Enabling Real-Time Communication in Multi-Agent Systems: A Graph Neural Network Based Approach

Paula Stocco, Alan Hesu, and Steven J. Spencer *

Unmanned Systems and Autonomy, Sandia National Laboratories, Albuquerque, USA
Email: stoccop@sandia.gov (P.S.); ahhesu@sandia.gov (A.H.); sjspenc@sandia.gov (S.J.S.)
*Corresponding author

***Abstract*—Global connectivity enables effective coordination in Multi-Agent Systems (MAS). Solving these connection problems under hardware constraints is an NP-hard non-Euclidean Degree Constrained Minimum Spanning Tree (DCMST) problem. Prior MAS controllers coordinate team movement for task completion and collision avoidance; some considering Line-of-Sight (LOS) maintenance but prioritizing flexibility over guarantees. Evolutionary Algorithms (EA) have been shown to find good solutions for DCMST, but their performance degrades with larger populations required to support a large MAS. We present a method based on edge graph attention networks, trained offline to reduce online computation times. Empirical comparisons with greedy polynomial-time solvers and EA show that our method leverages latent graph information to consistently find constraint-satisfying solutions in less time.**

***Keywords*—multi-agent system, connectivity maintenance, minimum spanning tree, graph neural network, evolutionary algorithm**

I. Introduction

Effective communication in cooperative mobile robotic systems is essential to ensure agents can efficiently achieve the goals of the system and operators maintain situational awareness [1]. Many researchers and applications use radios with mesh networking capabilities or mobile ad-hoc networks to communicate between agents. These schemes work quite well when the radio signal between agents is strong. But when agents move far apart or behind radio frequency blocking obstacles, communication can suffer.

When viewing a Multi-Agent Systems (MAS) as a graph with agents as vertices and communication links as edges, one way to maintain communication is by constraining agent motion, which prevent network graph disconnects [2, 3]. However, this limits the system if the graph must remain fully connected. Thus, finding a spanning tree, preferably a Minimum Spanning Tree (MST), and allowing that tree to change with time becomes an important aspect of the system controller. Furthermore, platforms using optical transceivers or directionally gained antennas can only maintain a limited number of connections at any time, in effect adding degree constraints to the graph vertices. The MAS can also be heterogeneous, supporting a mixture of more complex agents with more antennas and thus higher degree constraints in combination with simple agents with lower degree constraints. Here, we explore methods for solving the heterogeneous Degree Constrained Minimum Spanning Tree (DCMST) problem for multi-agent systems.

Previous exact methods, such as linear programming, for solving Combinatorial Optimization (CO) problems in large spaces have proven too slow for deployment on hardware for systems with more than five agents. This problem requires solvers that optimize non-Euclidean edge weights and multi-objective global cost functions while ensuring viable solutions without over-constraining the system or hindering group movement.

We address the challenge of solving large problem instances by proposing a method to solve for valid graph connections while optimizing non-Euclidean optimality measures. Our problem and solution contribute to current MAS and graph learning research. First, we extend previous DCMST Evolutionary Algorithm (EA) solvers [4, 5] to optimize connections in this MAS problem and find solutions in a fraction of the exact solver time. Second, we use Supervised Learning (SL) with a Graph Neural Network (GNN) to rapidly approximate these heuristic EA solutions, further reducing computation time and empirically showing generalization to larger group sizes and new environments.

II. Literature Review

Narula and Ho introduce the DCMST problem in 1980, noting that linear integer programming approaches become quickly overwhelmed as the number of nodes increases [6]. A related problem, the Traveling Salesman Problem (TSP), a well-studied MST problem with a maximum degree constraint of two is NP-hard even in the Euclidean case [7]. Given the similarities between the TSP and DCMST, prior approaches for solving the TSP offer opportunities for adaptation to the DCMST problem.

Manuscript received February 15, 2025; revised February 25, 2025; accepted May 9, 2025; published August 26, 2025.

Previous research studies how Machine Learning (ML) can be applied to solve NP-hard CO problems in practical time. Bengio et al. categorize these methods broadly into SL, which imitates expert decisions, experiential learning, such as Reinforcement Learning (RL), or hybrid methods integrating ML into existing solvers [8]. Mele *et al*. [9] provide a catalogue of and insight into SL and RL methods for TSP specifically. SL requires large datasets for training, while RL and unsupervised methods struggle with constraints and require long training times.

GNNs have emerged as particularly suited for solving CO problems, as many such problems can be encoded as network data [10, 11]. Peng *et al*. [10] divide these GNN-based CO methods into autoregressive, which iteratively constructs solutions, and non-autoregressive methods, which predicts a solution but requires a decoding step. Notable end-to-end autoregressive RL approaches include Bello *et al.* [12] and Dai *et al*. [13], with both Kool *et al*. [14] for TSP and recently Shi *et al*. [15] for DCMST using REINFORCE [16], though Shi *et al*. [15] test only with a degree constraint of two. Joshi *et al*. [17] is a compelling example of a non-autoregressive method for TSP. Subsequent work improves out-of-distribution generalization, comparing different GNN pipelines and key design choices [18]. Recent advances focus on constructing GNN architectures that execute the individual steps of classic algorithms, including MST solvers [11, 19, 20].

Cappart *et al*. [11] emphasize that while exact solvers, such as integer linear programming, can guarantee optimality, and approximation algorithms or metaheuristics can find feasible solutions for large, intractable problems, GNNs offer fast, scalable solutions where other methods struggle. The potential of GNNs lies in tackling even the most complex problems, extending beyond TSP, which can already be solved for instances of up to 10^7 nodes with less than a 1% optimality gap [11, 21].

Other works have explored the application of learning specifically for MAS problems. Huang *et al*. [22] and Lyu *et al*. [23] explore the use of Deep Deterministic Policy Gradient (DDPG) for connectivity maintenance, though they pose connectivity as a term in the reward function as opposed to an inviolable constraint. Li *et al*. [24] instead impose these constraints using Constrained Policy Optimization (CPO). Li *et al*. [25] leverage supervised GNN learning for multi-agent path finding, but decentralized localized communication rather than the enforcement of global connectivity constraints was pursued. Yang *et al*. [26] use a control barrier function to create a weighted graph based on MAS criteria, and then an MST solver to find the final acceptable connectivity graph. However, all these works do not explore our specific Point of Interest (POI) objectives and inviolable node degree constraints.

