Self-Supervised Graph Structure Learning for Cyber-Physical Systems with Denoising Autoencoders

First Author[a;*], **Second Author**[b] **and Third Author**[b]

[a]Short Affiliation of First Author
[b]Short Affiliation of Second Author and Third Author
First Author https://orcid.org/....-....-....-...., Second Author https://orcid.org/....-....-....-....,
Third Author https://orcid.org/....-....-....-....

Abstract. Cyber-physical systems (CPS) often exhibit complex and dynamic behaviors that depend on the underlying connections between their sensors. Knowing these connections is crucial for understanding a system's normal behavior or detecting changes in its state. In this paper, we introduce **S**elf-**S**upervised Graph **S**tructure **L**earning for **C**yber-**P**hysical **S**ystems using denoising autoencoders, or CP4SL. Effectively learning the underlying graph structure is our primary goal. We propose a model consisting of a graph generator module and a denoising autoencoder module in the form of a graph neural network (GNN). Our approach addresses the challenge of inferring the structure of a CPS in terms of weighted connections between signals of multivariate time series. To account for temporal dependencies characteristic of CPS, we extend existing graph structure learning methods by providing custom graph convolutional layers that use dilated causal convolutions. In contrast to prior works that have explored GNNs for CPS focusing on task-specific metrics, we measure the quality of learned connections explicitly. We evaluate our model on data generated using the Kuramoto model of synchronizing coupled oscillators. Our results demonstrate that our approach offers an adaptive way to learn the structure of such systems. Finally, we apply our method to a real-world dataset and discuss limitations due to challenges such as lack of ground truth connections, heterogeneous graph structures, and stationary signals. The code is available on GitHub [10].

1 Introduction

Cyber-Physical Systems (CPS) are complex, interconnected systems that involve a broad array of sensors and devices, producing substantial amounts of time series data. These systems are integral to critical infrastructures, such as power grids, water treatment plants, transportation networks, and communication systems. Given the magnitude of their role, understanding and modeling the intricate relationships between the sensors within these systems is paramount to assure their optimal performance and security. Yet, the high dimensionality and complexity of the sensor data pose significant challenges for manual monitoring and modeling of sensor interconnections.

Furthermore, the graphs delineating these connections are often unavailable or indeterminable from raw data. Consequently, the need for effective, scalable, and automatic methods to learn and infer the structure of CPS from time series data has emerged. This paper addresses this gap by proposing a novel self-supervised graph structure learning method for CPS, utilizing denoising autoencoders [23]. Our approach aims to learn the underlying structure of CPS without the necessity for labeled data.

Specifically, we introduce a model comprised of a graph generator module and a denoising autoencoder module, organized within a graph neural network (GNN). These modules are trained end-to-end to minimize the reconstruction error, as portrayed in Figure 1. The reconstruction error serves as an indirect supervision signal, steering the graph generator to produce weighted edges that approximate the connections of the original structure from which the data is derived.

To the best of our knowledge, our method is the first that is applicable to the CPS setting in which we aim to discover a single graph that delineates the underlying structure using a high number of samples containing node features in the form of time series. Current research in graph structure learning primarily deals with a setting in which each adjacency matrix corresponds to a single set of node features. In contrast, our approach manages to extract structural information from a multitude of node feature samples, thus enhancing its applicability to CPS.

In an experimental study conducted on synthetically generated data based on the Kuramoto model [18], we measure our method's performance using reconstruction error and graph adjacency error as metrics. The findings underscore the efficiency of our approach in learning the structure of CPS and highlight its prospective applications in understanding complex systems or producing outputs that could inform downstream tasks.

This paper makes the following contributions:

1. We introduce two graph generator modules: a static version for learning a fixed structure and a dynamic version that generates graphs based on time series inputs.
2. We devise a custom GNN layer specially adapted for time series signals and the structure learning task.
3. We conduct a comprehensive experimental study on synthetically generated data, demonstrating the efficacy of our method.

The paper is structured as follows: Section 2 offers an overview of related work in graph representation learning linked to CPS. Section 3 explicates our proposed method in detail. Section 4 presents the experimental setup, results, and ensuing discussion and limitations. Lastly, Section 5 concludes the paper and outlines future work.

* Corresponding Author. Email: somename@university.edu.

2 Related Work

2.1 Graph neural networks

In recent years, graph neural networks (GNNs) have gained significant attention for their ability to model complex patterns in graph-structured data effectively. GNNs generally assume that the state of a node is influenced by the states of its neighbors, which allows them to capture the relationships between nodes in the graph [28, 20, 2, 8, 13]. Graph Convolution Networks (GCNs) [17, 8] model a node's feature representation by aggregating the representations of its one-step neighbors. This approach has been extended by numerous works including Graph Attention Networks (GATs) [22], which employ an attention function to compute different weights for different neighbors during the aggregation process. GNN-based models have demonstrated success in various problems, such as traffic prediction tasks [30] [3]), recommendation systems [26], and other multi-relational data applications [21]. However, conventional GNNs face certain limitations when applied to multivariate time series data from CPS. Specifically, they require a known graph structure as an input, while in the setting considered in this work, the graph structure is initially unknown and must therefore be learned from the data.

2.2 Graph structure learning

Lately, deep graph structure learning (GSL) has emerged as an active research domain [34], focusing on enhancing GNN performance for downstream tasks by improving a given graph adjacency matrix. These methods represent the adjacency using learnable parameters and jointly optimize it with the GNN under the guidance of a downstream task such as node classification [11, 31, 14, 24, 4]. To parameterize the adjacency matrix, different techniques including Bernoulli probability models [12] or stochastic block models [25] have been explored. Others methods make use of metric learning functions like cosine similarity [5] and dot product [9, 32] to calculate structures based on node-wise similarity. Alternatively each element in the adjacency matrix can be represented as a learnable parameter directly [9, 15]. Although deep GSL approaches have shown promise, they often depend on a supervised scenario, where node labels are necessary to refine a given graph structure.

