GRAPH-BASED MOTION PLANNING NETWORKS

Anonymous authors
Paper under double-blind review

ABSTRACT

Differentiable planning network architecture has shown to be powerful in solving transfer planning tasks while possesses a simple end-to-end training feature. Many great planning architectures that have been proposed later in literature are inspired by this design principle in which a recursive network architecture is applied to emulate backup operations of a value iteration algorithm. However existing frameworks can only learn and plan effectively on domains with a lattice structure, i.e. regular graphs embedded in a certain Euclidean space. In this paper, we propose a general planning network, called Graph-based Motion Planning Networks (GrMPN), that will be able to i) learn and plan on general irregular graphs, hence ii) render existing planning network architectures special cases. The proposed GrMPN framework is invariant to task graph permutation, i.e. graph isormophism. As a result, GrMPN possesses the generalization strength and data-efficiency ability. We demonstrate the performance of the proposed GrMPN method against other baselines on three domains ranging from 2D mazes (regular graph), path planning on irregular graphs, and motion planning (an irregular graph of robot configurations).

1 INTRODUCTION

Reinforcement learning (RL) is a sub-field of machine learning that studies about how an agent makes sequential decision making (Sutton et al., 1998) to interact with an environment. These problems can in principle be formulated as Markov decision process (MDP). (Approximate) Dynamic programming methods such as value iteration or policy iterations are often used for policy optimization. These dynamic programming approaches can also be leveraged to handle learning, hence referred as model-based RL (Kober et al., 2013). Model-based RL requires an estimation of the environment model hence is computationally expensive, but it is shown to be very data-efficient. The second common RL paradigm is model-free which does not require a model estimation hence has a lower computation cost but less data-efficiency (Kober et al., 2013). With a recent marriage with deep learning, deep reinforcement learning (DRL) has achieved many remarkable successes on a wide variety of applications such as game (Mnih et al., 2015; Silver et al., 2016), robotics (Levine et al., 2016), chemical synthesis (Segler et al., 2017), news recommendation (Zhang et al., 2019) etc. DRL methods also range from model-based (Kurutach et al., 2018; Lee et al., 2018a) to model-free (Mnih et al., 2015; Heess et al., 2015) approaches.

On the other hand, transfer learning across tasks has long been desired because it is much more challenging in comparison to single-task learning. Recent work (Tamar et al., 2016) has proposed a very elegant idea that suggests to encode a differentiable planning module in a policy network architecture. This planning module can emulate the recursive operation of value iterations, called Value Iteration Networks (VIN). Using this network, the agent is able to evaluate multiple future planning steps for a given policy. The planning module is designed to base on a recursive application of convolutional neural networks (CNN) and max-pooling for value function updates. VIN not only allows policy optimization with more data-efficiency, but also enables transfer learning across problems with shared transition and reward structures. VIN has laid foundation for many later differentiable planning network architectures such as QMDP-Net (Karkus et al., 2017), planning under uncertainty(Gupta et al., 2017), Memory Augmented Control Network (MACN) (Khan et al., 2018), Predictron (Silver et al., 2017), planning networks (Srinivas et al., 2018) etc. However, these approaches including VIN is limited to learning with regular environment structures, i.e. the transition function forms an underlying 2D lattice structure.

Recent works have tried to mitigate this issue by resorting to graph neural networks. These work exploit geometric intuition in environments which have irregular structures such as generalized VIN (Niu et al., 2018), planning on relational domains (Toyer et al., 2018; Bajpai et al., 2018), (Ma et al., 2018), automated planning for scheduling (Ma et al., 2018), etc. The common between these approaches are in the use of graph neural networks to process irregular data structures like graphs.

Among these frameworks, only GVIN is able to emulate the value iteration algorithm on irregular graphs of arbitrary sizes, e.g. generalization to arbitrary graphs. GVIN has a differentiable policy network architecture which is very similar to VIN. GVIN can also have a *zero-shot planning* ability on unseen graphs. However, GVIN requires domain knowledge to design a graph convolution which might limit it to become a universal graph-based path planning framework.

In this paper, we aim to demonstrate different formulations for value iteration networks on irregular graphs. These proposed formulations are based on different graph neural network models. These models are capable of learning optimal policies on general graphs where their transition and reward functions are not provided a priori and yet to be estimated. These models are known to be invariant to graph isomorphism, therefore they are able to have a generalization ability to graphs of different sizes and structures. As a result, they enjoy the ability of *zero-shot learning to plan*. Specifically, it is known that Bellman equations are written as the form of message passing, therefore we propose using message passing neural networks (MPNN) to emulate the value iteration algorithm on graphs. We will show two most general formulations of graph-based value iteration network that are based on two general-purpose approaches in the MPNN family: Graph Networks (GN) (Battaglia et al., 2018) and Graph Attention Networks (GAT) (Velickovic et al., 2018). In particular, our contributions are three-fold:

- We develop a MPNN based path planning network (GrMPN) which can learn to plan on general graphs, e.g. regular and irregular graphs. GrMPN is an differentiable end-to-end planning network architecture trained via imitation learning. We implement GrMPN via two formulations that are based on GN and GAT.
- GrMPN is a general graph-based value iteration network that will render existing graph-based planning algorithms special cases. GrMPN is invariant to graph isomorphism which enables transfer planning on graphs of different structure and size.
- We will demonstrate the efficacy of GrMPN which achieves state of the art results on various domains including 2D maze with regular graph structures, irregular graphs, and motion planning problems. We show that GrMPN outperforms existing approaches in terms of data-efficiency, performance and scalability.

2 BACKGROUND

This section provides background on Markov decision process (MDP), value iteration algorithm, value iteration networks (VIN) and graph neural networks (GNN).

2.1 MARKOV DECISION PROCESS AND VALUE ITERATION

A MDP is defined as $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R})$, where $\mathcal{S}$ and $\mathcal{A}$ represent state and action spaces. $\mathcal{P}$ defines a transition function $\mathcal{P}(s, a, s') = P(s'|s, a)$, where $s, s' \in \mathcal{S}, a \in \mathcal{A}$. A planning algorithm, e.g. dynamic programming (Bertsekas et al., 1995), aims to find an optimal policy $\pi : \mathcal{S} \mapsto \mathcal{A}$ so that a performance measure $V^\pi(s) = \mathbb{E}\left(\sum_t \gamma^t r_t | s_0 = s\right)$ is maximized, for all state $s \in \mathcal{S}$; where $\gamma \in (0, 1)$ is a discount factor. The expectation is w.r.t stochasticity of $\mathcal{P}$ and $\mathcal{R}$. Value iteration (VI) is one of dynamic programming algorithms that can *plan* on $\mathcal{M}$. It starts by updating the value functions $V(s), \forall s \in \mathcal{S}$ iteratively via the Bellman backup operator T, $V^{(k)} = TV^{(k-1)}$ as follows:

$$V^k(s) = \max_{a \in \mathcal{A}} \left[\mathcal{R}(s, a) + \gamma \sum_{s'} P(s'|s, a) V^{(k-1)}(s') \right]$$

where k is an update iteration index. T is applied iteratively until $V^{(k)}(\cdot)$ converges to optimal values. It is proved that the operator T is a Lipschitz map with a factor of γ. In other words, as $k \to \infty$, $V^{(k)}$ converges to a fixed-point value function V^*. As a result, the optimal policy π^* can be computed as, $\pi^*(s) = \arg\max_{a \in \mathcal{A}} \left[\mathcal{R}(s, a) + \gamma \sum_{s'} P(s'|s, a) V^*(s')\right]$. Q-value functions are also defined similarly as $Q(s, a) = \mathbb{E}\left(\sum_t \gamma^t r_t | s_0 = s, a_0 = a\right)$. In addition, we have the relation between V and Q value functions as $V(s) = \max_a Q(s, a)$. For goal-oriented tasks, the reward function $\mathcal{R}(s, a)$ can be designed to receive low values at intermediate states and high values at goal states s^*.

