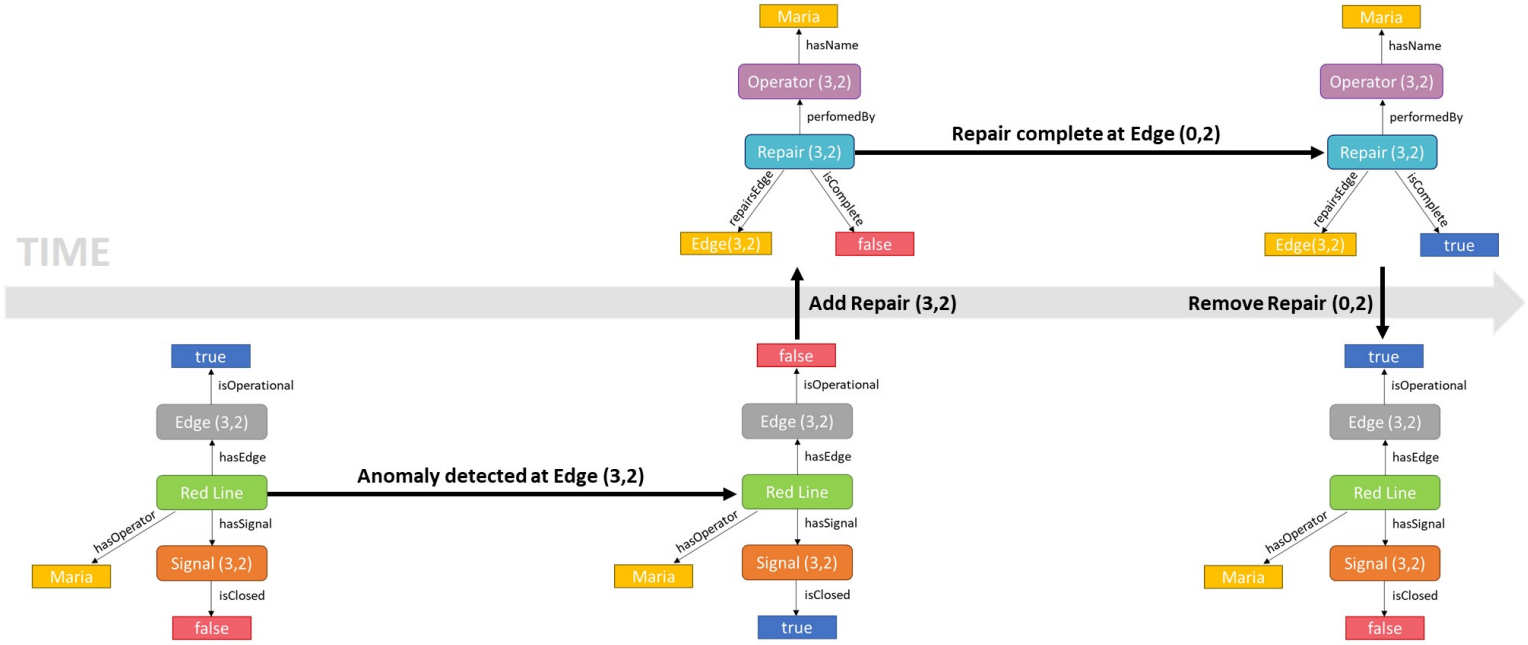


Figure 5.12: Visual representation of semantic graph transformations in Case Study 6.

Chapter 6: **Architecting and Testing Digital Twin**

This chapter addresses the connection between the Semantic Model and ML modules. The first section will give an overview of the the basic mechanisms for semantic/ML interaction. Later in this chapter, we discuss design approaches that should be taken into account for ensuring cohesion of digital twin modules.

6.1 Mechanisms for Semantic Model/ML Interaction

Up to this point, this dissertation has described the role of the ML module and semantic modeling module in the larger digital twin framework proposed in Chapter 1. It was shown in Chapters 3 and 4 that ML enables end-to-end optimization of the whole anomaly detection pipeline, starting from learning graph topology to building anomaly detection models. Chapter 5 focused on decision making, and presented strategies for building semantic models that are capable of performing reasoning, making decisions, and taking actions once an event is identified.

Although these modules have individual purposes of their own, they work hand-in-hand in order to achieve a greater common purpose: monitoring and controlling urban processes. Therefore, it is extremely important that their designs are cohesive with one

another for achieving an integrated digital twin. Part of that integration is dependent on an efficient protocol for transmission of events. In order to exchange data from one module to another, the modules need to have a common understanding of how the data will be formatted.

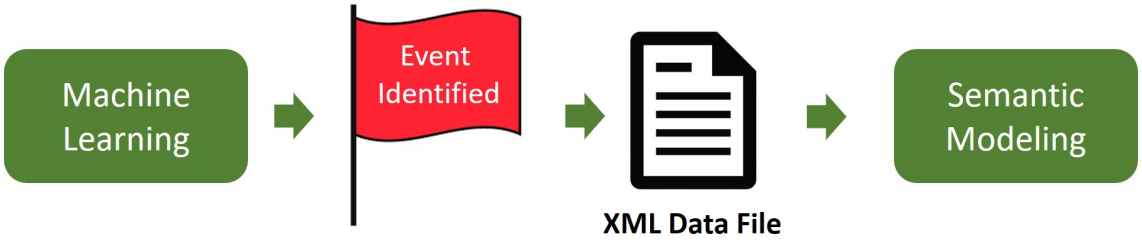


Figure 6.1: Data exchange mechanism between ML module and Semantic Model module.

As discussed in Section 5.5.2 and shown in Figure 6.1, events are identified through anomaly detection exercises and transmitted from the ML module to the semantic modeling module via XML datafiles. There are several advantages for using XML for data transfer:

- XML is a “self-describing” format, making it readable and understandable
- XML is compatible with most coding languages
- XML is extendable, allowing developers to create their own tags, using the natural language of their application’s domain.

XML datafiles are populated with information regarding network structure (i.e. graph components, edge numbers, etc.) and operation (i.e. values of "0" for operational components and "1" for non operational components). The datafile is a critical component of the digital twin as it stores and transfers information regarding anomalies, their

locations and time of occurrence. Both modules need to have an accurate understanding of anomalies in order to enact a timely response. To create a common understanding between modules, the XML datafile key/value pairs have to follow a predetermined network structure definition. For instance, "Edge (0,2)" in the ML module has to correspond to the same "Edge (0,2)" in the semantic model module.

6.2 Digital Twin Design Approach

In a step toward integrating digital twin modules, we propose a design framework that supports: (1) concurrent data-driven development of semantic models and machine learning models, and (2) executable processing of incoming events. Figure 6.2 shows details of our proposed design framework. A key advantage of this approach is that it forces a common understanding about the physical system between modules. The co-development of rules, ontologies, data models, anomaly detection models, and network models forces developers to think about the chain of dependency relationships that allow the rules to work. Rules require information from the ontologies, which in turn, require data values from the data models, which in turn, require anomaly cases from anomaly models, which in turn, require definitions of normal behavior from network models. In this way, we support common understanding across domains and establish a powerful mechanism that allows a digital twin to be fully integrated.