2.3 Graphs and CPS

Graph-like structures are ubiquitous in CPS due to their modular design. Therefore, GSL has been used as an integral part of methods for anomaly detection for CPS in order to enable the use of GNNs. However, these methods use the graph structure as a means to enhance the performance of the GNN measured by a task-specific metric [7, 27]. In contrast, learning the underlying structure is the main focus of our method. In the absence of labels we propose an approach inspired by SLAPS [9].

3 Proposed method: CP4SL

3.1 Preliminaries

3.1.1 Problem statement

In this paper, we address the challenge of learning the underlying graph structure of cyber-physical systems (CPS) from multivariate time series data. The primary goal is to learn the correct structure without requiring labeled data or prior knowledge of the relationships between sensors.

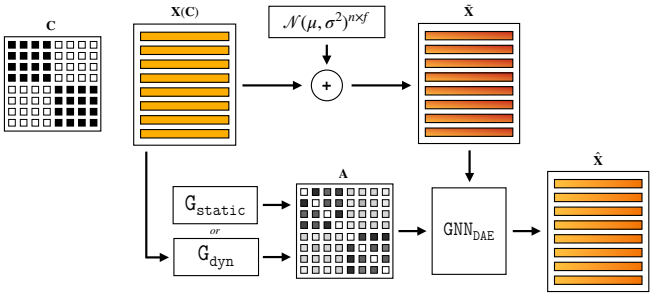


Figure 1: Overview of the CP4SL architecture. Given a noisy version of input features $\tilde{\mathbf{X}}$ that are generated from a system with an unknown underlying structure $\mathbf{C}$, a denoising autoencoder $\mathtt{GNN}_{\mathtt{DAE}}$ (Section 3.4) is trained to reconstruct the original features $\mathbf{X}$ using adjacency $\mathbf{A}$. Graph generators (Section 3.3) $\mathtt{G}_{\mathtt{static}}$ or $\mathtt{G}_{\mathtt{dyn}}$ are guided towards providing an adjacency $\mathbf{A}$ that resembles $\mathbf{C}$.

The problem can be formally defined as follows: We are given multivariate time series data, $\mathbf{X} = \mathbf{x_1}, ..., \mathbf{x_n}$, with n time series, each representing a sequence of readings from a sensor in the CPS. The dependencies between sensors are defined by a coupling matrix $\mathbf{C}$. Let $\mathbf{C} \in \{0, 1\}^{n \times n}$ be the binary coupling matrix where $\mathbf{C}_{ij} = 1$ if sensor i is related to $_j$, otherwise $\mathbf{C}_{ij} = 0$.

We treat the problem at hand as a graph structure learning problem in which the graph is defined as $\mathcal{G} = \{\mathcal{V}, \mathbf{A}, \mathbf{X}\}$, where $\mathcal{V} = \{v_1, ..., v_n\}$ is a set of nodes, $\mathbf{A} \in [0, 1]^{n \times n}$ is a weighted adjacency matrix with $\mathbf{A}_{ij}$ indicating the weight of the edge from v_i to v_j. Our objective is to train the model in such a way that the adjacency matrix generated by the graph generator module $\mathbf{A}$ matches the ground truth matrix $\mathbf{C}$.

To better understand the relationship between terminologies Table 1 highlights similarities of GNNs, the Kuramoto model and a CPS in the form of a tank system.

Table 1: Terminologies for GNNs, Kuramoto and tank systems.

	GNN	Kuramoto model	Tank system
Entity	Node	Oscillator	Tank
Signal	Features $\mathbf{x_i}$	Trajectory $\mathbf{x_i}$	Fill level $\mathbf{x_i}$
Connections	Edge	Coupling	Pipes
Weight	Weight $\mathbf{A_{ij}}$	Strength $\mathbf{C_{ij}}$	Flow rate
Matrix	Adjacency $\mathbf{A}$	Coupling $\mathbf{C}$	

3.1.2 Graph neural networks

GNNs use the graph structure and node features $\mathbf{x_i}$ to learn a representation vector of a node, $\mathbf{h_i}$. Representation of nodes are iteratively updated by aggregating representations of their neighbors. Following the notation of the Graph Isomorphism Network (GIN) [29], after k iterations of aggregation, a node's representation captures information from its k-hop network neighborhood. Formally, the k-th layer of a GNN is

$$a_i^{(k)} = AGGREGATE^{(k)}\left(\left\{h_j^{(k-1)} : u \in \mathcal{N}(v)\right\}\right), \quad (1)$$

$$h_i^{(k)} = COMBINE^{(k)}\left(h_i^{(k-1)}, a_i^{(k)}\right), \quad (2)$$

where $\mathbf{h_i^{(k)}}$ is the feature vector of node i at the k-th iteration/layer. In the first layer $\mathbf{h_i^{(0)}} = \mathbf{x_i}$, and $\mathcal{N}(i)$ is a set of nodes adjacent to node i.

3.2 Architectural choices

GNNs are built upon the smoothness assumption, which posits that connected nodes in a graph are likely to have similar features or labels [33]. Homophily is a fundamental concept in graph theory that refers to the tendency of nodes with similar attributes or features to be connected or form edges with one another. It provides the basis for the smoothness assumption and ensures that the aggregated information from neighboring nodes is relevant and consistent.

We make use of this property by choosing a GNN for our denoising autoencoder module. Since the GNN requires information to come from relevant neighboring nodes to reconstruct the original node features in the best possible way, the graph generator is incentivized to provide a graph with high homophily.

Given that inputs consist of time series data, we incorporate an additional inductive biases to help guide the learning process by applying Temporal Convolutional Networks (TCNs) [1] in both the (dynamic) graph generator and the denoising autoencoder module. TCN ensures the model respects the causal structure of the time series data by using causal convolutions, makes nearby time steps more relevant than distant ones, utilizes dilated convolutions to increase the model's receptive field and includes residual connections to facilitate the flow of gradients.

