Graph Neural Networks for Decentralized Multi-Robot Path Planning

Qingbiao Li¹, Fernando Gama², Alejandro Ribeiro², Amanda Prorok ¹

ABSTRACT
Efficient and collision-free navigation in multi-robot systems is fundamental to advancing mobility. Scenarios where the robots are restricted in observation and communication range call for decentralized solutions, whereby robots execute localized planning policies. From the point of view of an individual robot, however, its local decision-making system is incomplete, since other agents’ unobservable states affect future values. The manner in which information is shared is crucial to the system’s performance, yet is not well addressed by current approaches. To address these challenges, we propose a combined architecture, with the goal of learning a decentralized sequential action policy that yields efficient path plans for all robots. Our framework is composed of a convolutional neural network (CNN) that extracts adequate features from local observations, and a graph neural network (GNN) that communicates these features among robots. We train the model to imitate an expert algorithm, and use the resulting model online in decentralized planning involving only local communication. We evaluate our method in simulations involving teams of robots in cluttered workspaces. We measure the success rates and sum of costs over the planned paths. The results show a performance close to that of our expert algorithm, demonstrating the validity of our approach. In particular, we show our model’s capability to generalize to previously unseen cases (involving larger environments and larger robot teams).

KEYWORDS
Multi-Agent Path Finding; Decentralized Planning; Deep Learning; Graph Neural Networks;

1 INTRODUCTION
Efficient and collision-free navigation in multi-robot systems is fundamental to advancing mobility. The problem, generally referred to as Multi-Robot Path Planning (MRPP) or Multi-Agent Path Finding (MAPF), aims at generating collision-free paths leading robots from their origins to designated destinations. Solutions to this problem lend themselves to search and rescue operations [15], product pickup and delivery [12], item retrieval in warehouses [6], and mobility-on-demand services [20, 33]. Current approaches can be classified as either coupled or decoupled, depending on the structure of the state space that is searched. While coupled approaches are able to ensure the optimality and completeness of the solution, they involve centralized components, and tend to scale poorly with the number of robots [32, 34]. Decoupled approaches, on the other hand, compute trajectories for each robot separately, and re-plan only in case of conflicts [37, 39, 41]. This can significantly reduce the computational complexity of the planning task, but generally produces sub-optimal and incomplete solutions. It is noteworthy, however, that decoupled approaches lend themselves naturally to decentralized solutions. Balancing optimality and completeness with the complexity of computing a solution, however, is still an open research problem [2, 25].

This work focuses on multi-robot path planning for scenarios where the robots are restricted in observation and communication range. This naturally arises when considering physical robots equipped with hardware constraints that limit their perception and communication capabilities [16]. These scenarios impose a decentralized structure, where at any given point in time, robots have only partial information of the system state. This condition requires adapted control algorithms. In this paper, we propose a combined architecture, where we train a convolutional neural network (CNN) that extracts adequate features from local observations, and a graph neural network (GNN) to communicate these features among robots [10] with the ultimate goal of learning a decentralized sequential action policy that yields efficient path plans for all robots. The GNN implementation seamlessly adapts to the partial information structure of the problem, since it is computed in a decentralized manner. We train this architecture to imitate an optimal coupled planner with global information that is available offline at training time. We utilize a dataset aggregation method that leverages an online expert to resolve hard cases, thus expediting the learning process. The resulting trained model is used online in an efficient, decentralized manner, involving communication only with nearby robots. Furthermore, we show that the model can be deployed on much larger robot teams than the ones it was trained on.

This paper is organized as follows. In Sec. 2, we discuss related work and in Sec. 3, we formulate the problem. We then introduce fundamental concepts of graph signal processing and graph convolutions in Sec. 4, so that GNNs can be formally defined in Sec. 4.2. Sec. 5 presents the proposed architecture, and its core components. Finally, Sec. 6 summarizes our experimental setup and results, and is followed by a discussion in Sec. 7. Conclusions are drawn in Sec. 8.

2 RELATED WORK
Classical approaches to multi-robot path planning can generally be described as either centralized (assuming the existence of a central component that knows the state of the whole robot system) or decentralized (where no single component has the full picture, but cooperation must still be achieved). Centralized approaches are facilitated by a planning unit that monitors all robots’ positions and desired destinations, and returns a coordinated plan of trajectories (or way-points) for all the robots in the system. These plans are communicated to the respective robots, which use them for real-time
on-board control of their navigation behavior. Coupled centralized approaches, which consider the joint configuration space of all involved robots, have the advantage of producing optimal and complete plans, yet tend to be computationally very expensive [28]. Indeed, solving for optimality is NP-hard [42], and although significant progress has been made towards alleviating the computational load [8, 29, 43], these approaches still scale poorly in environments with a high number of potential path conflicts.

Decentralized approaches provide an attractive alternative to centralized approaches, firstly, because they reduce the computational overhead, and secondly, because they relax the dependence on centralized units. This body of work considers the generation of collision-free paths for individual robots that cooperate only with immediate neighbors [4, 41], or with no other robots at all [1, 38]. In the latter case, coordination is reduced to the problem of reciprocally avoiding other robots (and obstacles), and can generally be solved without the use of communication. Yet, by taking purely local objectives into account, global objectives (such as path efficiency) cannot be explicitly optimized. In the former case, it has been shown that monotonic cost reduction of global objectives can be achieved. This feat, however, relies on strong assumptions (e.g., problem convexity and invariance of communication graph [23, 40]) that can generally not be guaranteed in real robot systems.

