Optimisation of the D2D Topology Formation Using a Novel Deep ML Approach for ~~in~~ 6G Mobile Networks

Iacovos Ioannou* Marios Raspopoulos‖, Prabagarane Nagaradjane‡, Christophoros Christophorou*, Ala' Khalifeh¶, Vasos Vassiliou*
* Department of Computer Science, University of Cyprus and CYENS - Centre of Excellence, Cyprus
‡ Department of ECE, Sri Sivasubramaniya Nadar College of Engineering Chennai, India
¶ German Jordanian University, Amman, Jordan
‖ INSPIRE Research Centre, University of Central Lancashire, Larnaca, Cyprus

Abstract—**Optimizing device-to-device (D2D) topologies is pivotal for enhancing the performance and efficiency of 6G networks. This paper introduces a novel approach for forming optimal subnet trees within ~~these~~ the 6G networks using BDIx agents and advanced Minimum-Weight Spanning Tree (MWST/MST) algorithms augmented by Graph Neural Networks (GNNs). Our solution aims to significantly boost network performance, particularly in high-demand scenarios such as urban areas, large-scale events, and remote locations. ~~By minimizing power consumption and maximizing throughput, our approach dynamically adapts to changing network conditions, user movements, and traffic patterns~~. We implement various MWST algorithms, including Kruskal's, Prim's, and Boruvka's algorithms, and introduce a GNN model to predict edge weights. We propose a "weighted distance" metric to analyze network performance comprehensively. Our AI/ML-driven solution integrates BDIx agents with MWST algorithms, focusing on optimizing subnets under gNodeB in 6G networks, enhancing data transmission efficiency, reducing latency, and increasing throughput. Our approach dynamically adapts to changing network conditions, user movements, and traffic patterns by minimizing power consumption and maximizing throughput. This research contributes to developing scalable and flexible network management solutions suitable for diverse configurations and architectures.**

Keywords—**Device-to-Device (D2D) communication, 6G networks, Minimum-Weight Spanning Tree (MWST), Graph Neural Networks (GNNs), BDIx agents, network optimization, power consumption, throughput, dynamic adaptation.**

I. Introduction

Optimising D2D topologies is essential in the ever-changing telecommunications industry to improve the performance and efficiency of 6G networks. This research focuses on the issue of creating optimal subnet trees in these 6G networks. This is crucial for effectively managing the complex weighted tree structures needed for efficient network governance. Our proposal involves utilising BDIx agents and sophisticated Minimum-Weight Spanning Tree (MWST) algorithms, while employing Graph Neural Networks (GNNs). The objective of this research is to provide a strong and flexible solution to greatly improve network performance, particularly in situations with high demand like densely populated urban areas, large-scale events, and remote locations where maximising coverage and connectivity is crucial [1], [2].

The primary challenge in optimizing D2D topologies is to minimize power consumption and maximize throughput, such as sum rate, across the network by selecting paths with the shortest distances and highest data rates. Traditional spanning tree methods often fail to adapt dynamically to changing network conditions, user movements, and traffic patterns, highlighting the need for a more robust and intelligent solution [3], [4]. Our research aims to overcome these limitations by introducing a dynamic and adaptive service to optimize subnet trees in real-time. Using BDIx agents based on the Belief-Desire-Intention (BDI) framework ~~and~~ augmented with machine learning, we provide a comprehensive depiction of the network's current state, including node positions and connection quality, which are critical for effective MWST algorithm application [5], [2]. Our approach involves creating random trees and trees that are generated using the DAI framework with BDIx agents with coordinates and distances, calculating data rates, and applying various MWST algorithms. This includes implementing a GNN model to predict edge weights in the spanning tree and evaluating the performance of different algorithms such as Kruskal's, Prim's and Boruvka's algorithms. Additionally, we calculate power consumption and data rates, introducing a "weighted distance" metric for comprehensive network performance analysis. This AI/ML-driven solution focuses on optimizing subnets in 6G networks under gNodeB, integrating BDIx agents with MWST algorithms using GNNs [6], [2].

The core purpose of our proposed strategy is to manage the complex weighted tree structures produced by BDIx agents, crucial for the closed-loop governance module. By employing innovative MWST algorithms, we aim to optimize network topology efficiently, considering key metrics like minimizing delay and maximizing throughput. The dynamic and flexible characteristics of our solution allow it to process new data from the localization submodule, adjusting to network condition fluctuations, user movement, and traffic patterns for continuous optimization. Our approach is designed for scalability and flexibility and is compatible with various gNodeB configurations and 5G architectures. Furthermore, it can assist in improving data transmission efficiency, reducing latency, and increasing throughput, thereby offering significant advantages in different operating scenarios. ~~It offers significant advantages in high-density urban areas, large-scale events, emergencies, and remote areas with limited infrastructure"~~[7], [8], [9].

The novelty of our approach lies in combining BDIx agents with GNN-based MWST algorithms. This integration allows dynamic network configuration adjustments in response to real-time changes or improvements that can be enabled, ensuring peak network efficiency. The use of GNNs enhances the capability to handle complex graph structures, providing a scalable solution for various network sizes and complexities [10], [11].