3.3 Graph generators

Given that the main goal of the method is to discover the underlying structure, the graph generator modules are core components which are trained to output such structure in the form of adjacency matrices. The generator is a function $\mathtt{G} : \mathbb{R}^{\mathtt{n}\times\mathtt{f}} \to \mathbb{R}^{\mathtt{n}\times\mathtt{n}}$ with parameters θ_G which takes the node features $\mathbf{X} \in \mathbb{R}^{n\times f}$ as input and produces a matrix $\tilde{\mathbf{A}} \in \mathbb{R}^{n\times n}$ as output. We apply an activation function to $\tilde{\mathbf{A}}$ before symmetrizing and normalizing it to obtain the adjacency matrix $\mathbf{A}$ which is used by the GNN layers of the denoising autoencoder. For the static generator we apply the exponential linear unit (ELU) to avoid gradient flow problems in case any edge becomes negative and for the dynamic generator we apply the sigmoid function. We consider the following two generators:

3.3.1 Static

For this generator, $\theta_{\mathbf{G}} \in \mathbb{R}^{n\times n}$ and the generator function is defined as $\tilde{\mathbf{A}} = \mathtt{G}_{\mathtt{static}}(\mathbf{X}, \theta_{\mathbf{G}}) = \theta_{\mathbf{G}}$. That is, the generator directly optimizes the fully parametrized adjacency matrix while ignoring the input node features. Consequently, the static generator provides a single matrix $\tilde{\mathbf{A}}_{\mathbf{static}}$ that is optimized on all samples of the training set. $\tilde{\mathbf{A}}_{\mathbf{static}}$ can be viewed as a fixed description of the connections within a CPS.

3.3.2 Dynamic

Depending on the system at hand and whether connections or connection strengths can be expected to change, it may be beneficial to generate an adjacency that, in contrast to the static one, does depend on input features. To enable this, we introduce a dynamic graph generator for which $\theta_{\mathbf{G}}$ corresponds to the weights of a TCN [1] and $\tilde{\mathbf{A}} = \mathtt{G}_{\mathtt{dyn}}(\mathbf{X}, \theta_{\mathbf{G}}) = \mathtt{TCN}(\mathbf{X}) \cdot \mathtt{TCN}(\mathbf{X})^T = \mathbf{X}' \cdot \mathbf{X}'^T$, where $\mathtt{TCN} : \mathbb{R}^{n\times f} \to \mathbb{R}^{n\times f'}$ produces a matrix with updated node representations $\mathbf{X}'$. Dot product is applied as a similarity measure that maps $\mathbb{R}^{n\times f'} \to \mathbb{R}^{n\times n}$. Note that TCN is applied to each univariate time series separately making $\mathtt{G}_{\mathtt{dyn}}(\mathbf{X}, \theta_{\mathbf{G}})$ permutation equivariant.

3.4 Graph denoising autoencoder

The task of the denoising autoencoder $\mathtt{GNN}_{\mathtt{DAE}} : \mathbb{R}^{n\times f} \times \mathbb{R}^{n\times n} \to \mathbb{R}^{n\times f}$ with parameters $\theta_{\mathbf{GNN_{DAE}}}$ is to take a noisy version $\tilde{\mathbf{X}}$ of the node features $\mathbf{X}$ and the generated adjacency matrix $\mathbf{A}$ as inputs and produce the updated denoised node features $\hat{\mathbf{X}}$ as output. We add independent Gaussian noise to $\mathbf{X}$ to obtain $\tilde{\mathbf{X}} = \mathbf{X} + \mathcal{N}(\mu, \sigma^2)^{n\times f}$.

During training, we minimize:

$$\mathcal{L}_{\mathcal{DAE}} = \mathtt{L}(X, \mathtt{GNN}_{\mathtt{DAE}}(\tilde{\mathbf{X}}, \mathbf{A}; \theta_{\mathbf{GNN_{DAE}}})) \tag{3}$$

where $\mathbf{A}$ is the generated adjacency matrix and L is a loss function, in this case the mean-squared error loss.

3.4.1 Graph TCN layers

We design a custom graph layer with similarities to the popular GIN layers [29] (Equation 4):

$$h_i^{(k)} = \mathtt{MLP}^{(k)}\left(\left(1+\epsilon^{(k)}\right)\cdot h_i^{(k-1)} + \sum_{j\in\mathcal{N}(i)} h_j^{(k-1)}\right) \tag{4}$$

However, connections are weighted, we use a TCN [1] instead of an MLP and only apply it to the representations of neighboring nodes, excluding self-loops. We use residual connections for all layers except the output layer to allow for a deeper network while also forcing the GNN to use $\mathbf{A}$. Note that TCN parameters are shared within a layer, i.e. the same TCN is applied to each aggregated node representation of that layer. Hence, the AGGREGATE step in Equation 1 becomes a weighted sum

$$a_v^{(k)} = \sum_{j\in\mathcal{N}(i)} \mathbf{A}_{ij} \cdot \mathbf{h}_j^{(k-1)} \tag{5}$$

and the COMBINE step from Equation 2 for a graph TCN layer reads as

$$h_i^{(k)} = \mathbf{h}_i^{(k-1)} + \sigma\left(\mathtt{TCN}\left(a_v^{(k)}\right)\right) \tag{6}$$

so that the entire layer can be described by

$$\mathbf{h}_i^{(k)} = \mathbf{h}_i^{(k-1)} + \sigma\left(\mathtt{TCN}\left(\sum_{j\in\mathcal{N}(i)} \mathbf{A}_{ij}\cdot\mathbf{h}_j^{(k-1)}\right)\right). \tag{7}$$

For our use case the node representations are noisy features so that $\mathbf{h}_i^{(k)} = \tilde{\mathbf{x}}^{(k)}$. Rewritten in matrix form the hidden layers are defined by

$$\tilde{\mathbf{X}}^{(k)} = \mathbf{I}\cdot\tilde{\mathbf{X}}^{(k-1)} + \sigma\left(\mathtt{TCN}\left(\mathbf{A}\cdot\tilde{\mathbf{X}}^{(k-1)}\right)\right), \tag{8}$$

while the output layer without residual connection is defined as

$$\hat{\mathbf{X}}^{(k)} = \mathtt{TCN}\left(\mathbf{A}\cdot\tilde{\mathbf{X}}^{(k-1)}\right) \tag{9}$$

forcing the model to optimize $\mathbf{A}$.