Learning-based methods have proven effective at designing robot control policies for an increasing number of tasks [21, 35]. The application of learning-based methods to multi-robot motion planning has attracted particular attention due to their capability of handling high-dimensional joint state-space representations, by offloading the online computational burden to an offline learning procedure. The work in [7] proposes a decentralized multi-agent collision avoidance algorithm based on deep reinforcement learning. Their results show that significant improvement in the quality of the path (i.e., time to reach the goal) can be achieved with respect to current benchmark algorithms (e.g., ORCA [38]). Also in recent work, Sartoretti et al. [26] propose a hybrid learning-based method for multi-agent path-finding that uses both imitation learning (based on an expert algorithm) and multi-agent reinforcement learning. Robots controlled by this method perform well in various environments, useful when planning for large-scale collective robotic systems, by offloading the online computational burden to an offline learning procedure [7, 36]. Key to this learning procedure is the fact that each robot must be able to accumulate information from other robots in its neighborhood. From the point of view of an individual robot, its local decision-making system is incomplete, since other agents’ unobservable states affect future values. The manner in which information is shared is crucial to the system’s performance, yet is not well addressed by current machine learning approaches. Graph Neural Networks (GNNs) promise to overcome this deficiency [10, 27]. They capture the relational aspect of robot communication and coordination by modeling the collective robot system as a graph: each robot is a node, and edges represent communication links [19]. Although GNNs have been applied to a number of problem domains, including molecular biology [5], quantum chemistry [11], and simulation engines [3], they have only very recently been considered within the multi-robot domain [19, 36].

Contributions. The application of GNNs to the problem of multi-robot path planning is novel. GNNs offer an efficient architecture that operates in a localized manner, where information is shared over a multi-hop communication network, through communication with nearby neighbors only [10]. Our key contribution in this work is the development of a framework that can learn what information needs to be shared between robots to facilitate efficient path planning. This framework is composed of a convolutional neural network (CNN) that extracts adequate features from local observations, and a graph neural network (GNN) that learns to communicate these features among robots. By jointly training these two components, the system is able to best determine what information is relevant for the team as a whole, and share this. The proposed model is trained to imitate a centralized coupled planner, and makes use of a dataset aggregation method that leverages an online expert to resolve hard cases, thus expediting the learning process. Our aim is to achieve comparable performance in terms of success rate and flowtime (sum of path costs), while also being able to generalize to previously unseen cases, such as larger robot teams and environments.

3 PROBLEM FORMULATION

Let \( V = \{v_1, \ldots, v_N\} \) be the set of \( N \) robots. At time \( t \), each robot perceives its surroundings within a given field of vision, and knows where its own target destination is located. This map perceived by robot \( i \) is denoted by \( Z^i_t \in \mathbb{R}^{W_{FOV} \times H_{FOV}} \) where \( W_{FOV} \) and \( H_{FOV} \) are the width and height, respectively, and are determined by the field of vision radius \( \rho \).

The robots can communicate with each other as determined by the communication network. We can describe this network at time \( t \) by means of a graph \( G^t = (V, E^t, W^t) \) where \( V \) is the set of robots, \( E^t \subseteq V \times V \) is the set of edges and \( W^t : E^t \to \mathbb{R} \) is a function that assigns weights to the edges. Robots \( v_i \) and \( v_j \) can communicate with each other at time \( t \) if \((v_i, v_j) \in E^t\). The corresponding edge weight \( W^t(v_i, v_j) = w^t_{ij} \) can represent the strength of the communication (or be equal to 1 if we are only modeling whether there is a link or not). In this work, we formulate the multi-agent path planning problem as a sequential decision making problem, that each robot solves at every time instant \( t \) with the objective of reaching its destination. More formally, the objective of this work is to learn a mapping \( f \) that takes the maps \( \{Z^i_t\}_{v_i \in V} \) and the communication network \( G^t \) at time \( t \) and determines an appropriate action \( u^t \). We want the action \( u^t = f((Z^i_t), G^t) \) to be such that it contributes to the global objective of moving the robots towards their destinations in the shortest possible time while avoiding collisions with other robots and with obstacles that might be present. The mapping \( f \) has to be restricted to involve communication only among nearby robots, as dictated by the network \( G^t \) at each time instant \( t \). Finally, note that
for scalability, the mapping $F$ cannot depend on time $t$, allowing
the system to process sequences of arbitrary duration.

4 GRAPH NEURAL NETWORKS

To guarantee that the mapping $F$ is restricted to communications
only among nearby robots, we parametrize it by means of a GNN,
which is a naturally decentralized solution (Sec. 4.2). We then train
this GNN to learn appropriate actions that contribute to the global
objective by means of supervised learning through an expert algo-
rithm (i.e., imitation learning) (Sec. 5.5).

4.1 Graph Convolutions

Assume that each robot has access to $F$ observations $\tilde{x}_f^i \in \mathbb{R}^F$
at time $t$. Let $X_t \in \mathbb{R}^{N \times F}$ be the observation matrix where each row
collects these $F$ observations at each robot $\tilde{x}_f^i$, $i = 1, \ldots, N$,

$$ X_t = \begin{bmatrix} (\tilde{x}_1^T) & \cdots & (\tilde{x}_t^T) \\ \vdots & \ddots & \vdots \\ (\tilde{x}_N^T) & \cdots & (\tilde{x}_t^T) \end{bmatrix}. $$ (1)

Note that the columns $\tilde{x}_f^i \in \mathbb{R}^N$ represent the collection of the
observation $f$ across all nodes, for $f = 1, \ldots, F$. This vector $\tilde{x}_f^i$
is a graph signal [17], since it assigns a scalar value to each node,
$\tilde{x}_f^i : V \rightarrow \mathbb{R}$ so that $[\tilde{x}_f^i]_i = x_f^i \in \mathbb{R}$.

To formally describe the communication between neighboring
agents, we need a concise way of describing the graph and relating
it to the observations $X_t$. Let $S_t \in \mathbb{R}^{N \times N}$ be a matrix description
of $G_t$. That is, $S_t$ is such that $[S_t]_{ij} = s_{ij}^f = 0$ if $(v_j, v_i) \notin E_t$.
Matrix $S_t$ thus respects the sparsity of the graph and is called a
graph shift operator (GSO) [17]. Examples of GSO typically used in
the literature include the adjacency matrix of the graph [24], the
Laplacian matrix [31], the Markov matrix [13], among others.