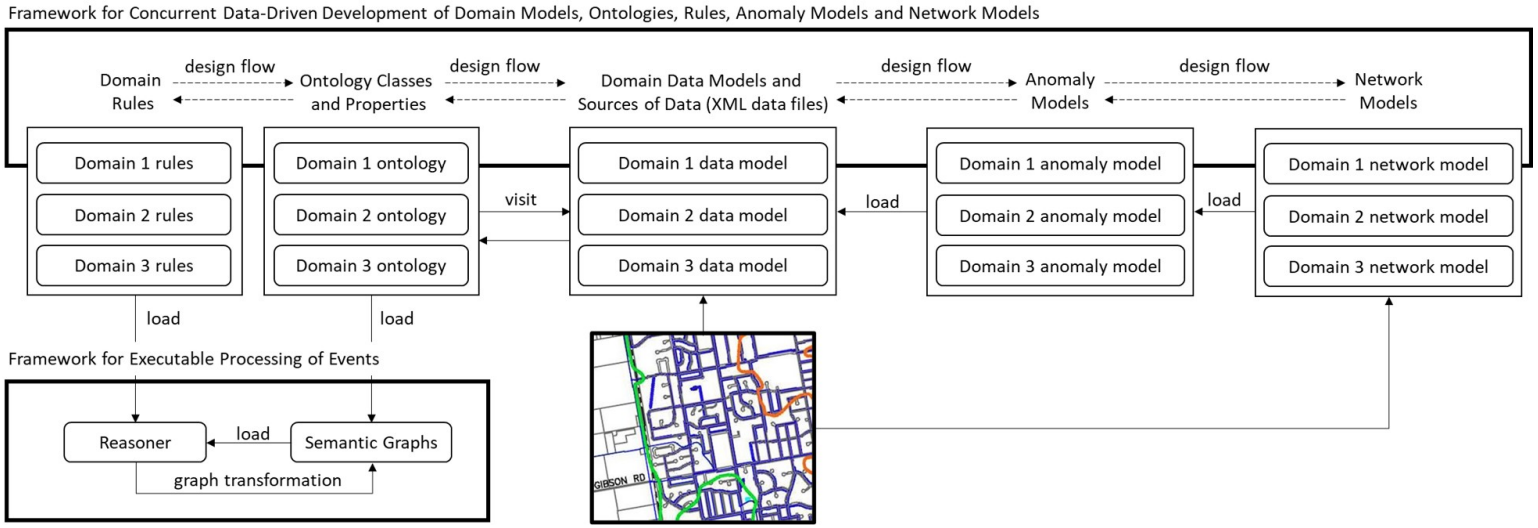


Figure 6.2: Proposed design framework for the integration of ML and Semantic Model modules.

Chapter 7: **Conclusions and Future Work**

In this dissertation, we have:

- Identified machine architectures and strategies of learning for a variety of graph structures.
- Exercised the machine architectures and strategies of learning on case study problems (i.e., a water distribution system and an urban metrorail system).
- Explored the effects of graph size on learning performance.
- Identified the scalability of our approaches for learning large-scale graphs.
- Identified events in learned network models via time-series anomaly detection.
- Established basic mechanisms for semantic / machine learning interaction.

The results presented in this work are a step towards our objective of architecting smart city digital twins. We have contributed to the advancement of knowledge in the large urban digital twin design problem by investigating ways in which Semantic Models and ML can work together to support event- and data-driven decision making. Specifically, we now understand how to build machine architectures to learn graph representations that

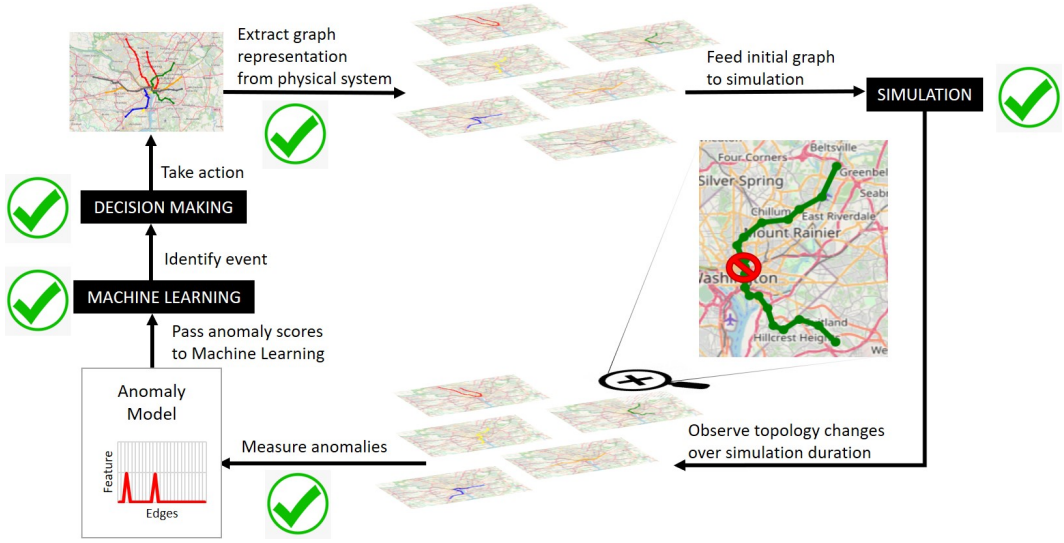


Figure 7.1: Process flowchart for training and executing machine.

are accurate and complete. We have discovered strategies for learning large-scale graphs that are efficient. In addition, we have learned to model anomalies in behavior using learned graph representations and identify outliers using automated means. Lastly, we have established basic mechanisms for semantic/ML interaction, and design approaches for ensuring cohesion of digital twin modules.

Our discoveries have allowed us to build a simplified digital twin demonstration. Figure 7.1 shows how our case studies have accomplished this task. We started by extracting a graph representation of the urban infrastructure and determining the initial parameters of the system. As time progresses, the digital twin monitors changes in the system. Anomaly models were generated, and machines were be trained to understand a number of salient features of acceptable and unacceptable behavior. When an unacceptable behavior was identified, urban recovery procedures were triggered.

Although our contributions are a step toward building fully operational digital twin demonstration, there are still important remaining questions. This section describes

opportunities to improve our approach and possible future research directions.

7.1 Learning Graph Attributes

Urban networks are associated with a rich set of node attributes that are in constant change due to day-to-day temporal variations in demand and, in the longer term, network expansion due to urban growth. These variations cause the emergence of new content patterns and the fading of old content patterns. In addition, it has been widely studied and reported that there exists a strong correlation among the attributes of linked nodes [46]. These node correlations and changing characteristics motivate us to seek an effective neural network architecture to capture both network structure and attribute evolving patterns, which is of fundamental importance to learning in a dynamic environment.

Our preliminary results revealed machine architectures required to learn different types of graph structures. More work is needed to design a machine architecture that captures graph attributes. A crucial research direction is to further develop case studies 1 and 2 to incorporate graph attributes and study how attribute information can be learned.

7.2 Network Decomposition

In our approach to network decomposition, we have considered multi-layer networks with just one aspect. That is, we divide layers based on a single common characteristic such as component type, connections, time, or space. However, in more complex urban networks, the layers may be divided into several groups, which indicates that mul-