4 Experiments

To demonstrate the effectiveness of the proposed method we conduct empirical experiments. We aim to answer three research questions:

- **RQ1:** How well can the method infer the underlying structure?
- **RQ2:** What are the characteristics of the static and the dynamic graph generator?
- **RQ3:** How do the generators adapt to changes in the structure?

4.1 Metrics

4.1.1 Adjacency error

Since the goal is inferring the structure of the underlying system, when evaluating the performance of the model we are mainly interested in the agreement between the ground truth coupling matrix $\mathbf{C}$ and the inferred adjacency matrix $\mathbf{A}$. Since self-loops are ruled out in order to force the $\texttt{GNN}_{\texttt{DAE}}$ to make use of informative nodes, entries of the matrix diagonals are masked out and hence also not considered for the evaluation of $\mathbf{A}$. To adjust for differences in scale, $\mathbf{A}$ is rescaled to range $[0, 1]$ before calculating the adjacency error, $e_{adj} = \texttt{MAE}(\mathbf{A}, \mathbf{C})$.

4.1.2 Reconstruction error

Even though minimizing the reconstruction error is merely used as a means for the model to learn the structure of the underlying system, it is still a metric worth tracking since it could be used for downstream tasks such as anomaly detection. Same as for the loss, the reconstruction error e_{rec} is calculated as the mean squared error between the input signal $\mathbf{X}$ and the denoised output $\hat{\mathbf{X}}$, $e_{rec} = \texttt{MSE}(\hat{\mathbf{X}}, \mathbf{X})$.

4.2 Datasets

We generate our synthetic dataset using simulations of phase-coupled oscillators (Kuramoto model [18]). The Kuramoto model is a nonlinear system of phase-coupled oscillators that can exhibit a range of complicated dynamics based on the distribution of the oscillators' internal frequencies and their coupling strengths. We use the common form for the Kuramoto model given by the following differential equation:

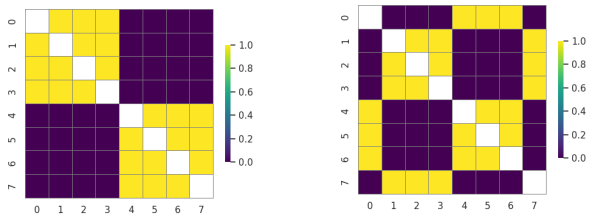
$$\frac{d\phi_i}{dt} = \omega_i + \sum_{j \neq i} k_{ij} \sin\left(\phi_i - \phi_j\right)$$

with phases ϕ_i, coupling constants k_{ij}, and intrinsic frequencies ω_i. We simulate 8 phase-coupled oscillators in 1D with intrinsic frequencies $\omega_i \sim \mathcal{N}(\mu, \sigma)^{n \times d}$ and initial phases $\phi_i^{t=1}$ uniformly sampled from $[0, 2\pi)$. We create two subsystems of connected oscillators by coupling all of the first four as well as all of the last four oscillators.

We connect pairs of oscillators v_i and v_j by setting the corresponding coupling constant $k_{ij} = k_{ji} = k$. All other coupling constants are set to 0. The resulting matrix $\mathbf{K}$ describes the underlying structure of the system of coupled oscillators and can therefore be treated as the ground truth coupling matrix $\mathbf{C}$ when using our proposed method. To generate samples for the training and validation datasets the normal coupling matrix $\mathbf{C}$ (Figure 2a) is used, whereas for the test set the shuffled coupling matrix $\hat{\mathbf{C}}$ (Figure 2b) is used to analyze the effect of changes in the underlying structure on the model outputs.

The model is built to denoise multivariate time series with T time steps. We run the simulation for $2T$ time steps so that with each run of the simulation we obtain a signal matrix $\mathbf{S} \in \mathbb{R}^{n \times 2T}$. Samples of the training set comprise the first T time steps whereas validation and test set include the last T time steps of the corresponding simulation runs. See Figure 3 for a visualization of a training sample. Additional visualizations for validation and test data are provided in the supplementary material (Figure 9).

Parameters of the Kuramoto model such as intrinsic frequencies and coupling constants are chosen in such a way that within the first T time steps, depending on the initial phases, the oscillators are generally not yet fully synchronized making the training task non-trivial. On the other hand, validation and test data cover a time span in which the system is already largely synchronized. This makes the reconstruction even more dependent on a correct adjacency matrix $\mathbf{A}$. For our experiments we use the kuramoto package [6] to run simulations utilizing five different seeds to account for randomness. For each seed we generate 400 training samples as well as 100 validation and 100 test samples, respectively. For details of the data generation process please refer to the provided code.



(a) Normal $\mathbf{C}$ (b) Shuffled $\hat{\mathbf{C}}$

Figure 2: Coupling matrices defining the underlying structure from which data is generated

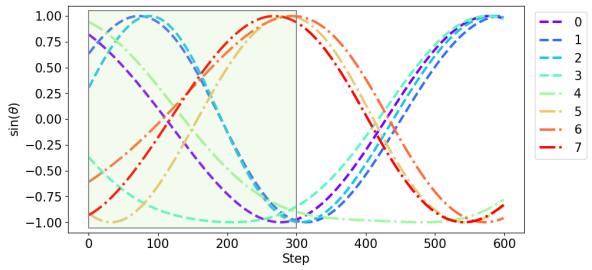


Figure 3: Visualization of a training sample $\mathbf{X}_{\mathbf{train}}$ (green box) as part of a signal matrix $\mathbf{S}$. Nodes 0-3 and nodes 4-7 synchronize due to coupling $\mathbf{C}$ as shown in Figure 2a.

4.3 Experimental Setup

We train models using either the static or the dynamic graph generator along with the same denoising autoencoder module. Since both training and validation set are generated using $\mathbf{C}$, $e_{adj_{val}}$ indicates whether the model is learning the correct structure. In contrast, as the test set is generated using the shuffled coupling matrix $\hat{\mathbf{C}}$, $e_{adj_{test}}$ is a measure indicating how the graph generator adapts to changes in the underlying structure. Similarly, $e_{rec_{val}}$ measures the reconstruction error achieved when the underlying structure is identical to that of the training data, while $e_{rec_{test}}$ shows to which extent the model is still able to denoise/reconstruct the input as the underlying structure is changed.