The operation $S_t X_t$ represents a linear combination of neighbor-

Figure 1: Graph convolution. Every node takes its data value $x$ and weighs it by $a_0$ (first graph). Then, all the nodes exchange information with their one-hop neighbors to build $Sx$, and weigh the result by $a_1$ (second graph). Next, they exchange their values of $Sx$ again to build $S^2x$ and weigh it by $a_2$ (third graph). This procedure continues for $K$ steps until all $a_k S^k x$ have been computed for $k = 0, \ldots, K - 1$, and added up to obtain the output of the graph convolution operation (3). To avoid cluttering, this operation is illustrated on only 5 nodes. In each case, the corresponding neighbors accessed by successive relays of information are indicated by the colored disks.

Graph Neural Networks for Decentralized Multi-Robot Path Planning

precisely, note that the value at node $i$ for observation $f$ after operation
$S_t X_t \in \mathbb{R}^{N \times F}$ becomes

$$ [S_t X_t]_{if} = \sum_{j=1}^N [S_t]_{ij} [X_t]_{jf} = \sum_{j : v_j \in N_i} s_{ij}^f x_f^j $$ (2)

where $N_i = \{v_j \in V : (v_j, v_i) \in E_t\}$ is the set of nodes $v_j$ that are
neighbors of $v_i$. Note that the second equality in (2) holds because
$s_{ij}^f = 0$ for all $j \notin N_i$.

The linear operation $S_t X_t$ is essentially shifting the values of $X_t$
through the nodes, since the application of $S_t$ updates the value
at each node by a linear combination of values in the neighbor-
hood. With the shifting operation in place, we can define a graph
convolution [10] as linear combination of shifted versions of the signal

$$ \mathcal{A}(X_t; S_t) = \sum_{k=0}^{K-1} S_t^k X_t A_k $$ (3)

where $\{A_k\}$ is a set of $F \times G$ matrices representing the filter co-
efficients combining different observations. Several noteworthy
comments are in order with respect to (3). First, multiplications to
the left of $X_t$ need to respect the sparsity of the graph since these
multiplications imply combinations across different nodes. Multipli-
cations to the right, on the other hand, can be arbitrary, since they
imply linear combination of observations within the same node in
a weight sharing scheme. Second, $S_t^k X_t = S_t (S_t^{k-1} X_t)$ is computed
by means of $k$ communication exchanges with 1-hop neighbors,
and is actually computing a summary of the information located
at the $k$-hop neighborhood. Therefore, the graph convolution is
an entirely local operation in the sense that its implementation is
naturally distributed. Third, the graph convolution is actually com-
puting the output of a bank of $FG$ filters where we take as input $F$
observations per node and combine them to output $G$ observations
per node, $\mathcal{A}(X_t; S_t) \in \mathbb{R}^{N \times G}$. There are $FG$ graph filters involved
in (3) each one consisting of $K$ filter taps, i.e., the $(f,g)$ filter can
be described by filter taps \( \delta^g = \{\delta^g_1, \ldots, \delta^g_{K} \} \in \mathbb{R}^K \) and these filter taps are collected in the matrix \( A_k \) as \( [A_k]_{f,g} = \delta^g_k \).

4.2 Graph Neural Networks

A convolutional GNN [10] consists of a cascade of \( L \) layers, each of which applies a graph convolution (3) followed by a pointwise nonlinearity \( \sigma : \mathbb{R} \to \mathbb{R} \) (also known as activation function)

\[
X_{\ell} = \sigma\left[ A_{\ell}(X_{\ell-1}; S) \right] \quad \text{for} \quad \ell = 1, \ldots, L
\]

where, in a slight abuse of notation, \( \sigma \) is applied to each element of the matrix \( A_{\ell}(X_{\ell-1}; S) \). The input to each layer is a graph signal consisting of \( F_{\ell-1} \) observations and the output has \( F_{\ell} \) observations so that \( X_{\ell} \in \mathbb{R}^{N \times F_{\ell}} \). The input to the first layer is \( X_0 = X_t \) so that \( F_0 = F \) and the output of the last layer corresponds to the action to be taken at time \( t \). \( X_L = U_t \) which could be described by a vector of dimension \( F_L = G \). The GSO \( S \) to be used in (4) is the one corresponding to the communication network at time \( t \). \( S = S_t \).

At each layer \( \ell \) we have a bank of \( F_t F_{\ell-1} \) filters \( A_{\ell} \) described by a set of \( K_t F_t F_{\ell-1} \) total filter taps \( \{A_{\ell,k}\}_{k=0}^{K} \).

We note that, in the present framework, we are running one GNN (4) per time instant \( t \), where each time step is determined by the moment the action is taken and the communication network changes. This implies that we need to carry out \( \sum_{t=1}^{T} (K_t - 1) \) total communications before deciding on an action. Therefore, it is important to keep the GNN shallow (small \( L \)) and the filters short (small \( K_t \)).

In summary, we propose to parametrize the mapping \( \mathcal{F} \) between maps \( Z_{t} \) and actions \( U_{t} \) by using a GNN (4) acting on observations \( X_{t} = CNN(Z_{t}) \) obtained by applying a CNN to the input maps. We note that, by choosing this parametrization we are obtaining a map of the network structure of the data.

5 ARCHITECTURE

The following sections describe the architecture, of which all components are illustrated in Fig. 2.

5.1 Processing Observations

In an environment \((W \times H)\) with static obstacles, each robot has a local field-of-view (FOV), the size of which is defined by \( \rho \), beyond which it cannot ‘see’ anything. The data available at each robot is a vector \( \tilde{x}_i \in \mathbb{R}^{F} \) containing \( F \) observations (1), \( \tilde{x}_i = CNN(Z_{i}^{t}) \). These observations can then be communicated to nearby robots. The intuition behind using a CNN is to process the input map \( Z_{i}^{t} \) into a higher-level feature tensor \( \tilde{x}_i \) describing the observation, goal and states of other robots. This feature tensor is then transmitted via the communication network, as described in the following section, Sec. 5.2.