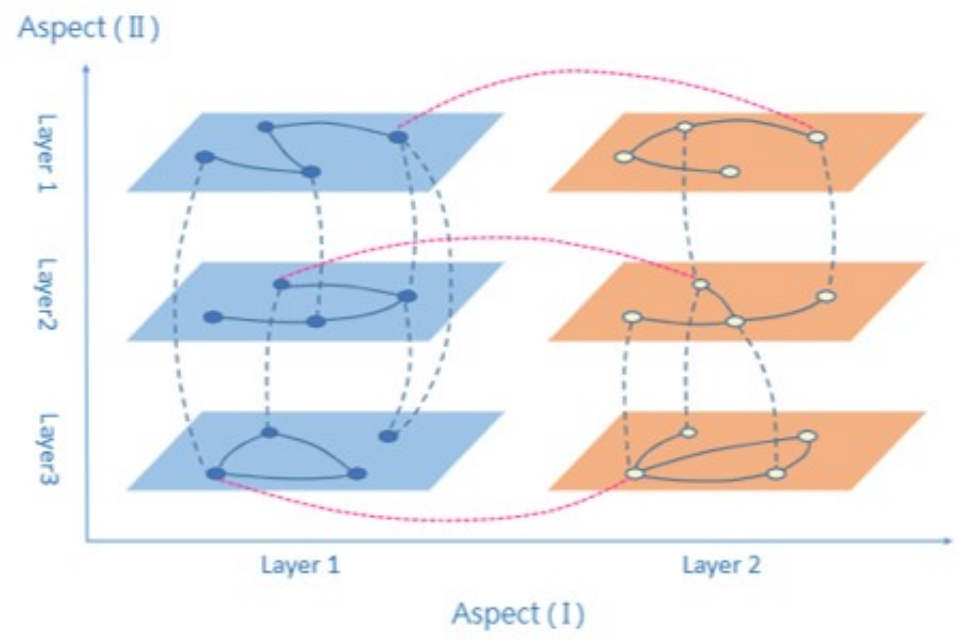


Figure 7.2: Multi-aspect representation of the multi-layer network model [60].

tilayer networks should also be distinguished when observed from different aspects. For example, a transportation network can be characterized by different means such as railways, airports and roads. Meanwhile, this network is also temporal since there are new railways, airports and roads being built at any time point. Thus, this layer is divided into two groups: according to time stamps or according to transportation means. In order not to lose the information of the networks from either aspect, we have to construct a more complex multilayer network. An interesting research direction would be to study complex multi-layer decomposition approaches and if it can improve the learning process by preserving more information about the network. In addition, network decomposition and its representation through a supra-adjacency matrix allows for the study of inter-layer relationships (i.e. interdependencies). Another interesting research direction would be to explore approaches to learning inter-layer relationships, and if it can improve the overall

graph learning process.

7.3 Multi-domain Behavior Modeling

In our infrastructure behavior modeling applications we have not considered the interactions between domains belonging to different disciplines. We envision that further iterations of this work will model the dynamic behavior of systems, that belong to different disciplines, with collections of domain specific networks, that evolve in response to events as shown in Figure 7.3.

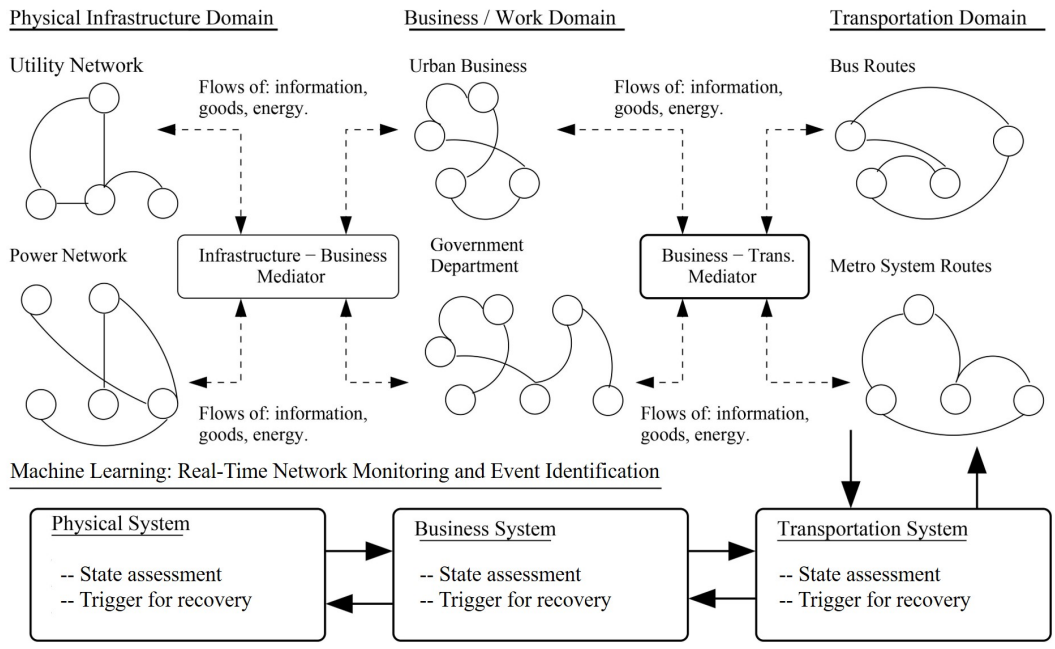


Figure 7.3: Architecture for multi-domain behavior modeling with many-to-many associations.

Individual domains will operate as concurrent processes each having their own thread of execution, and will respond to streams of incoming data from external domains. Domains will interact and negotiate when needed in order to achieve system-level objectives.

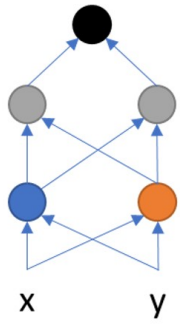
Machine learning tools will provide support for real-time assessment of individual domain's state and will trigger recovery actions in response to severe events. In this way, multi-disciplinary interactions can be represented, which is of fundamental importance to behavior modeling in an urban environment.

Appendix A: Additional Topology Classification Results

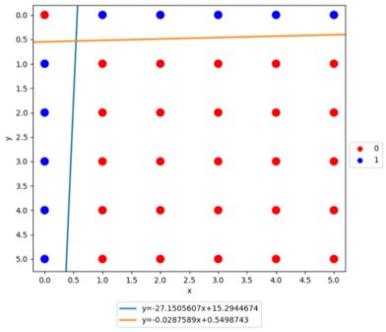
This Appendix includes classification results for urban topologies presented in Chapter 3 and other additional topologies of interest.

$$A = \begin{bmatrix} 0 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

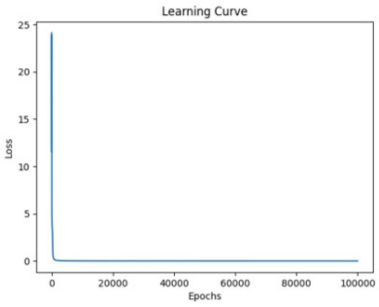
(a) Network Representation.



(b) Neural Network.



(c) Decision boundaries.

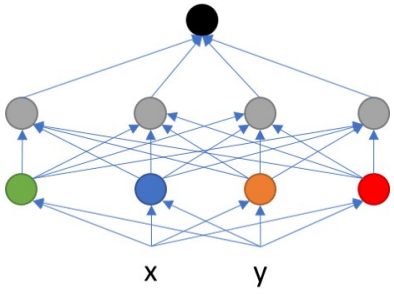


(d) Learning curve.

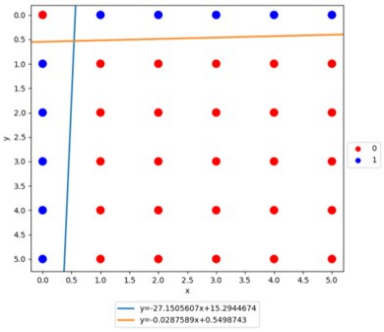
Figure A.1: Six node star topology problem with node 1 centered.

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix}$$

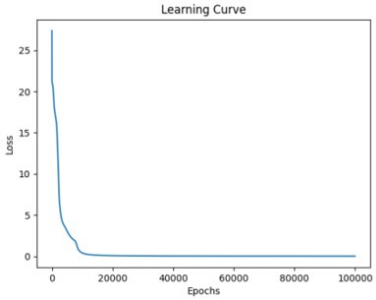
(a) Network Representation.



(b) Neural Network.



(c) Decision boundaries.