Our work presents a non-autoregressive, supervised learning approach for a novel non-Euclidean, multi-objective heterogeneous MAS DCMST problem. We move away from prior unsupervised, end-to-end learning approaches, which may be difficult to train to convergence. Instead, we focus on supervised graph learning because we can use our EA algorithm as an oracle for training examples, and we enforce hard constraints via a centralized MAS controller.

III. Preliminaries

A. Combinatorial Optimization Graph Problems

The following defines a graph based combinatorial optimization problem.

Definition: Given a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with vertices $\mathcal{V}$ and edges $\mathcal{E}$ and cost function $c(\cdot) \forall G = (\mathcal{V}, \mathcal{E}^C) \in \mathcal{G}$, $\mathcal{E}^C \subseteq \mathcal{E}$ for each spanning tree subgraph, a CO problem is to find the optimum value of c or the corresponding set of edges $\mathcal{E}^C$ that produces that optimum value [10].

The DCMST problem seeks a selection of edges $\mathcal{E}^C$ from the graph $\mathcal{G}$ such that the resulting subgraph G minimizes the objective function. In the following sections, we define the constraints and objective function for the MAS and some preliminaries on the chosen GNN based algorithms. Table I is a quick reference for symbol notations.

TABLE I. Symbols

Symbol	Representation
$\mathcal{G}$	Starting graph that contains all possible connections
G	Spanning tree (subset of $\mathcal{G}$)
$\mathcal{V}$	Vertex set
$\mathcal{E}$	Edge set
n	Number of agents
R_i	Agent i
x_i	State (position) of R_i
a_{ij}	Edge weight between R_i and R_j
$\bar{l}_{ij}$	Euclidean distance between R_i and R_j
ρ	Max effective communication range
ρ_0	Min communication degradation distance
γ_{ij}	Communication range strength factor
d_{min}^{β}	Min obstacle radius
d_{max}^{β}	Max obstacle radius
β_{ij}	Obstacle occlusion strength factor
A	Adjacency matrix
D	Degree matrix
L	Laplacian
λ_2	Algebraic connectivity (Fiedler value)
v_2	Fiedler vector
V	Control value function
Δx_i	Desired change in agent position
u_i	Agent control input
h_i	Node i features
e_{ij}	Edge features between nodes i and j
F	Learned weight vector
h_i'	Hidden node i features
f_{ij}'	Hidden edge features between i and j
α_{ij}	Attention score between i and j

B. Line-of-Sight Connectivity Maintenance

We begin with N robots, whose goal is to reach various POIs, specified by a high-level controller or system operator, while avoiding obstacles and maintaining connectivity. To account for obstacles that may disrupt communication, we build on prior work that considers Line-of-Sight (LOS) between agents.

Gradient-based control equations calculated from graph properties have been practically demonstrated on

unmanned aerial and ground vehicles [2]. Other work has used control inputs based on control barrier functions, which act as constraints [26, 27]. Shetty *et al*. [3] use a decentralized LOS controller incorporating sensor and motion uncertainty with confidence regions, connectivity maintenance, and obstacle avoidance using the graph Fiedler value. Our approach to LOS connectivity maintenance for MAS is similar, using the gradient of the constraints in a control barrier function [2, 3].

The system of agents can be defined as a weighted graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V} = \{R_1, R_2, \dots, R_n\}$ denotes the set of n agents, and $\mathcal{E}$ denotes the set of edges representing possible communication links between agents. $\mathcal{E}$ only contains the set of edges where the edge weight a_{ij}, is greater than zero. The state of the i^{th} agent is defined by the state vector x_i, while the weight a_{ij} of each edge between agents R_i and R_j is defined by

$$a_{ij} = \gamma_{ij}\beta_{ij}, \tag{1}$$

where γ_{ij} and β_{ij} represent the connection strength factors for communication range and obstacle occlusion, respectively.

We model the communication range factor as

$$\gamma_{ij} = \begin{cases} 1 & 0 \le \bar{l}_{ij} \le \rho_0 \\ 1 - \frac{\bar{l}_{ij} - \rho_0}{\rho - \rho_0} & \rho_0 < \bar{l}_{ij} \le \rho \\ 0 & \bar{l}_{ij} > \rho \end{cases}, \tag{2}$$

where $\bar{l}_{ij}$ is the Euclidean distance between the positions of R_i and R_j, and ρ and ρ_0 are constants that define the maximum effective communication range and the distance at which communication begins to degrade, respectively, such that $0 < \rho_0 < \rho$. Agents closer together than ρ_0 have full communication strength, those further than ρ apart cannot communicate, and the range factor falls off linearly in between.

The obstacle occlusion factor is modeled as

$$\beta_{ij} = \begin{cases} 1 & \bar{l}_{ij,o} > d^{\beta}_{max} \\ \frac{\bar{l}_{ij,o} - d^{\beta}_{min}}{d^{\beta}_{max} - d^{\beta}_{min}} & d^{\beta}_{min} < \bar{l}_{ij,o} \le d^{\beta}_{max} \\ 0 & \bar{l}_{ij,o} \le d^{\beta}_{min} \end{cases}, \tag{3}$$

where $\bar{l}_{ij,o}$ is the distance from the closest obstacle o to the edge e_{ij}, and d^{β}_{min} and d^{β}_{max} are constants that define the minimum and maximum obstacle occlusion ranges such that $0 < d^{\beta}_{min} < d^{\beta}_{max}$. When the edge is further than d^{β}_{max} from any obstacle it has full communication strength, edges closer than d^{β}_{min} are broken, and the occlusion factor falls off linearly in between.

In addition to these edge weight equations, the gradient-based approach incorporates algebraic measures of connectivity. From graph G, the Laplacian matrix can be calculated as $L = D - A$, where the adjacency matrix, A, is defined such that $A_{ij} = a_{ij} \forall (i,j) \in \mathcal{E}^C, i \neq j$ and the degree matrix, D, is a diagonal matrix such that $D_{ij} = \sum_{j=1}^{n} a_{ij} \; \forall i \in N, i = j, (i,j) \in \mathcal{E}^C$. As observed in prior work, the second smallest eigenvalue of the graph Laplacian matrix, $\lambda_{2,G}$, is greater than 0 if and only if the graph is connected.