5.2 Communication

Let \( p_i \in \mathbb{R}^2 \) be the global position of robot \( i \) in a given planar workspace. Two robots \( i \) and \( j \) can communicate if \( |p_i - p_j| \leq r \) for some given communication radius \( r > 0 \). Each individual robot communicates its compressed observation vector \( \tilde{x}_j \) over the multi-hop communication network, whereby the number of executed hops is limited by \( K \). As described in Sec. 4.2, we apply our GNN to aggregate and fuse the states \( \tilde{x}_j \) within this \( K \)-hop neighborhood of robots \( j \in N_i \), for each robot \( i \). The output of the communication GNN is a hyper-representation of the fused information of the robot itself and its \( K \)-hop neighbors, which is passed to the action policy, as described in Sec. 5.3. We note that each robot carries a local copy of the GNN, hence resulting in a localized decision-making policy.

5.3 Action Policy

We formulate the path-finding problem as a sequential classification problem, whereby an optimal action is chosen at each time step. We adopt a local multi-layer perceptron (MLP) to train our action network. More specifically, each node applies a MLP to the aggregated features resulting from the communication GNN. This MLP is the same across all nodes, resembling a weight-sharing scheme. The action \( u_i \) taken by robot \( i \) is computed by a softmax over the probability distribution of motion primitives, which, in our case, consists of five discrete options (up, left, down, right, idle), and are represented by one-hot vectors. The final path is represented by the series of sequential actions.

5.4 Network Architecture

In the proposed framework, the CNN architecture is composed by the following set of instructions, Sequential I(32)-Sequential II(32)-Sequential I(64)-Sequential II(64)-Sequential I (128) followed by a Linear-Softmax layer. The instruction Sequential I is composed by the blocks Conv2d-BatchNorm2d-Relu-MaxPool2d, and Sequential II is the composition of Conv2d-BatchNorm2d-Relu, where the number in parentheses indicates the number of channels. All kernels are of size 3 with a stride of 1 and zero-padding. In the GNN architecture, we deploy a single layer GNN (as described in Sec. 4.2) and set 128 as the number of input observations \( F \) and output observations \( G \). Note that we can tune the filter taps \( K \) for non-communication \( (K = 1) \) and multi-hop communication \( (K > 1) \). In the action policy, we use a Linear-Sofmax layer to decode the output observations \( G \) from GNN with the number of features, 128, into the five motion primitives.

5.5 Learning from Expert Data

To train our models, we propose a supervised learning approach based on expert data (i.e., imitation learning). We assume that, at training time, we have access to an optimal trajectory of actions \( U^{t}_{\star} \) for all the robots, and the corresponding maps obtained for this trajectory \( \{Z_{i}^{t}\} \), collected in a training set \( T = \{\{U_{t}\},\{Z_{i}^{t}\}\} \).

Then, we train the mapping \( \mathcal{F} \) so that the output is as close as possible to the corresponding optimal action \( U^{\star} \). If the mapping \( \mathcal{F} \) is parametrized in terms of a GNN (4) then this optimization problem becomes

\[
\min_{\text{CNN}, \{A_{\ell,k}\}, \text{MLP}} \sum_{(U_{t},\{Z_{i}^{t}\}) \in T} \sum_{t} |U_{t}^{\star} - \mathcal{F}(\{Z_{i}^{t}\}, G)_{t}|.
\]
We are optimizing over the filters in the CNN required to process the map as well as the set of matrices \( \{ A_k \} \) that contains the \( \sum_{\ell=1}^{L} K_{\ell} F_{\ell-1} F_{\ell} \) learnable parameters of the communication GNN. Note that the number of parameters is independent of the size of the network \( N \).

Imitation learning rests on the availability of an optimal solution (further elaborated in Sec. 5.6, below). While this solution might be computationally expensive, or even intractable for large networks, we only need it at training time. Once trained, the GNN models can be deployed in different communication topologies [9], including those with a larger number of robots as is evidenced in the numerical experiments of Sec. 6.4. Given the decentralized nature of the parametrizations, the trained models are efficient in the sense that their computation is distributed among the agents, demanding only communication exchanges with one-hop neighbors.

Training the parametrized mapping \( F \) through imitation learning guarantees that the actions taken by the individual robots contribute to the global objective of moving the robots towards their pre-specified destination while avoiding collisions with other robots and obstacles. Furthermore, the output actions are computed in a distributed manner.

### 5.6 Expert Data Generation

As described in our problem statement, Sec. 3, the robots operate in a grid world of size \( W \times H \) with static obstacles. We generate random cases, i.e., problem instances, which consist of pairs of start and goal positions for all robots (we also refer to this as a configuration). We filter duplicates or invalid cases, and store the remaining cases in a setup pool, which is randomly shuffled at training time. For each case, we generate the optimal solution. Towards this end, we run an expert algorithm: Conflict-Based Search (CBS) [30] (which is a similar approach as taken in [26]). This expert algorithm computes our ‘ground truth paths’ (the sequence of actions for individual robots), within a 300s timeout, for a given initial configuration. Our data set comprises 30,000 cases for any given grid world and number of agents. This data is divided into a training set (70%), a validation set (15%), and a testing set (15%).
5.7 Policy Execution with Collision Shielding

At inference stage, we execute the action policy with a protective mechanism that we name collision shielding. Since it is not guaranteed that robots learn collision-free paths, we require this additional mechanism to guarantee that no collisions take place. Collision shielding is implemented as follows: (i) if the inferred action would result in a collision with another robot or obstacle, then that action is replaced by an idle action; (ii) if the inferred actions of two robots would result in an edge collision (essentially having them swap positions), then those actions are replaced by idle actions. It is entirely possible that robots remain stuck in an idle state until the timeout is reached. When this happens, we count it as a failure case, degrading the measured performance. The overall inference process is summarized in Alg. 1 and Fig. 3.