Value iteration network Planning on large MDPs might be very computationally expensive, hence transfer-planning would be desirable, especially for tasks sharing similar structures of $\mathcal{P}$ and $\mathcal{R}$. Value Iteration Networks (VIN) (Tamar et al., 2016) is an differentiable planning framework that can i) do *transfer-planning* for goal-oriented tasks with different goal states, ii) and learn the shared underlying MDP $\mathcal{M}$ between tasks, i.e. learning the transition $\mathcal{P}$ and reward $\mathcal{R}$ functions.

Let's assume that we want to find an optimal plan on a MDP M with unknown $\mathcal{P}$ and $\mathcal{R}$. VIN's policy network with embedded approximate reward and transition functions $\bar{\mathcal{R}}$ and $\bar{\mathcal{P}}$ is trained end-to-end through imitation learning. $\bar{\mathcal{R}}$ and $\bar{\mathcal{P}}$ are assumed to be from an unknown MDP $\bar{M}$ whose optimal policy can form useful features about the optimal policy in M. Based on observation feature $\phi(s)$ on state s, the relation between M and $\bar{M}$ is denoted as $\bar{R} = f_R(\phi(\cdot))$ and $\bar{\mathcal{P}} = f_P(\phi(\cdot))$. More specifically, inputs are 2D images, e.g. of the $m \times m$ maze with start and goal states; outputs are optimal paths going from start to goal. VIN embeds value iteration as a recursive application of convolutions and max-pooling over the feature channels. VIN consists of a convolutional layer $Q_a \in \Re^{m \times m}$ with $|\mathcal{A}|$ channels. The trainable parameters are $W^R_{a,i,j}$ and $W^P_{a,i,j}$ with $|\mathcal{A}|$ channels, which account for the reward and transition embeddings. The recursive process contains two following convolution and max-pooling operations,

$$Q^{(k)}_a = W^R_a \bar{R} + W^P_a V^{(k-1)}, \qquad V^{(k)} = \max_a Q^{(k)}_a \tag{1}$$

where the convolution operators on R.H.S in the first equation is written as:

$$W^R_a \bar{R} = \sum_{i,j} W^R_{a,i,j} \bar{R}_{i'-i,j'-j}, \quad W^P_a V^{(k-1)} = \sum_{i,j} W^P_{a,i,j} \bar{V}^{(k-1)}_{i'-i,j'-j}$$

where i, j are cell index in the maze. VIN has later inspired many other differentiable planning algorithms. For example, VIN's idea can again be exploited for diffirentiable planning architectures for planning on partially observable environments such as QMDP-Net (Karkus et al., 2017), (Gupta et al., 2017), Memory Augmented Control Network (MACN) (Khan et al., 2018). A related differentiable planning network is also used in the Predictron framework (Silver et al., 2017) where its core planning module aims to estimate a Markov reward process that can be rolled forward for many imagined planning steps. A notable extension of VIN is proposed by (Lee et al., 2018b), called Gated path planning networks (GPPN), in which they use LSTM to replace the recursive VIN update, i.e.

$$h^{(k)}_{i',j'}, c^{(k)}_{i',j'} = \text{LSTM}\left(\sum_{i,j}\left(W^R_{a,i,j}\bar{R}_{i'-i,j'-j} + W^P_{a,i,j}\bar{V}^{(k-1)}_{i'-i,j'-j}\right), c^{(k-1)}_{i'.j'}\right) \tag{2}$$

These algorithms show great success at path planning on many different grid-based navigation tasks in which the states are either fully or partially observable. However the underlying state space must assume regular lattices in order to exploit local connectivity through the help of convolution operations of CNN. This limits their applications to domains whose state spaces might be in the forms of irregular graphs.

Generalized value iteration networks There is recent effort considering planning and reinforcement learning whose state transitions form a general graph. Niu et. al. (Niu et al., 2018) propose such a graph-based model-based deep reinforcement learning framework that generalizes VIN to differentiable planning on graphs, called generalized value iteration (GVIN). GVIN takes a graph with a certain start node and a goal node as input, and output an optimal plan. GVIN can learn an underlying MDP and an optimal planning policy via either imitation learning or reinforcement learning. Inspired by VIN, GVIN applies recursive graph convolution and max-pooling operators to emulate the value iteration algorithm on general graphs. With a specially designed convolution kernel, GVIN can also transfer planning to unseen graphs of arbitrary size.

Specifically, an input is a directed, weighted spatial graph $G = (\mathcal{V}, \mathcal{E})$ where $\mathcal{V} = \{v_i\}_{i=1:N^v}$ is a set of nodes with node attribute $v_i \in \Re^{d^v}$. $\mathcal{E} = \{e_k\}_{k=1:N^e}$ is a set of edges where $e_k \in \Re^{d^e}$ is the edge attribute, e.g. edge weights; d^v, d^e are the node and edge features' dimension, respectively. In addition, we denote $A \in \Re^{N^v \times N^v}$ as the adjacency matrix. We denote $W^P_a \in \Re^{N^v \times N^v}$ as a convolution operator. Each input graph with a goal state $v^* \in \{0, 1\}^{N^v}$ (one-hot vector labels the goal state) describes one task. GVIN constructs convolution operators W^P_a as a function of the input graph G. The reward graph signal is denoted as $\bar{R} = f_R(G, v^*)$, where f_R is a CNN in VIN, but an identity function in GVIN. The value functions $V(v)$ with $v \in \mathcal{V}$ on graph nodes can be computed recursively as follows,

$$Q^{(k)}_a = W^P_a(\bar{R} + \gamma V^{(k-1)}), \qquad V^{(k)} = \max_a Q^{(k)}_a. \tag{3}$$

While VIN uses CNN to construct W^P_a that could only capture 2D lattice structures, GVIN designs directional and spatial kernels to construct W^P_a that try to capture invariant translation on irregular graphs. However we will show that these kernels are not enough to capture invariance to graph isomorphism, which leads to a poor performance in domains with a complex geometric structure. In particular, though it works well on multiple navigation tasks, GVIN is shown to be sensitive to the

choice of hyperparameters and specially designed convolution kernels, e.g. directional discretization.

2.2 GRAPH NEURAL NETWORKS

Graph neural networks (GNN) (Scarselli et al., 2008) have received much attention recently as they can process data on irregular domains such as graphs or sets. The general idea of GNN is to compute an encoded feature h_i for each node v_i based on the structure of the graph, node v_i and edge e_{ij} features, and previous encoded features as $h_i = \sum_{j \in \mathcal{N}(i)} f\left(h_i, h_j, v_i, v_j, e_{ij}\right)$, where f is a parametric function, and $\mathcal{N}(i)$ denotes the set of neighbour nodes of node i. After computing h_i (probably apply f for k iterations), an additional function is used to compute the output at each node, $y_i = g(h_i, v_i)$, where g is implemented using another neural network, called a read-out function.

Graph convolution network Many earliest work on GNN propose extending traditional CNNs to handle convolution operations on graphs through the use of spectral methods (Bruna et al., 2013; Henaff et al., 2015; Kipf & Welling, 2017). For example, graph convolution networks (GCN) (Kipf & Welling, 2017) is based on fundamental convolution operations on spectral domains. GCN must assume graphs of same size. Therefore, these methods which rely on the computation of graph eigenvectors are either computationally expensive or not able to learn on graphs of arbitrary sizes. Many later introduced graph convolutional networks on spatial domain such as Neural FPs (Duvenaud et al., 2015), PATCHY-SAN (Niepert et al., 2016), DCNN (Atwood & Towsley, 2016), etc., are able to learn on graphs of arbitrary sizes. However they are limited to either the choice of a subset of node neighbours or random walks of k-hop neighborhouds. These drawbacks limit graph convolution based methods to applications on large-scale graphs with highly arbitrary sizes and structures, hence not favourable for transfer planning in MDPs.

Graph attention network (GAT) GAT (Velickovic et al., 2018) is inspired by the attention mechanism by modifying the convolution operation in GCN in order to make learning more efficient and scalable to domains of large graphs. Specifically, the encoding at each node is recursively computed as, $h_i^{(k)} = \sigma\left(\sum_{j \in \mathcal{N}(i)} \alpha_{ij}^{(k)} W^{(k)} h_j^{(k-1)}\right)$, where σ is an activation function, and $\alpha_{ij}^{(l)}$ is the attention coefficients which are computed as

$$\alpha_{ij}^{(k)} = \frac{\exp\left(\text{LeakyReLU}(\mathbf{a}^\top [W^{(k)} h_i, W^{(k)} h_j])\right)}{\sum_{k \in \mathcal{N}(i)} \exp\left(\text{LeakyReLU}(\mathbf{a}^\top [W^{(k)} h_i, W^{(k)} h_k])\right)}$$

where $\mathbf{a}$ is weight vector, and k denotes the embedding layer k whose weights are $W^{(k)}$.