4.4 Baselines

- **True** assumes that the underlying structure is known. $\mathbf{A}_{\mathbf{true}} = \mathbf{C}$ replaces the output of the graph generator.
- **Correlations** replaces the output of the graph generator with $\mathbf{A}_{\mathbf{corr}}$, a correlation matrix calculated by averaging the Pearson correlation over all samples of the training set.
- **Full** replaces the output of the graph generator with $\mathbf{A}_{\mathbf{full}}$ in which all connections are set to 1.

- **Features** ignores the TCN in the dynamic graph generator to calculate similarity between nodes directly from the untransformed features.

4.4.1 *Implementation details*

We implemented our model in PyTorch [19] and used the kuramoto package [6] to run simulations utilizing five different seeds to account for randomness. We trained the model for 50 epochs using the Adam [16] optimizer with a learning rate of 0.001, and a batch size of 16. We identified suitable hyperparameters [layers: 2, dropout: 0.5] for the denoising autoencoder based on the reconstruction error on the validation set. For the TCN proven parameters were chosen based on the original publication [1]. Details can be found in the configuration files and code provided which will also be made publicly available after the review process. We ran all experiments on a n1-standard-8 instance with 8 vCPUs, 30 GB of memory and a single Tesla T4 GPU.

4.5 *Results*

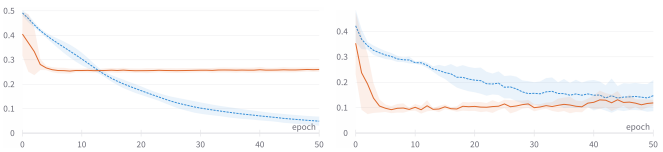
4.5.1 *Learning underlying structure (RQ1)*

Validation results given in Table 2 show that the static generator improves upon the Correlation baseline in terms of adjacency error and yields results comparable to True and Correlation baselines with respect to reconstruction error. On the other hand the dynamic generator shows the best results with respect to reconstruction error but poorer adjacency error.

Table 2: Validation results given as mean $\pm$ standard deviation for five different seeds. Validation data is generated from $\mathbf{C}$.

Configuration	Adjacency	Reconstruction
True	0.000 ± 0.000	0.138 ± 0.067
Static	$\mathbf{0.050 \pm 0.018}$	0.144 ± 0.058
Correlation	0.119 ± 0.027	0.139 ± 0.057
Dynamic	0.260 ± 0.009	$\mathbf{0.117 \pm 0.014}$
Full	0.500 ± 0.000	0.244 ± 0.106
Features	0.457 ± 0.002	0.240 ± 0.034

To get a better understanding, Figure 4 shows the mean and standard deviation of the validation error for the static as well as for the dynamic generator model evolving during the training process. The adjacency error in Figure 4a displays a a continuous gradual decrease for the static generator while the dynamic generator curve is characterized by a sudden drop before the adjacency error stays constant.



(a) Adjacency (b) Reconstruction

Figure 4: Mean $\pm$ standard deviation of validation error over epochs

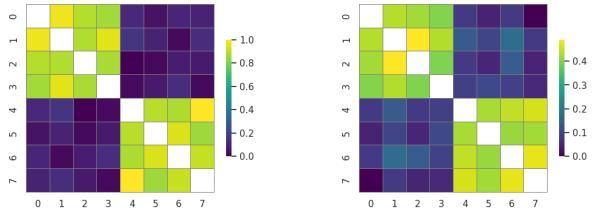
The mean of the validation reconstruction error for the static generator in Figure 4b is steadily decreasing as the adjacency error is decreasing but its variation increases. Similarly, for the dynamic generator the adjacency error correlates to the reconstruction error. Overfitting starts as soon as the graph generator stops improving.

The correlation between adjacency error and reconstruction error shows that learning the correct structure is crucial for the denoising module. Both models converge towards the correct underlying structure.

4.5.2 *Generator characteristics (RQ2)*

Why does the lower adjacency error of the static generator not result in a lower reconstruction error? To understand the different behavior of the two generators we need to consider the characteristics of the two generator types. While the static generator learns a set of parameters that best describe the system on average, the dynamic generator can and will adjust the adjacency to the inputs.

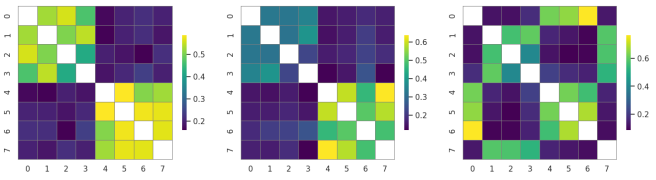
Figure 5 shows the adjacency learned by the static generator alongside the correlations averaged over all samples from the same training dataset. Both represent the average relations of nodes within a single matrix. Consequently, randomness of individual samples cannot be represented. As our results show, this property is advantageous given the goal is to identify the underlying structure.



(a) Static (b) Correlations

Figure 5: Examples of adjacency matrices generated by the static generator (a) as well as a Pearson correlation baseline (b)

On the other hand, the dynamic generator learns to provide adjacencies that depend on the transformed inputs and are optimally suited for the denoising module. More specifically, in our Kuramoto experiments, edge weights are low if two oscillators are not in sync even though they are coupled. This is caused by randomness, i.e. random initial phase of the oscillators. To illustrate this effect, Figure 6a shows the adjacency generated for a sample with highly synchronized coupled nodes which looks almost identical to the static adjacency (5a). In contrast, if the coupled nodes are not synchronized as is the case in Figure 6b, exchanging information between these nodes is less beneficial for the denoising module resulting in lower edge weights.



(a) Dynamic (val) (b) Dynamic (val) (c) Dynamic (test)

Figure 6: Examples of adjacency matrices generated by the dynamic generator for two validation samples (a-b) and a test sample (c)

This property of the dynamic generator improves reconstruction results but increases the adjacency error which cannot be lower than

a value that is determined by the amount of randomness within the data.

