predictCSMA: Graph Neural Networks for CSMA Network Throughput Prediction

USC Autonomous Networks Research Group (ANRGUSC)

September 21, 2025

1 Overview

This repository implements Graph Neural Network (GNN) architectures for predicting per-node saturation throughput in CSMA (Carrier Sense Multiple Access) wireless networks, comparing them against theoretical Markov chain models and renewal approximation methods `https://ieeexplore.ieee.org/document/11133870`.

Repository: `https://github.com/ANRGUSC/predictCSMA`

2 Project Structure

```
predictCSMA/
          Dataset/                          # Generated simulation datasets
          GNN_Code/                         # Graph Neural Network implementations
                GNN_Architecture.py          # Core model architectures
                GCN.py                      # Graph Convolutional Network
                DGCN.py                     # Dynamic/Enhanced GCN
                GIN.py                      # Graph Isomorphism Network
                GINE.py                     # GIN with Edge attributes
                SAGE.py                     # GraphSAGE
                PredictWithGCN.py           # Inference utilities
          Markove.py                       # Markov chain exact throughput
   calculation
          Renewal.py                       # Renewal approximation approach
          Simulation.py                     # CSMA simulation engine
          Generate_data_simulation.py       # Data generation
          AggregateData.py                 # Dataset aggregation
          Data_cleaning.py                 # Data preprocessing
          README                            # This file
```

3 Methods Implemented

3.1 Analytical Methods

1. Markov Chain Analysis (`Markove.py`)

- Exact throughput calculation using steady-state probabilities
- Provides ground truth for validation

2. Renewal Approximation (`Renewal.py`)

- Approximate throughput using renewal theory

3.2 Machine Learning Methods

3. Graph Neural Networks (`GNN_Code/`)

- Multiple architectures: GCN, D-GCN, GIN, GINE, GraphSAGE
- Input: Network topology and transmission probabilities
- Output: Per-node saturation throughput
- Training on simulated data with Markov chain ground truth

4 Installation

4.1 Clone the Repository

```
git clone https://github.com/ANRGUSC/predictCSMA.git
cd predictCSMA
```

4.2 Dependencies

```
# Core requirements
torch>=1.9.0
torch-geometric>=2.0.0
numpy>=1.19.0
pandas>=1.2.0
matplotlib>=3.3.0
scikit-learn>=0.24.0
networkx>=2.5

# Install PyTorch (adjust for your CUDA version)
pip install torch torchvision --index-url https://download.pytorch.org/whl/cu118

# Install PyTorch Geometric
pip install torch-geometric

# Install other dependencies
pip install numpy pandas matplotlib scikit-learn networkx
```

5 Data Generation Pipeline

5.1 Step 1: Generate Simulation Data

```
python Generate_data_simulation.py
```

Parameters:

- `num_nodes`: Network size (default: 6)
- `num_graphs`: Number of topologies (default: 5000)

- `total_time_slots`: Simulation duration (default: 1,000,000)
- `T`: Packet transmission duration (default: 8)
- `sigma`: Backoff duration after collision (default: 1)
- `prob_edge`: Edge probability for Erdős-Rényi graphs (default: 0.5)

Output format: `T_numNodes_node_simulation_data_million.csv`

5.2 Step 2: Data Cleaning (Optional)

Removes zero-throughput samples that can affect training.

5.3 Step 3: Aggregate Multiple Configurations

```
python AggregateData.py
```

Combines datasets from different node counts into unified training data.

6 Training Graph Neural Networks

Each GNN model can be trained independently:

```
# Graph Convolutional Network
python GNN_Code/GCN.py

# Enhanced Dynamic GCN
python GNN_Code/DGCN.py

# Graph Isomorphism Network
python GNN_Code/GIN.py

# GIN with Edge Attributes
python GNN_Code/GINE.py

# GraphSAGE
python GNN_Code/SAGE.py
```

Training configuration:

- Data split: 70% train, 10% validation, 20% test
- Epochs: 200-250 with early stopping
- Batch size: 32
- Learning rate: 0.001 with scheduling
- Loss: Mean Squared Error (MSE)
- Output: `T_best{MODEL}.pt`

7 Network Utility Optimization

Demonstrates network utility maximization using:

- Gradient-based optimization
- Comparison between Markov chain and GNN predictions
- Log-utility and proportional fairness objectives

The optimization problem:

$$\max_{p} \sum_{i=1}^{N} \alpha_i U(\theta_i(p)) \tag{1}$$

where p are transmission probabilities, θ_i is throughput of node i, and $U(\cdot)$ is the utility function.

8 Model Inference

```
python GNN_Code/PredictWithGCN.py
```

Predict throughput for new network configurations using trained models.
Input format:

```
{
    'adj_matrix': [[0,1,0],[1,0,1],[0,1,0]],  # Network topology
    'transmission_prob': [0.1, 0.07, 0.05],    # p values
    'saturation_throughput': [...]              # Optional ground truth
}
```

9 Performance Metrics

- **MAE**: Mean Absolute Error
- **NMAE**: Normalized MAE = $\frac{\text{MAE}}{\text{mean}(\theta_{\text{actual}})}$
- **Computational Time**: Seconds per prediction
- **Scalability**: Performance vs. network size

10 CSMA Protocol Parameters

- T: Packet transmission duration (time slots)
- σ: Backoff duration after collision
- p_i: Transmission probability of node i
- θ_i: Saturation throughput of node i

11 Key Contributions

1. Fast throughput prediction using GNNs as surrogate models
2. Comparison of multiple analytical and ML approaches
3. Scalable alternative to exact Markov chain analysis
4. Application to network utility optimization
5. Open-source implementation and datasets

12 Citation

If you use this code in your research, please cite `https://ieeexplore.ieee.org/document/11133870`

13 Contact

USC Autonomous Networks Research Group (ANRGUSC)
Repository: `https://github.com/ANRGUSC/predictCSMA`