The Fiedler value, its corresponding eigenvector, v_2, and corresponding edge weights are all used to calculate the desired agent motion, Δx_i, as follows.

$$\Delta x_i = -\frac{\partial V(\lambda_{2,G})}{\partial \lambda_{2,G}} \sum_{j=1}^{n} \nabla_{x_i}(a_{ij})(v_{2i} - v_{2j})^2, \tag{4}$$

where $\nabla_{x_i}(a_{ij})$ is the gradient of a_{ij} with respect to x_i, v_{2i} and v_{2j} are the i^{th} and j^{th} components of the Fiedler vector, and V is the value function

$$V = \begin{cases} \coth(\lambda_{2,G} - \epsilon) & \lambda_{2,G} > \epsilon > 0 \\ 0 & \text{otherwise} \end{cases}. \tag{5}$$

Finally, Δx_i is converted to a control input u_i for each agent via the kinematic motion model $u_i = B_i \Delta x_i$.

C. Graph Attention Layer for Node and Edge Features

Eq. (1) describes how each edge weight of the MAS graph captures communication strength and obstacle proximity. However, that information alone does not capture the full MAS state and does not account for the state of surrounding agents. GNNs can calculate a latent representation incorporating neighborhood information.

The EGAT architecture, introduced with the Rossman Toolbox [28], uses both node and edge features while leaving their representations separate. This is particularly valuable for making decisions on graph edges, unlike other GNN architectures that either rely solely on node information or encode edges into latent node representations. EGAT uses edge information and latent edge representations, reasoning directly at the edge level. We refer the reader to the Rossman supplementary materials for details with mathematical formulations and provide an overview of how EGAT convolutional layers (EGATConv) are implemented in Deep Graph Library [29]. Fig. 1 visualizes this formulation.

(1) The node and edge features, h_i, h_j, and e_{ij} respectively, between a principal node, i, and a secondary node, j, are concatenated and then linearly transformed by the learned weight matrix W. The resulting vector is then passed through the leaky rectified linear unit function to create the new latent edge feature f'_{ij}.

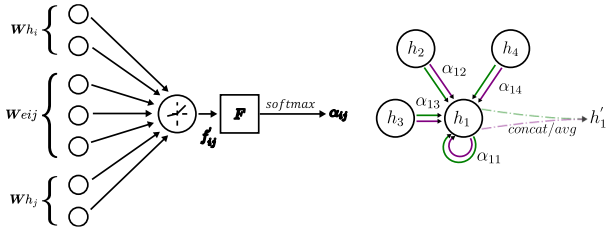

Fig. 1. Diagram of the information aggregation structure for the PyTorch DGL EGATConv layer with two attention heads and one neighboring node. Left: single-head attention. Right: multi-head attention; different colors represent the different attention calculations. Adapted from [30].

(2) The new latent edge features are passed through F, a learned weight vector, then normalized using softmax to produce the attention score α_{ij}.
(3) The attention scores are further weighted and multiplied by the secondary node features. When concatenated or averaged for all neighboring nodes, this results in the new node feature h_i' using multi-headed and self-attention mechanisms.

Following multiple EGATConv layers, the latent edge features, referred to as edge scores here, are used for edge selection. The full architecture and method are detailed in Section IV.

IV. Methodology

A. Degree Constrained Connectivity Maintenance

In addition to always maintaining connectivity, the MAS optimization problem includes additional constraints and objectives. First, point-to-point communication capacity of each agent is modeled as $\delta_i \geq 1$, a maximum degree for each node in G. Second, the POI objectives are added by introducing a second control input term $\tilde{u}_i$ for each agent. Accordingly, the DCMST problem is defined as the selection of the edges $\mathcal{E}^C$ from the graph $\mathcal{G}$ to find G that minimizes the objective function

$$u^* = \arg\min_{\mathcal{G},u} \sum_{i=1}^{n} \left\| \hat{u}_i + \hat{\tilde{u}}_i \right\|_2, \quad \text{s.t. } |E_i| \leq \delta_i \quad \forall i \in 1, \dots, N \tag{6}$$

where $\hat{u}_i$ and $\hat{\tilde{u}}_i$ are the normalized control inputs for LOS maintenance and POI tracking, respectively. Each G denotes a potential viable spanning tree such that the degree $|E_i|$ of each node i for all N nodes satisfies that node's degree constraint.

B. Evolutionary Algorithm

Finding exact solutions to Eq. (6) is computationally intractable for large MAS due to the NP-hard property of the DCMST problem. One approach to finding approximate solutions is to use an EA [31]. In addition to being used as an online computational tool for smaller MAS, we use the EA as an approximator of an oracle to generate supervised training samples offline.

The EA fitness function is as defined in Eq. (6), such that the EA selects for an optimal spanning tree from a population of individual potential spanning tree candidates. Edge set encoding, recombination, and mutation operators from [4] are incorporated to solve the DCMST problem. To initialize each individual in the population, the constraint that a possible edge must have $a_{ij} > 0$ is temporarily lifted, and a randomized Kruskal's algorithm [32] is used to select from any edge between two nodes while maintaining the degree constraint for each node. A set of tree reduction heuristics [33] is used to eliminate potential nonviable solutions up front.

While the EA may select individuals with connections between any two nodes, a penalty directs the EA to favor constraint satisfying individuals. The penalty constant ϕ is used, where each nonviable edge in the individual is penalized and added to the resulting fitness value:

$$\Phi = \sum_{(i,j)\in\mathcal{E}^C} \phi e(a_{ij}) \tag{7}$$

where $e(a_{ij}) = 0$ if $a_{ij} = 0$ and $e(a_{ij}) = 1$ if $a_{ij} > 0$. At each time step $t_i > 0$, the EA is invoked for a set number of generations, starting with the resulting population from the previous time step t_{i-1} and selecting the most fit individual after the final generation.

C. Graph Neural Network with Decoding

The neural network input is all node and edge features of the initial graph $\mathcal{G}$,which contains all possible edge connections. Each node represents an agent and carries attributes for its position and total number of other agents potentially connected to it. Each edge represents a physically possible agent connection, with a_{ij} as its feature. The first EGATConv layer computes new latent node features similarly to GATConv [30], but with attention scores calculated using edge features in addition to node features. Significantly, EGATConv also computes new latent edge features using both node and edge information, which are preserved and later reused in subsequent layers.

