

---

# Neural Online Graph Exploration

---

Ioannis Chiotellis

Technical University of Munich  
john.chiotellis@tum.de

Daniel Cremers

Technical University of Munich  
cremers@tum.de

## Abstract

Can we *learn* how to explore unknown spaces efficiently? To answer this question, we study the problem of Online Graph Exploration, the online version of the Traveling Salesperson Problem. We reformulate graph exploration as a reinforcement learning problem and apply Direct Future Prediction (Dosovitskiy and Koltun, 2017) to solve it. As the graph is discovered online, the corresponding Markov Decision Process entails a dynamic state space, namely the observable graph and a dynamic action space, namely the nodes forming the graph’s frontier. To the best of our knowledge, this is the first attempt to solve online graph exploration in a data-driven way. We conduct experiments on six data sets of procedurally generated graphs and three real city road networks. We demonstrate that our agent can learn strategies superior to many well known graph traversal algorithms, confirming that exploration can be learned.

## 1 INTRODUCTION

In online graph exploration, an agent is immersed in a completely unknown environment. Located at a node of an unknown graph, they can only see the node’s immediate neighbors. The agent moves and discovers the graph as they go. Whenever they visit a new node, all incident edges are revealed, along with their weights and their end nodes. To visit a new node, the agent has to traverse a path of known edges in the discovered graph. For each of these edges, the agent pays their weight as a cost. The goal of the agent is to visit all nodes in the graph, while paying the minimum cost.

By removing any particular geometric constraints, a large number of problems can be reduced to online graph exploration, as it basically is *search* with partial information in a discrete space. We revisit online graph exploration for undirected unweighted connected graphs. This task can be directly associated with many major problems in robotics such as planning, navigation, tracking and mapping (Yamauchi, 1997). All these subfields have been thoroughly investigated and a large number of algorithms have been devised, both classical and recently also learning-based. While path planning and navigation algorithms consider the question “How can I get from A to B the fastest?”, exploration algorithms consider a more abstract question: “Where should I go beginning from A in order to discover the world the fastest?”. In other words, while path planning studies *how* to reach a given destination, exploration is concerned with *which* destination should be reached next, a problem more akin to dynamic planning. We argue that exploration is a fundamental sequential decision making problem and it is therefore worth investigating if algorithms can *learn* which destinations are worth reaching and when.

The best known exploration strategy remains a simple greedy method - the nearest neighbor algorithm (NN). However, as NN selects the nearest (in terms of shortest path distance) unexplored node, its decisions are optimal only when considering a horizon of a single decision step. Therefore, a reasonable question is whether there are algorithms that can consider a longer horizon and thus minimize the *cumulative* path length, which is the true objective of exploration.

We present a learning algorithm that does exactly this. Our contributions can be summarized as follows:

- reformulate online graph exploration as a reinforcement learning problem,
- propose a neural network that can handle the associated dynamic state and action space,
- show experimentally that the proposed approach solves graph exploration as fast or faster than many classical graph exploration algorithms.

## 2 RELATED WORK

The problem of graph exploration has been studied by the graph theory community for decades. A large number of works has been conducted, studying the problem for specific classes of graphs (Miyazaki et al., 2009; Higashikawa et al., 2014), with multiple collaborative agents (Dereniowski et al., 2013) or for variations of the problem with additional information (Dobrev et al., 2012) or energy constraints (Duncan et al., 2006; Das et al., 2015). Successful exploration algorithms are often sophisticated variations of depth first search (DFS), where the algorithm has to decide when to diverge from DFS (Kalyanasundaram and Pruhs, 1994; Megow et al., 2012). A similar problem has been studied by Deng and Papadimitriou (1999) for unweighted *directed* graphs, where the agent has to traverse all edges instead of visiting all nodes. In this setting, the offline equivalent problem is known as the Chinese Postman Problem (CPP) (Guan, 1962) which is solvable in polynomial time. In contrast, the offline equivalent of Online Graph Exploration is the Traveling Salesperson Problem, an NP-hard problem.

Besides the large volume of research, for general graphs, the best known exploration algorithm remains a simple greedy method - the nearest neighbor algorithm (NN). The trajectories followed by NN are provably at most  $O(\log n)^1$  longer than the optimal ones (Rosenkrantz et al., 1977).