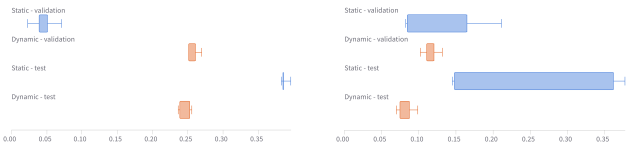
4.5.3 *Effect of changing underlying structure (RQ3)*

As described in Section 4.2, training and validation datasets are generated using the normal coupling matrix $\mathbf{C}$ while for the test set the shuffled version $\hat{\mathbf{C}}$ is used to simulate a perturbed system and examine the effect on the model outputs. Hence, while $e_{adj_{val}} = MAE(\mathbf{A}, \mathbf{C})$ tells us how well the model has learnt the underlying structure of the normal system, $e_{adj_{test}} = MAE(\mathbf{A}, \hat{\mathbf{C}})$ indicates the deviation between the adjacencies generated for the test set and the modified system. Similarly, by comparing $e_{adj_{val}}$ to $e_{adj_{test}}$ we can measure to which extent the model is still able to denoise the signals even for the test system with modified structure.

Table 3: Test results given as mean $\pm$ standard deviation for five different seeds. Test data is generated from $\hat{\mathbf{C}}$.

Configuration	Adjacency	Reconstruction
True	0.375 ± 0.000	0.269 ± 0.140
Static	0.388 ± 0.005	0.282 ± 0.123
Correlation	0.407 ± 0.015	0.270 ± 0.131
Dynamic	$\mathbf{0.248 \pm 0.009}$	$\mathbf{0.084 \pm 0.012}$
Full	0.500 ± 0.000	0.240 ± 0.115
Features	0.454 ± 0.002	0.247 ± 0.033

Table 3 and Figure 7 show results on the test set generated from $\hat{\mathbf{C}}$. Since the output of the static generator remains fixed after the training process (Figure 5a), e_{adj} increases significantly. In contrast, the outputs of the dynamic generator are permutation equivariant and only depend on the individual inputs. This allows the dynamic generator to adpapt to the change in structure (see Figure 6c) leading to similar performance for both validation and test data.



(a) Adjacency (b) Reconstruction

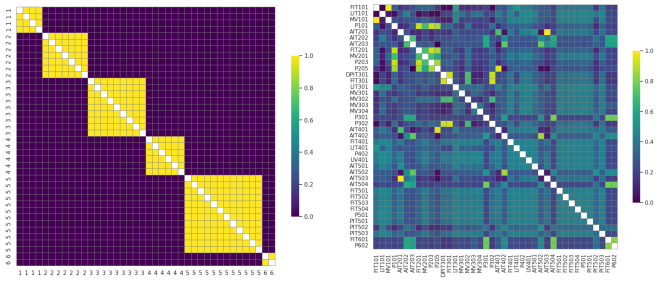
Figure 7: Mean $\pm$ standard deviation of test error

Again, we observe that the reconstruction error depends on the adjacency error. The adjacency matrix generated by the static does not match the underlying structure of the test data, leading to a higher reconstruction error. In contrast, the dynamic generator adapts to the modified structure allowing the model to denoise the data well for both validation and test data.

4.5.4 *Limitations in real-world scenarios*

We apply our method to the Secure Water Treatment (SWaT) dataset, a collection of data gathered from a scaled-down version of a real water treatment plant. The dataset includes data from sensors, actuators, and programmable logic controllers. Our model was trained on data collected over seven days of normal operation, excluding signals that remained constant throughout that time period.

Our results, as shown in Figure 8b, partially resemble the block structure of the processes, and the adjacency errors (static: 0.36, dynamic: 0.29) converge significantly below a full graph baseline.



(a) SWaT processes (b) SWaT static

Figure 8: SWaT structure if defined by processes (a) and results using static generator (b)

However, compared to our experiments on the Kuramoto datasets, the SWaT dataset presents several complex challenges, which impacts the overall effectiveness of our model.

Firstly, the lack of obvious choices for ground truth connections in real-world data can lead to a degree of uncertainty in our model's ability to accurately learn the true underlying structure. This uncertainty can be compounded by the heterogeneity of the system, as different sensor types imply a heterogeneous graph. The model must learn to treat nodes differently, adding another layer of complexity.

Secondly, the presence of categorical or binary signals in addition to continuous ones can lead to difficulties in signal processing and interpretation. This issue necessitates careful preprocessing of the data to ensure that all types of signals can be effectively used by our model.

Finally, signals are often stationary in the real-world datasets, which may cause a lack of guidance for the model and potentially result in sub-optimal learning of the graph structure. This issue can affect the ability of our model to adapt and learn effectively in more stationary environments.

Given these challenges, the performance of our model on the SWaT dataset should be considered as an initial exploration into applying our method to real-world data. Further research is required to better handle the complexity and heterogeneity of real-world CPS, and to improve the robustness and generalizability of our model.

5 Conclusion

In this paper, we proposed CP4SL, a method for learning the graph structure of a cyber-physical system (CPS) with the help of a self-supervised denoising task. We introduced two graph generator modules, each with different characteristics and potential use cases. We also showed that the method can be used to analyze changes in the underlying structure of a CPS. The adjacency matrix learned by the static generator can be used as a representation of the interdependencies between the components of a CPS in normal state, which could serve as a starting point for creating ontologies. Additionally, comparing the static adjacency matrix to the dynamically updated adjacency may enable performing anomaly detection directly on the graph structures. While our approach shows promising results, future research could improve upon it by categorizing the challenges posed by the characteristics of different types of CPS and proposing customized solutions.

Acknowledgements

Kept anonymous for review

References

[1] Shaojie Bai, J Zico Kolter, and Vladlen Koltun, 'An empirical evaluation of generic convolutional and recurrent networks for sequence modeling', *arXiv preprint arXiv:1803.01271*, (2018).

[2] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun, 'Spectral networks and locally connected networks on graphs', *arXiv preprint arXiv:1312.6203*, (2013).