Following the first attention layer, a dropout mechanism is applied for regularization, mitigating the risk of overfitting by preventing the model from relying too heavily on any single feature. The hidden features are independently concatenated for nodes and edges and then input to the second EGATConv.

The output latent edge features from the second EGATConv layer are averaged across the multiple attention heads such that each edge representation matches the input dimensionality of the final fully connected layer. This Multi-Layer Perceptron (MLP) takes in the processed edge features and outputs logits. The network is trained using a binary cross entropy loss function, and the normalized logits are treated as scores and used as edge priority ordering for which edge should be chosen.

D. Decoders

A decoding step is required for our non-autoregressive method to produce a valid graph from edge predictions. *Decoding* refers to a post-processing step that converts a sorted list of edges into a valid graph. Prim's [34] and Kruskal's algorithms [35] are polynomial-time algorithms for the unconstrained MST problem. They are used as greedy second stage decoders for this DCMST problem.

Prim's algorithm starts with the best edge, the first in the given list, and adds edges connecting to existing graph vertices, choosing from this subset the one highest on the list. Thus, an initial tree is grown until complete. In Kruskal's algorithm, any edge can be added regardless of whether its nodes already exist in the graph, such that smaller trees are expanded until complete [32]. For both, the node degree constraints are checked at each iteration.

The abbreviations GNN-K and GNN-P refer to solvers that use the GNN's output edge scores for Kruskal's and Prim's edge ordering, respectively. These are compared to

Kruskal's and Prim's algorithms applied directly to edge weights (Eq. (1)). Fig. 2 highlights differences in edge ordering and solutions.

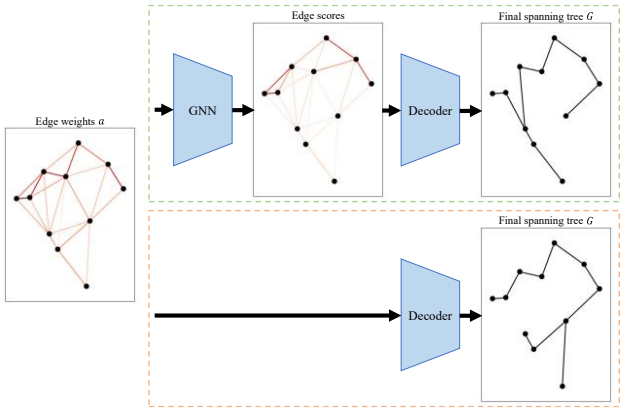

Fig. 2. Algorithm steps showing edge rankings being used to decode a valid graph. The upper path shows the GNN providing edge scores as rankings and the resultant graph, and the lower path shows the edge weights being used as rankings directly.

If a solution cannot be found from the initial edge list ordering, the algorithm restarts with a modified list, placing the second-highest edge as the first edge selected and preserving the order of the remaining edges. Reordering lists repeatedly causes them to progressively deviate from the original configuration. The number of attempts performed before finding a successful solution is recorded, as well as whether the algorithm is unable to find a successful solution within the predefined maximum number of restarts. After the maximum iterations, the solver falls back to using the EA to produce a valid solution. This approach provides a rigorous comparison between the efficacy of the GNN-generated edge score list ordering versus using the edge weights alone.

V. Experiments

The MAS testbed is a simulation of a two-dimensional environment, run for a fixed duration with a number of obstacles and POI objectives, examples of which are shown in Fig. 3. Obstacles, POIs and one agent (the base station) remain stationary. All mobile agents have a unicycle kinematic model. Over several trials, the initial positions of the agents, obstacles, and POIs are uniformly sampled from their respective regions in the environment. When one POI is reached by an agent based on a proximity threshold, a new one is spawned in a random location to maintain the maximum POI number of ten. Two different field setups are used referred to as low difficulty and high difficulty. Experiments were conducted with teams of either 10 or 100 agents. Teams either had a uniform degree constraint of two or were heterogeneous with 75% of agents limited to two connections and 25% limited to three connections. Details of the simulated environment are described in Table II.

The GNN model was trained on EA solutions for a 100 agent MAS simulation, totaling 6,040 samples (40 trials of 60 s with 0.4 s timesteps). The simulations used a low-difficulty field setting with a maximum of two connections per agent. The GNN and MLP was trained using a binary cross entropy loss function using the Adam optimizer [36] for seven epochs with a batch size of one graph per batch, a learning rate of 0.001, and a dropout rate of 0.1. All model training and experimental evaluation was performed on one node of a High Performance Computing (HPC) cluster [37] with an AMD EPYC 7543P 32-Core CPU with 500 GB DRAM and an Nvidia A100 GPU with 40 GB VRAM. End-to-end training, including dataset generation, took a few days to complete.

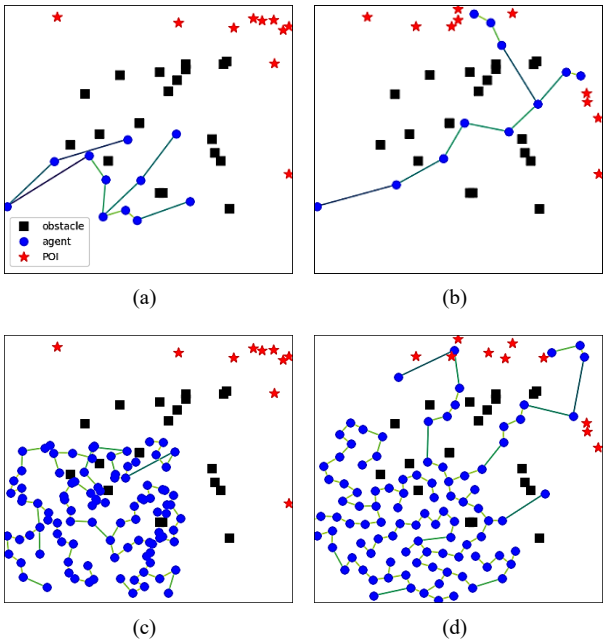

Fig. 3. Example simulated MAS environments, showing variants with 10 and 100 agents early (initial configuration) and later (evolved configuration) on in the simulation. For each environment, the agents are spawned in the lower left quadrant with the base station on the far left side, the obstacles are spawned in the central region, and POIs are spawned along the upper and right side edges. (a) Initial, 10 agents; (b) Evolved, 10 agents; (c) Initial, 100 agents; (d) Evolved, 100 agents.