5.8 Dataset Aggregation during Training

The use of collision shielding leads to failure cases due to potential deadlocks in the actions taken by the robots, where some of them remain stuck in an idle state. To overcome such deadlocks, we propose a dataset aggregation method that makes use of an online expert (OE) algorithm, during training. More specifically, every $C$ epochs, we select $n_{OE}$ random cases from the training set and identify which ones are stuck in a deadlock situation. Then, we run the expert starting from the deadlock configuration in order to unlock them into moving towards their goal. The resulting successful trajectory is added to the training set and this extended training set is then used in the following epochs. This process is detailed in Alg. 2. We note that no change is made to the validation or test sets. This dataset aggregation method is similar to the approach in DAgger [22], but instead of correcting every failed trajectory, we only correct trajectories from a randomly selected pool of cases, as calls to our expert algorithm are time-consuming. Another key difference is that we need to resort to an explicit measure of failure (i.e., through the use of a timeout), since focusing on any deviations from the optimal path (as in the DAgger approach) may be misleading, because those paths may still lead to very competitive solutions in our problem setting.

6 PERFORMANCE EVALUATION

To evaluate the performance of our method, we perform two sets of experiments, (i) on networks trained and tested on the same number of robots, and (ii) on networks trained on a given number of nodes and tested on previously unseen team sizes (both larger and smaller).

In Section 6.1 we discuss the performance metrics, in Section 6.2 we present the baseline centralized planner and in Section 6.3 we describe the hardware setup. Section 6.4 presents the results obtained from the proposed framework, evaluated by the corresponding metrics, and compared to the baseline.

6.1 Metrics

We consider two key performance metrics:
Graph Neural Networks for Decentralized Multi-Robot Path Planning

- **Success Rate** ($\alpha$): A case is considered successful (complete) when all robots reach their goal prior to the timeout, i.e., when all robots find their paths from $p_i^t$ to $p_{goal}^i$ for $i \in [0, N]$. The success rate is hence quantified by the proportion of successful cases over the total number of tested cases $n$:

$$\alpha = \frac{n_{success}}{n} \quad (6)$$

- **Flowtime Increase** ($\delta_{FT}$): At the end of the system’s inference stage (see Fig. 3), the sequence of actions result in a final path, for each robot. The sum of the executed path lengths ($FT$) may be larger than the sum of expert (target) path lengths ($FT^*$). This deterioration is computed as

$$\delta_{FT} = \frac{FT - FT^*}{FT^*}. \quad (7)$$

Note that if a robot does not reach its goal, the length of the predicted path is considered to be the length of the maximum allowed path length ($T_{max} = 3T_{MP}$). Here, $T_{MP}$ is the makespan of the solution generated by expert algorithm. We also note that computing the flowtime increase with respect to an expert algorithm requires that we can actually solve a case using the expert algorithm in tractable time.

### 6.2 Baseline: Centralized Planner

In order to compare our decentralized action policy with a centralized solution, we implement a centralised path planner. For a fair comparison, we train and test this centralized planner on the same data we use for our decentralized policies. The centralized planner (true to its formal definition [30]) receives synchronized global state information (i.e., the states and goals of all robots, at each point in time). Hence, the input tensor ($W \times H \times (1 + 2 \times N)$) consists of the concatenation of the global map ($W \times H$, same for all robots), and the goals and initial states ($W \times H \times 2$) in global coordinates for all robots $i \in [0, N]$. Convolutional neural networks are applied to process the input tensor ($x_i^t$). The same CNN architecture as for the decentralized framework (previously described in Sec. 5.4) is used for fair comparison. The main difference to the decentralized architecture is that we use a multi-output network to predict the actions ($\hat{a}_i^t$) of all robots simultaneously, instead of an action policy based on exchanged information with nearby agents. Each output-network is a Linear-Softmax layer for each individual robot.

### 6.3 Experimental Setup

Our simulations were conducted using a 12-core, 3.2Ghz i7-8700 CPU and an Nvidia GTX 1080Ti GPU with 32 and 11GB of memory, respectively. The proposed network was implemented in PyTorch v1.1.0 [18], and was accelerated with Cuda v10.0 APIs. We use the Adam optimizer [14] with momentum 0.9. The learning rate $\gamma$ is scheduled to decay from $10^{-3}$ to $10^{-6}$ within 150 epochs, using cosine annealing. We set the batch size to 64. In the proposed framework, we set L2 regularization as $10^{-5}$. The online expert on the GNN is deployed every $C = 4$ epochs on $n_{OE} = 500$ randomly selected cases from the training set. In the centralized planner (Sec. 6.2), we set L2 regularization as $3 \times 10^{-3}$ and kept the optimizer, learning rate, maximum epochs and batch size the same as in the experiments with the GNN.

### 6.4 Results

We instantiate a map of size $20 \times 20$. The robots’ FOV is $\rho = 9$, and the communication radius is $r = 5$. The obstacle density is set to 10%, corresponding to the proportion of occupied over free space in the environment. At each time step, each robot runs a forwards pass of its local action policy (i.e., the trained network). At the end of each case (i.e., it either solved or the timeout is reached), we record the length of each robot’s path and the number of robots that reach their goals, to compute performance metrics according to Sec. 6.1.

#### 6.4.1 Effect of Communication on Flowtime and Success Rates

Figures 4a and 4b show results for the success rate ($\alpha$) and flowtime increase ($\delta_{FT}$), as a function of the number of robots. Panels (a) and (b) show results for our GNN implementation. For each panel, we vary the number of communication hops ($K \in [1, 2, 3, 4]$), including results obtained through training with online expert (OE) (for $K = 3$ and $K = 4$). We compare the results with the centralized planner (baseline).