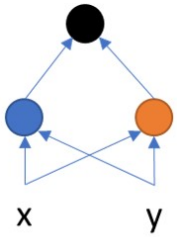


(d) Learning curve.

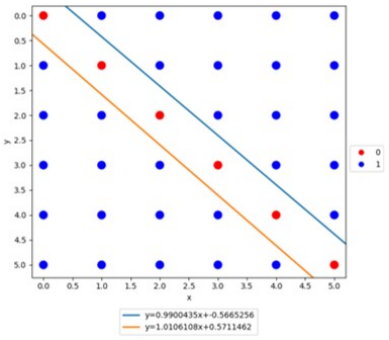
Figure A.2: Six node star topology problem with node 2 centered.

$$A = \begin{bmatrix} 0 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 \end{bmatrix}$$

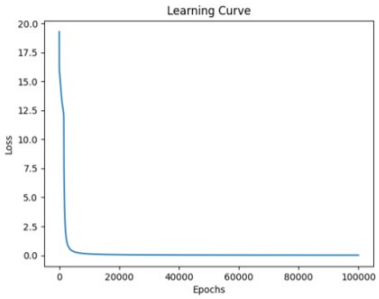
(a) Network Representation.



(b) Neural Network.

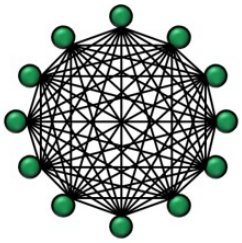


(c) Decision boundaries.



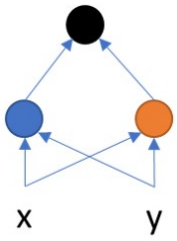
(d) Learning curve.

Figure A.3: Six node fully connected topology problem.

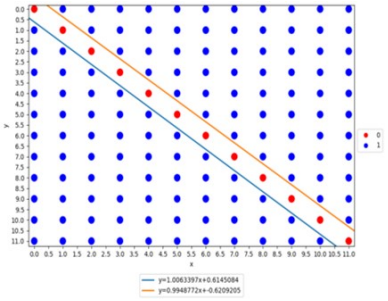


$$A = \begin{bmatrix} 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \end{bmatrix}$$

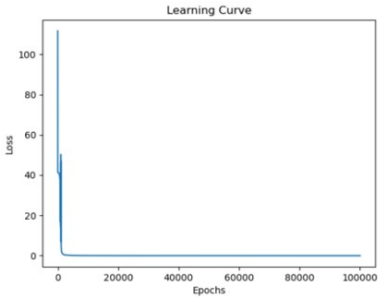
(a) Network Representation.



(b) Neural Network.

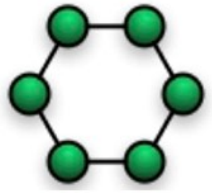


(c) Decision boundaries.



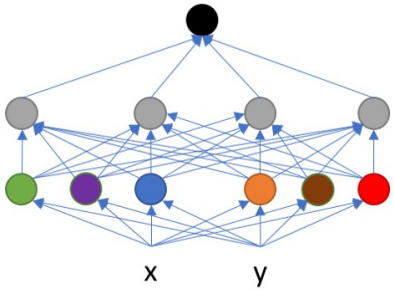
(d) Learning curve.

Figure A.4: Twelve node fully connected topology problem.

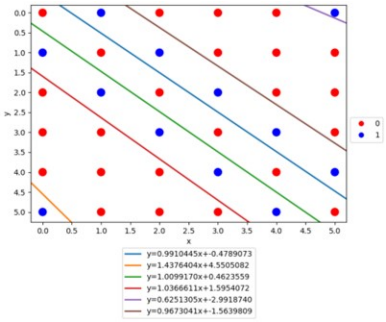


$$A = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

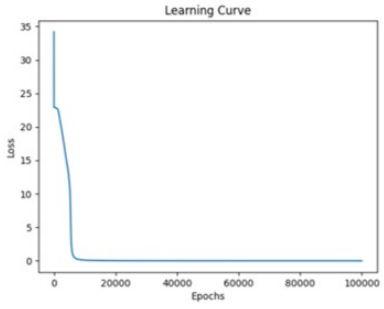
(a) Network Representation.



(b) Neural Network.

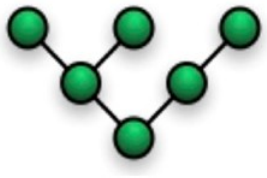


(c) Decision boundaries.



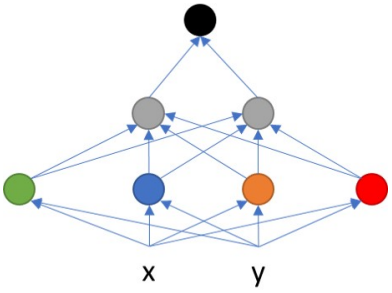
(d) Learning curve.

Figure A.5: Six node ring topology problem.

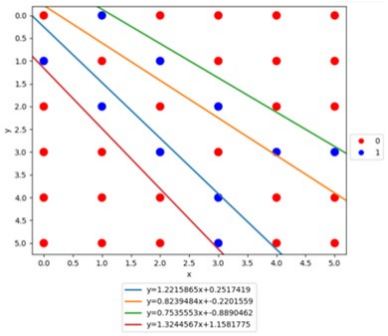


$$A = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

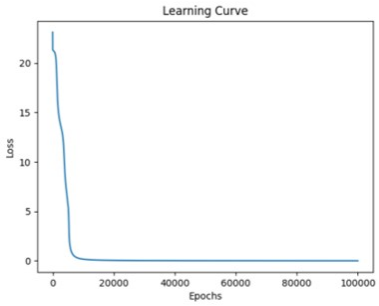
(a) Network Representation.



(b) Neural Network.



(c) Decision boundaries.

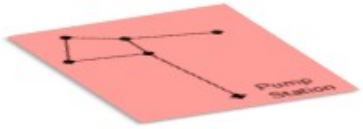


(d) Learning curve.

Figure A.6: Six node tree topology problem.

Appendix B: Water Distribution Multi-Layer Network Classification Results

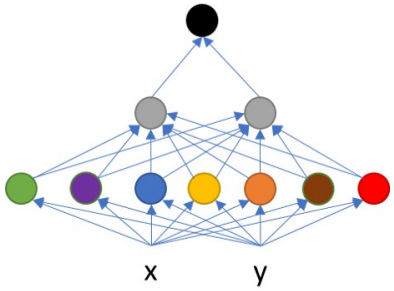
This Appendix includes classification results for the water distribution network presented in Chapter 3 using a multi-layer network approach.



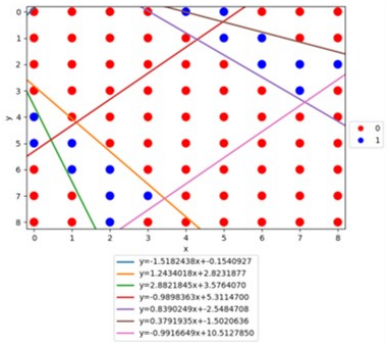
(a) Network Topology.

$$A = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

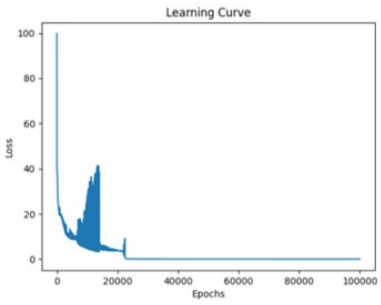
(b) Network Adjacency Matrix.



(c) Neural Network.

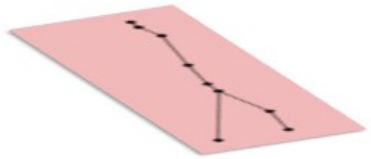


(d) Decision boundaries.