TABLE II. Simulation Field Metrics of Low Difficulty and High Difficulty Situations. Locations Are Rectangles Described by the Lower Left Corner Coordinates and Upper Right Corner Coordinates

Field Setup	Low Difficulty	High Difficulty
# of Obstacles	5	20
Obs Spawn Location	[(20, 20), (65, 65)]	[(20, 20), (80, 80)]
d^{β}_{min}	0.5	2.0
d^{β}_{max}	4.0	
Agent Spawn Location	[(1, 1), (65, 65)]	
POI Spawn Location	[(1,90),(99,99)] + [(90,1),(99,99)]	
Field Size	[(0, 0), (100, 100)]	

Experiments are conducted across ten trials, with randomized environment and starting team configurations, to compare the GNN model (IV-C), the EA baseline (IV-B), and polynomial-time MST solver variants (IV-D).

Performance is evaluated using optimization time (wall time), domain-specific metrics such as POIs reached and

time to reach them (simulation time), and behavioral metrics including total kinetic energy and mechanical cost (related to velocity and acceleration). Training on smaller problem sizes reduces computational costs and time for collecting supervised samples from a slower expert. We test GNN generalizability to larger problems by comparing a model trained with 10 agent simulations (GNN_{10}) and one trained with 100 agent simulations (GNN_{100}), both trained and tested on low-difficulty problems. We also test generalizability from uniform degree constraints to heterogeneous degree constraints, since we never trained on a heterogeneous MAS.

VI. Results

Table III contains the major simulation results. The primary algorithm performance goal was to reduce the computation time with minimal impact on other system metrics. On average, the GNN-informed methods are faster than just using Prim's or Kruskal's algorithms alone.

TABLE III. Performance Metrics by Solver Types and Degree Constraint. High Difficult Field Setup with 10 Agents. GNN is Trained on 100 Agents. Mean and Standard Deviation across All Time across All Runs Shown. ▼ Indicates Lower is Better, ▲ Indicates Higher is Better. Best Results for Each Connection Type are Marked with a Dagger†

Solver Type	Optimization Time [s] ▼	% Fail ▼	Attempts Required ▼	POI Lifetime [s] ▼	POIs Found ▲	Connection Type
GNN-P	**0.012 ± 0.054** †	**0.33 ± 0.84** †	0.25 ± 1.27	42.38 ± 14.31	0.90 ± 1.52	Two
	0.010 ± 0.039 †	***0.20 ± 0.63*** †	0.14 ± 1.36	44.87 ± 13.96	0.40 ± 0.70	Heterogeneous
GNN-K	0.089 ± 0.218	12.72 ± 9.04	**0.02 ± 0.15** †	44.92 ± 7.34	1.40 ± 1.71	Two
	0.055 ± 0.175	7.62 ± 6.73	***0.02 ± 0.12*** †	44.26 ± 5.70	1.90 ± 2.33	Heterogeneous
Prim	0.041 ± 0.168	3.64 ± 9.10	0.71 ± 3.30	**31.82 ± 11.93** †	**3.90 ± 3.35** †	Two
	0.028 ± 0.133	2.38 ± 7.54	0.57 ± 3.43	31.18 ± 14.28	2.90 ± 2.77	Heterogeneous
Kruskal	0.183 ± 0.280	29.40 ± 25.32	**0.02 ± 0.15** †	36.55 ± 12.37	2.70 ± 2.75	Two
	0.108 ± 0.229	17.48 ± 25.62	0.02 ± 0.16	31.66 ± 8.79	***4.50 ± 2.59*** †	Heterogeneous
EA	0.051 ± 0.018	-	-	34.80 ± 17.67	0.70 ± 1.16	Two
	0.053 ± 0.015	-	-	***18.00 ± 0*** †	0.20 ± 0.63	Heterogeneous

Number of attempts required measures how many times a decoder had to restart before finding a valid solution, while the frequency of failures measures how often the restart limit was reached and the solver had fall back on the EA. Non-EA methods that revert to the EA require random population initialization, increasing convergence time and resulting in higher worst-case times.

Both GNN-methods outperform their non-GNN counterparts, suggesting that GNN edge order accounting for neighborhood information better encodes priority. The EA had the slowest, but most consistent performance and is used as the fallback method. As shown in Fig. 4, GNN-P achieves the fastest average times and fails the least of all non-EA methods.

Table IV summarizes behavioral metrics. GNN-P and GNN-K showed higher total kinetic energy and mechanical cost than Prim's and Kruskal's algorithms, suggesting these connection optimizer algorithms' agents switch connections and priorities often. The Fiedler value $\lambda_{2,\mathcal{G}}$ is an analogous measure of the relative spread of the agents, with a higher value indicating the agents are likely grouped closer together. For behavioral metrics such as graph spread, there is no single correct value—shorter edges improve communication, Eq. (2), but clustering can limit exploration.

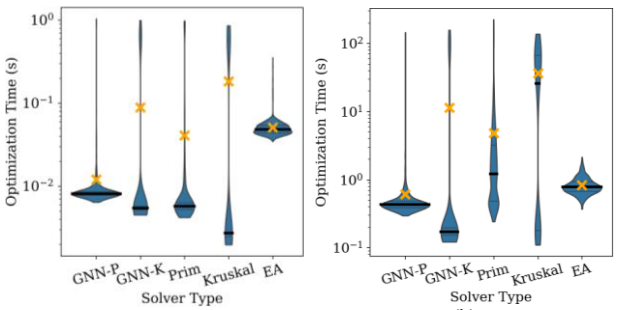

Fig. 4. Violin plots of optimization times for high difficulty simulations with uniform degree constraints of two, with mean values overlaid. GNN trained on 100 agents. (a) 10 agent simulation; (b) 100 agent simulation.