Figure 4: Results for success rate ($\alpha$) and flowtime increase ($\delta_{FT}$), as a function of the number of robots. Panels (a) and (b) show results for our GNN implementation. For each panel, we vary the number of communication hops ($K \in [1, 2, 3, 4]$), including results obtained through training with online expert (OE) (for $K = 3$ and $K = 4$). We compare the results with the centralized planner (baseline).
6.4.2 Generalization. Tables 5a and 5b summarize the generalization capability of our model for success rate and flowtime increase, respectively. The experiment was carried out by testing networks across previously unseen cases. The tables specify the number of robots trained on in the rows, and the number of robots tested on in the columns.

The generalization performance of the network is visualized by a heatmap, which maps performance values into a color range from purple to red, where purple indicates the best performance and red indicates the worst performance.

![Heatmap](image)

**Figure 5:** Success rate and flowtime increase (trained with OE, for \(K = 3\)). The rows represent the number of robots on which each model was trained, and columns represent the number of robots at test time. In other words, the values on the diagonal are the values for environments with the same number of robots during test and training. The generalization performance of the network is visualized by a heatmap, which maps performance values into a color range from purple to red, where purple indicates the best performance and red indicates the worst performance.

7 DISCUSSION AND FUTURE WORK

The intrinsic limitations of the centralized planner (CP) are that it requires a larger network capacity (as described in Sec. 6.2), and that it needs to be re-trained for each new number of robots \(N\) (i.e., it does not have the capability of generalizing across robot team sizes). Moreover, as the results in Fig. 4 demonstrate, it exhibits a steep performance decrease for growing robot team sizes; this weakness can potentially be mitigated by increasing the network capacity, but clearly, this strategy does not scale.

In contrast, our decentralized framework generalizes to different numbers of robots, as seen in Sec. 6.4.1 and Sec. 6.4.2. With communication (\(K > 1\)), the network performs more stably than the centralized (CP) and non-communicative GNN (\(K = 1\)) as the number of robots increases. We note that the centralized solution [30], implemented in C++, can compute the solution for a given case in less than 0.02 s for 8 agents. However, a centralized unit is required to obtain the states and goals of all robots, and to broadcast back a solution. In contrast, a single forward pass of our model (enabling a robot to predict its action) takes only 0.0019 ± 2.15e-06 s on the workstation described in Sec. 6.3. In addition to the decentralized nature of our solution, this speed of computation is beneficial in real-world deployments, where each robot runs its own (localized) action policy. We note that, in contrast, the expert algorithm [30] is intractable for more than 14 agents in dense environments within the given timeout; this is corroborated by results in [2, 26].

The experiment in Sec. 6.4.2 showed the capability of our decentralized policy to generalize to robot teams of larger scales. Table 5a and Table 5b showed that the framework trained in smaller robot teams (\(n = 2, 4\)) tends to perform worse than those trained in larger teams (\(n = 6, 8, 10\)), across any unseen instances (larger as well as smaller robot teams).

We perform subsequent experiments on larger robot teams to further test the generalization. We train a GNN (\(K = 3\)) network with the online expert on 10 robots, and test it on 20 and 40 robots in 28 × 28 and 40 × 40 environments, respectively (gridmaps are scaled to preserve the same effective density). Different from our success rate metric, which only considers complete cases (all robots reach their goals), Fig. 6 presents the proportion of cases distributed over the number of robots reaching their goals. The distributions show that, in both cases, more than 50% of all robots always reach their goals, and in 70% of cases, more than 80% of robots reach their goals. For instance, in the 20-robot example, in 395 of 500 cases (79%), at least 16 robots reach their goals. In the 40-robot example, in 141 of 200 cases (70.5%), at least 32 robots reach their goals.

![Histogram](image)

**Figure 6:** Histogram of proportion of cases distributed over the number of robots reaching their goal; the network is trained on 10 robots, and tested on 20 and 40 robots respectively with hop count \(K = 3\) (OE).
implementation (without data aggregation) could not. Moreover, a significant right-shift of the distribution is visible.

There are some assumptions and corresponding limitations in the current implementation, which will be improved in future work. Firstly, we assumed that communication between robots was achieved instantly without delay. Time-delayed aggregation GNNs [36] can be introduced to extend our framework to handle the time-delayed scenario.

Secondly, inter-robot collisions, especially position swaps, are the main reason causing the lower success rates for larger teams. However, the current decentralized policy was only able to learn collision-avoidance implicitly from the solution of the expert algorithm. The optimal planner tends to avoid inter-robot collision several steps ahead through collision-based search. This artifact reduces the number of ‘interesting’ cases in our training set, where collisions happen in coming steps. One potential solution to this is to deploy a policy gradient to add a penalty on the action causing a collision. However, such a strategy (e.g., as implemented in [26]) is harder to train, and is left for future work.

8 CONCLUSIONS

We considered the problem of collision-free navigation in multi-robot systems where the robots are restricted in observation and communication range. We proposed a combined architecture, composed of a convolutional neural network that extracts adequate features from local observations, and a graph neural network that communicates these features among robots. The key idea behind our approach is that we jointly trained these two components, enabling the system to best determine what information is relevant for the team of robots as a whole. We devised a data aggregation strategy (through an online expert) that facilitated faster learning. We performed experiments in cluttered environments with robot teams of varying sizes, and measured the success rates and sum of costs over the resulting planned paths. The results demonstrated the validity of our approach; in particular, we showed our model’s capability to generalize to previously unseen cases involving much larger robot teams.

This work is the first to apply GNNs to the problem of multi-robot path planning. Our results show that we are very close to achieving the same performance as first-principles-based methods. Of particular importance is that fact that we can already scale our system to sizes that are intractable for coupled centralized solvers, while remaining computationally feasible through our decentralized approach.

9 ACKNOWLEDGMENTS

We gratefully acknowledge the support of ARL grant DCIST CRA W911NF-17-2-0181. A. Prorok was supported by the Engineering and Physical Sciences Research Council (grant EP/S015493/1). We gratefully acknowledge their support.