(e) Learning curve.

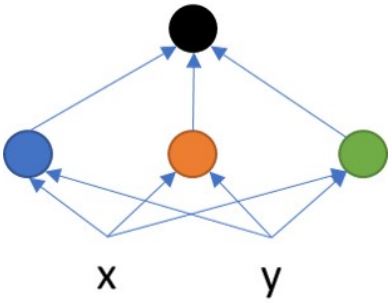
Figure B.1: Layer 1 of water distribution network topology problem.



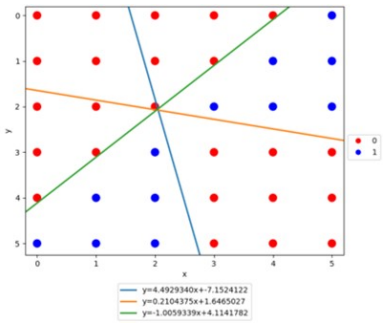
(a) Network Topology.

$$A = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \end{bmatrix}$$

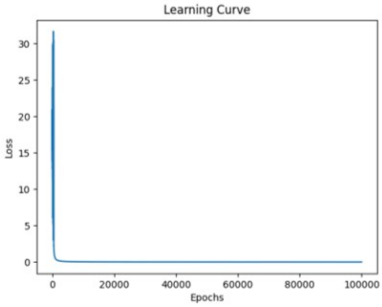
(b) Network Adjacency Matrix.



(c) Neural Network.

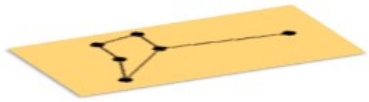


(d) Decision boundaries.



(e) Learning curve.

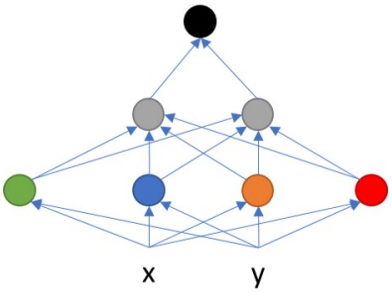
Figure B.2: Layer 2 of water distribution network topology problem.



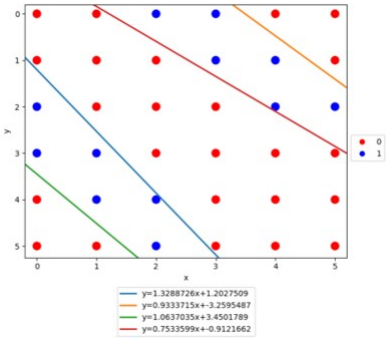
(a) Network Topology.

$$A = \begin{bmatrix} 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix}$$

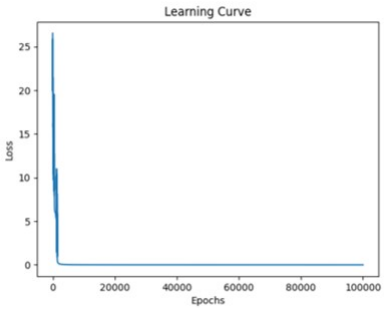
(b) Network Adjacency Matrix.



(c) Neural Network.

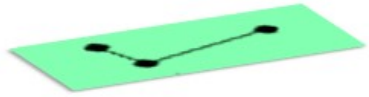


(d) Decision boundaries.



(e) Learning curve.

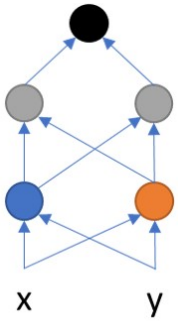
Figure B.3: Layer 3 of water distribution network topology problem.



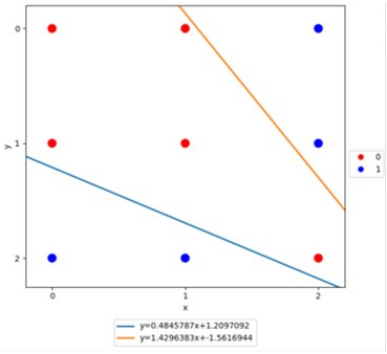
(a) Network Topology.

$$A = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix}$$

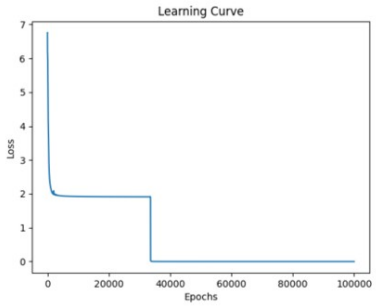
(b) Network Adjacency Matrix.



(c) Neural Network.

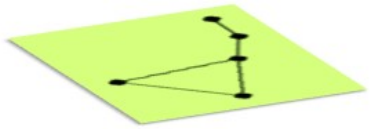


(d) Decision boundaries.



(e) Learning curve.

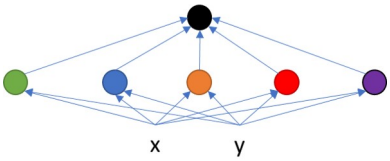
Figure B.4: Layer 4 of water distribution network topology problem.



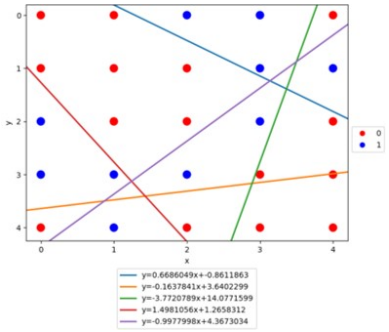
(a) Network Topology.

$$A = \begin{bmatrix} 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \end{bmatrix}$$

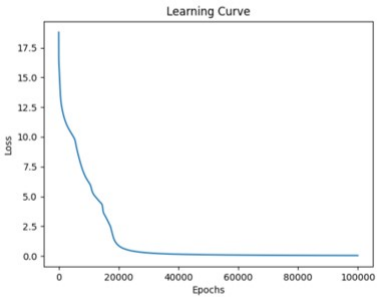
(b) Network Adjacency Matrix.



(c) Neural Network.

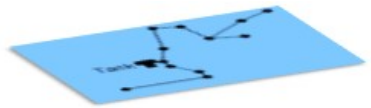


(d) Decision boundaries.



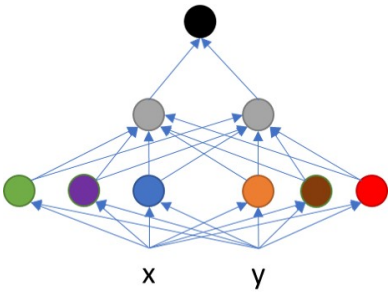
(e) Learning curve.

Figure B.5: Layer 5 of water distribution network topology problem.

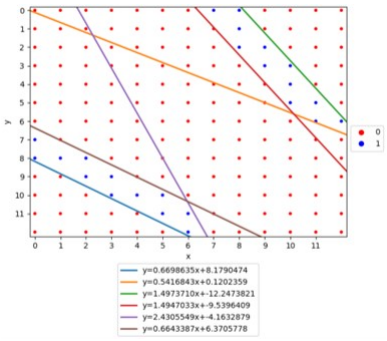


(a) Network Topology.

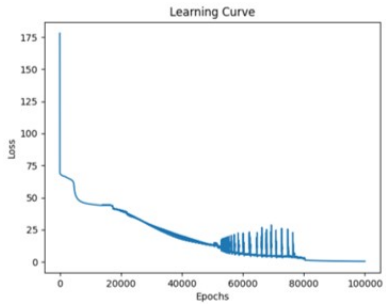
(b) Network Adjacency Matrix.



(c) Neural Network.



(d) Decision boundaries.



