Revista ELECTRO, Vol. 46, 2024, pp. 272 - 277
https://itchihuahua.mx/revista_electro
ISSN Electrónico: 3061-774X

REDES NEURONALES GRÁFICAS (GNN) Y SU APLICACIÓN EN LA PROGRAMACIÓN DE CONTROLADORES LÓGICOS PROGRAMABLES (PLC)

Márquez Gutiérrez, Pedro Rafael; Acosta Cano de los Ríos, José Eduardo;
López Flores, David Ricardo; Baray Arana, Rogelio Enrique
Tecnológico Nacional de México/Tecnológico de Chihuahua
División de Estudios de Posgrado e Investigación
Avenida Tecnológico 2909, Chihuahua, Chih., México, C.P. 31200
Tel. +52 (614) 201-2000, Fax +52 (614) 413-5187
{pedro.mg, jose.ac, david.lf,rogelio.ba}@chihuahua.tecnm.mx

RESUMEN.

Las Redes Neuronales Gráficas (GNN) se han convertido en una herramienta significativa en la modelización y análisis de sistemas complejos representados como grafos. Estas redes permiten el aprendizaje profundo en datos estructurados, lo que las hace altamente efectivas en diversas aplicaciones, incluyendo redes sociales, estructuras moleculares y sistemas de control industrial. Este artículo explora la integración de las GNN en la programación y optimización de los Controladores Lógicos Programables (PLC), dispositivos esenciales en la automatización industrial. El estudio destaca cómo las GNN pueden utilizarse para mejorar la eficiencia, seguridad y fiabilidad de los sistemas PLC mediante la optimización automática de diagramas de escalera. Se presenta también una implementación práctica en Python que ilustra el potencial de las GNN para revolucionar la programación de los PLC.

ABSTRACT.

Graph Neural Networks (GNNs) have become a significant tool in the modeling and analysis of complex systems represented as graphs. These networks enable deep learning on structured data, making them highly effective in various applications, including social networks, molecular structures, and industrial control systems. This paper explores the integration of GNNs into the programming and optimization of Programmable Logic Controllers (PLCs), essential devices in industrial automation. The study highlights how GNNs can be utilized to enhance the efficiency, safety, and reliability of PLC systems by automatically optimizing ladder logic diagrams. A practical implementation in Python is also presented, illustrating the potential of GNNs to revolutionize the programming of PLCs.

Keywords: 1. Redes Neuronales Gráficas (GNN), Controladores Lógicos Programables (PLC), Automatización industrial, Programación de PLC, Diagrama de escalera (Ladder Logic), Optimización automática.

1. INTRODUCCIÓN

Las Redes Neuronales Gráficas (Graph Neural Networks, GNN) [2] han revolucionado la manera en que modelamos y analizamos sistemas complejos que pueden representarse como grafos, Figura 1.

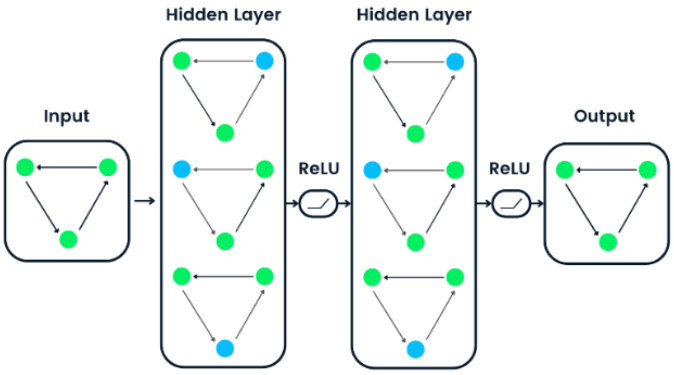


Figura 1. Esquema de una Red Neuronal Gráfica

Estas redes permiten el aprendizaje profundo en datos estructurados, como redes sociales [3], estructuras moleculares y sistemas eléctricos [10], proporcionando una herramienta poderosa para capturar relaciones intrínsecas y patrones en datos no euclidianos, Figura 2.