Message passing neural network (MPNN) MPNN (Gilmer et al., 2017) uses the mechanism of message passing to compute graph embedding. In particular, the calculation of a feature on each node involves two phases: i) message passing and readout. The message passing operation at a node is based on its state and all messages received from neighbor nodes. The readout is based on the node state and the calculated message. These phases are summarised as follows

$$m_i^{(k)} = \sum_{j \in \mathcal{N}(i)} f(h_i^{(k-1)}, h_j^{(k-1)}, e_{ij}), \quad h_i^{(k)} = g(h_i^{(k-1)}, m_i^{(k)})$$

MPNN is a unified framework for graph convolution and other existing graph neural networks back to that time, e.g. Graph convolution network and Laplacian Based Methods (Duvenaud et al., 2015; Bruna et al., 2013; Henaff et al., 2015; Kipf & Welling, 2017), Gated Graph Neural Networks (GG-NN) (Li et al., 2016), Interaction Nework (IN) (Battaglia et al., 2016), Molecular Graph Convolutions (Kearnes et al., 2016), Deep Tensor Neural Networks (Schütt et al., 2017). Gilmer et.al. (Gilmer et al., 2017) has made great effort in converting these frameworks to become a MPNN variant. MPNN is designed similarly to GG-NN in which GRU is applied to implement the recursive message operation, but different at message and output functions. Specifically, the message sent from node j to i is implemented as $f(h_i, h_j, e_{ij}) = A(e_{ij})h_j$, where $A(e_{ij})$ is implemented as a neural network that maps edge feature e_{ij} to an $d^v \times d^v$ operator. Another variant of the message funcion that is additionally based on the receiver node feature h_i was also implemented in the paper. Updating message with all received information h_i, h_j, e_{ij} is inspired by (Battaglia et al., 2016). MPNN has shown the state-of-the-art performance in prediction tasks on large graph dataset, e.g. molecular properties.

Graph network (GN) Graph networks (GN) (Sanchez-Gonzalez et al., 2018; Battaglia et al., 2018) is a general framework that combines all previous graph neural networks. The update operations of GN involve nodes, edges and global graph features. Therefore it renders MPNN, GNN,

GCN, GAT as special cases. Specifically, if we denote an additional global graph feature as u, the updates of GN which consist of three update functions g, and three $aggregation$ functions f. These functions are implemented based on the message passing mechanism. The aggregation functions are 1) $m_i = f^{e\to v}(\{e_{ij}\}_{j\in\mathcal{N}(i)})$ aggregate messages sent from edges to compute information of node i; 2) $m^e = f^{e\to u}(\{e_{ij}\}_{j\in\mathcal{N}(i),\forall i})$ aggregate messages sent from all edges to the global node u; 3) $m^v = f^{v\to u}(\{v_i\}_{\forall i})$ aggregate messages sent from all nodes to the global node u. These aggregation functions must be invariant to the order of nodes, which is critical to the Weisfeiler-Lehman (WL) graph isomorphism test (Weisfeiler & Lehman, 1968) and regarded as an important requirement for graph representation learning. Using aggregated information, the three update functions to node, edge and global features are defined as follows

$$e_{ij}^{(k+1)} = g^e(e_{ij}^{(k)}, v_i^{(k)}, v_j^{(k)}, u^{(k)}), \quad v_i^{(k+1)} = g^v(m_i, v_i^{(k)}, u^{(k)}), \quad u^{(k+1)} = g^u(m^e, m^v, u^{(k)})$$

The aggregation functions could be element-wise summation, averages, or max/min operations. The update functions could use general neural networks. The use of edge features and the global graph node makes GN distinct from MPNN. In addition, the use of a recursive update from immediate neighbors is in contrast to multi-hop updates of many spectral methods.

3 GRAPH-BASED MOTION PLANNING NETWORKS

In this section, we propose to use a general graph neural network to construct inductive biases for "*learning to plan*", called graph-based motion planning network (GrMPN). Similar to GVIN, our framework is trained on a set of motion planning problems as inputs and associated optimal paths (or complete optimal policies) as outputs. The inputs are general graphs without knowing its reward function and transition. At test time, GrMPN takes i) a test problem defined as a general graph and ii) a pair of starting node and a goal node. The target is to return an optimal plan for the test problem.

Problem setting: Given a dataset consists of input-output pairs $\mathcal{D} = \{G_i, \tau_i^*\}_{i=1}^N$, where G_i is a general graph $G = (\mathcal{V}, \mathcal{E})$, and τ_i^* is an *optimal path* (or an optimal policy) which can be generated by an expert as a demonstration. The target is to learn an *optimal policy* for a new graph G with a starting node and a goal node. This learning problem can be either formulated as imitation learning or reinforcement learning (Tamar et al., 2016; Niu et al., 2018). While it is straightforward to train with RL, within the scope of this paper we only focus on the formulation of imitation learning.

General GrMPN framework We propose a general framework for graph-based value iteration that is based on the principle of message passing. First, we also design a feature-extract function to learn a reward function r: $r = f_R(G; v^*)$ given a graph G and a goal node v^*. We use a similar setting from GVIN (Niu et al., 2018) for f_R (a more general architecture is depicted in Appendix A.1. Second, GrMPN also consists of the following recurrent application of graph operations at all nodes i and edges ij.

$$q_{ai}^{(k)} = f^{e\to v}(r, \{e_{ij}, v_j^{(k-1)}\}_{j\in\mathcal{N}(i)}), \quad e_{ij}^{(k)} = g^e(e_{ij}^{(k-1)}, v_i^{(k-1)}, v_j^{(k-1)}), \quad v_i^{(k)} = g^v(q_{ai}^{(k)})$$

where k is the processing step index; v_i is the node feature (which also contains the node's value function); $f^{e\to v}, g^e, g^v$ are defined as aggregation and update functions. Note that we use q_{ai} as edge features. If the transition is stochastic, we can use $|\mathcal{A}|$ channels on edges.

3.1 GRMPN VIA GRAPH NETWORKS

GrMPN can be implemented based on GN (Sanchez-Gonzalez et al., 2018; Battaglia et al., 2018). GrMPN does not represent the global node u. It consists of the representation of nodes and edges, hence uses one aggregation function and two update functions, as described below:

$$m_i = \sum_{j\in\mathcal{N}(i)} e_{ij}, \quad e_{ij} = g^e(r_j + \gamma e_{ij}), \quad v_i = g^v(m_i)$$

Note that for brevity the above updates assume deterministic actions, similar to the setting in VIN and GVIN. The message aggregation is equivalent to a sum over actions $\sum_a \sum_{s'} p(s'|s,a)V(s')$, similar to the implementation of GPPN (Lee et al., 2018b). The algorithm of GrMPN via Graph Networks (GrMPN-GN) is summarized in Algorithm 1.