The contributions of this research are:

1) Leveraging the Distributed Artificial Intelligence (DAI) framework with machine learning to provide ~~provides~~ an adaptive solution for network management [12].
2) Implementing over the state-of-the-art MWST [1] algorithm based on Deep ML that ensures optimal path selection, minimizing latency and maximizing throughput.
3) To propose an approach that ensures the D2D topology to adapt ~~The service adapts~~ in real-time to changing conditions, user movements, and traffic patterns, ensuring consistent performance.
4) To propose D2D topology optimisation scheme that ~~The solution~~ is scalable and flexible, suitable for various network sizes and complexities.
5) To quantify that the proposed approach results in improved data transmission efficiency, reduced latency, increased throughput, and optimised resource utilization ~~Improvements in data transmission efficiency reduce latency, increase throughput, and optimize resources~~ [13], [14].

The rest of this article is arranged as follows: Section II provides a comprehensive discussion of the background information that is relevant to our inquiry. The proposed system description is explained in detail in Section III. Section IV elaborates on the approach employed and the deep learning models utilized for the estimation along with the methodology used. The simulation results of the suggested system under different settings are presented and analyzed in Section V. Section VI discusses the results drawn from the research and outlines potential future directions.

II. Background Work

To understand traditional Minimum Spanning Tree (MST) algorithms, we review classical approaches and a machine learning method. MST algorithms find a subset of edges forming a tree with minimized total weight. The key algorithms are Kruskal's, Prim's, Boruvka's, and a Graph Neural Network (GNN) approach.

1) **Kruskal's Algorithm**: A greedy method sorting edges by weight and adding them to the MST without forming cycles until $V-1$ edges are included [8], [15].
2) **Prim's Algorithm**: Constructs the MST by starting with one vertex and repeatedly adding the smallest edge in terms of the metric examined connecting a vertex in the tree to one outside it [7], [16].

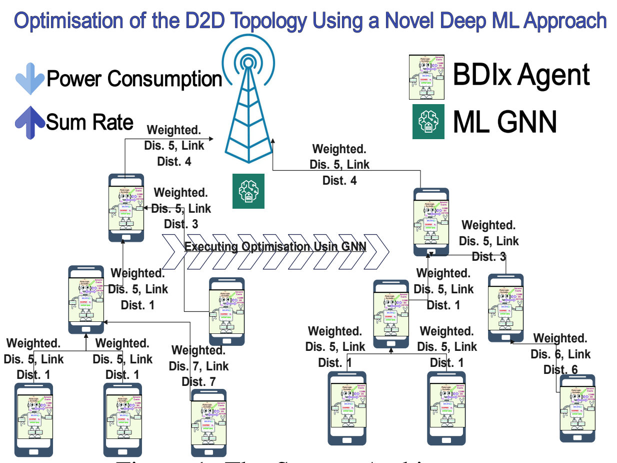
[1]MWST algorithms are called like this because we use the weighted metric. They are the same as the MST.

3) **Boruvka's Algorithm**: Treats each vertex as a separate component, merging the closest components iteratively until a single component remains [17], [18].
4) **GNN Approach**: Uses SAmple and aggreGatE Convolution (SAGEConv) layers to aggregate information from node neighborhoods, generating embeddings for tasks like node classification and MST prediction [19], [20].

These algorithms have different strengths and weaknesses. Kruskal's and Prim's are general-purpose algorithms, while Boruvka's excels in parallel computing. The GNN approach leverages learned embeddings for potentially improved MST predictions.

III. System Description

This section outlines the problem and system components. The proposed system comprises a base station (BS), user equipments (UEs) forming tree-style D2D communication subnetworks, and a BS controller responsible for optimizing the network tree as shown in Figure. 1. The BS coordinates the network, while UEs participate in D2D communications. The BS controller ensures efficient network management and performance. BDIx agents, based on the DAI framework


Optimisation of the D2D Topology Using a Novel Deep ML Approach
Power Consumption
Sum Rate
BDIx Agent
ML GNN
Weighted. Dis. 5, Link Dist. 4
Weighted. Dis. 5, Link Dist. 4
Weighted. Dis. 5, Link Dist. 3
Weighted. Dis. 5, Link Dist. 3
Executing Optimisation Usin GNN
Weighted. Dis. 5, Link Dist. 1
Weighted. Dis. 5, Link Dist. 1
Weighted. Dis. 5, Link Dist. 1
Weighted. Dis. 5, Link Dist. 1
Weighted. Dis. 7, Link Dist. 7
Weighted. Dis. 5, Link Dist. 1
Weighted. Dis. 5, Link Dist. 1
Weighted. Dis. 6, Link Dist. 6
Figure 1: The System Architecture