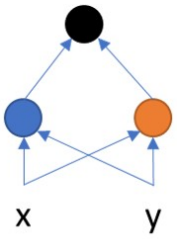
(e) Learning curve.

Figure B.6: Layer 6 of water distribution network topology problem.

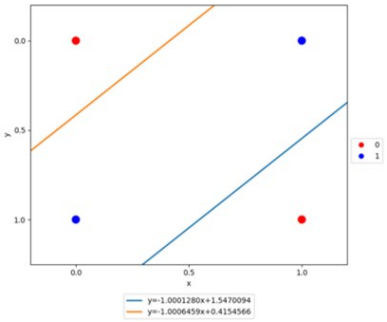
(a) Network Topology.

$$A = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

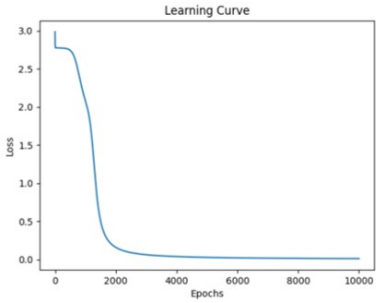
(b) Network Adjacency Matrix.



(c) Neural Network.



(d) Decision boundaries.



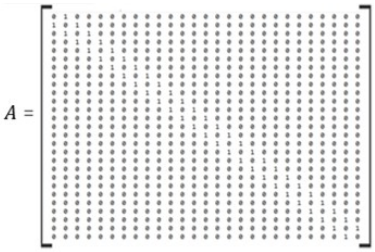
(e) Learning curve.

Figure B.7: Layer 7 of water distribution network topology problem.

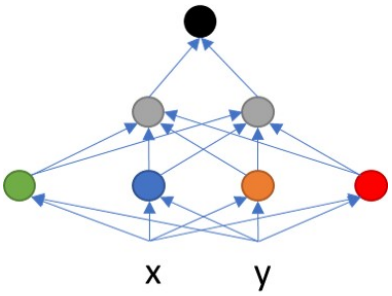
Appendix C: Metro Multi-Layer Network Classification Results

This Appendix includes classification results for the metro network presented in Chapter 3 using a multi-layer network approach.

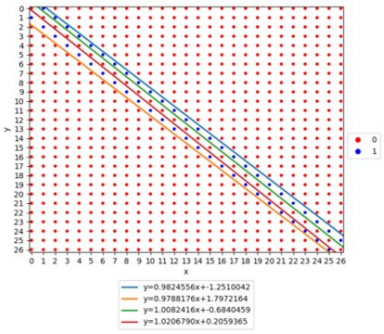
(a) Network Topology.



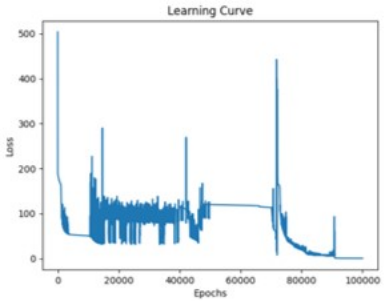
(b) Network Adjacency Matrix.



(c) Neural Network.



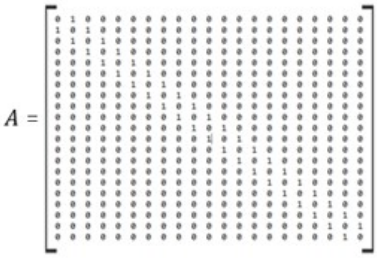
(d) Decision boundaries.



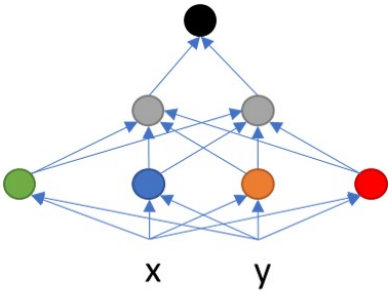
(e) Learning curve.

Figure C.1: Red line layer of metro network topology problem.

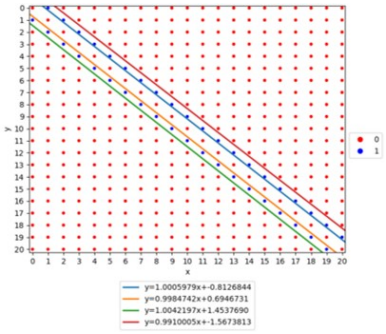
(a) Network Topology.



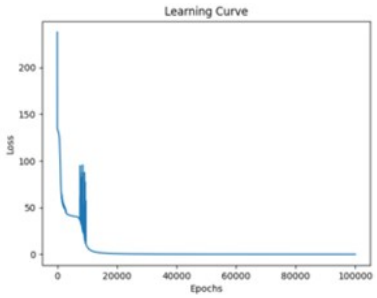
(b) Network Adjacency Matrix.



(c) Neural Network.



(d) Decision boundaries.

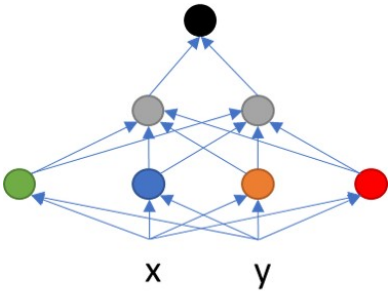


(e) Learning curve.

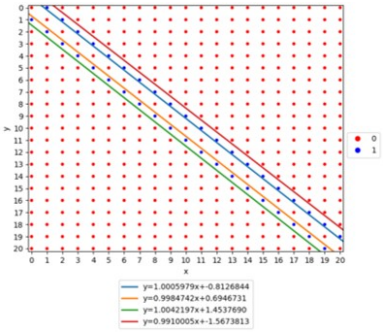
Figure C.2: Green line layer of metro network topology problem.

(a) Network Topology.

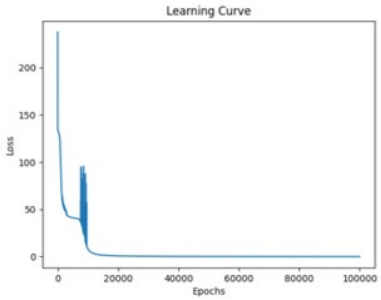
(b) Network Adjacency Matrix.



(c) Neural Network.



(d) Decision boundaries.

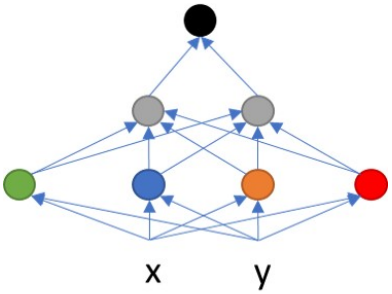


(e) Learning curve.

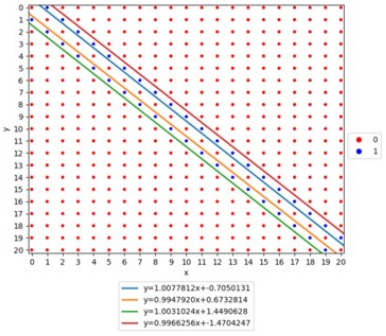
Figure C.3: Yellow line layer of metro network topology problem.

(a) Network Topology.

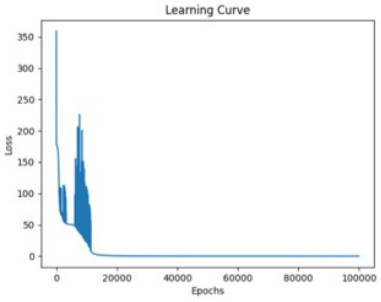
(b) Network Adjacency Matrix.



(c) Neural Network.



(d) Decision boundaries.