However, to the best of our knowledge, it has not been attempted to solve graph exploration in a data-driven way. In this work, we investigate whether, given a training set of graphs, a learning algorithm is able to find good exploration strategies that are competitive or even superior to traditional methods. Recently, there has been growing interest in applying learning to combinatorial optimization problems. In the work of Vinyals et al. (2015), three problems were studied and solved by learning to “point” to elements of a set with a neural network. Among these problems was the Traveling Salesperson Problem, which can be thought of as the offline equivalent to graph exploration. Vinyals et al. (2015) proposed a recurrent neural network (RNN) architecture, called Pointer Network, based on the attention mechanism introduced by Bahdanau et al. (2015). However, using an RNN might introduce a bias due to the ordering of the elements in the input sequence. To alleviate this bias, several works have studied neural architectures that can preserve the permutation-invariance property of sets (Edwards and Storkey, 2017; Zaheer et al., 2017; Lee et al., 2019). Moreover, in recent years, Graph Neural Networks (GNNs) (Battaglia et al., 2018) have emerged.

<sup>1</sup>where  $n$  is the number of nodes

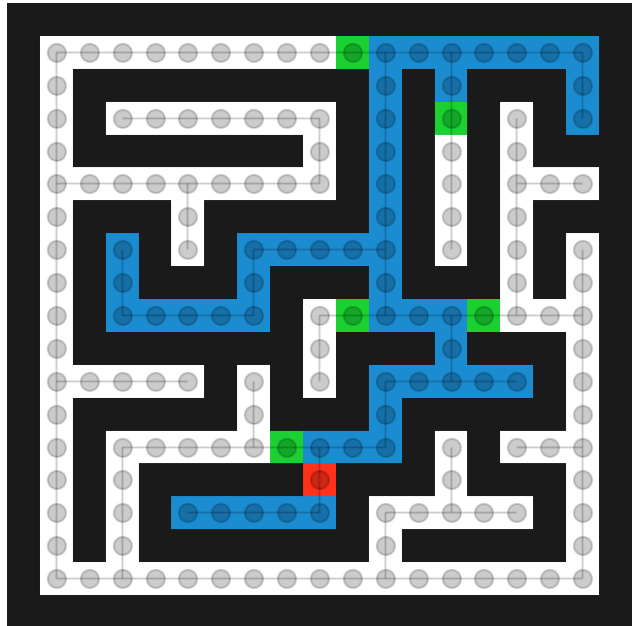


Figure 1: A graph exploration agent (**red**) keeps track of the nodes already visited (**blue**) and the *frontier* nodes that could be visited next (**green**). The frontier nodes form the boundary between known and unknown (**white**) space. The goal is to discover and visit all nodes as fast as possible.

These neural networks consider not just sets of nodes but also their pairwise connections or relationships as inputs and can learn how to solve problems such as node classification (Kipf and Welling, 2017) and link prediction (Zhang and Chen, 2018). Further, methods such as DeepWalk (Perozzi et al., 2014) and node2vec (Grover and Leskovec, 2016) have been used to learn node embeddings in an unsupervised way, borrowing ideas from natural language processing (Mikolov et al., 2013). These methods make an implicit assumption that nodes that co-appear in a random walk on the graph are more similar than nodes that don’t. Nevertheless, it remains a challenge to learn node embeddings for dynamic graphs that are changing while an algorithm makes decisions about the graph’s structure.

Close to our work is the work of Dai et al. (2019) which studies exploration on environments with graph-structured state-spaces, such as software testing. In contrast to (Dai et al., 2019), we study the original problem as defined by the graph theory community which prohibits the revisiting of past nodes. This allows us to study the problem in isolation, without the interference of other tasks such as visual feature learning and the perceptual aliasing problem.