~~and~~ enhanced with machine learning, autonomously form subnetworks under the BS using a transmission selection algorithm that employs the Weighted Data Rate (WDR) metric. This allows BDIx agents to dynamically select transmission modes and establish efficient subnets by considering data rates and UE positions. These agents provide a comprehensive view of the network's current state, including node positions and connection quality, aiding the application of MWST algorithms for optimizing network topology. Initially, BDIx agents form a subnetwork under the BS. Our target is to optimize this subnetwork to enhance the existing network's performance (i.e., sum rate and power consumption). When traffic and user demand increase in a specific 5G subnetwork, the telecom operator can activate ~~activates~~ the BDIx agents to form a D2D communication network, to increase the data rates, reduce power consumption, and support required bandwidth. ~~increasing data rates, reducing power consumption, and supporting the required bandwidth.~~

IV. Deep Learning Aided Topology Optimisation Methodology

In this section, we provide the introduction of the weighted distance, the methodology, and the steps that are used in the research. Specifically,~~So,~~ this study focuses on creating and analyzing Minimum Spanning Trees (MST) to minimize the maximum weighted distance in graphs representing random trees and trees generated by BDIx agents with coordinates. We utilize both traditional MST algorithms and a graph neural network (GNN) approach to predict parent-child relationships in the graph. This section details our methodology, including data generation, algorithmic steps, training process, hyperparameter search, evaluation, and visualization. Our methodology involves several key steps. First, we create a structure to track the results for each model, organizing various metrics such as data rates and power consumption values in a dictionary. This organization ensures that all necessary data is easily accessible. We then iterate through different node sizes, generating sample data for each size to assess model performance as the network scales. The current node size being tested is recorded to maintain clear conditions for each result set. Next, we use Kruskal's algorithm to compute the MST for the generated data, finding the subset of edges that connects all nodes with the minimum total edge weighted distance. The total weighted distance, data rate, and power consumption for the MST are calculated and recorded, serving as benchmark data for training GNN-based models. We then retrieve the necessary parameters for each model, configure them, and execute them to generate predictions and construct trees. Performance metrics for these predicted trees, such as total weight, data rate, and power consumption, are calculated and recorded to compare each model's effectiveness against the benchmark.

Finally, we compile all recorded results into a comprehensive dataset, containing performance metrics for each model across different node sizes. This dataset is returned for further analysis and comparison, ensuring that all evaluation data is systematically organized and ready for detailed examination.

A. Formulation of Weighted Distance

To calculate the MST towards the root node, referred to as the BS, we need ~~needed~~ to introduce a new metric, the weighted distance. The weighted distance is defined as the maximum link distance of an edge on a path towards the root. Specifically, for any two nodes u and v, the weighted distance (physical distance in meters) $w(u,v)$ is given by:

$$w(u,v) = \max_{e \in P(u,v)} \{d(e)\}$$

where $P(u,v)$ represents the path between nodes u and v, and $d(e)$ is the distance of edge e on this path (as shown in 1). This metric allows us to create a complete graph where the edge weights reflect the maximum link distance on a path towards the root. This setup enables a comprehensive comparison of the performance of different MST algorithms and our GNN-based model and also provide us the tool to calculate the minimum spanning tree that its links have the minimum weighted distance towards the BS.

B. Methodology Steps

1) Data Creation: The data creation process involves several steps to generate synthetic data using random trees along with trees that are generated from BDIx agents and complete graphs, focusing on calculating maximum distance paths and setting appropriate edge weights. Here's a detailed description of the procedure:

1) **Generating random trees and trees resulted from with the use of DAI Framework and BDIx agents with Coordinates [2]:** Start by creating a random tree and a tree with a specified number of nodes that have been installed on them the BDIx agents. Each node is assigned coordinates representing the spatial location of the nodes.
2) **Calculating Distances for Tree Edges:** Calculate the distance between the connected nodes for each edge in the tree. Assign this distance to the edge as both its distance and weight attribute, ensuring that each edge has a corresponding distance value representing the separation between the nodes.
3) **Calculating Maximum Distance Paths:** Designate a root node (e.g., node 0) and compute the maximum distance paths from this root node to all other nodes in the tree. This step is crucial for understanding the tree's structure and the farthest points from the root.
4) **Creating a Complete Graph:** Generate a complete graph from the nodes of the tree. In this graph, every pair of nodes is connected by an edge, allowing for the consideration of all possible direct connections between nodes.
5) **Assigning Positions to Complete Graph Nodes:** Assign each node in the complete graph the same coordinates as its corresponding node in the tree, ensuring consistent spatial representation between the tree and the complete graph.
6) **Calculating Distances and Weights for Complete Graph Edges:** Recalculate the distance between connected nodes for each edge in the complete graph. Set the weight of each edge to the maximum of the pre-calculated maximum distances of the two nodes it connects. This step assigns meaningful weights to the edges, reflecting the tree's maximum distance paths.
7) **Constructing the Minimum Spanning Tree:** Use Kruskal's algorithm (selected because it is superior to other classic approaches [8]) to construct a MST from the complete graph. This MST connects all nodes with the minimum possible total edge weight, determining the minimum distance for each edge in the MST.
8) **Converting to a Suitable Data Format:** Convert the complete graph and the MST into a suitable format,

likely an array, for efficient processing in further analysis or model training.