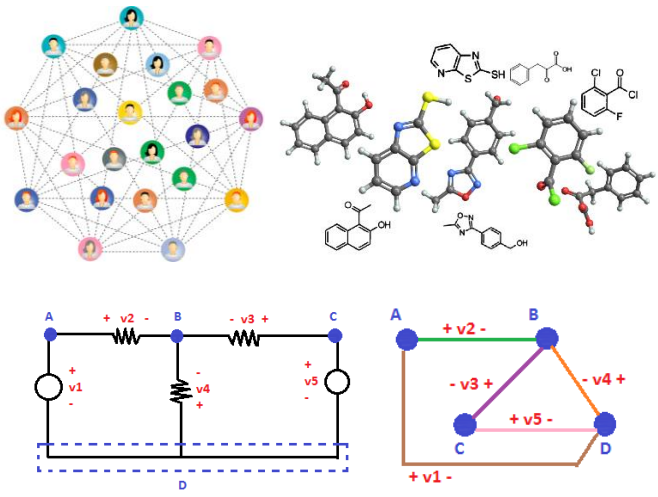


Figura 2. Grafos y aplicaciones

Revista ELECTRO, Vol. 46, 2024, pp. 272 - 277
https://itchihuahua.mx/revista_electro
ISSN Electrónico: 3061-774X

Una aplicación emergente y de gran relevancia es el uso de GNN en la programación y optimización de Controladores Lógicos Programables (PLC) [4], dispositivos clave en la automatización industrial, Figura 3.

Figura 3. PLC Logo! Siemens

Este artículo explora cómo las GNN pueden integrarse en la programación de PLC para mejorar la eficiencia y la seguridad en sistemas automatizados complejos.

2. GRAFOS Y SU REPRESENTACIÓN.

Un grafo es una estructura matemática que consta de un conjunto de nodos (o vértices) y un conjunto de aristas que conectan pares de nodos. Formalmente, un grafo $G = (V, E)$ se define como un par ordenado de conjuntos, donde V es el conjunto de nodos y E es el conjunto de aristas, cada una de las cuales conecta dos nodos [11]. Esta estructura es particularmente útil para modelar sistemas donde las entidades están interconectadas de manera compleja, como las redes de transporte, las redes neuronales biológicas y los sistemas de control industrial [6], Figura 3.

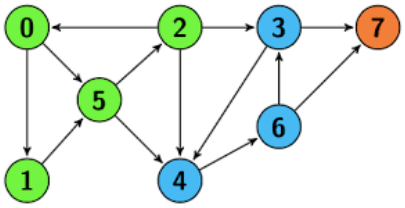


Figura 3. Diagrama de un Grafo Dirigido

2.1 Redes Neuronales Gráficas.

Las GNN extienden los conceptos de las redes neuronales convencionales para trabajar directamente con datos en forma de grafos. Utilizan un mecanismo de propagación de mensajes en el que las representaciones de los nodos se actualizan iterativamente en función de la información agregada de sus vecinos. Esta capacidad de incorporar la topología del grafo en el proceso de aprendizaje permite a las GNN capturar relaciones de alto nivel entre las entidades representadas por los nodos.
Las GNN han demostrado ser efectivas en tareas como la clasificación de nodos, la predicción de enlaces y la clasificación de grafos enteros.

Su capacidad para modelar dependencias complejas las convierte en una herramienta esencial para sistemas en los que la interconexión entre elementos es crítica, como en los sistemas de control industrial gestionados por PLC.

2.2 Mecanismo de Propagación de Mensajes en Redes Neuronales Gráficas.

El mecanismo de propagación de mensajes es uno de los componentes fundamentales de las Redes Neuronales Gráficas (GNN). Este mecanismo permite que la información se comparta entre los nodos de un grafo, lo que facilita la actualización de las representaciones de los nodos en función de la estructura del grafo y las características de los nodos vecinos.

a. **Concepto Básico**
En una GNN, cada nodo en un grafo tiene una representación de características (un vector) que se actualiza iterativamente a lo largo de varias capas de la red. El objetivo del mecanismo de propagación de mensajes es permitir que cada nodo agregue y combine la información de sus nodos vecinos para actualizar su propia representación.

b. **Fases del Mecanismo de Propagación de Mensajes**
El proceso de propagación de mensajes en una GNN generalmente se divide en las siguientes fases:

- **Mensajería** (*Message Passing*): En esta fase, cada nodo v envía un mensaje a sus vecinos. Este mensaje suele ser una función de las características del nodo emisor y las características de la arista que conecta al nodo emisor con el nodo receptor. Formalmente, el mensaje que el nodo v envía a su vecino u puede representarse como:

$$m_{v \to u} = \text{Message}(h_v, h_{v,u})$$

  donde h_v son las características del nodo v y $h_{v,u}$ son las características de la arista que conecta v con u.
- **Agregación** (*Aggregation*): Una vez que los nodos han enviado sus mensajes, cada nodo receptor u agrega los mensajes recibidos de sus vecinos. Este paso puede implicar operaciones como la suma, el promedio o la concatenación de los mensajes. La operación de agregación se puede describir como:

$$h_u^{\text{agg}} = \text{Aggregate}(\{m_{v \to u} : v \in \mathcal{N}(u)\})$$

  donde $N(u)$ es el conjunto de vecinos del nodo u.
- **Actualización** (*Update*): Después de la agregación, cada nodo u actualiza su representación de características utilizando la información agregada. Esto se suele hacer aplicando una función de actualización, que podría ser una simple función lineal o una red neuronal más compleja. El nuevo estado del nodo se calcula como:

$$h_u^{(l+1)} = \text{Update}(h_u^{(l)}, h_u^{\text{agg}})$$

donde $h_u^{(l)}$ es la representación del nodo en la capa l y $h_u^{(l+1)}$ es la representación del nodo en la capa siguiente.

2.3 IMPORTANCIA EN TAREAS DE APRENDIZAJE

El mecanismo de propagación de mensajes es crucial para que las GNN puedan realizar tareas como la clasificación de nodos, la predicción de enlaces y la clasificación de grafos completos. Al permitir que los nodos intercambien información y actualicen sus representaciones, las GNN pueden aprender representaciones ricas que capturan tanto las propiedades locales como las relaciones globales dentro del grafo.

3. CONTROLADORES LÓGICOS PROGRAMABLES (PLC) Y SU ROL EN LA AUTOMATIZACIÓN INDUSTRIAL.

Los Controladores Lógicos Programables (PLC) son dispositivos digitales diseñados para controlar procesos industriales mediante la ejecución de programas lógicos almacenados en su memoria. Los PLC son ampliamente utilizados en la automatización industrial debido a su fiabilidad, flexibilidad y capacidad para gestionar operaciones en tiempo real. Tradicionalmente, los PLC son programados utilizando diagramas de escalera (Ladder Logic), una forma gráfica de representar circuitos lógicos que es fácil de interpretar para los ingenieros eléctricos [7,8], Figura 4.

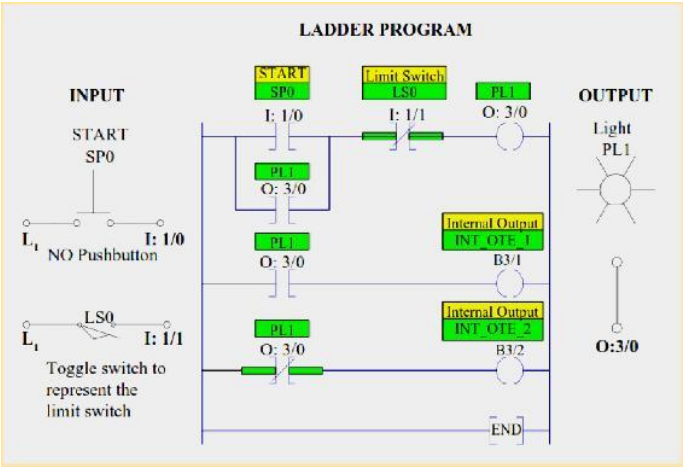


Figura 4. Diagrama de Escalera

4. DESAFÍOS EN LA PROGRAMACIÓN DE PLC.

La programación de PLC mediante Ladder Logic puede volverse extremadamente compleja en sistemas de gran escala, donde múltiples subsistemas deben coordinarse y sincronizarse. Además, la optimización de estos programas para mejorar la eficiencia y reducir los tiempos de ciclo es una tarea que requiere habilidades avanzadas y una considerable inversión de tiempo. Aquí es donde las GNN pueden jugar un rol crucial, al permitir la optimización automática de los programas de PLC mediante el aprendizaje a partir de grafos que representen los diagramas de escalera.