3.2 GRMPN VIA GRAPH ATTENTION NETWORKS

In general, many graph neural networks can be used to represent the graph-based value iteration module. As MPNN, GGNN and other similar approaches are special cases of GN, therefore we can

Algorithm 1 GrMPN via Graph Networks (GrMPN-GN)

Input: graph $G = \{V, E\}$, a goal node v^*
Extract rewards: $r = f_R(G, v^*)$
for k iterations **do**
  for each edge $\{e_{ij}\}$ **do**
    Compute edge update: $e_{ij} = g^e(r_j + \gamma e_{ij})$
  end for
  for each node $\{v_i\}$ **do**
    Compute node aggregation: $m_i = \sum_{j \in \mathcal{N}(i)} e_{ij}$
    Compute node update: $v_i = g^v(m_i)$
  end for
end for
Output: $\{v_i^{(k)}\}_{i=1}^{N^v}$

easily rewrite GrMPN to become an application of these approaches. In this section, we draw a connection of GrMPN to GAT, and show how GAT can also be used to represent a graph-based VI module. We use multi-heads to represent $|\mathcal{A}|$ channels for the Q value functions. Each head represents a value function corresponding to an action Q_a, $V_i^{(k)} = \sigma\left(\sum_{j \in \mathcal{N}(i)} \alpha_{ij}^a W^a (r_{ij} + \gamma V_j^{(k-1)})\right)$, where σ is a *Maxout* activation function or $\sum_a$ over possible actions (in our implementation we chose the latter); and α_{ij}^a is the attention coefficients of channel a which are computed as

$$\alpha_{ij}^a = \frac{\exp\left(\text{LeakyReLU}(\beta^\top [W^a V_i, W^a V_j])\right)}{\sum_{k \in \mathcal{N}(i)} \exp\left(\text{LeakyReLU}(\beta^\top [W^a V_i, W^a V_k])\right)}$$

We denote this formulation as GrMPN-GAT. We can also make a connection between GrMPN-GN and GrMPN-GAT by reformulating the updates and coefficients of GATPPN to message passing in which edge attentions become edge updates. The edge feature now have the attention coefficients α as additional information. GrMPN-GAT is rewritten as a GN module with attentions as:

$$e_{ij} = (\alpha_{ij}, q_{ij}) = g^e\left(r_j + \gamma e_{ij}\right), \quad m_i = \frac{1}{\sum_{j \in \mathcal{N}(i)} \alpha_{ij}} \sum_{j \in \mathcal{N}(i)} \alpha_{ij} q_{ij}, \quad v_i = g^v(m_i)$$

where e_{ij} is the feature of the edge i-j that contains attention coefficients and q_{ij} (equivalently to the Q-value function q_{ai}). The algorithm of GrMPN-GAT can be described similar to GrMPN-GN in Algorithm 1.

4 EXPERIMENTS

We evaluate the proposed framework GrMPN-GAT and GrMPN-GN on 2D mazes, irregular graphs, and motion planning problems. These problems range from simple regular graphs (2D mazes) to irregular graphs of simple (irregular grid) and complex geometric structures (motion planning). Through these experiments we want to confirm the following points:

1. GrMPN frameworks based on general graph neural networks can not only perform comparably to VIN for lattice structures, but also with more data-efficiency, because GrMPN-GAT and GrMPN-GN are invariant to graph isomorphism.
2. GrMPN is able to generalize well to unseen graphs of arbitrary sizes and structures, because they exploit graph isomorphism in a principled way. In particular, GrMPN handles long-range planning well by providing a great generalization ability a) across nodes in graphs b) across graphs. Therefore they are able to cope with planning on larger problems with high complexity, hence significantly improve task performance and data-efficiency.
3. GrMPN-GAT exploiting attended (weighted) updates for value functions, i.e. non-local updates (Battaglia et al., 2018), would outperform GrMPN-GN which only based on uniform sum over actions.

Settings: In all experiments, we follow the standard encode-decode graph network design. Firstly, each graph component (node, edge) feature is passed through a two-layered MLP encoder with ReLu activation and 16 hidden units each. To increase the robustness, this two-layered network is also used in other graph network components, include both the graph block module (GrMPN-GAT or GrMPN-GN) and the decoder network. Notably, in the lattice $20\times$ 20 experiment, as the number

of nodes is significantly large, we instead increase the hidden units up to 32. We use an additional node encoding f_v, to compute note features v in the motion planning problems (see Appendix A.1 a full policy network architecture). Finally, we use the standard RMSProp algorithm with a learning rate of 0.001 in all experiments. The numbers of message passing step k, is set differently in different experiments. Specifically, k is set respectively equal to 10, 15, 20 for 12×12, 16×16 and 20×20 2D mazes. Meanwhile, $k = 20$ in all other irregular graph experiments, except for the last motion planing task tested on a roadmap of 500 configuration where we set $k = 40$ to handle large graphs. We use other standard settings as used in the papers of VIN and GVIN (Note that VIN set the recurrent parameter as: k=40 on 2D mazes, k=200 on irregular graphs, and motion planning).

Metrics: We use three different metric which are also used in different work: i) %Accuracy (Acc) is the percentage of nodes whose predicted actions are optimal, ii) %Success (Succ) is the percentage of nodes whose predicted path reach the goal, and iii) path difference is the Euclidean distance over all nodes between a predicted path vs. optimal path.

Training: While extensions to RL training is straightforward, we are only focused on imitation learning. Therefore we use the same objective function used by VIN and GVIN, i.e. a cross-entropy loss for the supervised learning problem with dataset $\{v, a^* = \pi^*(v)\}$ where v is a state node with an optimal action a^* demonstrated by an expert.

4.1 DOMAIN I: 2D MAZES

In this experiment we carry out evaluations on 2D mazes which have a regular graph structure. We compare GrMPN against VIN, GVIN, and GPPN. The environment and experiment settings are set similar to VIN, GVIN, and GPPN. We use the same script used in VIN, GVIN, and GPPN to generate graphs. For each each graph, we generated seven optimal trajectories corresponding to different start and goal nodes. Note that only GPPN requires a complete optimal policy which gives an optimal action at every graph node. This setting makes GPPN have a little advantage. We train and test on the same graph with sizes 12×12, 16×16, 20×20. The number of generated graphs for training is chosen from small to large with values $\{200, 1000, 5000\}$. The size of testing data is fixed to 1000 graphs of corresponding size.

Table 1: 2D Mazes: Test performance with varying training data size $\{200, 1000, 5000\}$ and different graph sizes. GrMPN-GN and GrMPN-GN use a smaller training size of 200.

	12×12		16×16		20×20	
	Acc	Succ	Acc	Succ	Acc	Succ
VIN-200	81.0	86.9	74.5	82.1	66.3	74.5
VIN-1000	85.0	87.6	78.1	83.8	79.0	84.0
VIN-5000	93.8	95.0	90.9	93.5	90.2	91.6
GPPN-200	80.1	87.6	75.6	83.4	69.7	85.0
GPPN-1000	85.8	90.1	85.0	89.5	84.1	90.9
GPPN-5000	93.4	95.7	91.5	94.5	90.0	94.4
GVIN-200	73.6	77.7	68.2	74.3	77.0	80.3
GVIN-1000	89.8	91.0	87.3	89.2	88.7	90.3
GVIN-5000	92.8	93.9	93.7	94.2	90.1	91.4
GrMPN-GAT-200	94.90	94.92	95.72	95.74	95.67	95.68
GrMPN-GAT-1000	94.85	94.88	95.71	95.72	95.67	95.68
GrMPN-GAT-5000	94.96	94.99	95.73	95.75	95.67	95.68
GrMPN-GN-200	93.17	93.50	94.48	94.77	95.35	95.42
GrMPN-GN-1000	95.78	95.87	95.88	95.89	**96.07**	**96.12**
GrMPN-GN-5000	**96.34**	**96.46**	**96.34**	**96.39**	95.83	95.84

The results shown in Table 1 tells that GrMPN-GAT and GrMPN-GN not only outperform other baselines but also more data-efficient. GPPN has a slightly better performance with a large amount of data. We note that GPPN must assume the data consists of optimal policies instead of a small set of optimal demonstration trajectories as in VIN, GVIN and ours. GVIN is a graph-based VIN but it relies on specially designed convolution kernels that are based on the choice of discretised directions. Therefore GVIN is not able to perform as well as GrMPN-GAT and GrMPN-GN which are based on principled graph networks and known to be invariant to graph isomorphism. GrMPN-GAT and GrMPN-GN show significant better in terms of data-efficiency on large domains 20×20. On these large domains, learning algorithms often require a large amount of data. However GrMPN-GAT and GrMPN-GN can still learn well with a limited amount of data. This shows how important invariance to graph isormophism is for learning on graphs. Performance of GrMPN-GAT on bigger domain is better than small domains, because the amount of nodes involved in training is bigger in large graphs. We show more ablation results in Appendix.

4.2 DOMAIN II: IRREGULAR GRAPHS

Table 2: Irregular Dense Graphs: Test performance with varying number of nodes used in training $\{10, 100\}$.