In summary, the data creation process generates a synthetic dataset by first creating a random tree and a tree generated by BDIx agents and calculating distances between nodes. It then builds a complete graph, assigns appropriate weights to its edges based on maximum distance paths, constructs a minimum spanning tree, and formats the data.

2) The Training Features: The dataset used for training the GNN model consists of several key fields as outlined in Table I. These fields include the node features, which are the coordinates of each node in the graph; the edge index, which defines the connections between nodes; and the edge attributes, which are the weights of the edges based on node distances. Additionally, the dataset includes parent indices, representing the parent-child relationships in the Minimum Spanning Tree (MST), which are used as labels for training the model.

Table I: Fields of the Dataset Used for Training the GNN Model

Dataset Field	**Description**	**Tensor Shape**
Node Features (pos)	Coordinates (x, y) of each node	`(num_nodes, 2)`
Edge Index (edge_index)	Indices of nodes forming each edge	`(2, num_edges)`
Edge Attributes (edge_attr)	Weights of edges based on distance	`(num_edges, 1)`
Parent Indices (parent_indices)	Parent node index in the MST	`(num_nodes)`

3) Hyperparameter Search: To optimize the model performance, we conduct a hyperparameter search using the Optuna library over a predefined grid of parameters, selecting the best combination based on validation loss [21]. The search process involves exploring various combinations of hyperparameters to identify the best set for a given model. This is achieved through a systematic search over a predefined grid of hyperparameter values, using K-Fold cross-validation to evaluate each combination. The hyperparameter search process begins with the initialization step. Next, the grid search step iterates over all combinations of the hyperparameters, and the process continues until, at the end, the best hyperparameters of the model are identified.

In the model training and evaluation step, the model is trained and evaluated using the function `TrainAndEvaluate` with K-Fold cross-validation for each combination of hyperparameters. The average validation loss, *avg_val_loss*, is computed from the validation losses obtained from the K-Folds using the formula:

$$avg\_val\_loss = \frac{1}{k}\sum_{i=1}^{k} val\_loss_i$$

During the best parameters update step, if $avg\_val\_loss < best\_score$, the variable *best_score* is updated to *avg_val_loss*. The variable *best_params* is then set to the current combination of hyperparameters, which includes $hidden\_channels1$, $hidden\_channels2$, and *output_dim*. Finally, after evaluating all combinations, the best parameters and the best score are returned. In summary, the hyper parameter search process identify the best set that minimizes the validation loss as described above, and ~~In summary, the hyperparameter search process systematically explores various combinations of hyperparameters using K-Fold cross-validation to identify the best set that minimizes the validation loss. This approach~~ ensures the selection of optimal hyperparameters for the given model.

4) Validation using K-Fold Cross-Validation Approach: The training process involves K-Fold cross-validation to ensure robust evaluation. The models are trained on the dataset using backpropagation and the Adam optimizer. Evaluation metrics include mean absolute error (MAE), mean squared error (MSE), root mean square error (RMSE), and R^2 score [22]. The detailed training procedure is as follows:

First, the number of classes is determined by finding the maximum parent index in the training data. The K-Fold cross-validation is initialized with the specified number of splits, and an empty list is created to store the results from each fold. For each fold in the K-Fold split, the training and testing subsets are created from the training data list based on the indices provided by the K-Fold split. For each batch in the training loader, the optimizer gradients are zeroed, and the batch data is loaded. The model computes node embeddings from the batch data, and the parent predictor produces output from these node embeddings. The loss is computed using the criterion on the output and the batch's parent indices. The loss is then backpropagated, and the optimizer steps to update the model parameters. Finally, the training loss for the batch is added to the total training loss. After processing all batches, the average training loss is computed. The model is evaluated on the test loader using the defined evaluation metrics, and the results are appended to the list of fold results. Finally, the function returns the accumulated results from all folds, and the best split of training and test percentage is selected, which is 80% to 20%.

5) Model Implementation: Graph Neural Networks (GNNs) have emerged as a powerful tool for learning on graph-structured data. Specifically, we implemented a GNN using SAGEConv layers to learn node embeddings for predicting MST-related properties [19]. The SAGEConv layer, or GraphSAGE convolution, aggregates features from a node's local neighborhood to generate its embedding. This method allows for efficient computation and scalability to large graphs. The key steps in our Graph Neural Network (GNN) implementation involve:

1) Initializing the SAGEConv layers.
2) Forward propagation through the network to generate node embeddings.
3) Using the embeddings to predict MST-related properties.