### 3 FORMULATION

#### 3.1 Graph Exploration Overview

An agent explores a connected unweighted graph  $G = (V, E)$ . At time step  $t = 0$ , they start at an arbitrary node  $v_0 \in V$  and they can only observe an initial map  $G_0 = (V_0, E_0)$  comprised of the neighbors and incident edges of  $v_0$ . We assume the agent has a memory where it can store and integrate observations. Therefore, at any time step  $t$ , the agent observes a subgraph  $G_t = (V_t, E_t)$  with a subset of visited nodes  $C_t$  and a subset of frontier nodes  $F_t$  that are to be explored. Being at node  $v_t \in C_t$ , the agent has to choose a node  $v_{t+1} \in F_t$  to visit next. Once a decision is made, the agent follows a path from  $v_t$  to  $v_{t+1}$  of length  $l_{t+1} = d_{G_t}(v_t, v_{t+1})$ . Note that this is a shortest path in  $G_t$  but not necessarily in  $G^2$ . The new node  $v_{t+1}$  gets removed from the frontier and becomes part of the set of visited nodes:  $C_{t+1} \leftarrow C_t \cup \{v_{t+1}\}$ . Finally, the agent observes the neighbors  $N(v_{t+1})$  of  $v_{t+1}$  and the frontier gets expanded by the subset of neighbors that have not been observed in the past:

$$F_{t+1} \leftarrow F_t \cup N(v_{t+1}) \setminus C_{t+1}. \quad (1)$$

The goal is to visit the nodes in such an order that the total path length is minimized. Notice that we use a different timescale than commonly used *e.g.* in navigation problems. In a single time step, the exploration agent can traverse a path of arbitrary length in the known graph  $G_t$ . The differences between exploration algorithms lie in the way they choose the node  $v \in F_t$  to visit next. For instance, DFS considers the order of entry in the frontier and chooses the most recently entered node. The nearest neighbor algorithm (NN) chooses the node closest to the current node  $v_t \in C_t$ :

$$v_{t+1}^{NN} = \arg \min_{v \in F_t} d(v_t, v) \quad (2)$$

However, since the NN selection rule is greedy, it might be suboptimal. Namely, an algorithm  $A$  could exist that takes into account the *expected* future path lengths and could therefore make better decisions:

$$v_{t+1}^A = \arg \min_{v \in F_t} \mathbb{E}[d(v_t, v) + \sum_{i=t+1}^{\infty} d(v_i, v_{i+1})]. \quad (3)$$

This formulation is reminiscent of reinforcement learning (RL). In RL an agent in a state  $s_t$  and following a policy  $\pi$ , chooses action  $a_t = \pi(s_t)$  and receives an immediate reward  $r_{t+1}$ . The true objective of the agent

is to maximize the cumulative reward:

$$a_t = \arg \max_{a \in \mathcal{A}} \mathbb{E}[r_{t+1} + \sum_{i=t+1}^{\infty} \gamma^{i-t} r_{i+1} | \pi, a], \quad (4)$$

where  $\gamma \in [0, 1]$  is a discount factor that weighs distant future rewards less than imminent rewards. The expectation of cumulative rewards is also known as the action value  $Q^\pi(s_t, a_t)$ . Notice that the NN algorithm can be exactly recovered for  $\gamma = 0$ .

#### 3.2 Markov Decision Process

RL problems are formally described as Markov Decision Processes (MDPs). An MDP is defined as a 5-tuple  $(\mathcal{S}, \mathcal{A}, p, r, \gamma)$ , namely a state space  $\mathcal{S}$ , an action space  $\mathcal{A}$ , a state transition probability function  $p : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \mapsto [0, 1]$ , a reward function  $r : \mathcal{S} \times \mathcal{A} \mapsto \mathbb{R}$  and a discount factor  $\gamma \in [0, 1]$ . A partially observable Markov decision process (POMDP) is a generalization of a MDP, where the agent cannot directly observe the state  $s_t \in \mathcal{S}$  but has partial information through observations  $o_t \in \mathcal{O}$ . An agent with a memory component can integrate partial observations to cumulative observations  $\mathbf{x}_t \in \mathcal{X}$ . Notice that the observation space  $\mathcal{X}$  is a subset of the state space  $\mathcal{S}$ . Thus, we refer to the setting of Online Graph Exploration as a memory-augmented POMDP. In Figure 2, we illustrate this setting. In the following, we describe the components of this MDP.

**State Space** Let  $\mathcal{G}$  be the set of all conceivable graphs, and let  $\mathcal{P}_G$  denote the set of all conceivable visit orderings  $P_G$  for a graph  $G \in \mathcal{G}$ . Then the state space  $\mathcal{S}$  is defined as the set of all pairs  $(G, P_G)$  of graphs  $G \in \mathcal{G}$  and associated visit orderings  $P_G \in \mathcal{P}_G$ :

**Observation Space** At each time step, the environment reveals the neighborhood of the visited node. Therefore, the observation space is exactly the subset of graphs that are star graphs.