	Dense (100 nodes)			Dense (150 nodes)		
	Acc	Succ	Diff	Acc	Succ	Diff
GVIN-10	58.3	**99.9**	0.046	53.4	**99.8**	0.042
GrMPN-GAT-10	**63.1**	99.2	**0.030**	**58.4**	99.5	**0.028**
GrMPN-GN-10	55.8	95.0	0.057	50.4	94.4	0.062
GVIN-100	56.6	97.3	0.064	52.9	**99.2**	0.059
GrMPN-GAT-100	**62.7**	**97.7**	**0.032**	**58.6**	98.8	**0.029**
GrMPN-GN-100	61.5	97.7	0.038	56.6	98.4	0.037

This experiment uses the same script used by GVIN, which is based on Networkx (Hagberg et al., 2008), to create synthetic graphs that are with random coordinates from box $[0, 1]^2$ in 2D space. We vary the parameters of the generation program to create three types of irregular graphs: Dense, Sparse, and Tree-like. For Tree-like graphs, we use use the Networkx's function, geographical_threshold_graph by setting the connectedness probability between nodes to a small value. We create Tree-like graphs which are not considered in GVIN, because there are two main challenges on these graphs. *First*, with the same number of nodes and amount of generated graphs, tree-like graphs would have much fewer nodes for training. *Second*, Tree-like graphs result in a major issue which are ideal to evaluate generalization for long-range planning which requires propagation of value functions across nodes in graphs well. We generate 10000 graphs, with varying number of nodes $\{10, 100\}$. The label for each graph is an optimal policy, i.e. an optimal action at every graph node. Training is 6/7 of the generated data, while testing is 1/7.

The comparing results are described in Tables 2 (on Dense), 3 (on Sparse), and 7 (on Tree-like). Testing is performed on irregular graphs of different size: 100 and 150 nodes on Dense, 100 nodes on Sparse. The results show that GrMPN methods perform comparably with GVIN on Dense graphs in terms of Success rate, but slightly better in terms of Accuracy and Distance difference. On Sparse graphs, GrMPN-GAT and GrMPN-GN based on the principled of message passing are able to have fast updates across nodes.

4.3 Domain III: Motion Planning

Sampling-based methods such as probabilistic roadmaps (PRM) (LaValle, 2006) have been shown to be every efficient in practice. PRM is one of the most widely used techniques in robotic motion planning, especially for applications in navigation. In such an application, a motion planning algorithm must find an optimal path that must satisfy i) the environment's geometry constraints, i.e. collision-free path, and ii) the robot system constraint, e.g. differential constraints. PRM is multiple-query methods that are very useful in highly structured environments such as large buildings.

Table 3: Irregular Sparse Graphs: Test performance with varying number of nodes used in training $\{10, 100\}$.

	Sparse (100 nodes)		
	Acc	Succ	Diff
GVIN-10	57.9	80.5	0.053
GrMPN-GAT-10	**61.7**	**91.2**	**0.048**
GrMPN-GN-10	59.0	85.1	0.052
GVIN-100	60.3	85.7	0.053
GrMPN-GAT-100	**74.5**	98.1	**0.027**
GrMPN-GN-100	73.9	**98.3**	**0.027**

We evaluate GrMPN on two motion planning problems: 2D navigation with a holonomic mobile robot and manipulation with a simulated 7-DoF Baxter robot arm. We aim to improve PRM by bringing it closer to an online-planning method through transfer planning. In this section, we show that GrMPN would outperform GVIN under such tasks of a complex geometric structure.

Setting: Note that the input of the simulated 7-DoF Baxter robot forms a kinematic tree. Therefore we propose to use three alternative encoding layers f_v to comppute node features: GN, GCN or MLP. For the 2D navigation with a mobile robot, we only use a MLP to encode robot locations. In addition, we use a simple MLP layer of 16 nodes to encode the one-hot goal value. Then, the outputs from these two encoders are concatenated and used as node features. This architecture renders our implementation a hierarchical graph-based planning method.

Data generation: For the mobile robot navigation, we use a standard PRM algorithm to construct 2000 roadmaps for training, each with 200 robot configurations. A graph node is represented as a robot configuration which is a generalized coordinate (x, y) where x, y are 2D translation coordinates. We generate two different test sets consisting of 1000 roadmaps: 200 and 500 configurations. Each generation time uses a different setting of environment obstacles. For each generated graph, we use the Dijkstra algorithm to provide one optimal trajectory corresponding to one random pair of start and goal states.

For the simulated 7-DoF Baxter robot arm, we use the same scrift as in Qureshi et al. (2018) to generate different environment settings (with different obstacles), and set different start and goal configurations. For each environment setting, the roadmap and the optimal trajectories from 20 randomly selected start nodes to the goal are then found by using PRM*(Karaman & Frazzoli, 2011) and the Dijkstra algorithm provided by the OMPL library (Şucan et al., 2012). In total, we generate 280 roadmaps, each with 300 configurations. The test set contains 80 roadmaps, each with about 1200 configurations.

Analysis: Table 4 show test performance results on 2D navigation with a mobile robot. It shows that GrMPN methods not only outperform GVIN but also possess a great generalization ability.We additionally evaluate the generalization ability of GrMPN methods by using the trained model using Tree-like data as described in Irregular Graphs section to test on the created roadmap test set. The distance difference (Diff) computes the cost difference between the predicted path and the optimal path planned by Dijkstra. We skip reporting on GVIN on large testing graphs (500 configuration nodes) due to its degraded performance. The trained model using Tree-like graph data could also generalize well on unseen graphs generated by PRM on different environments. In addition, they can generalize to much bigger graphs (with 500 nodes). This suggests GrMPN is able to do *zero-shot learning to plan*.

Table 4: 2D Navigation motion planning problems: Test performance with varying number of configurations $\{200, 500\}$. Note that algorithms with suffix -T denote trained models using tree-like graph data.

	200 configurations			500 configurations		
	Acc	Succ	Diff	Acc	Succ	Diff
GVIN-T	62.2	58.6	0.210	-	-	-
GrMPN-GAT-T	**82.2**	**83.8**	**0.172**	**61.3**	**72.5**	**0.523**
GrMPN-GN-T	80.0	79.6	0.181	57.1	59.3	0.650
GVIN	53.9	40.6	1.3260	-	-	-
GrMPN-GAT	**82.2**	**84.4**	**0.164**	**62.3**	**73.1**	**0.504**
GrMPN-GN	**82.7**	81.2	0.176	61.0	68.4	0.575

Table 5 show the test performance results on the motion planning task with a simulated 7-DoF Baxter arm. We skip reports on GVIN due to its poor performance and scalability to learning on large graphs. The results show that GrMPN methods are able to do motion planning on this complex geometric domain with high accuracy and success rates.

5 Conclusion

In this paper we have proposed a general graph-based motion planning network, GrMPN. The proposed framework leverages the idea of graph neural networks to handle planning with graphs. The main idea is to integrate Graph processing modules into a differentiable planning network with the aim to capture graph isomorphism in order to achieve i) generalization for transfer planning to graphs of arbitrary structures and ii) data-efficiency when dealing with complex graphs across task instances. Through various experiments on 2D mazes, irregular graphs and motion planning tasks, we have shown that GrMPN is able to improve data-efficiency significantly and improve planning task performance in comparisons to existing approaches. GrMPN outperform baselines for regular graphs and existing approaches for irregular graphs in terms of data-efficiency and generalization ability. For future researches, there is a promising direction in combining GrMPN with other powerful graph networks in order to further exploit the factored structure in planning problems, e.g. factored MDP planning or planning on high-ordered Markov models.

Table 5: Simulated 7-DoF Baxter arm motion planning: Test performance with different uses of node encoding (MLP-, GCN-, GN-) and graph-based planning (GrMPN-GAT and GrMPN-GN).

	Acc	Succ	Diff
MLP-GrMPN-GAT	**78.5**	89.1	**0.466**
GCN-GrMPN-GAT	70.4	83.4	0.875
GN-GrMPN-GAT	77.4	**90.3**	0.561
MLP-GrMPN-GN	**77.9**	89.0	**0.503**
GCN-GrMPN-GAT	67.7	76.1	1.024
GN-GrMPN-GN	76.6	**89.4**	0.552

References

James Atwood and Don Towsley. Diffusion-convolutional neural networks. In *NIPS*, pp. 1993–2001, 2016.