The GNN implementation using SAGEConv layers takes a data object as input, which contains node features and edge indices. A sequential model is created, and it consists of a SAGEConv layer that takes the input channels and transforms them into hidden channels, followed by a ReLU activation function. Another SAGEConv layer takes the hidden channels and transforms them into output channels. An optimizer is created using the Adam optimization algorithm, with the model's parameters and a learning rate of 0.01. The training process runs for a specified number of epochs. During training, the model is set to training mode, and the gradients of the optimizer are zeroed. Forward propagation is performed on the input data to generate node embeddings, using the node features and edge indices as inputs to the model. The loss is calculated using a specified loss function, which compares the model's output with the target labels. Backward propagation is performed to compute the gradients, and the optimizer steps to update the model parameters. After completing the training epochs, the function returns the trained model. This model learns node embeddings and uses a fully connected neural network to predict parent-child relationships. The architecture of the GNN model is described in Algorithm 1. The `GNNModel` function is called to initialize the model with the specified layers and dimensions. The `Forward` function is called during the forward pass to compute the node embeddings.

Algorithm 1 GNN Model

```
1: function GNNModel(hidden_channels1, hidden_channels2, output_dim)
2:     $self.conv1 \leftarrow$ SAGEConv(2, hidden_channels1)
3:     $self.conv2 \leftarrow$ SAGEConv(hidden_channels1, hidden_channels2)
4:     $self.conv3 \leftarrow$ SAGEConv(hidden_channels2, output_dim)
5:     $self.linear \leftarrow$ Linear(output_dim, output_dim)
6: end function
7: function Forward(data)
8:     $x, edge\_index \leftarrow data.x, data.edge\_index$
9:     $x \leftarrow$ ReLU(self.conv1(x, edge_index))
10:    $x \leftarrow$ ReLU(self.conv2(x, edge_index))
11:    $x \leftarrow self.conv3(x, edge\_index)$
12:    $x \leftarrow self.linear(x)$
13:    return $x$
14: end function
```

Algorithm 2 Parent Predictor Model

```
1: function ParentPredictor(input_dim, hidden_dim, num_classes)
2:     $self.lin1 \leftarrow$ Linear(input_dim, hidden_dim)
3:     $self.lin2 \leftarrow$ Linear(hidden_dim, hidden_dim)
4:     $self.output \leftarrow$ Linear(hidden_dim, num_classes)
5: end function
6: function Forward(x)
7:     $x \leftarrow$ ReLU(self.lin1(x))
8:     $x \leftarrow$ ReLU(self.lin2(x))
9:     return $self.output(x)$
10: end function
```

The `GNNModel` function is called during model initialization to set up the convolutional layers (`conv1`, `conv2`, `conv3`) and the linear layer (`linear`). The `Forward` function is executed during each forward pass through the network, where it takes the input data, applies the convolutional layers with ReLU activations, and then applies the final linear transformation to produce the output node embeddings. To predict parent nodes, we implemented a Parent Predictor model (Algorithm 2). The `ParentPredictor` function initializes the model with the specified dimensions for the input, hidden layers, and output. The `Forward` function is called to compute the predictions for the parent nodes. The `ParentPredictor` function is called during model initialization to set up the linear layers (`lin1`, `lin2`, `output`). The `Forward` function is executed during each forward pass, where it takes the input node embeddings, applies the linear transformations with ReLU activations, and then produces the final output, which is the predicted parent indices for each node. These models work together to first learn the node embeddings using the GNN with SAGEConv layers and then use these embeddings to predict the parent-child relationships in the graph. **the above steps have been discussed in detail in the previous section..should we do it once again??–Please check the source file...If you feel to retain, please uncomment**

The trained model can then be used to predict MST-related properties based on the node embeddings generated during forward propagation.

6) Evaluation: We evaluate the performance of our models on a range of node sizes, comparing the GNN-based approach with traditional MST algorithms in terms of total weighted distance, data rate, and power consumption. The detailed evaluation procedure is structured to ensure a comprehensive assessment of each model's performance across different network ~~node~~ sizes. To thoroughly evaluate the performance of our models on varying network ~~node~~ sizes, we compare ~~compared~~ the GNN-based approach with traditional MST algorithms. The focus was on metrics such as total weighted distance, data rate, and power consumption. ~~The evaluation process systematically tests each model across a range of node sizes and compares their performance against the Kruskal MST algorithm, serving as a benchmark.~~the previous statement is repeated The entire evaluation procedure is encapsulated in Algorithm 3. This algorithm outlines the step-by-step process for evaluating the models, ensuring a systematic and recursive ~~repeatable~~ method for performance assessment.

Algorithm 3 Evaluate Models

```
 1: function EvaluateModels(models, best_params, num_nodes_range)
 2:     $test\_results \leftarrow$ dictionary of result lists
 3:     for all $model\_class \in models$ do
 4:         $model\_name \leftarrow model\_class.\_\_name\_\_$
 5:         Initialize $test\_results$ lists for $model\_name$ data rate and power consumption
 6:     end for
 7:     for all $num\_nodes \in num\_nodes\_range$ do
 8:         $data, initial\_BDIx\_tree, full\_graph, kruskal\_mst \leftarrow$ CreateSampleData(num_nodes)
 9:         $test\_results['num\_nodes'].append(num\_nodes)$
10:         $mst, kruskal\_weight \leftarrow$ KruskalMST(full_graph)
11:         $kruskal\_data\_rate, kruskal\_power\_loss \leftarrow$ CalculateMSTMetrics(full_graph, kruskal_mst)
12:         Append $kruskal\_weight$, $kruskal\_data\_rate$, and $kruskal\_power\_loss$ to $test\_results$
13:         for all $model\_class \in models$ do
14:             $model\_name \leftarrow model\_class.\_\_name\_\_$
15:             Retrieve model-specific parameters and instantiate model
16:             Construct $predicted\_tree$ from $predicted\_parents$
17:         end for
18:     end for
19:     return $test\_results$
20: end function
```