5. APLICACIÓN DE GNN EN LA PROGRAMACIÓN Y OPTIMIZACIÓN DE PLC.

5.1 Modelado de Diagramas de Escalera como Grafos.

Un diagrama de escalera puede representarse como un grafo en el que los contactos (entradas) y bobinas (salidas) son los nodos, y las conexiones lógicas entre ellos son las aristas. Este grafo captura la estructura lógica del programa de PLC, permitiendo que una GNN lo procese y optimice.

La idea central es utilizar GNN para analizar y optimizar la secuencia lógica del diagrama de escalera, detectando patrones de uso común y potenciales puntos de fallo, así como proponiendo mejoras que reduzcan los tiempos de ciclo o que mejoren la seguridad del sistema.

6. OPTIMIZACIÓN AUTOMATIZADA DE PROGRAMAS DE PLC.

Una vez que el diagrama de escalera ha sido convertido en un grafo, se puede entrenar una GNN para realizar diversas tareas, como la detección de bucles innecesarios, la identificación de rutas críticas que deben ser optimizadas, y la sugerencia de reestructuraciones que simplifiquen el código. Este proceso no solo mejora la eficiencia del sistema, sino que también ayuda a minimizar errores humanos en la programación de los PLC.

7. IMPLEMENTACIÓN EN PYTHON.

A continuación, se presenta un ejemplo ilustrativo de cómo una GNN puede aplicarse para optimizar un programa de PLC modelado como un grafo. Para ilustrar el funcionamiento de las Redes Neuronales Gráficas (GNN) y su aplicación en la programación de PLC, podemos crear un programa en Python que simule un sistema de control sencillo. En este ejemplo, se utiliza un grafo para modelar las conexiones entre varios componentes de un sistema de control, como sensores y actuadores, y se emplea una GNN para predecir las acciones del PLC basadas en las entradas de los sensores.

7.1 Configuración del entorno.

```
pip install torch torch_geometric networkx matplotlib
```

7.2 Grafo del Sistema de Control.

El código comienza creando un grafo que representa un sistema de control sencillo. Los nodos representan sensores, un PLC, y actuadores. Los sensores están conectados al PLC, que a su vez está conectado a los actuadores, Figura 5.

```python
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
import networkx as nx
import matplotlib.pyplot as plt
from torch_geometric.data import Data
from torch_geometric.nn import GCNConv

# Crear un grafo que representa un sistema de control con sensores y actuadores
G = nx.Graph()
G.add_edges_from([
    ('Sensor1', 'PLC'),
    ('Sensor2', 'PLC'),
    ('Sensor3', 'PLC'),
    ('PLC', 'Actuador1'),
    ('PLC', 'Actuador2')
])
```

Figura 5. Grafo de Control

Visualización del Grafo.

Se dibuja el grafo para visualizar cómo están conectados los diferentes componentes del sistema, Figura 6.

```python
# Visualizar el grafo del sistema de control
plt.figure(figsize=(8, 6))
nx.draw(G, with_labels=True, node_color='lightblue', edge_color='g
plt.title("Grafo del Sistema de Control")
plt.show()
```

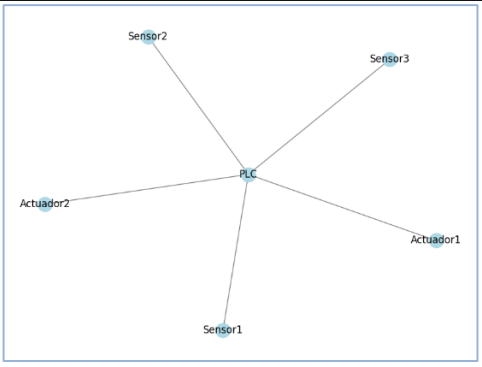


Figura 6. Visualización del Grafo de Control

7.4 Preparación de los Datos.

Se asignan características simples a cada nodo (un vector binario para los sensores y un vector constante para el PLC y los actuadores). Luego, el grafo se convierte en un formato compatible con *PyTorch Geometric*, Figura 7.

```python
# Crear un mapeo de nodos a índices numéricos
node_mapping = {node: idx for idx, node in enumerate(G.nodes())}

# Convertir las aristas a índices numéricos
edges = [(node_mapping[u], node_mapping[v]) for u, v in G.edges()]

# Convertir a un tensor de PyTorch
edge_index = torch.tensor(edges, dtype=torch.long).t().contiguous()

# Asignar características a los nodos
node_features = {
    'Sensor1': [1, 0, 0],
    'Sensor2': [0, 1, 0],
    'Sensor3': [0, 0, 1],
    'PLC': [0.5, 0.5, 0.5],
    'Actuador1': [0, 0, 0],
    'Actuador2': [0, 1, 0]
}
```

Figura 7. Preparación de Datos.