Aniket Nick Bajpai, Sankalp Garg, et al. Transfer of deep reactive policies for mdp planning. In *NIPS*, pp. 10965–10975, 2018.

Peter W. Battaglia, Razvan Pascanu, Matthew Lai, Danilo Jimenez Rezende, and Koray Kavukcuoglu. Interaction networks for learning about objects, relations and physics. In *NIPS*, pp. 4502–4510, 2016.

Peter W Battaglia, Jessica B Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, et al. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*, 2018.

Dimitri P Bertsekas, Dimitri P Bertsekas, Dimitri P Bertsekas, and Dimitri P Bertsekas. *Dynamic programming and optimal control*, volume 1. Athena scientific Belmont, MA, 1995.

Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. Spectral networks and locally connected networks on graphs. *arXiv preprint arXiv:1312.6203*, 2013.

David K Duvenaud, Dougal Maclaurin, Jorge Iparraguirre, Rafael Bombarell, Timothy Hirzel, Alán Aspuru-Guzik, and Ryan P Adams. Convolutional networks on graphs for learning molecular fingerprints. In *NIPS*, pp. 2224–2232, 2015.

Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. Neural message passing for quantum chemistry. In *ICML*, pp. 1263–1272, 2017.

Saurabh Gupta, James Davidson, Sergey Levine, Rahul Sukthankar, and Jitendra Malik. Cognitive mapping and planning for visual navigation. In *CVPR*, pp. 7272–7281, 2017.

Aric Hagberg, Pieter Swart, and Daniel S Chult. Exploring network structure, dynamics, and function using networkx. Technical report, Los Alamos National Lab.(LANL), Los Alamos, NM (United States), 2008.

Nicolas Heess, Gregory Wayne, David Silver, Timothy P. Lillicrap, Tom Erez, and Yuval Tassa. Learning continuous control policies by stochastic value gradients. In *NIPS*, pp. 2944–2952, 2015.

Mikael Henaff, Joan Bruna, and Yann LeCun. Deep convolutional networks on graph-structured data. *arXiv preprint arXiv:1506.05163*, 2015.

Sertac Karaman and Emilio Frazzoli. Sampling-based algorithms for optimal motion planning. *IJRR*, 30(7):846–894, 2011.

Péter Karkus, David Hsu, and Wee Sun Lee. Qmdp-net: Deep learning for planning under partial observability. In *NIPS*, pp. 4694–4704, 2017.

Steven Kearnes, Kevin McCloskey, Marc Berndl, Vijay Pande, and Patrick Riley. Molecular graph convolutions: moving beyond fingerprints. *Journal of computer-aided molecular design*, 30(8): 595–608, 2016.

Arbaaz Khan, Clark Zhang, Nikolay Atanasov, Konstantinos Karydis, Vijay Kumar, and Daniel D. Lee. Memory augmented control networks. In *ICLR*, 2018.

Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *ICLR*, 2017.

Jens Kober, J Andrew Bagnell, and Jan Peters. Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research*, 32(11):1238–1274, 2013.

Thanard Kurutach, Ignasi Clavera, Yan Duan, Aviv Tamar, and Pieter Abbeel. Model-ensemble trust-region policy optimization. In *ICLR*, 2018.

Steven M LaValle. *Planning algorithms*. Cambridge university press, 2006.

Gilwoo Lee, Brian Hou, Aditya Mandalika, Jeongseok Lee, and Siddhartha S Srinivasa. Bayesian policy optimization for model uncertainty. *arXiv preprint arXiv:1810.01014*, 2018a.

Lisa Lee, Emilio Parisotto, Devendra Singh Chaplot, Eric Xing, and Ruslan Salakhutdinov. Gated path planning networks. *arXiv preprint arXiv:1806.06408*, 2018b.

Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deep visuomotor policies. *J. Mach. Learn. Res.*, 17:39:1–39:40, 2016.

Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard S. Zemel. Gated graph sequence neural networks. In *ICLR*, 2016.

Tengfei Ma, Patrick Ferber, Siyu Huo, Jie Chen, and Michael Katz. Adaptive planner scheduling with graph neural networks. *arXiv preprint arXiv:1811.00210*, 2018.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.

Mathias Niepert, Mohamed Ahmed, and Konstantin Kutzkov. Learning convolutional neural networks for graphs. In *ICML*, pp. 2014–2023, 2016.

Sufeng Niu, Siheng Chen, Hanyu Guo, Colin Targonski, Melissa C. Smith, and Jelena Kovacevic. Generalized value iteration networks: Life beyond lattices. In *AAAI*, pp. 6246–6253. AAAI Press, 2018.

Ahmed H Qureshi, Mayur J Bency, and Michael C Yip. Motion planning networks. *arXiv preprint arXiv:1806.05767*, 2018.

Alvaro Sanchez-Gonzalez, Nicolas Heess, Jost Tobias Springenberg, Josh Merel, Martin Riedmiller, Raia Hadsell, and Peter Battaglia. Graph networks as learnable physics engines for inference and control. *arXiv preprint arXiv:1806.01242*, 2018.

Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2008.

Kristof T Schütt, Farhad Arbabzadah, Stefan Chmiela, Klaus R Müller, and Alexandre Tkatchenko. Quantum-chemical insights from deep tensor neural networks. *Nature communications*, 8:13890, 2017.

Marwin Segler, Mike Preuß, and Mark P Waller. Towards" alphachem": Chemical synthesis planning with tree search and deep neural network policies. *arXiv preprint arXiv:1702.00020*, 2017.

David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Vedavyas Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy P. Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

David Silver, Hado van Hasselt, Matteo Hessel, Tom Schaul, Arthur Guez, Tim Harley, Gabriel Dulac-Arnold, David P. Reichert, Neil C. Rabinowitz, André Barreto, and Thomas Degris. The predictron: End-to-end learning and planning. In *ICML*, pp. 3191–3199, 2017.

Aravind Srinivas, Allan Jabri, Pieter Abbeel, Sergey Levine, and Chelsea Finn. Universal planning networks: Learning generalizable representations for visuomotor control. In *ICML*, pp. 4739–4748, 2018.

Ioan A. Şucan, Mark Moll, and Lydia E. Kavraki. The Open Motion Planning Library. *IEEE Robotics & Automation Magazine*, 19(4):72–82, December 2012. doi: 10.1109/MRA.2012.2205651. `http://ompl.kavrakilab.org`.

Richard S Sutton, Andrew G Barto, et al. *Introduction to reinforcement learning*, volume 2. MIT press Cambridge, 1998.

Aviv Tamar, Sergey Levine, Pieter Abbeel, Yi Wu, and Garrett Thomas. Value iteration networks. In *NIPS*, pp. 2146–2154, 2016.

Sam Toyer, Felipe Trevizan, Sylvie Thiébaux, and Lexing Xie. Action schema networks: Generalised policies with deep learning. In *AAAI*, 2018.

Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. In *ICLR*, 2018.

Boris Weisfeiler and Andrei A Lehman. A reduction of a graph to a canonical form and an algebra arising during this reduction. *Nauchno-Technicheskaya Informatsia*, 2(9):12–16, 1968.

Shuai Zhang, Lina Yao, Aixin Sun, and Yi Tay. Deep learning based recommender system: A survey and new perspectives. *ACM Computing Surveys (CSUR)*, 52(1):5, 2019.

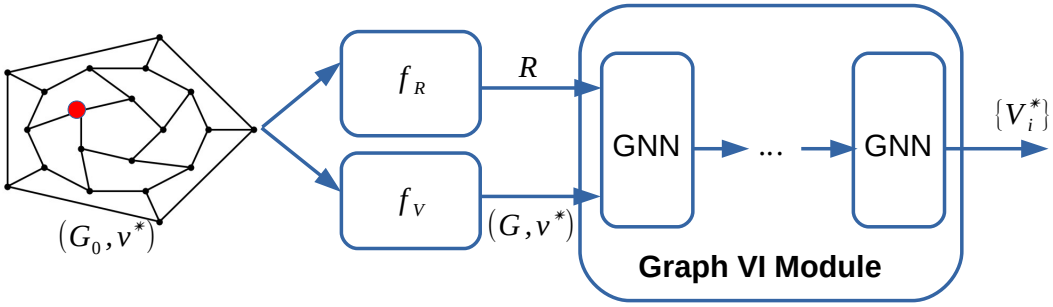

Figure 1: GrMPN based on graph neural networks.