**Action Space** It is common in RL problems with discrete action spaces, for the agent to have access to a fixed set of actions  $\mathcal{A}$  as in Eq. (4). Instead, in graph exploration, a new unique action set  $\mathcal{A}_t$  is induced from the state at each time step. This action set corresponds to the nodes that have been observed but not visited yet, namely the nodes in the frontier:

$$\mathcal{A}_t = F_t. \quad (5)$$

The general action space can be described by the power set of all nodes:  $\mathcal{A} = 2^V$ . Note that the frontier can be derived from the known graph  $G_t = (V_t, E_t)$  and the path  $P_t$  as  $F_t = V_t \setminus C_t$ , where  $C_t$  denotes the set of visited nodes found in the sequence  $P_t$ .

<sup>2</sup>In the rest of the paper, we omit  $G_t$  and use the shorter notation  $d(v_t, v_{t+1})$  wherever possible.

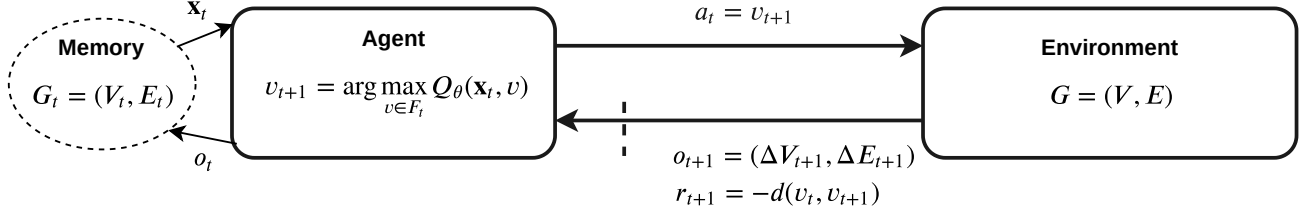


Figure 2: The task of a graph exploring agent is to select the frontier node  $v_{t+1} \in F_t$  to visit next. Once the node is visited, the environment reveals a new set of nodes  $\Delta V_{t+1}$  and edges  $\Delta E_{t+1}$ , expanding the agent’s knowledge to the graph  $G_{t+1} = (V_{t+1}, E_{t+1})$ . The distance traveled by the agent from  $v_t$  to  $v_{t+1}$  is paid as a negative reward  $r_{t+1}$ . The agent acts upon an integrated observation  $\mathbf{x}_{t+1}$  retrieved from its memory. The subset of nodes that have been discovered but are not yet visited are labeled as the frontier  $F_{t+1}$ .

**Reward Function** As defined in section 3.1, the rewards correspond to negative geodesic distances. Therefore, assuming unweighted graphs, all rewards are strictly negative:

$$r(s_t, a_t) = -d(v_t, v_{t+1}) < 0. \quad (6)$$

**State Transition Function** Let  $v_{t+1}$  be the node to visit next. Then, if  $s_t = (G, P_t)$  is the state described by the graph  $G$  and the path  $P_t$ , the new state is described by the same graph  $G$  and the extended path  $P_{t+1} \leftarrow P_t \parallel v_{t+1}$ <sup>3</sup>.

**Memory Update** Upon observing  $\Delta V_{t+1} = N(v_{t+1})$ , namely the neighbors of  $v_{t+1}$ , and  $\Delta E_{t+1} = E(v_{t+1})$ , the edges from  $v_{t+1}$  to  $N(v_{t+1})$ , the agent’s memory is updated as:

$$V_{t+1} \leftarrow V_t \cup \Delta V_{t+1} \quad (7)$$

$$E_{t+1} \leftarrow E_t \cup \Delta E_{t+1} \quad (8)$$

$$P_{t+1} \leftarrow P_t \parallel v_{t+1} \quad (9)$$

$$C_{t+1} \leftarrow C_t \cup \{v_{t+1}\} \quad (10)$$

$$F_{t+1} \leftarrow F_t \cup \Delta V_{t+1} \setminus C_{t+1}. \quad (11)$$

## 4 METHODOLOGY

### 4.1 Predicting the Future Path Lengths

Our premise is that a learning agent can perform better than traditional exploration algorithms, as long as they can predict the future distances to be traveled. Inspired by the framework introduced by Dosovitskiy and Koltun (2017), we use **Direct Future Prediction** (DFP) to learn a predictor of future path lengths. Confirming the authors’ observations, we found that reducing policy learning to a supervised regression problem makes training faster and more stable. In