[3] Weiqi Chen, Ling Chen, Yu Xie, Wei Cao, Yusong Gao, and Xiaojie Feng, 'Multi-range attentive bicomponent graph convolutional network for traffic forecasting', in *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pp. 3529–3536, (2020).

[4] Yu Chen, Lingfei Wu, and Mohammed Zaki, 'Iterative deep graph learning for graph neural networks: Better and robust node embeddings', *Advances in neural information processing systems*, **33**, 19314–19326, (2020).

[5] Yu Chen, Lingfei Wu, and Mohammed Zaki, 'Iterative deep graph learning for graph neural networks: Better and robust node embeddings', *Advances in neural information processing systems*, **33**, 19314–19326, (2020).

[6] Fabrizio Damicelli. Python implementation of the kuramoto model. https://github.com/fabridamicelli/kuramoto, 2019.

[7] Ailin Deng and Bryan Hooi, 'Graph neural network-based anomaly detection in multivariate time series', in *Proceedings of the AAAI conference on artificial intelligence*, volume 35, pp. 4027–4035, (2021).

[8] David K Duvenaud, Dougal Maclaurin, Jorge Iparraguirre, Rafael Bombarell, Timothy Hirzel, Alán Aspuru-Guzik, and Ryan P Adams, 'Convolutional networks on graphs for learning molecular fingerprints', *Advances in neural information processing systems*, **28**, (2015).

[9] Bahare Fatemi, Layla El Asri, and Seyed Mehran Kazemi, 'Slaps: Self-supervision improves structure learning for graph neural networks', *Advances in Neural Information Processing Systems*, **34**, 22667–22681, (2021).

[10] Anonymous For Review. Self-supervised structure learning for cyber-physical systems. https://github.com/alemamm/cp4sl-lightning, 2023.

[11] Luca Franceschi, Mathias Niepert, Massimiliano Pontil, and Xiao He, 'Learning discrete structures for graph neural networks', in *International conference on machine learning*, pp. 1972–1982. PMLR, (2019).

[12] Luca Franceschi, Mathias Niepert, Massimiliano Pontil, and Xiao He, 'Learning discrete structures for graph neural networks', in *Proceedings of the 36th International Conference on Machine Learning*, eds., Kamalika Chaudhuri and Ruslan Salakhutdinov, volume 97 of *Proceedings of Machine Learning Research*, pp. 1972–1982. PMLR, (2019).

[13] Will Hamilton, Zhitao Ying, and Jure Leskovec, 'Inductive representation learning on large graphs', *Advances in neural information processing systems*, **30**, (2017).

[14] Wei Jin, Yao Ma, Xiaorui Liu, Xianfeng Tang, Suhang Wang, and Jiliang Tang, 'Graph structure learning for robust graph neural networks', in *Proceedings of the 26th ACM SIGKDD international conference on knowledge discovery & data mining*, pp. 66–74, (2020).

[15] Wei Jin, Yao Ma, Xiaorui Liu, Xianfeng Tang, Suhang Wang, and Jiliang Tang, 'Graph structure learning for robust graph neural networks', in *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '20, pp. 66–74, New York, NY, USA, (August 2020). Association for Computing Machinery.

[16] Diederik P Kingma and Jimmy Ba, 'Adam: A method for stochastic optimization', *arXiv preprint arXiv:1412.6980*, (2014).

[17] Thomas N Kipf and Max Welling, 'Semi-supervised classification with graph convolutional networks', *arXiv preprint arXiv:1609.02907*, (2016).

[18] Yoshiki Kuramoto, 'Self-entrainment of a population of coupled non-linear oscillators', in *International Symposium on Mathematical Problems in Theoretical Physics*, pp. 420–422. Springer Berlin Heidelberg, (1975).

[19] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al., 'Pytorch: An imperative style, high-performance deep learning library', *Advances in neural information processing systems*, **32**, (2019).

[20] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini, 'The graph neural network model', *IEEE transactions on neural networks*, **20**(1), 61–80, (2008).

[21] Michael Schlichtkrull, Thomas N Kipf, Peter Bloem, Rianne van den Berg, Ivan Titov, and Max Welling, 'Modeling relational data with graph convolutional networks', in *The Semantic Web*, pp. 593–607. Springer International Publishing, (2018).

[22] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio, 'Graph attention networks', *arXiv preprint arXiv:1710.10903*, (2017).

[23] Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol, 'Extracting and composing robust features with denoising autoencoders', in *Proceedings of the 25th international conference on Machine learning*, ICML '08, pp. 1096–1103, New York, NY, USA, (July 2008). Association for Computing Machinery.

[24] Ruijia Wang, Shuai Mou, Xiao Wang, Wanpeng Xiao, Qi Ju, Chuan Shi, and Xing Xie, 'Graph structure estimation neural networks', in *Proceedings of the Web Conference 2021*, pp. 342–353, (2021).

[25] Ruijia Wang, Shuai Mou, Xiao Wang, Wanpeng Xiao, Qi Ju, Chuan Shi, and Xing Xie, 'Graph structure estimation neural networks', in *Proceedings of the Web Conference 2021*, WWW '21, pp. 342–353, New York, NY, USA, (June 2021). Association for Computing Machinery.

[26] Shiwen Wu, Fei Sun, Wentao Zhang, Xu Xie, and Bin Cui, 'Graph neural networks in recommender systems: A survey', *ACM Comput. Surv.*, **55**(5), 1–37, (December 2022).

[27] Weiqiang Wu, Chunyue Song, Jun Zhao, and Zuhua Xu, 'Physics-informed gated recurrent graph attention unit network for anomaly detection in industrial cyber-physical systems', *Information Sciences*, **629**, 618–633, (2023).

[28] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and S Yu Philip, 'A comprehensive survey on graph neural networks', *IEEE transactions on neural networks and learning systems*, **32**(1), 4–24, (2020).

[29] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka, 'How powerful are graph neural networks?', *arXiv preprint arXiv:1810.00826*, (2018).

[30] Bing Yu, Haoteng Yin, and Zhanxing Zhu, 'Spatio-temporal graph convolutional networks: A deep learning framework for traffic forecasting', *arXiv preprint arXiv:1709.04875*, (2017).