Table 6: Information on irregular graphs.

	# Edges	%Sparsity	Regularity
Sparse (100 nodes)	285	0.057	1.63
Dense (100 nodes)	1097	0.219	3.37
Dense (150 nodes)	2493	0.221	3.77

A Appendix

A.1 Graph-based Motion Planning Networks

Our proposed graph-based motion planning network is inspired by VIN and GVIN, and depicted in Fig. 1.

A.2 Domain I: 2D Mazes

The results in Figs. 2 and 3 show ablation of GrMPN-GAT and GrMPN-GN on different number of graph processing steps. The figure show the value function maps after training. The results show how value functions on a test graph (after training) are computed depending on the value k of processing steps. More nodes would be updated with a larger number of processing step which corresponding to more batches of value iterations updates. This ablation also shows that GrMPN-GAT is able to generalize across nodes better than GrMPN-GN. This generalization ability would significantly help with long-range planning problems.

A.3 Domain II: Irregular Graphs

The information on generated graphs is summarized in Table 6.

As seen in Fig. 4, GVIN is not able to spread the update to nodes that are far from the goal node. This figure also shows that GrMPN-GAT has a slightly better generalization ability across nodes than GrMPN-GN. The value functions of GrMPN-GN have more un-updated nodes (see the color of nodes that are far from the goal node as labeled in black) than that of GrMPN-GAT. This explains why GrMPN-GAT performs slightly better than GrMPN-GN.

The results in Table 7 tell that VIN is not able to cope with very sparse graphs and long-range planning. This shows GVIN has a weak generalization ability.

Table 7: Irregular Tree-like Graphs: Test performance on graphs of 150 nodes. Training uses tre-like graphs of 50 nodes.

	Tree-like (150 nodes)		
	Acc	Succ	Diff
GVIN	66.5	44.67	1.6263
GrMPN-GAT	**88.2**	**98.25**	**0.0270**
GrMPN-GN	87.1	94.8	0.032

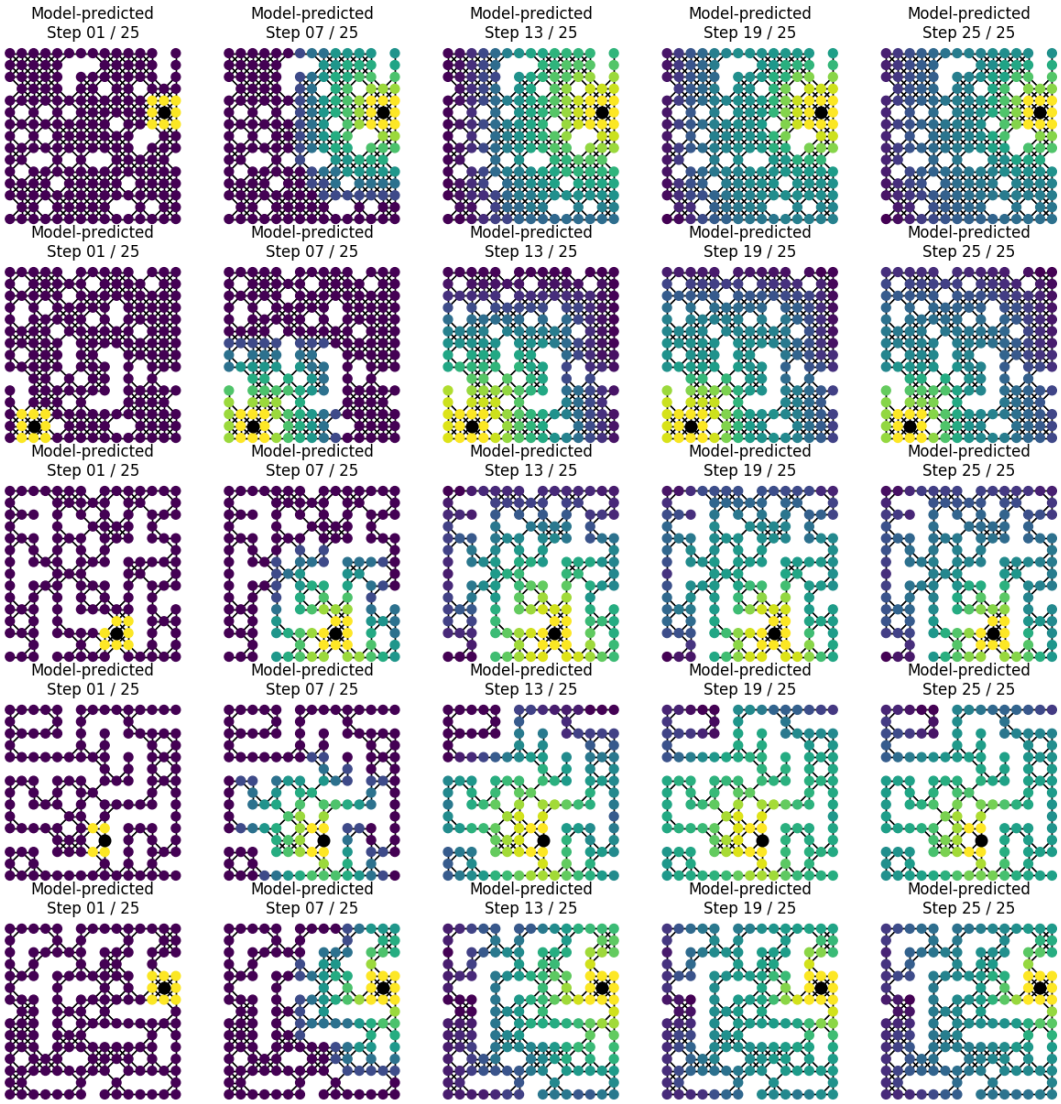

Figure 2: 2D mazes: Value functions of GrMPN-GAT on 16×16 domains w.r.t the different number of processing step k. The left-most figure shows an initial map with a goal state.