particular, at time  $t$ , we aim to predict the vector

$$\mathbf{y}_t = (\mathbf{m}_{t+\tau_1} - \mathbf{m}_t, \mathbf{m}_{t+\tau_2} - \mathbf{m}_t, \dots, \mathbf{m}_{t+\tau_M} - \mathbf{m}_t), \quad (12)$$

where  $\mathbf{m}_t$  is a low-dimensional measurement vector augmenting the agent’s high-dimensional observation  $\mathbf{x}_t$ , and  $\{\tau_j\}_{j=1}^M$  are temporal offsets. Following Dosovitskiy and Koltun (2017), we choose exponential offsets  $\tau_j = 2^{j-1}$ . We could directly use a scalar measurement  $L_t = \sum_{i=0}^t l_i$ , namely the cumulative path length up to time  $t$ . However, there are several disadvantages with this choice. For unweighted graphs, we know that any *one-step* path length  $l_t$  lies in the range  $[1, N_{max} - 1]$ , where  $N_{max}$  is the maximum number of nodes we are considering. Thus, directly predicting path lengths would limit our ability to generalize to graphs larger than our training graphs. Second, the distribution of path lengths naturally grows over time together with the observable graph’s diameter. To avoid these problems, instead of minimizing path lengths, we maximize the agent’s exploration rate  $u_t = \frac{|C_t|}{L_t} = \frac{t}{L_t}$  which always lies in the  $[0, 1]$  interval and thus the entries of  $\mathbf{y}$  always lie in  $[-1, 1]$ . At test time, we choose the node to visit next by simply taking the arg max:

$$v_{t+1} = \arg \max_{v \in F_t} \mathbf{g}^\top \mathbf{f}_\theta(\mathbf{x}_t, \mathbf{m}_t, v), \quad (13)$$

where  $\mathbf{f}_\theta$  is our parameterized predictor network,  $\mathbf{x}_t = (V_t, E_t, X_t)$  is the observable graph with node features  $X_t$  and  $\mathbf{g}$  is a goal vector expressing how much we care about different future horizons. Another advantage of using DFP instead of RL is that, given information about the remaining time available, we can directly incorporate it in the goal vector, both at training and at test time without the need of retraining the network. In contrast to Dosovitskiy and Koltun (2017), we don’t use  $\mathbf{g}$  as an input to the network<sup>4</sup>, but only as a weighting of the predictions to obtain a policy.

<sup>3</sup>By  $\parallel$  we denote concatenation of a sequence with a new element.

<sup>4</sup>We found that adding a goal module does not improve and some times even hurts performance.