REFERENCES

[1] Javier Alonso-Mora, Andreas Breitenmoser, Paul Beardsley, and Roland Siegwart. 2012. Reciprocal collision avoidance for multiple car-like robots. In Robotics and Automation (ICRA). 2012 IEEE International Conference on. IEEE, 360–366.

[2] Max Barer, Guni Sharon, Roni Stern, and Ariel Felner. 2014. Suboptimal variants of the conflict-based search algorithm for the multi-agent pathfinding problem. In Seventh Annual Symposium on Combinatorial Search.

[3] Peter Battaglia, Razvan Pascanu, Matthew Lai, Danilo Jimenez Rezende, and others. 2016. Interaction networks for learning about objects, relations and physics. In Advances in Neural Information Processing Systems (NIPS). 4502–4510.

[4] Vinay R Desaraju and Jonathan P How. 2012. Decentralized path planning for multi-agent teams with complex constraints. Autonomous Robots 32, 4 (2012), 385–403.

[5] David K Duvenaud, Dougal Maclaurin, Jorge Ibarra, Rafael Bombarell, Timothy Hirzel, Alan Aspuru-Guzik, and Ryan P Adams. 2015. Convolutional networks on graphs for learning molecular fingerprints. In Advances in Neural Information Processing Systems (NIPS). 2224–2232.

[6] John Enright and Peter R Wurman. 2011. Optimization and coordinated autonomy in mobile fulfillment systems. In Automated action planning for autonomous mobile robots. 33–38.

[7] M. Everett, Y. F. Chen, and J. P. How. 2018. Motion Planning Among Dynamic, Decision-Making Agents with Deep Reinforcement Learning. In 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). 3052–3059. https://doi.org/10.1109/IROS.2018.8593871

[8] Cornelia Ferner, Glenn Wagner, and Howie Choset. 2013. OdMr*: Optimal multirobot path planning in low dimensional search spaces. In IEEE International Conference Roboticists and Automation (ICRA). 3854–3859.

[9] F. Gama, J. Brunia, and A. Ribeiro. 2019. Stability Properties of Graph Neural Networks. arXiv:1905.04497v2 [cs.LG] (4 Sep. 2019). http://arxiv.org/abs/1905.04497.

[10] F. Gama, A. G. Marques, G. Leus, and A. Ribeiro. 2019. Convolutional Neural Network Architectures for Signals Supported on Graphs. IEEE Trans. Signal Process. 67, 4 (Feb 2019), 1034–1049.

[11] Justin Gilmer, Samuel Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. 2017. Neural message passing for quantum chemistry. Proceedings of the 34th International Conference on Machine Learning 70 (2017).

[12] Pasquale Grippa, Doris A Behrens, Christian Bettstetter, and Friederike Wall. 2017. Job selection in a network of autonomous UAVs for delivery of goods. Robotics: Science and Systems (2017).

[13] A. Heimowitz and Y. C. Eldar. 2017. A Unified View of Diffusion Maps and Signal Processing on Graphs. In 2017 Int. Conf. Sampling Theory and Appl. IEEE, Tallinn, Estonia, 308–312.

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

[15] Yugang Lin and Goldie Nejat. 2013. Robotic urban search and rescue: A survey from the control perspective. Journal of Intelligent & Robotic Systems 72, 2 (2013), 147–165.

[16] Léaëtizia Matignon, Laurent Jeanprêtre, and Abdel-Bihl Mouaddib. 2012. Coordinated multi-robot exploration under communication constraints using decentralized markov decision processes. In Twenty-sixth AAAI conference on artificial intelligence.

[17] A. Ortega, P. Frossard, J. Kováčević, J. M. F. Moura, and P. Vanderdelenst. 2018. Graph Signal Processing: Overview, Challenges and Applications. Proc. IEEE 106, 5 (May 2018), 808–828.

[18] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. 2017. Automatic Differentiation in PyTorch. In NIPS Autodiff Workshop. Advances in Neural Information Processing Systems. Long Beach, CA, USA, 1–4.

[19] Amanda Prorok. 2018. Graph Neural Networks for Learning Robot Team Coordination. Federated AI for Robotics Workshop, IFCAI-ECAI/ICML/AAMAS 2018, arXiv:1805.03737 [cs] (May 2018). http://arxiv.org/abs/1805.03737 arXiv: 1805.03737.

[20] Amanda Prorok and Vijay Kumar. 2017. Privacy-preserving vehicle assignment for mobility-on-demand systems. In 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). IEEE, 1869–1876.

[21] Aravind Rajeswaran, Kendall Lowrey, Emanuel V. Todorov, and Sham M Kakade. 2017. Towards Generalization and Simplicity in Continuous Control. In Advances in Neural Information Processing Systems (NIPS). I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (Eds.). 6550–6561. http://papers.nips.cc/paper/ 7233-towards-generalization-and-simplicity-in-continuous-control.pdf

[22] Stéphane Ross, Geoffrey Gordon, and Drew Bagnell. 2011. A reduction of imitation learning and structured prediction to no-regret online learning. In Proceedings of the fourteenth international conference on artificial intelligence and statistics. 627–635.

[23] M. Rotkowitz and S. Lall. 2005. A characterization of convex problems in decentralized control. IEEE Trans. Automatic Control. 50, 12 (Dec. 2005), 1984–1996. https://doi.org/10.1109/TAC.2005.868365

[24] A. Sandryhaila and J. M. F. Moura. 2013. Discrete Signal Processing on Graphs. IEEE Trans. Signal Process. 61, 7 (April 2013), 1644–1656.