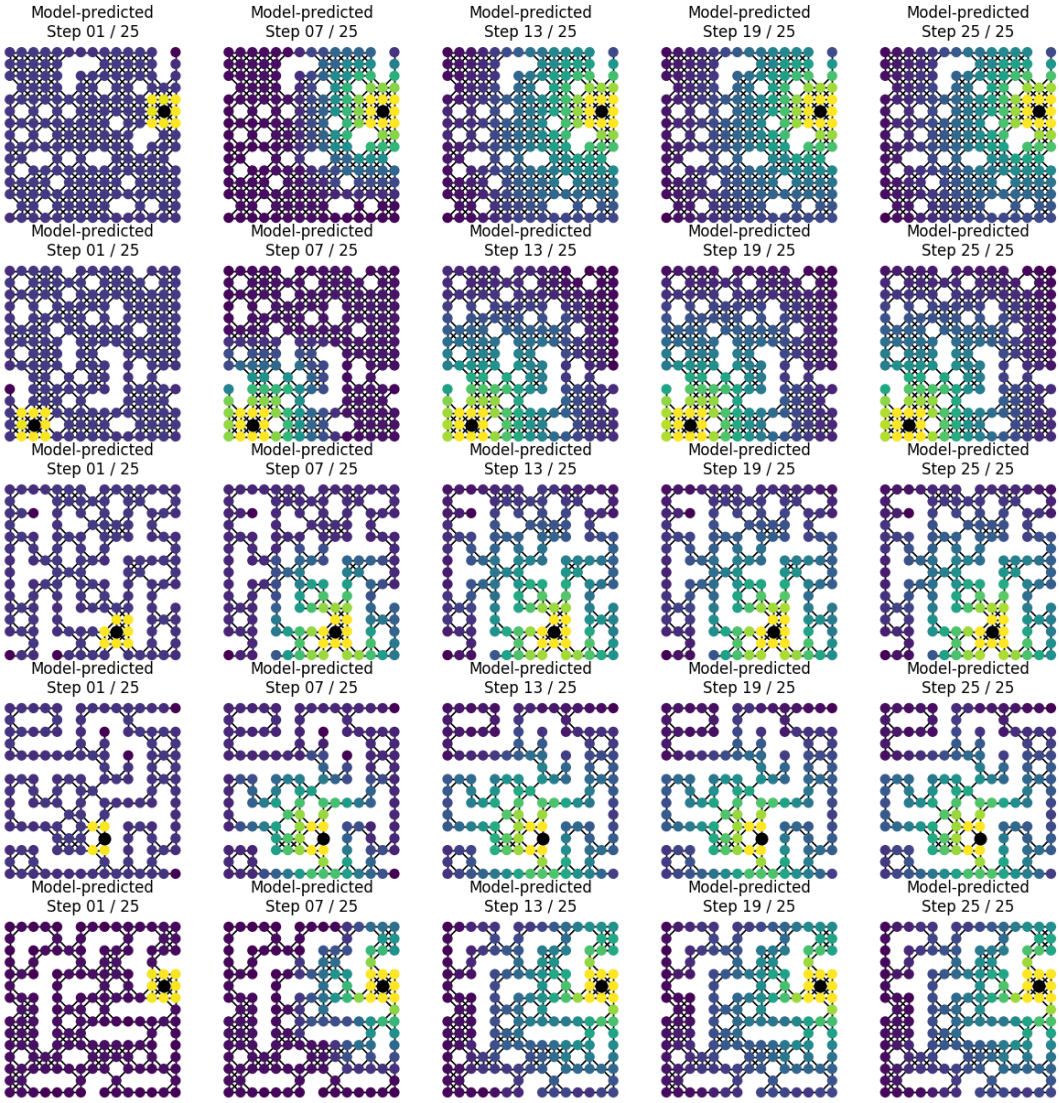

Figure 3: 2D mazes: Value functions of GrMPN-GN on 16×16 domains w.r.t the different number of processing step k. The left-most figure shows an initial map with a goal state.

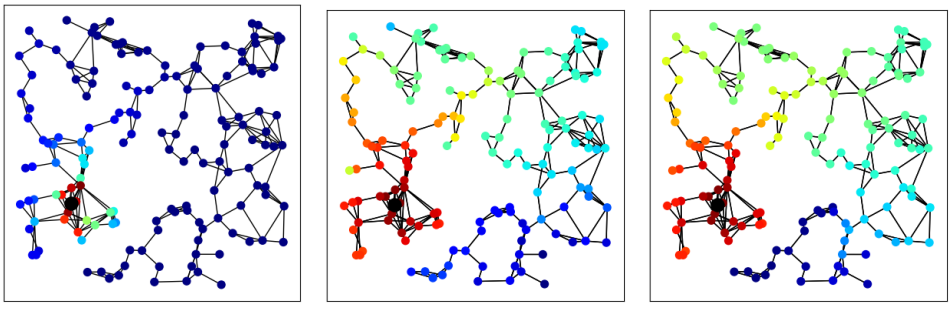

Figure 4: Value functions on tree-like graphs: left) GVIN, middle) GrPN-GN; right) GrMPN-GAT.

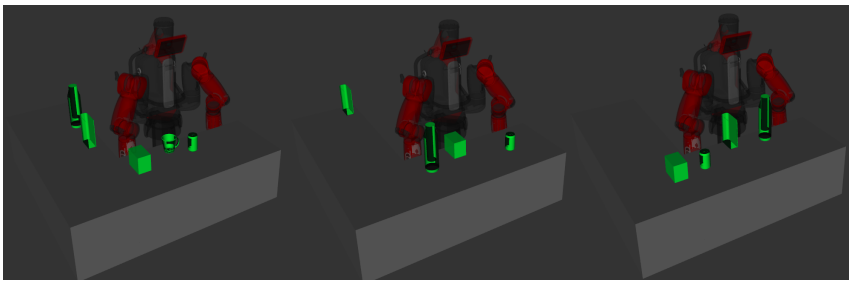

Figure 5: An environment instance generated for the simulated 7-DoF Baxter arm.

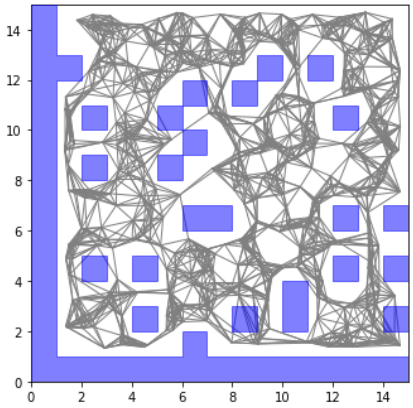

Figure 6: An environment instance with a sampled graph (the roadmap) using PRM.

A.4 Domain III: Motion Planning

Data generation: We use a standard PRM algorithm to construct 2000 graph maps for training, each with 200 robot configurations. A graph node is represented as a robot configuration which is a generalized coordinate (x, y) where x, y are 2D translation coordinates. We generate two different test sets consisting of 1000 roadmaps: 200 and 500 configurations. Each generation time uses a different setting of environment obstacles. For each generated graph, we use the Dijkstra algorithm to provide one optimal trajectory corresponding to one random pair of start and goal states, for an example in Fig. 6.

For the simulated 7-DoF Baxter robot arm, we use the same scrift as in Qureshi et al. (2018) to generate different environment settings (with different obstacles), and set different start and goal configurations. An example of a generated environment is depicted in Fig. 5.

Analysis: Further ablation results in Figures 7 and 8 show that GVIN is not able to update value functions of nodes far from the goal, while GrMPN-GAT and GrMPN-GN can generalize value updates well to such nodes. The color map in Figure 8 also suggests that GrMPN-GAT slightly has

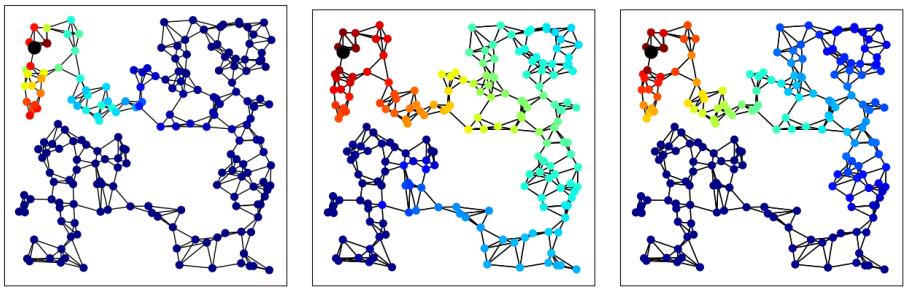

Figure 7: Value functions on PRM graphs of 200 configurations: left) GVIN, middle) GrPN-GN; right) GrMPN-GAT.

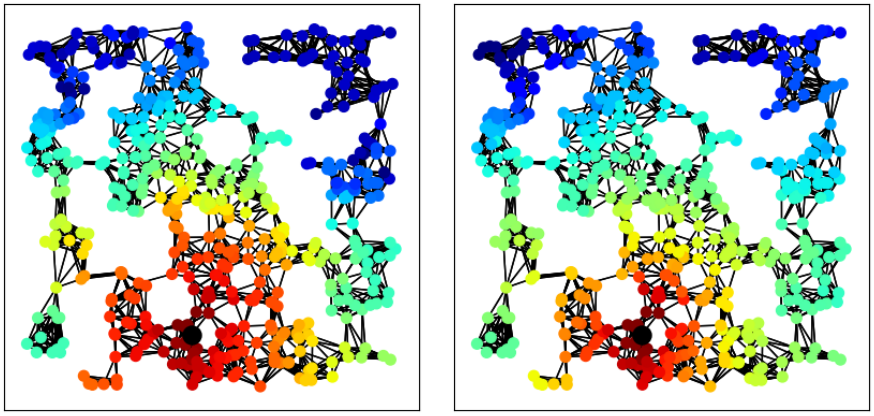

Figure 8: Value functions on PRM graphs of 500 configurations: left) GrPN-GN; right) GrMPN-GAT.

wider value propagation, which means better generalization for long-range planning and across task graphs.