TABLE IV. Behavioral Metrics by Solver Types and Degree Constraint. High Difficulty Field Setup with 10 Agents. GNN is Trained on 100 Agents. Mean and Standard Deviation across all Runs Shown. Maximum and Minimum Results for Each Connection Type are Marked with an Up-Triangle▲ and Down-Triangle▼ Respectively

Solver Type	Total Kinetic Energy	Mechanical Cost	$\lambda_{2,\mathcal{G}}$	$\lambda_{2,G}$	Edge Length	Connection Type
GNN-P	**5.65 ± 1.94** ▲	4610 ± 2969	**0.339 ± 0.209** ▲	**0.018 ± 0.010** ▼	22.02 ± 2.82	Two
	5.71 ± 2.01 ▲	4574 ± 2965	0.341 ± 0.192	0.024 ± 0.017	21.90 ± 2.84	Heterogeneous
GNN-K	5.56 ± 2.00	**4650 ± 2954** ▲	0.302 ± 0.219	0.018 ± 0.012	21.58 ± 3.13	Two
	5.62 ± 2.03	***4745 ± 3072*** ▲	0.313 ± 0.194	***0.021 ± 0.013*** ▼	22.11 ± 2.58	Heterogeneous
Prim	**4.48 ± 2.02** ▼	**1743 ± 1211** ▼	**0.165 ± 0.136** ▼	**0.035 ± 0.013** ▲	**18.20 ± 2.99** ▼	Two
	4.50 ± 1.91	***1844 ± 1248*** ▼	0.194 ± 0.137	0.041 ± 0.019	19.20 ± 2.58	Heterogeneous
Kruskal	4.73 ± 2.09	2371 ± 1670	0.208 ± 0.193	0.032 ± 0.016	18.93 ± 4.40	Two
	4.33 ± 1.95 ▼	2016 ± 1426	***0.182 ± 0.149*** ▼	***0.041 ± 0.020*** ▲	***18.87 ± 4.03*** ▼	Heterogeneous
EA	5.55 ± 1.94	2774 ± 1744	0.336 ± 0.215	0.022 ± 0.010	**22.38 ± 2.94** ▲	Two
	5.46 ± 1.95	2686 ± 1728	***0.428 ± 0.262*** ▲	0.027 ± 0.016	***22.61 ± 3.13*** ▲	Heterogeneous

Together, these values indicate teams using the edge weight decoding algorithms tend to move further apart, potentially explaining why they reach more POIs more quickly and why the EA, the baseline for POI performance, appears to reach the fewest POIs of all. Fig. 5 illustrates how the agents spread out over time. Fiedler values of the selected spanning tree, $\lambda_{2,G}$, are highest for the non-GNN decoder methods, with the total edge lengths being the lowest. This indicates these methods optimize for the minimum spanning tree with respect to edge weights, while the EA and GNN methods are optimizing for minimum control effort, a non-Euclidean property.

Table V contains the experimental performance results for the GNN_{100}-GNN_{10} comparison, and Table VI the behavioral results. While the $GNN\text{-}K_{100}$ shows significant outliers in the optimization time and failure rates, the general trends in these metrics across algorithms are similar to those in the initial 10 agent simulations.

The GNN_{100} and GNN_{10} results show significant differences. The GNN_{10} finds far fewer POIs, expends more energy, has a higher mechanical cost, and spreads its agents out less effectively. For energy metrics, the goal is to minimize energy consumption without compromising task completion. These indicate a performance gap when GNN models are trained on smaller graphs that lack the global graph structure of larger graphs. However, GNN_{10} still outperforms the EA, offering significant computation savings. Behavioral measures show that the GNN_{100} used on 100 agent system is similar in Total Kinetic Energy, Mechanical Cost, and λ_2 values to the non GNN-informed methods. Finally, the EA's poor POI performance, combined with the relatively high $\lambda_{2,\mathcal{G}}$, suggests rapid decrease in population diversity, limiting adaptability to dynamic scenarios.

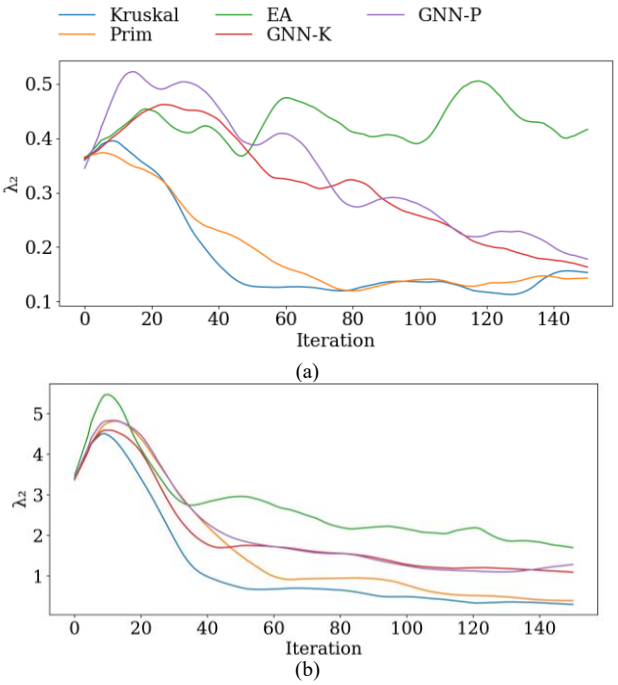

Fig. 5. Time history plots of $\lambda_{2,G}$ for high difficulty simulations with heterogeneous degree constraints, averaged across each time step of all trials and smoothed. (a) 10 agent simulation; (b) 100 agent simulation.

TABLE V. PERFORMANCE METRICS BY SOLVER TYPES AND NUMBER OF AGENTS IN TRAINING DATA FOR LOW DIFFICULTY FIELD SETUP. 100 AGENTS IN SIMULATION. MEAN AND STANDARD DEVIATION ACROSS ALL TIME ACROSS ALL RUNS. ▼ INDICATES LOWER IS BETTER, ▲ INDICATES HIGHER IS BETTER. BEST RESULTS ARE MARKED WITH A DAGGER†

Solver Type	Optimization Time [s] ▼	% Fail ▼	Attempts Required ▼	POI Lifetime [s] ▼	POIs Found ▲	Agents in Training Data
GNN-P	**0.516 ± 0.325** †	**0.00 ± 0.00** †	0.09 ± 0.67	31.01 ± 3.53	9.80 ± 0.42	100
	0.764 ± 0.252	**0.00 ± 0.00** †	0.03 ± 0.40	45.48 ± 7.10	3.90 ± 2.60	10
GNN-K	8.989 ± 28.112	9.60 ± 5.71	**0.00 ± 0.05** †	29.31 ± 3.78	9.80 ± 0.63	100
	0.607 ± 2.817	1.39 ± 2.17	0.00 ± 0.06	44.45 ± 6.33	3.70 ± 2.75	10
Prim	2.204 ± 7.353	0.33 ± 0.64	3.70 ± 8.85	**20.58 ± 3.30** †	**10.00 ± 0.00** †	-
Kruskal	21.002 ± 31.196	43.63 ± 15.56	0.02 ± 0.14	31.99 ± 6.00	9.20 ± 1.14	-
EA	1.117 ± 0.247	-	-	45.49 ± 6.40	2.20 ± 1.69	-