Our proposed methodology combines both traditional and modern machine learning techniques to tackle the problem of ~~optimising~~ MSTs optimisation, leveraging the strengths of both approaches. ~~for comprehensive analysis and comparison.~~

Table II: Simulation Parameters

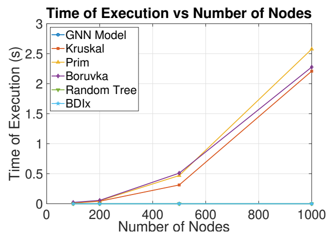
Parameter	Value
Frequency	2.4 GHz
Transmit Power	20 dBm
Gain	2 dB
Noise Figure	10 dB
Bandwidth using WiFi Direct	1 MHz
Path Loss Exponent	2
Noise Floor Level	-174 dBm/Hz

V. Simulated Results and Analysis

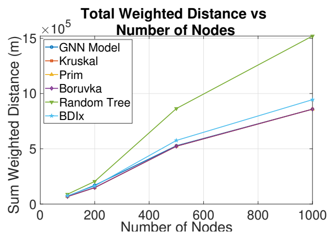
This section provides the description of the metrics that are used in the examination. It also evaluates the performance of the proposed deep learning-enhanced optimization of the D2D communication model with traditional approaches using specific network metrics. Moreover, it examines the impact of NN and GNN estimation over the ML metrics (i.e., Train Loss, Validation Loss, MAE, MSE, RMSE, R^2, adj R^2[2]) measured in terms of weighted distance in various scenarios. The training and testing sets consist of 8000 and 2000 samples, respectively. The simulation parameters are presented in Table II.

A. Results and Analysis Regarding Network Metrics

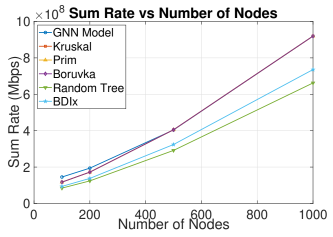
This section presents an in-depth analysis of the network metrics for optimizing D2D communication topologies using various algorithms, including a novel GNN-based approach. The metrics include time of execution, total weighted distance, data rate, and total power consumption. The results are based on the number of nodes in the network.



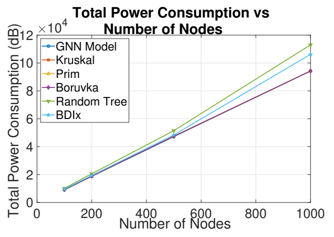
(a) Time of Execution vs Number of Nodes



(b) Total Weighted Distance vs Number of Nodes



(c) Data Rate vs Number of Nodes



(d) Total power consumption vs Number of Nodes

Figure 2: Comparison of network metrics for different algorithms as the number of nodes increases.

1) Time of Execution: Figure 2a shows the time of execution for different algorithms as the number of nodes increases. The GNN model consistently demonstrates lower execution times compared to most other algorithms, particularly for larger networks. This efficiency is crucial for real-time applications where rapid computation is necessary. The GNN model's ability to quickly compute optimal topologies stems from its deep learning-based approach, which effectively generalizes from training data to make fast predictions during {can inference be replaced with an appropriate word} inference. Kruskal's algorithm shows a similar trend but with slightly higher execution times, making it another viable option for

[2]Train Loss: Measures how well the model fits the training data. A lower value indicates a better fit. Validation Loss: Evaluates model performance on unseen validation data. Lower values signify better performance. MAE (Mean Absolute Error): Averages the absolute differences between predicted and actual values. Indicates model accuracy in terms of average error. Lower values are better. MSE (Mean Squared Error): Averages the squared differences between predicted and actual values. Penalizes larger errors more than smaller ones. Lower values are preferable. RMSE (Root Mean Squared Error): Square root of the MSE. Maintains the same unit as the target variable. R^2 (R-squared): Represents the proportion of variance in the target variable explained by the model. Ranges from 0 to 1. Higher values indicate better explanatory power. Adj R^2 (Adjusted R-squared): Adjusts the R^2 value for the number of predictors in the model [22].

real-time applications. Prim's algorithm, however, has the highest execution time among all the compared algorithms, indicating it may not be suitable for real-time applications in large networks. Both Boruvka and Random Tree algorithms have higher execution times than the GNN model but are still lower than Prim's. The BDIx algorithm shows competitive execution times, although not as low as the GNN model.