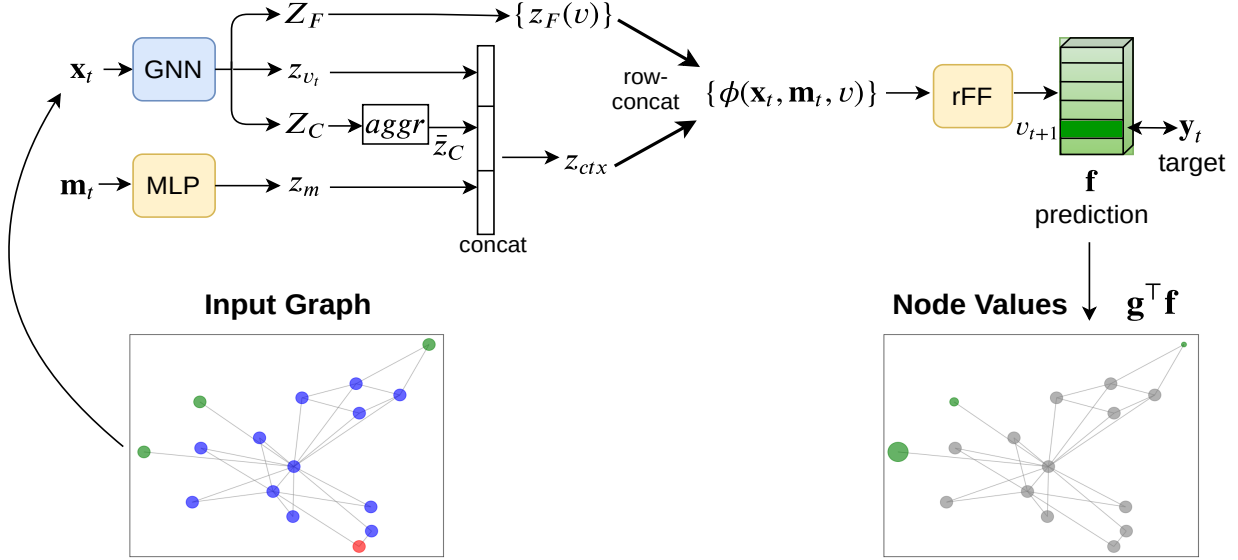


Figure 3: The proposed neural network architecture for online graph exploration.

#### 4.2 Network Architecture

In Figure 3 we show our network architecture. We first obtain node embeddings  $Z_F$ ,  $Z_C$  and  $z_{v_t}$  by passing the observable graph  $\mathbf{x}_t = (V_t, E_t, X_t)$  through a graph neural network (GNN). The embeddings correspond to the node subsets  $F_t$ ,  $C_t$  and the current node  $v_t$ . We use a standard graph convolutional network (GCN) (Kipf and Welling, 2017), but any graph neural network that produces node embeddings can be used. We then aggregate the node embeddings  $Z_C$  of the visited set to obtain a subgraph embedding  $\bar{z}_C$ . Even though more sophisticated pooling methods (e.g. attention (Velickovic et al., 2018)) may be used, we use simple mean pooling.

We pass  $\mathbf{m}_t$  through a multi-layer Perceptron (MLP) to obtain a measurement embedding  $z_m$ . The vectors  $\bar{z}_C$ ,  $z_{v_t}$  and  $z_m$  are concatenated to form a *context* vector  $z_{ctx}$ . Each frontier node embedding  $z_F(v) \in Z_F$  is concatenated with  $z_{ctx}$ , resulting in a set of state-action encodings  $\phi(\mathbf{x}_t, \mathbf{m}_t, v)$ , one for each node  $v \in F_t$ . Finally we pass these encodings through a row-wise feed-forward network (another MLP) to obtain a prediction vector  $\mathbf{f}(\mathbf{x}_t, \mathbf{m}_t, v)$  for each node  $v \in F_t$ . The state-action value  $Q(\mathbf{x}_t, \mathbf{m}_t, v, \mathbf{g})$  of each node in the frontier can be obtained by multiplying the associated prediction vector  $\mathbf{f}(\mathbf{x}_t, \mathbf{m}_t, v)$  with the goal vector  $\mathbf{g}$ .

#### 4.3 Input Features

Solving online graph exploration as a learning problem depends critically on what the learning algorithm “sees” as input. The state  $s_t$  consists of the known

graph  $G_t$  and the path  $P_t$ . Therefore, we have to decide on the nodes and perhaps also edge input features of the graph  $G_t$ . Second, we have to consider a representation of the path  $P_t$ .

We used categorical node features indicating if a node belongs to the visited set  $C_t$ , the frontier set  $F_t$  or if it is the current node  $v_t$ :

$$x(v_i) = [\mathbf{1}(v_i \in C_t), \mathbf{1}(v_i \in F_t), \mathbf{1}(v_i = v_t)]. \quad (14)$$

Note that a categorical node feature space can incorporate many classical graph traversal algorithms (e.g. BFS, DFS and NN) as special cases by simply adding a binary channel that indicates the node that would be selected by the respective algorithm. Furthermore, this representation allows the learning algorithm to potentially learn a *hyper-policy* (Precup et al., 1998) by combining greedy algorithms in novel ways.

In preliminary experiments, we investigated ways to utilize the order of visit of the nodes,  $P_t$ , by using positional encodings (Vaswani et al., 2017) as continuous node features. We found that these features degraded the agent’s performance both when used on their own and when combined with the categorical features.