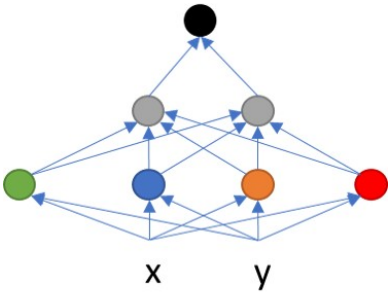


(e) Learning curve.

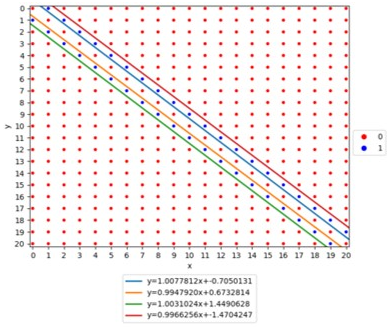
Figure C.4: Orange line layer of metro network topology problem.

(a) Network Topology.

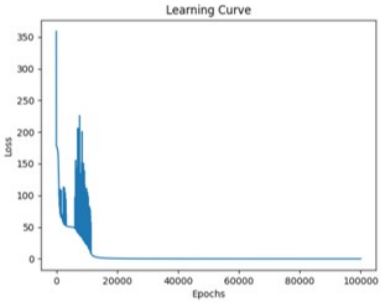
(b) Network Adjacency Matrix.



(c) Neural Network.



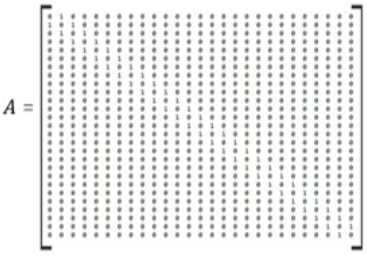
(d) Decision boundaries.



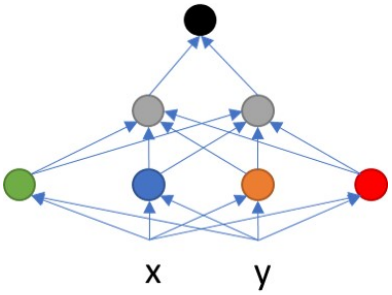
(e) Learning curve.

Figure C.5: Blue line layer of metro network topology problem.

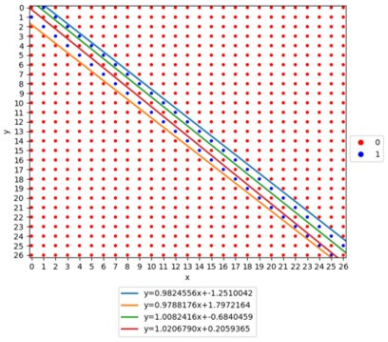
(a) Network Topology.



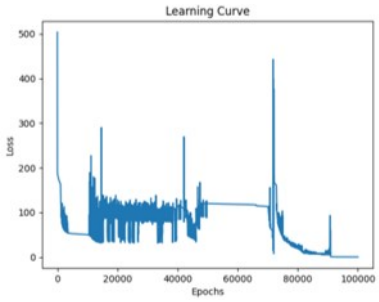
(b) Network Adjacency Matrix.



(c) Neural Network.



(d) Decision boundaries.



(e) Learning curve.

Figure C.6: Silver line layer of metro network topology problem.

Bibliography

[1] State of the Global Climate 2021: WMO Provisional Report. Tech. rep., World Meteorological Organization, 2021.

[2] Aleta A. and Moreno Y. Multilayer Networks in a Nutshell. *Annual Review of Condensed Matter Physics 10*, 1 (March 2019), 45–62.

[3] Apache Jena:. An Open Source Java framework for building Semantic Web and Linked Data Applications. For details, see https://jena.apache.org/, 2016.

[4] Austin M.A., Delgoshaei P., Coelho M., and Heidarinejad M. Architecting Smart City Digital Twins: A Combined Semantic Model and Machine Learning Approach. *ASCE Journal of Management in Engineering 36*, 4 (2020).

[5] Behrisch M., Benjamin B., Riche H., Tobias S., and Jean-Daniel F. Matrix Reordering Methods for Table and Network Visualization. *Computer Graphics Forum 35* (June 2016).

[6] Berners-Lee T., Hendler J., and Lassila O. The Semantic Web. *Scientific American* (May 2001), 35–43.

[7] Blackburn M.R., and Austin M.A. WRT 1025: Architecting for Digital Twins and MCE with AI/ML Part II. Tech. Rep. TR-007, Systems Engineering Research Center, 2021.

[8] Blackburn M.R., Austin M.A., and Coelho M. Modeling and Cross-Domain Dependability Analysis of Cyber-Physical Systems. *The 12th Annual IEEE International Systems Conference (SYSCON 2018)* (April 23-26 2018), 23–30.

[9] Blackburn M.R., Bone M., Dzielski J., Kruse B., Peak R., Cimtalay S., Ballard M., Baker A., Carnevale A., Stock W., Ramaswamy A., Szostak M., Rizzo G., Rouse W., Rhodes D., Austin M.A., and Coelho M. Transforming Systems Engineering through Model-Centric Engineering. Tech. Rep. TR-005, Systems Engineering Research Center, 2019.

[10] Bonino D. and Corno F. DogOnt - Ontology Modeling for Intelligent Domotic Environments. In *The Semantic Web - ISWC 2008* (Karlsruhe, Germany, October 26-30 2008), Springer Berlin Heidelberg, pp. 790–803.

[11] Breiman L. Random Forests. In *Machine Learning* (Norwell, MA, USA, October 2001), vol. 45, Kluwer Academic Publishers, pp. 5–32.

[12] Cao S., Lu W., and Xu Q. Deep Neural Networks for Learning Graph Representations. In AAAI.

[13] Coelho M. Distributed system behavior modeling of urban systems with ontologies, rules and message passing mechanisms. *M.S. Thesis, University of Maryland, College Park, MD 20742, USA* (2017).

[14] Coelho M. and Austin M.A. Teaching Machines to Understand Urban Networks. *The Fifteenth International Conference on Systems (ICONS 2020)* (February 23-27 2020), 37–42.

[15] Coelho M., Austin M.A., and Blackburn M.R. Distributed System Behavior Modeling of Urban Systems with Ontologies, Rules and Many-to-Many Association Relationships. *The Twelth International Conference on Systems (ICONS 2017)* (April 23-27 2017), 10–15.

[16] Coelho M., Austin M.A., and Blackburn M.R. Semantic Behavior Modeling and Event-Driven Reasoning for Urban System of Systems. *International Journal on Advances in Intelligent Systems 10*, 3 and 4 (December 2017), 365–382.

[17] Coelho M., Austin M.A., and Blackburn M.R. The Data-Ontology-Rule Footing: A Building Block for Knowledge-Based Development and Event-Driven Execution of Multi-Domain Systems. In *Systems Engineering in Context, Proceedings of the 16th Annual Conference on Systems Engineering Research* (2019), Springer, pp. 255–266.

[18] Coelho M., Austin M.A., Mishra S., and Blackburn M.R. Teaching Machines to Understand Urban Networks: A Graph Autoencoder Approach. *International Journal on Advances in Nteworks and Services 13*, 3 and 4 (December 2020), 70–81.

[19] Dehmamy N., Barabási A., and Yu R. Understanding the Representation Power of Graph Neural Networks in Learning Graph Topology. *CoRR abs/1907.05008* (2019).

[20] Delgoshaei P., Austin M.A., and Pertzborn A. A Semantic Framework for Modeling and Simulation of Cyber-physical Systems. *International Journal on Advances in Systems and Measurements 7*, 3-4 (December 2014), 223–238.