7.5 Definición del modelo GNN

1. **Definición de la Red Neuronal Gráfica (GNN)**: Se define una red neuronal gráfica utilizando GCNConv, una capa de convolución sobre grafos. La red tiene dos capas: la primera expande las características y la segunda produce la salida que indica si los actuadores deben activarse o no.

2. **Entrenamiento del Modelo**: El modelo se entrena utilizando un pequeño conjunto de etiquetas predefinidas que indican si los actuadores deben activarse o no, basándose en las entradas de los sensores [1,5]. Ver Figura 8.

```python
# Definir los datos para PyTorch Geometric
data = Data(x=x, edge_index=edge_index)

# Definir la red neuronal gráfica (GNN) utilizando GCNConv
class GNN(nn.Module):
    def __init__(self):
        super(GNN, self).__init__()
        self.conv1 = GCNConv(data.num_node_features, 8)
        self.conv2 = GCNConv(8, 2)  # 2 salidas, para activar o no el actuador

    def forward(self, data):
        x, edge_index = data.x, data.edge_index
        x = F.relu(self.conv1(x, edge_index))
        x = self.conv2(x, edge_index)
        return F.log_softmax(x, dim=1)

# Crear el modelo, definir la pérdida y el optimizador
model = GNN()
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.01)

# Crear etiquetas para entrenar el modelo (1 = activar, 0 = no activar)
# Solo entrenamos en los nodos de los actuadores
labels = torch.tensor([0, 0, 0, 0, 1, 0], dtype=torch.long)  # Solo Actuador1 se activ

# Definir la máscara de entrenamiento
train_mask = torch.tensor([0, 0, 0, 0, 1, 1], dtype=torch.bool)  # Solo entrenamos en

# Entrenar la GNN
for epoch in range(100):
    model.train()
    optimizer.zero_grad()
    out = model(data)
    loss = criterion(out[train_mask], labels[train_mask])
    loss.backward()
    optimizer.step()

    if epoch % 10 == 0:
        print(f'Epoch {epoch}, Loss: {loss.item():.4f}')
```

Figura 8. GNN

7.6 Evaluación y Visualización.

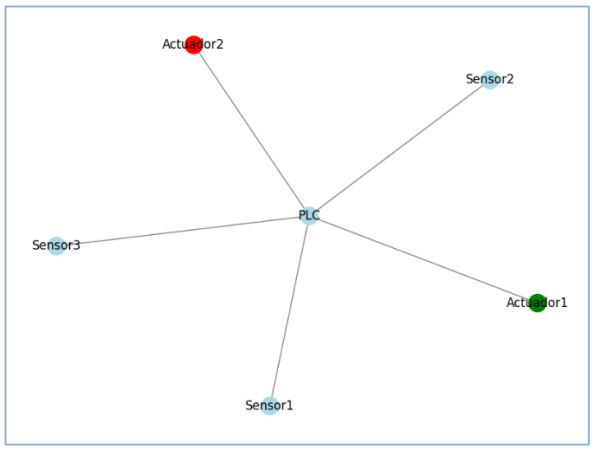
Después del entrenamiento, se realizan predicciones sobre los nodos del grafo y se visualizan los resultados. Los actuadores que deben activarse se colorean en verde, mientras que los que no deben activarse se colorean en rojo, Figura 9.

```python
# Evaluar la GNN (predicciones)
model.eval()
pred = model(data).argmax(dim=1)
print("Predicciones de los actuadores (0 = no activar, 1 = activar):")
for node, p in zip(G.nodes, pred):
    if "Actuador" in node:
        print(f'{node}: {p.item()}')

# Visualización de las predicciones en el grafo
color_map = []
for node in G.nodes:
    if "Actuador" in node:
        # Usar node_mapping[node] directamente para acceder al índice en
        if pred[node_mapping[node]] == 1:
            color_map.append('green')
        else:
            color_map.append('red')
    else:
        color_map.append('lightblue')

plt.figure(figsize=(8, 6))
nx.draw(G, with_labels=True, node_color=color_map, edge_color='gray')
plt.title("Predicciones de los Actuadores en el Sistema de Control")
plt.show()
```