[25] Guillaume Sartoretto, Justin Kent, Yuanfei Shi, Glenn Wagner, TK Kumar, Sven Koenig, and Howie Choset. 2019. PRIMAL: Pathfinding via Reinforcement and Imitation Multi-Agent Learning. IEEE Robotics and Automation Letters 4 (2019), 2378–2385. Issue 3.
[26] G. A. Sartoretti, J. Kerr, Y. Shi, G. Wagner, T. K. S. Kumar, S. Koenig, and H. Choset. 2019. PRIMAL: Pathfinding via Reinforcement and Imitation Multi-Agent Learning. IEEE Robotics and Automation Letters (2019), 1–1. https://doi.org/10.1109/LRA.2019.2903261

[27] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. 2009. The graph neural network model. IEEE Transactions on Neural Networks 20, 1 (2009), 61–80.

[28] Tom Schouwenaars, Bart De Moor, Eric Feron, and Jonathan How. 2001. Mixed integer programming for multi-vehicle path planning. In European Control Conference (ECC). IEEE, 2603–2608.

[29] Guni Sharon, Roni Stern, Ariel Felner, and Nathan R Sturtevant. 2015. Conflict-based search for optimal multi-agent pathfinding. Artificial Intelligence 219 (2015), 40–66.

[30] Guni Sharon, Roni Stern, Ariel Felner, and Nathan R Sturtevant. 2015. Conflict-based search for optimal multi-agent pathfinding. Artificial Intelligence 219 (2015), 40–66.

[31] D. I Shuman, S. K. Narang, P. Frossard, A. Ortega, and Gabriele Monfardini. 2009. The graph neural network model. IEEE Transactions on Neural Networks 20, 1 (2009), 61–80.

[32] David Silver. 2005. Cooperative Pathfinding. Artificial Intelligence and Interactive Digital Entertainment 1 (2005), 117–122.

[33] Kevin Spieser, Samitha Samaranayake, Wolfgang Gruel, and E Frazolli. 2016. Shared-vehicle mobility-on-demand systems: a fleet operator’s guide to rebalancing empty vehicles. In Transportation Research Board.

[34] Trevor Standlee and Richard Korf. 2011. Complete Algorithms for Cooperative Pathfinding Problems. In Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence - Volume Volume One (IJCAI'11). AAAI Press, Barcelona, Catalonia, Spain, Article 1, 6 pages. https://doi.org/10.5591/978-1-57735-516-8/IJCAI11-118

[35] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel. 2017. Domain randomization for transferring deep neural networks from simulation to the real world. In 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). 23–30. https://doi.org/10.1109/IROS.2017.8201133

[36] E. Tolstaya, F. Gama, J. Paulos, G. Pappas, V. Kumar, and A. Ribeiro. 2019. Learning Decentralized Controllers for Robot Swarms with Graph Neural Networks. In Conf. Robot Learning 2019. Int. Found. Robotics Res., Osaka, Japan.

[37] Jur Van Den Berg, Stephen J Guy, Ming Lin, and Dinesh Manocha. 2011. Reciprocal n-body collision avoidance. In Robotics research. Springer, Berlin, Heidelberg, 3–19.

[38] Jur Van den Berg, Ming Lin, and Dinesh Manocha. 2008. Reciprocal velocity obstacles for real-time multi-agent navigation. In IEEE International Conference on Robotics and Automation (ICRA). IEEE, 1928–1935.

[39] J. P. van den Berg and M. H. Overmars. 2005. Prioritized motion planning for multiple robots. In 2005 IEEE/RSJ International Conference on Intelligent Robots and Systems. IEEE, Edmonton, Alta., Canada, 430–435. https://doi.org/10.1109/IROS.2005.1545396

[40] Y. Wang, N. Matni, and J. C. Doyle. 2018. Separable and Localized System-Level Synthesis for Large-Scale Systems. IEEE Trans. Automat. Control 63, 12 (Dec. 2018), 4234–4249. https://doi.org/10.1109/TAC.2018.2819246

[41] Wenyong Wu, Subhraraj Bhattacharya, and Amanda Prorok. 2019. Multi-Robot Path Deconfliction through Prioritization by Path Prospects. arXiv:1908.02361 [cs] (Aug. 2019). http://arxiv.org/abs/1908.02361 arXiv: 1908.02361.

[42] Jingjin Yu and Steven M LaValle. 2013. Structure and intractability of optimal multi-robot path planning on graphs. In AAAI.

[43] Jingjin Yu and Steven M. LaValle. 2015. Optimal Multi-Robot Path Planning on Graphs: Complete Algorithms and Effective Heuristics. arXiv:1507.03290 [cs] (July 2015). http://arxiv.org/abs/1507.03290 arXiv: 1507.03290.
A SUPPLEMENTARY RESULTS

This appendix presents supplementary material for our approach. Fig. 7 exemplifies a typical failure case caused by inter-robot collision (i.e., in this case, a position-swap). Fig. 8 compares the GNN performance with and without the online expert (OE). We also investigate the impact of filter taps \( K \) (Fig. 9).

Fig. 7 illustrates the target path computed by the expert algorithm and the path computed by the network trained and tested on 8 robots. Different from a search-based method, the proposed framework can not explore the future collision ahead of time. This results in the inter-robot collision in Fig. 7 (b). Although the collision shielding can force robots to remain idle, the lack of solutions for such scenarios (e.g., position-swap) in the offline data-set causes the deadlock. This yields the need for an online expert (through a data-set aggregation method, as described in Sec. 5.8) that provides a solution from this configuration onward.

To further evaluate the improvement provided by the online expert, we trained the GNN with and without the online expert, and tested the networks on 16 and 32 robots in 28 \( \times \) 28 and 40 \( \times \) 40 environments. Fig. 8 shows a distribution shift from the GNN without the online expert to the GNN with the online expert. We see how the GNN network with the online expert tends to generalize better than the one without, since the proportion of robots reaching the goal is significantly larger. This holds for tests on 16 and 32 robots, indicating the network starts to learn how to solve those cases that the GNN network without the online expert could not solve.

To evaluate the impact of filter taps \( K \), we increases the hop count \( K \) from 2 into 3 in our experiment of testing 20 and 40 robots. We observed the proportion of robots reaching the goal slightly increases from \( K = 2 \) into 3 in Fig. 9.