TABLE VI. BEHAVIORAL METRICS BY SOLVER TYPES AND DEGREE CONSTRAINT. HIGH DIFFICULTY FIELD SETUP WITH 10 AGENTS. GNN IS TRAINED ON 100 AGENTS. MEAN AND STANDARD DEVIATION ACROSS ALL RUNS SHOWN. MAXIMUM AND MINIMUM RESULTS ARE MARKED WITH AN UP-TRIANGLE▲ AND DOWN-TRIANGLE▼ RESPECTIVELY

Solver Type	Total Kinetic Energy	Mechanical Cost	$\lambda_{2,\mathcal{G}}$	$\lambda_{2,G}$	Edge Length	Agents in Training Data
GNN-P	67.39 ± 7.23	35487 ± 21539	1.622 ± 2.129	3.35e−4 ± 1.42e−4	14.47 ± 1.75	100
	71.23 ± 7.19	54179 ± 32222	3.431 ± 4.106	9.01e−5 ± 5.68e−5	22.01 ± 1.08	10
GNN-K	67.75 ± 7.61	37527 ± 23426	1.615 ± 2.016	3.79e−4 ± 1.70e−4	15.19 ± 3.26	100
	71.65 ± 7.35▲	**54708 ± 32524**▲	2.854 ± 3.276	1.01e−4 ± 6.41e−5	22.04 ± 1.07	10
Prim	68.92 ± 7.32	**35022 ± 20943**▼	**1.368 ± 2.264**▼	**7.69e−4 ± 1.05e−4**▲	**7.98 ± 1.16**▼	-
Kruskal	69.82 ± 8.46	43939 ± 26632	1.834 ± 2.416	**0.49e−4 ± 3.49e−4**▼	14.75 ± 8.47	-
EA	**66.29 ± 7.08**▼	39458 ± 23157	**3.697 ± 2.737**▲	1.58e−4 ± 6.72e−5	**23.49 ± 0.93**▲	-

VII. DISCUSSION AND CONCLUSIONS

We introduce a novel method for solving LOS connectivity graphs in multi-agent teams using GNNs to learn latent information. This problem is difficult due to communication hardware constraints, making it a DCMST problem with a non-Euclidean measure of graph optimality. The proposed method consistently finds high-quality solutions, even when deployed on swarms of up to 100

agents. The centralized control architecture can be implemented on a modern laptop with a CUDA enabled GPU, and individual agents need only light-weight edge computing hardware.

Our novel GNN approach outperforms the EA by reaching more POIs with faster solution times. The method generalizes well to unseen environment conditions and heterogeneous connection constraints, though it is less optimal with larger team sizes. Future improvements could include training the GNN with a larger neighborhood of data or increasing the network depth. Either would incorporate information from more distant nodes into the edge scores, which may be important to communicate global information in larger graphs. Additional scalability testing would also be valuable to assess performance as graph size increases.

Future research could explore adding temporal context to the model or completely shifting to an experiential learning approach to predict both the connection graph and future agent locations, expanding the model's control. These methods would also reduce dependence on hyperparameter choices for edge weight equations and could potentially leverage an obstacle map constructed by aggregating information from all agents. With newer GNN architectures focusing on executing the individual steps of classic algorithms, mimicking evolutionary algorithms within the GNN would provide an interesting future direction with potential for generalization across a broader class of algorithms, beyond this SL approach.

COPYRIGHT AND GOVERNMENT LICENSE STATEMENT

CONFLICT OF INTEREST

The authors declare no conflict of interest.

AUTHOR CONTRIBUTIONS

Paula Stocco researched and developed algorithms for the GNN. Alan Hesu designed and implemented the EA and helped with GNN implementation and training. Steven Spencer was principal investigator on the project. All authors contributed to writing and approved the final version.

FUNDING

This work was supported by the Laboratory Directed Research and Development program at Sandia National Laboratories, a multimission laboratory managed and operated by National Technology and Engineering Solutions of Sandia LLC, a wholly owned subsidiary of Honeywell International Inc. for the U.S. Department of Energy's National Nuclear Security Administration under Contract No. DE-NA0003525.

ACKNOWLEDGMENT

The authors would like to thank Tamzidul Mina for LOS maintenance expertise, and Derya Tansel for simulation environment software support.

REFERENCES

[1] S. J. Spencer *et al*., "Autonomous detection and assessment with moving sensors," in *Proc. 2020 IEEE/RSJ Int. Conf. Intelligent Robots and Systems (IROS)*, 2020, pp. 8231–8238.

[2] P. R. Giordano *et al*., "A passivity-based decentralized strategy for generalized connectivity maintenance," *Int. J. Robot. Res.*, vol. 32, no. 3, pp. 299–323, March 2013. https://doi.org/10.1177/0278364912469671

[3] A. Shetty, T. Hussain, and G. Gao, "Decentralized connectivity maintenance for multi-robot systems under motion and sensing uncertainties," *Journal of the Institute of Navigation*, vol. 70, no. 1, March 2023.

[4] G. Raidl, "An efficient evolutionary algorithm for the degree-constrained minimum spanning tree problem," in *Proc. 2000 Congr. Evolutionary Computation*, 2000, vol. 1, pp. 104–111.

[5] K. Singh and S. Sundar, "A hybrid genetic algorithm for the degree-constrained minimum spanning tree problem," *Soft Computing*, vol. 24, pp. 2169–2186, May 2019.

[6] S. C. Narula and C. A. Ho, "Degree-constrained minimum spanning tree," *Computers & Operations Research*, vol. 7, no. 4, pp. 239–249, 1980.

[7] C. H Papadimitriou, "The Euclidean travelling salesman problem is NP-complete," *Theoretical Computer Science*, vol. 4, no. 3, pp. 237–244, June 1977.