Figura 9. Evaluación y Predicciones

8. RESULTADOS

En este ejemplo simple, la GNN aprende a predecir si la bobina debe activarse en función del estado de los contactos. Este es un caso extremadamente básico, pero ilustra cómo las GNN pueden aplicarse para modelar y optimizar la lógica de los PLC, Figura 10.




```
Epoch 0, Loss: 0.6593
Epoch 10, Loss: 0.5972
Epoch 20, Loss: 0.5525
Epoch 30, Loss: 0.5006
Epoch 40, Loss: 0.4377
Epoch 50, Loss: 0.3748
Epoch 60, Loss: 0.3115
Epoch 70, Loss: 0.2529
Epoch 80, Loss: 0.2027
Epoch 90, Loss: 0.1614
Predicciones de los actuadores (0 = no activar, 1 = activar):
Actuador1: 1
Actuador2: 0
```

Figura 10. Resultados

9. DISCUSIÓN: LA DIFICULTAD DE EMPLEAR AUTÓMATAS FINITOS DETERMINÍSTICOS PARA PROGRAMAR UN PLC.

Los PLC han sido tradicionalmente programados utilizando lenguajes gráficos como el Diagrama de Escalera (Ladder Diagram, LD) o lenguajes textuales como el Texto Estructurado (ST), los cuales están diseñados para ser intuitivos y accesibles para ingenieros y técnicos con formación en electricidad y electrónica. Sin embargo, cuando se trata de implementar lógicas de control secuencial complejas, los Autómatas Finitos Determinísticos (AFD) ofrecen una estructura matemática precisa que puede, en teoría, mejorar la claridad y la eficiencia de la programación. A pesar de esto, la integración de AFD en la programación de PLCs presenta una serie de desafíos técnicos.

9.1 Atractivo de los AFD en la Programación de PLC.

Los Autómatas Finitos Determinísticos [9], son modelos matemáticos que permiten describir un sistema en términos de un número finito de estados y transiciones bien definidas entre estos estados. Formalmente, un AFD es una quintupla $A=(Q,\Sigma,\delta,q0,F)$, donde:

1. **Q** es un conjunto finito de **estados**.
2. **Σ** es un conjunto finito de **símbolos**, conocido como el **alfabeto**.
3. **$\delta:Q\times\Sigma\rightarrow Q$** es una **función de transición**, que toma un estado y un símbolo de entrada, y devuelve un nuevo estado. Esta función es **determinista**, lo que significa que para cada par (q,a), donde $q\in Q$ y $a\in\Sigma$, hay un único estado $q'\in Q$ tal que $\delta(q,a)=q'$.
4. **$q0\in Q$** es el **estado inicial**, es decir, el estado en el que el autómata comienza su proceso de lectura de la cadena.
5. **$F\subseteq Q$** es un conjunto de **estados de aceptación** o **estados finales**. Si, después de procesar una cadena de símbolos, el autómata termina en uno de estos estados, se dice que la cadena es aceptada por el autómata.

Esta representación es particularmente útil en sistemas que dependen de secuencias específicas de eventos o de cambios de

estado, como los procesos industriales automatizados. En teoría, los AFD ofrecen ventajas significativas:

1. **Precisión en el Control Secuencial**: Los AFD permiten una descripción precisa de las secuencias de operaciones, lo que puede resultar en un control más exacto de procesos complejos.
2. **Claridad en el Diseño**: Al dividir un proceso en estados y transiciones claramente definidos, los AFD facilitan la visualización y el diseño de sistemas de control, reduciendo la ambigüedad y el riesgo de errores lógicos.
3. **Predictibilidad y Deter**minismo: En un AFD, para cada combinación de estado y entrada, el resultado es siempre el mismo. Esta característica es vital en aplicaciones donde la confiabilidad y la repetibilidad son críticas.

9.2 Desafíos en la Implementación de AFD en PLCs

A pesar de los beneficios teóricos de los AFD, su implementación en la programación de PLCs no es trivial y conlleva una serie de dificultades prácticas.