2) Total Weighted Distance: Figure 2b illustrates the total weighted distance for different algorithms. The GNN model maintains a lower total weighted distance compared to the Random Tree algorithm, indicating its effectiveness in optimizing the paths within the network. This metric is crucial for minimizing latency and ensuring efficient data routing. The GNN model's ability to learn from graph structures allows it to make intelligent decisions about path optimization, leading to lower weighted distances. Kruskal and Prim algorithms perform comparably well, with slightly higher total weighted distances than the GNN model. Boruvka also performs well but is not as consistent as the GNN model. The Random Tree algorithm shows significantly higher weighted distances, indicating poor path optimization. The BDIx algorithm performs better than Random Tree but not as well, as the GNN model.

3) Sum Rate: Figure 2c presents the data rate for different algorithms. The GNN model achieves a high data rate, close to that of Boruvka's algorithm, and significantly better than the Random Tree algorithm. This high data rate indicates the GNN model's efficiency in maximizing network throughput, which is essential for supporting bandwidth-hungry ~~high-bandwidth~~ applications. The model's capability to understand and optimize the network's data flow patterns contributes to this high performance. Boruvka's algorithm achieves a slightly higher data rate than the GNN model, but both are very close in performance. Kruskal and Prim algorithms perform well but have lower data rates compared to the GNN model. The Random Tree algorithm, on the other hand, shows a lower data rate, reflecting its inefficiency in optimizing data transmission paths. The BDIx algorithm performs comparably to the GNN model, indicating efficient data routing.

4) Total power consumption: Figure 2d depicts the total power consumption for different algorithms. The GNN model exhibits lower power consumption compared to the Random Tree algorithm, demonstrating its energy efficiency. Minimizing power consumption is critical for extending the lifespan of network devices and reducing operational costs. The GNN model's optimization process takes into account not only the network's connectivity but also its power consumption patterns, resulting in lower total power consumption. Kruskal and Prim algorithms perform comparably to the GNN model in terms of power consumption, indicating similar levels of energy efficiency. Boruvka has a slightly higher power consumption but remains efficient. The Random Tree algorithm shows the highest power consumption, indicating inefficiency in power management. The BDIx algorithm performs well, with power consumption close to the GNN model, highlighting its effectiveness in minimizing energy consumption.

The content seems repeating hence commented–refer to source file

B. Results and Analysis Regarding ML Metrics

The dataset provided includes various metrics for evaluating a machine learning model. Here is an in-depth analysis of the results:

Table III: Model Evaluation Metrics

Train Loss	**Val Loss**	**MAE**	**MSE**	**RMSE**	R^2	**Adj** R^2
0.0338	0.0044	0.0143	0.9855	0.9928	0.9661	0.9661

According to Table III, the training loss is 0.0338, while the validation loss is significantly lower at 0.0044. This indicates that the model performs better on the validation dataset than the training dataset. The MAE is 0.0143, showing the average magnitude of errors in a set of predictions without considering their direction. the above sentence is incomplete..Pls chk The MSE is 0.9855, indicating relatively low average squared errors. The RMSE of 0.9928, derived from the MSE, indicates the model's error magnitude is just below 1. The R^2 value is 0.9661, indicating that 96.6% of the variance in the dependent variable is predictable from the independent variables. The adjusted R^2 value is nearly identical at 0.9661, reinforcing the model's efficacy. ~~high explanatory power~~

The model exhibits excellent performance on both the training and validation sets, as indicated by the low loss values and high R^2 values. The error metrics (MAE, MSE, RMSE) suggest that our ~~the~~ model's predictions have very low errors. The high R^2 and adjusted R^2 values indicate that our ~~the~~ model has ~~explains~~ a significant portion of the variance in the target variable, showcasing its robustness and predictive power.

VI. Conclusions and Future Work

This paper presents a comprehensive evaluation of a deep learning-enhanced optimization of the D2D communication model compared to traditional methods. The results show that the GNN model, along with Kruskal and BDIx algorithms, generally performs better across various metrics compared to Prim, Boruvka, and Random Tree algorithms. The GNN model and Kruskal algorithm offer balanced performance with low execution times, optimized weighted distances, high data rates, and reduced power consumptions, making them suitable for real-time applications in large D2D networks. The GNN model has the lowest execution times, making it highly efficient for real-time applications, while Kruskal's algorithm also shows competitive performance in this regard. In terms of total weighted distance, the GNN model outperforms other algorithms, indicating superior path optimization, which is crucial for minimizing latency and ensuring efficient data routing. Regarding data rate, the GNN model achieves high data rates, indicating its efficiency in maximizing network throughput, essential for high-bandwidth applications. Moreover, the GNN model demonstrates lower power consumption, showcasing its energy efficiency and effectiveness in minimizing power consumption. These findings suggest that the GNN model is not only effective but also efficient for deployment in modern communication networks where real-time data processing and energy efficiency are paramount.