[31] Donghan Yu, Ruohong Zhang, Zhengbao Jiang, Yuexin Wu, and Yiming Yang, 'Graph-revised convolutional network', in *Machine Learning and Knowledge Discovery in Databases: European Conference, ECML PKDD 2020, Ghent, Belgium, September 14–18, 2020, Proceedings, Part III*, pp. 378–393. Springer, (2021).

[32] Donghan Yu, Ruohong Zhang, Zhengbao Jiang, Yuexin Wu, and Yiming Yang, 'Graph-Revised convolutional network', in *Machine Learning and Knowledge Discovery in Databases*, pp. 378–393. Springer International Publishing, (2021).

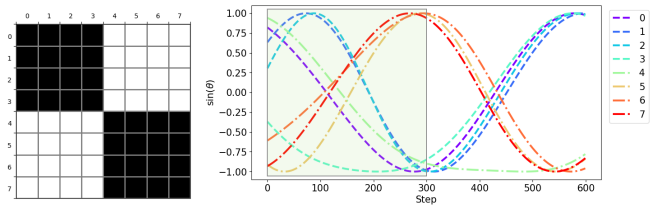
[33] Jiong Zhu, Yujun Yan, Lingxiao Zhao, Mark Heimann, Leman Akoglu, and Danai Koutra, 'Beyond homophily in graph neural networks: Current limitations and effective designs', *Advances in Neural Information Processing Systems*, **33**, 7793–7804, (2020).

[34] Yanqiao Zhu, Weizhi Xu, Jinghao Zhang, Qiang Liu, Shu Wu, and Liang Wang, 'Deep graph structure learning for robust representations: A survey', *arXiv preprint arXiv:2103. 03036*, **14**, (2021).

A Supplementary material

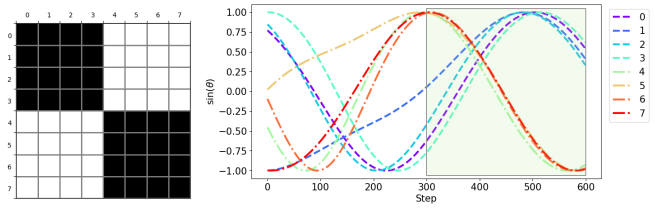
A.1 Kuramoto dataset examples

To illustrate common features and differences of training, validation and test data of the kuramoto dataset Figure 9 shows samples, i.e. trajectories of oscillators, alongside the coupling matrices that define the underlying structure from which the data has been generated. The normal coupling matrix $\mathbf{C}$ causes synchronization between oscillators 0, 1, 2, 3 and 4, 5, 6, 7. In contrast, the shuffled coupling matrix $\hat{\mathbf{C}}$ in the test leads to synchronization of 1, 2, 3, 7 and 0, 4, 5, 6.



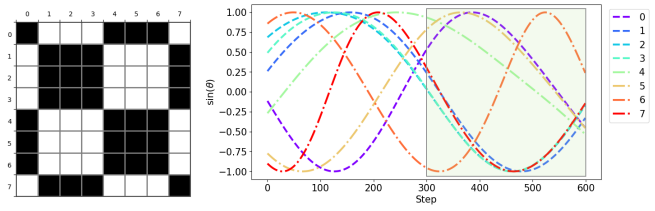
$\mathbf{C}$ $\quad$ $\mathbf{X}_{train}$ in green box

(a) A training data sample generated using $\mathbf{C}$. The first T (here $T = 300$) steps are used as training data.



$\mathbf{C}$ $\quad$ $\mathbf{X}_{val}$ in green box

(b) A validation data sample generated using $\mathbf{C}$. Steps 300-600 are used as validation data.



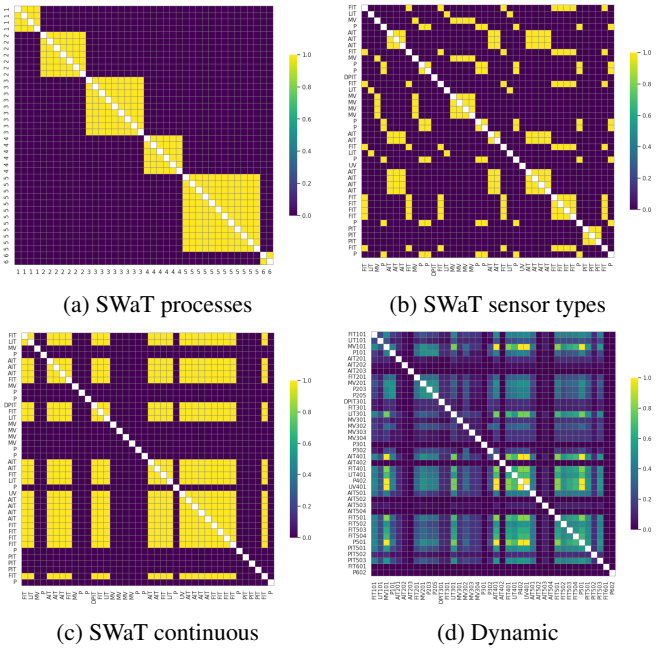
$\hat{\mathbf{C}}$ $\quad$ $\mathbf{X}_{test}$ in green box

(c) A test data sample generated using $\hat{\mathbf{C}}$. Steps 300-600 are used as test data.

Figure 9: Coupling matrices and samples for training, validation and test data in the Kuramoto dataset. Black indicates $\mathbf{C}_{ij} = 1$

A.2 SWaT

As mentioned in Section 4.5.4, for the SWaT system there is no obvious choice for ground truth connections. Structure learning results may be influenced equally as much by sensor types or the continuous or categorical nature of the signals. Figure 10b shows connections between sensors of the same type while Figure 10c highlights connections between continuous signals. We see that structures learned by the static (Figure 8b) and dynamic 10d partially resemble not only processes but also sensor or signal types.



(a) SWaT processes (b) SWaT sensor types

(c) SWaT continuous (d) Dynamic

Figure 10: Heterogeneous properties and dynamic generator sample result for the SWaT dataset