1. **Complejidad del Diseño**: Aunque los AFD pueden simplificar la descripción de sistemas secuenciales complejos, su diseño inicial puede ser extremadamente complejo, especialmente para sistemas con un gran número de estados y transiciones. A medida que aumenta la complejidad del sistema, el número de estados y transiciones en el AFD puede crecer exponencialmente, lo que dificulta tanto la creación como el mantenimiento del autómata.
2. **Traducción a Diagrama de Escalera**: Convertir un AFD en un Diagrama de Escalera, el lenguaje más comúnmente utilizado en la programación de PLCs, puede ser un proceso tedioso y propenso a errores. Cada estado y transición del AFD debe traducirse en contactos, bobinas y peldaños en el diagrama de escalera, lo que puede llevar a diagramas complejos y difíciles de seguir. La conversión manual aumenta la probabilidad de cometer errores de programación, que pueden ser difíciles de detectar y corregir.
3. **Limitaciones de los PLCs**: Los PLCs están diseñados para ser flexibles y fáciles de programar por técnicos que no necesariamente tienen formación en teoría de autómatas. Implementar AFDs en un PLC requiere un nivel de abstracción y comprensión matemática que puede exceder las capacidades de los lenguajes de programación y de los programadores de PLC típicos. Además, las plataformas de PLC pueden tener limitaciones en la cantidad de estados o en la complejidad de las lógicas que pueden implementar eficientemente.
4. **Mantenimiento y Escalabilidad**: Un sistema basado en AFD puede ser difícil de escalar y mantener a lo largo del tiempo. Cualquier modificación en el proceso controlado puede requerir cambios significativos en el AFD y, por consiguiente, en el código del PLC. Además, la documentación y el entendimiento de un sistema basado en AFD pueden ser más difíciles de transmitir a nuevos ingenieros o técnicos que se unan al proyecto.

10. CONCLUSIONES

Las Redes Neuronales Gráficas representan una herramienta poderosa para la programación y optimización de PLC en sistemas de control industrial. Al modelar diagramas de escalera como grafos y aplicar GNN, es posible mejorar la eficiencia, reducir los tiempos de ciclo y minimizar errores humanos. La integración de GNN en la programación de PLC abre nuevas posibilidades para la automatización avanzada y la mejora continua de procesos industriales. Las GNN son una herramienta poderosa para manejar datos estructurados en forma de grafos, lo que las hace ideales para aplicaciones en la programación de PLC y otros sistemas donde las relaciones entre componentes son cruciales para el funcionamiento del sistema.

11. REFERENCIAS

[1]. Kipf, T. N., & Welling, M. (2017). Semi-supervised Classification with Graph Convolutional Networks. arXiv preprint arXiv:1609.02907.
[2]. Wu, Z., Pan, S., Chen, F., Long, G., Zhang, C., & Yu, P. S. (2020). A Comprehensive Survey on Graph Neural Networks. IEEE Transactions on Neural Networks and Learning Systems, 32(1), 4-24.
[3]. Velickovic, P., Cucurull, G., Casanova, A., Romero, A., Lio, P., & Bengio, Y. (2018). Graph Attention Networks. arXiv preprint arXiv:1710.10903.
[4]. Bolton, W. (2015). Programmable Logic Controllers, 6th Edition
[5]. Sayed-Mouchaweh, M. (2018). Learning in Non-Stationary Environments: Methods and Applications. Springer.
[6]. Zhou, J., Cui, G., Zhang, Z., Yang, C., Liu, Z., Wang, L., ... & Sun, M. (2020). Graph Neural Networks: A Review of Methods and Applications. AI Open, 1, 57-81.
[7]. Lewis, R. W. (2016). Programming Industrial Control Systems Using IEC 61131-3. IET.
[8]. Bolton, W. (2015). Programmable Logic Controllers, Newnes.
[9]. Sipser, M. (2014). Introduction to the Theory of Computation, Cengage Learning.
[10]. Tuo, M; Li, X.; Zhao, T. (2023), Graph Neural Network-based Power Flow Model, arXiv:2307.02049.
[11]. Koh, K.M.; Dong, F.; Tay, E.G. (2024), Introduction to Graph Theory: With Solutions to Selected Problems, World Scientific Publishing, ISBN-13: 978-9811285011.