[8] Y. Bengio, A. Lodi, and A. Prouvost, "Machine learning for combinatorial optimization: A methodological tour d'horizon," *Eur. J. Oper. Res.*, vol. 290, no. 2, pp. 405–421, April 2021.

[9] U. J. Mele, L. M. Gambardella, and R. Montemanni, "Machine learning approaches for the traveling salesman problem: A survey," in *Proc. 8th Int. Conf. Ind. Eng. Appl.* (Europe), 2021, pp. 182–186.

[10] Y. Peng, B. Choi, and J. Xu, "Graph learning for combinatorial optimization: A survey of state-of-the-art," *Data Sci. Eng.*, vol. 6, no. 2, pp. 119–141, Jun. 2021. https://doi.org/10.1007/s41019-021-00155-3

[11] Q. Cappart *et al*., "Combinatorial optimization and reasoning with graph neural networks," *J. Mach. Learn. Res*., vol. 24, no. 1, pp. 5922–5982, January 2023.

[12] I. Bello *et al*., "Neural combinatorial optimization with reinforcement learning," presented at the International Conference on Learning Representations, 2017.

[13] H. Dai, E. B. Khalil, Y. Zhang *et al*., "Learning combinatorial optimization algorithms over graphs," *Advances in Neural Information Processing Systems*, no. 30, 2017.

[14] W. Kool, H. van Hoof, and M. Welling, "Attention, learn to solve routing problems!" presented at the International Conference on Learning Representations, 2019.

[15] Y. Shi, C. Han, and T. Guo, "NeuroPrim: An attention-based model for solving NP-hard spanning tree problems," *Science China Mathematics*, vol. 67. pp. 1359–1376, 2024.

[16] R. S. Sutton *et al*., "Policy gradient methods for reinforcement learning with function approximation," *Advances in Neural Information Processing Systems*, pp. 1057–1063, 1999.

[17] C. K. Joshi, T. Laurent, and X. Bresson. "An efficient graph convolutional network technique for the travelling salesman problem," arXiv preprint, arXiv:1906.01227, Oct. 2019.

[18] C. K Joshi *et al*., "Learning the travelling salesperson problem requires rethinking generalization," *Constraints*, pp. 1–29, April 2022.
[19] P. Veličković *et al.*, "Neural execution of graph algorithms. international conference on learning representations," in *Proc. International Conference on Learning Representations*, 2020.
[20] D. Georgiev *et al*., "Algorithmic concept-based explainable reasoning," in *Proc. AAAI Conference on Artificial Intelligence*, 2022, pp. 6685–6693.
[21] É. D. Taillard and K. Helsgaun, "POPMUSIC for the travelling salesman problem," *European Journal of Operational Research*," vol. 272, no. 2, pp. 420–429, January 2019.
[22] W. Huang, Y. Wang, and X. Yi, "A deep reinforcement learning approach to preserve connectivity for multi-robot systems," in *Proc. 2017 10th International Congress on Image and Signal Processing, BioMedical Engineering and Informatics (CISP-BMEI)*, Oct. 2017, pp. 1–7.
[23] Y. Lyu *et al*., "Multivehicle flocking control with deep deterministic policy gradient method," in *Proc. International Conference on Control and Automation*, 2018, pp. 306–311.
[24] M. Li *et al*., "Decentralized global connectivity maintenance for multi-robot navigation: A reinforcement learning approach," in *Proc. International Conference on Robotics and Automation*, 2022, pp. 8801–8807.
[25] Q. Li, F. Gama, A. Ribeiro, and A. Prorok, "Graph neural networks for decentralized multi-robot path planning," in *Proc. 2020 IEEE/RSJ Int. Conf. Intelligent Robots and Systems (IROS)*, Oct. 2020, pp. 11785–11792.
[26] Y. Yang, Y. Lyu, and W. Luo, "Minimally constrained multi-robot coordination with line-of-sight connectivity maintenance," in *Proc. 2023 IEEE Int. Conf. Robotics and Automation (ICRA)*, London, United Kingdom, May 2023, pp. 7684–7690.
[27] B. Capelli and L. Sabattini, "Connectivity maintenance: Global and optimized approach through control barrier functions," in *Proc. 2020 IEEE Int. Conf. Robotics and Automation (ICRA)*, Paris, France, May 2020, pp. 5590–5596.
[28] K. Kaminski *et al*., "Rossmann-toolbox: A deep learning-based protocol for the prediction and design of cofactor specificity in Rossmann fold proteins," *Briefings in Bioinformatics*, vol. 23, no. 1, bbab371, Sep. 2021.
[29] M. Wang *et al*., "Deep graph library: A graph-centric, highly-performant package for graph neural networks," arXiv preprint, arXiv:1909.01315, 2019.
[30] P. Velicković *et al*., "Graph attention networks," in *Proc. Int. Conf. Learning Representations*, 2018.
[31] T. T. Nguyen, S. Yang, and J. Branke, "Evolutionary dynamic optimization: A survey of the state of the art," *Swarm and Evolutionary Computation*, vol. 6, pp. 1–24, Oct. 2012.
[32] G. Raidl and B. Julstrom, "Edge sets: An effective evolutionary coding of spanning trees," *IEEE Trans. Evolutionary Computation*, vol. 7, no. 3, pp. 225–239, Jun. 2003.
[33] A. Ning, L. Ma, and X. Xiong, "A new algorithm for degree-constrained minimum spanning tree based on the reduction technique," *Progress in Natural Science*, vol. 18, no. 4, pp. 495–499, Apr. 2008.
[34] R. C. Prim, "Shortest connection networks and some generalizations," *The Bell System Technical Journal*, vol. 36, no. 6, pp. 1389–1401, 1957.
[35] J. B. Kruskal, "On the shortest spanning subtree of a graph and the traveling salesman problem," in *Proc. the American Mathematical Society*, 1956, vol. 7, no. 1, pp. 48–50.
[36] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," arXiv preprint, arXiv:1412.6980, Jan. 2017.
[37] C. Ulmer, J. Friesen, and J. Kenny, "Glinda: An HPDA cluster with ampere A100 GPUs and BlueField-2 VPI SmartNICs," Sandia National Laboratories, Albuquerque NM, USA, Tech. Rep. SAND2023-10451, Sep. 2023.