The promising results of the GNN model pave the way for several future research directions. One area is scalability studies, investigating the GNN model's performance in larger network scenarios to ensure robustness. Real-world deployments in D2D communication systems can validate its practical applicability. Additionally, hybrid models combining GNN with other techniques or traditional algorithms ~~could enhance~~ can be explored for possible enhancement in performance and efficiency. ~~Research on reducing power consumption and enhancing energy efficiency in large-scale networks is crucial.~~{the above sentence seems repeating hence strike-through} Adaptive learning mechanisms for dynamic network adjustments and robust security protocols within the GNN framework can also be delved into. ~~are also essential~~ Addressing these areas will further enhance the capabilities of deep learning-aided models in modern communication networks.

Acknowledgement

This work has received funding from the European Union's Horizon 2020 Research and Innovation Programme under Grant Agreement No. 739578, the ADROIT6G project of the SNS-JU under Grant Agreement No. 101095363, and the Government of the Republic of Cyprus through the Deputy Ministry of Research, Innovation and Digital Policy.

References

[1] S. Kim, "Optimal wireless resource allocation with random edge graph neural networks," *IEEE Transactions on Wireless Communications*, vol. 19, no. 1, pp. 11–23, 2024.

[2] I. Ioannou, V. Vassiliou, C. Christophorou, and A. Pitsillides, "Distributed artificial intelligence solution for d2d communication in 5g networks," *IEEE Systems Journal*, vol. 14, no. 3, pp. 4232–4241, 2020.

[3] P. W. Battaglia, J. B. Hamrick, V. Bapst, A. Sanchez-Gonzalez, V. Zambaldi, M. Malinowski, A. Tacchetti, D. Raposo, A. Santoro, R. Faulkner, C. Gulcehre, F. Song, A. Ballard, J. Gilmer, G. E. Dahl, A. Vaswani, K. R. Allen, C. Nash, V. Langston, C. Dyer, N. Heess, D. Wierstra, P. Kohli, M. Botvinick, O. Vinyals, Y. Li, and R. Pascanu, "Relational inductive biases, deep learning, and graph networks," *Journal of Big Data*, vol. 6, no. 1, 2020.

[4] S. Kim, "A survey of intelligent end-to-end networking solutions: Integrating graph neural networks and deep reinforcement learning approaches," *Electronics*, vol. 13, no. 5, pp. 345–359, 2024.

[5] H. Cai, V. W. Zheng, and K. C.-C. Chang, "A comprehensive survey on graph neural networks," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 31, no. 11, pp. 16–37, 2020.

[6] A. S. Rao and M. P. Georgeff, "Bdi agents: From theory to practice," *Proceedings of the First International Conference on Multi-Agent Systems (ICMAS-95)*, pp. 312–319, 1995.

[7] R. C. Prim, "Shortest connection networks and some generalizations," *The Bell System Technical Journal*, vol. 36, no. 6, pp. 1389–1401, 1957.

[8] J. B. Kruskal, "On the shortest spanning subtree of a graph and the traveling salesman problem," *Proceedings of the American Mathematical Society*, vol. 7, no. 1, pp. 48–50, 1956.

[9] J. Zhou, G. Cui, S. Hu, Z. Zhang, C. Yang, Z. Liu, and M. Sun, "Graph neural networks: A review of methods and applications," *AI Open*, vol. 1, pp. 57–81, 2020.

[10] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and P. S. Yu, "A comprehensive survey on graph neural networks," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 31, no. 11, pp. 4–24, 2020.

[11] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How powerful are graph neural networks?" in *Proceedings of the International Conference on Learning Representations (ICLR)*, 2019.

[12] V. Morreale, S. Bonura, G. Francaviglia, M. Cossentino, and S. Gaglio, "Practionist: A new framework for bdi agents," R&D Laboratory - Engineering Ingegneria Informatica S.p.A, Tech. Rep., 2024.

[13] W. L. Hamilton, R. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.

[14] J. Zhou, G. Cui, S. Hu, Z. Zhang, C. Yang, Z. Liu, and L. Wang, "Graph neural networks: A review of methods and applications," *arXiv preprint arXiv:1812.08434*, 2020.

[15] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*. MIT press, 2009.

[16] R. E. Tarjan, *Data Structures and Network Algorithms*. Society for Industrial and Applied Mathematics, 1983.

[17] O. Borůvka, "O jistém problému minimálním," *Práce Moravské Přírodovědecké Společnosti*, vol. 3, pp. 37–58, 1926.

[18] H. N. Gabow, Z. Galil, T. Spencer, and R. E. Tarjan, "Efficient algorithms for finding minimum spanning trees in undirected and directed graphs," *Combinatorica*, vol. 6, no. 2, pp. 109–122, 1986.

[19] W. L. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *Advances in Neural Information Processing Systems*, 2017, pp. 1024–1034.

[20] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *International Conference on Learning Representations (ICLR)*, 2017.

[21] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, "Optuna: A next-generation hyperparameter optimization framework," in *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, 2019, pp. 2623–2631.

[22] I. Ioannou, M. Savva, M. Raspopoulos, C. Christophorou, and V. Vassiliou, "Revolutionising iot network security by assessing ml localisation techniques against jamming attacks," 04 2024.