[21] Delgoshaei P., Austin M.A., and Veronica D.A. A Semantic Platform Infrastructure for Requirements Traceability and System Assessment. *The Ninth International Conference on Systems (ICONS 2014)* (February 2014), 215–219.

[22] Delgoshaei P., Heidarinejad M., and Austin M.A. Combined Ontology-Driven and Machine Learning Approach to Monitoring of Building Energy Consumption. In *2018 Building Performance Modeling Conference and SimBuild* (Chicago, IL, September 26-28 2018).

[23] Falquet G., Metral C., Teller J., and Tweed C. *Ontologies in Urban Development Projects*. Springer, 2005.

[24] Feigenbaum L., 2006. Semantic Web Technologies in the Enterprise.

[25] Fuller A., Fan Z., Day C., and Barlow C. Digital Twin: Enabling Technologies, Challenges and Open Research. *IEEE Access 8* (June 2020), 108952–108971.

[26] Glaessgen E.H. and Stargel D.S. The Digital Twin Paradigm for Future NASA and U. S. Air Force Vehicles. In *53rd AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics and Materials Conference* (2012).

[27] Goyal P. and Ferrara E. Graph Embedding Techniques, Applications, and Performance: A Survey. *CoRR abs/1705.02801* (2017).

[28] Hamilton W.L., Ying R., and Leskovec J. Representation Learning on Graphs: Methods and Applications. *CoRR abs/1709.05584* (2017).

[29] Hammoud Z., and Kramer F. Multilayer Networks: Aspects, Implementations, and Application in Biomedicine. *Big Data Analytics 5* (July 2020).

[30] Hendler J. Agents and The Semantic Web. *IEEE Intelligent Systems* (March/April 2001), 30–37.

[31] Hongyun C., Zheng V.W., and Chang K.C. A Comprehensive Survey of Graph Embedding: Problems, Techniques and Applications. *IEEE Transactions on Knowledge and Data Processing 30*, 9 (2018), 1616–1637.

[32] Kingma D. and Ba J. Adam: A Method for Stochastic Optimization. *International Conference on Learning Representations* (December 2014).

[33] Kipf T.N. and Welling M. Semi-Supervised Classification with Graph Convolutional Networks. *CoRR abs/1609.02907* (2016).

[34] Kipf T.N. and Welling M. Variational Graph Auto-Encoders. *ArXiv abs/1611.07308* (2016).

[35] Kitchin R. Data-Driven, Networked Urbanism. *SSRN Electronic Journal* (January 2015).

[36] Kivelä M. et al. Multilayer Networks. *Journal of Complex Networks 2*, 3 (July 2014), 203–271.

[37] Komninos N. et al. Towards High Impact Smart Cities: a Universal Architecture Based on Connected Intelligence Spaces. *Journal of the Knowledge Economy* (March 2021), 1–29.

[38] Kong J., Simonovic S., and Chao Z. Resilience Assessment of Interdependent Infrastructure Systems: A Case Study Based on Different Response Strategies. *Sustainability 11* (November 2019), 6552.

[39] LeCun Y., Bengio Y., and Hinton G. Deep learning. *Nature 521* (May 2015), 436–44.

[40] Lee J., Bagheri B., and Kao H. A Cyber-Physical Systems Architecture for Industry 4.0-based Manufacturing Systems. *Manufacturing Letters 3* (2015), 18–23.

[41] Lippmann, R. An Introduction to Computing with Neural Nets. *IEEE ASSP Magazine 4*, 2 (1987), 4–22.

[42] Liu J. et al. Artificial intelligence in the 21st century. *IEEE Access 6* (2018), 34403–34421.

[43] Mohammadi N. and Taylor J.E. Smart City Digital Twins, 2018 Global Smart Industry Conference (GloSIC). In *Proceedings of IEEE Symposium Series on Computational Intelligence* (January-February 2018), pp. 1–5.

[44] Negri E., Fumagalli L., and Macchi M. A Review of the Roles of Digital Twin in CPS-Based Production Systems. *Procedia Manufacturing 11* (2017), 939 – 948.

[45] Pang G., Shen C., Cao L., and Van Den Hengel A. Deep learning for anomaly detection: A review. *CoRR abs/2007.02500* (2020).

[46] Pfeiffer J.J., Moreno S., La Fond T., Neville J., and Gallagher B. Attributed Graph Models: Modeling Network Structure with Correlated Attributes. In *Proceedings of the 23rd International Conference on World Wide Web* (New York, NY, USA, 2014), ACM, pp. 831–842.

[47] Privat G., Coupaye T., Bolle S., and Raipin-Parvedy P. WoT Graph as Multiscale Digital-Twin for Cyber-Physical Systems-of-Systems. In *2019 International Conference on Artificial Intelligence and Computing Science (ICAICS 2019)* (Shanghai, China, 2019).

[48] Rinaldi S.M., Peerenboom J.M., and Kelly T.K. Identifying, Understanding, and Analyzing Critical Infrastructure Interdependencies. *IEEE Control Systems Magazine 21* (December 2001), 11–25.

[49] Rossman L.A. EPANET 2: Users Manual. Tech. Rep. EPA/600/R-00/057, Water Supply and Water Resources Division National Risk Management Research Laboratory, Cincinnati, OH, USA, September 2000.

[50] Saucedo J. Design and Development of Digital Twins: A Case Study in Supply Chains. *Mobile Networks and Applications* (2020), 1–20.

[51] Shahzad M., Shafiq M., Douglas D., and Kassem M. Digital Twins in Built Environments: An Investigation of the Characteristics, Applications, and Challenges. *Buildings 12* (January 2022), 120.

[52] Sharma A. et al. Digital Twins: State of the Art Theory and Practice, Challenges, and Open Research Questions. *CoRR abs/2011.02833* (2020).

[53] Steenwinckel B. *Adaptive Anomaly Detection and Root Cause Analysis by Fusing Semantics and Machine Learning*. August 2018, pp. 272–282.

[54] Strogatz S. Exploring Complex Networks. *Nature 410* (March 2001), 268–276.

[55] Tahmasebinia F., Fogerty D., Wu, L.O., Li Z., Sepasgozar S.M.E., Zhang K., Sepasgozar S., and Marroquin F.A. Numerical Analysis of the Creep and Shrinkage Experienced in the Sydney Opera House and the Rise of Digital Twin as Future Monitoring Technology. *Buildings 9*, 6 (2019).

[56] Wang D., Cui P., and Zhu W. Structural Deep Network Embedding. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (New York, NY, USA, 2016), Association for Computing Machinery, p. 1225–1234.

[57] Witten I.H., Frank E., Hall M.A., and Christopher J.P. *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kauffmann, 2017.

[58] Wu Z., Pan S., Chen F., Long G., Zhang C., and Yu P.S. A Comprehensive Survey on Graph Neural Networks. *CoRR abs/1901.00596* (2019).

[59] Xia F. et al. Graph learning: A survey. *IEEE Transactions on Artificial Intelligence 2*, 02 (April 2021), 109–127.

[60] Zhang H., Wang C., Lai J., and Yu P. Modularity in Complex Multilayer Networks with Multiple Aspects: A Static Perspective. *Applied Informatics 4* (05 2016).

[61] Zhang M., Li T., Yu Y., Li Y., Hui P., and Zheng Y. Urban Anomaly Analytics: Description, Detection, and Prediction. *CoRR abs/2004.12094* (2020).

[62] Zheng A. and Casari A. *Feature Engineering for Machine Learning*. O'Reilly Media, 2018.

[63] Zhou J., Cui G., Zhang Z., Yang C., Liu Z., and Sun M. Graph Neural Networks: A Review of Methods and Applications. *CoRR abs/1812.08434* (2018).