#### 4.4 Training

In Algorithm 1 we describe our training procedure. In each episode, we randomly sample a graph from the training set and then randomly set one of its nodes as source. This virtually increases the training set size from the number of training graphs  $|\mathcal{G}_{train}|$  to the total number of nodes in all training graphs  $\sum_{G \in \mathcal{G}_{train}} |V_G|$ , where by  $V_G$  we denote the vertices of graph  $G$ .



**Algorithm 1** Training NOGE

- 1: **Input:** network  $\mathbf{f}_\theta$ , training set of graphs  $\mathcal{G}_{train}$ , time limit  $T_{max}$ , goal vector  $\mathbf{g}$ , minibatch size  $B$ .
- 2: **Output:** trained network  $\mathbf{f}_\theta$ .
- 3: Initialize  $\theta$  randomly.
- 4: Initialize an experience replay buffer  $\mathcal{R}$ .
- 5: **while** *training* **do**
- 6:   Sample a graph  $G = (V, E) \sim \mathcal{G}_{train}$ .
- 7:   Sample a source node  $v_0 \sim V$ .
- 8:   Explore  $G$  using  $\mathbf{f}_\theta$ ,  $\mathbf{g}$  and  $\epsilon$ -greedy policy for up to  $T_{max}$  episode steps.
- 9:   Store  $G_T$  and the followed path  $P_T$  in  $\mathcal{R}$ .
- 10:   Sample a minibatch  $\{G_t^{(i)}, P_t^{(i)}\}_{i=1}^B$  from  $\mathcal{R}$ .
- 11:   Reconstruct tuples  $\{\mathbf{x}_t^{(i)}, \mathbf{m}_t^{(i)}, v_{t+1}^{(i)}, \mathbf{y}_t^{(i)}\}_{i=1}^B$ .
- 12:   Train  $\mathbf{f}_\theta$  with the minibatch, using Adam (Kingma and Ba, 2015) to minimize the mean squared error:  

$$L(\theta) = \frac{1}{B} \sum_{i=1}^B \|\mathbf{y}_t^{(i)} - \mathbf{f}_\theta(\mathbf{x}_t^{(i)}, \mathbf{m}_t^{(i)}, v_{t+1}^{(i)})\|^2.$$
- 13: **end while**

## 5 EXPERIMENTS

The complete set of hyperparameters used is reported in Appendix A and our full source code is available at <https://github.com/johnny-c/noge>.

### 5.1 Evaluation Protocol

We evaluate our algorithm - NOGE (Neural Online Graph Exploration) on data sets of generated and real networks. In addition to the basic version of our algorithm, we evaluate NOGE with an extra node feature, indicating the nearest neighbor, as described in section 4.3. We call this variant NOGE-NN. We use three well known graph exploration algorithms as baselines: Breadth First Search (BFS), Depth First Search (DFS) and Nearest Neighbor (NN). We note that these heuristics do not need any training. For completeness, we also report a random exploration baseline (RANDOM). We compare the algorithms in terms of the *exploration rate*  $u_T$ , namely the number of visited nodes over the total path length at the end of episodes:

$$u_T = \frac{|C_T|}{|P_T|} = \frac{T}{\sum_{i=0}^T l_i}. \quad (15)$$

For the test sets we fix a set of source nodes per graph, to compare all methods given the exact same initial conditions. The metrics reported are computed on the test sets after either all nodes have been explored or when a fixed number of  $T_{max} = 500$  exploration steps has been reached. For all experiments we report mean and standard deviation over 5 random seeds.

### 5.2 Procedurally Generated Graphs

We first examine six classes of procedurally generated graphs (Figure 6). We used the *networkx* library (<https://github.com/networkx/networkx>) to generate a diverse (in terms of size and connectivity) set of graphs for each class. In Table 1 we report basic statistics describing the size and connectivity of the graphs. We split each data set in a training (80%) and test set (20%) of graphs. For some of these data sets, an optimal strategy is known. For instance, DFS explores trees optimally by traversing each edge two times - once to explore and once to backtrack. Thus its exploration rate is approximately 0.5. It is worthwhile examining if NOGE can find this optimal strategy.

In Figure 4 we show the test performance of NOGE over 25600 training steps. In Table 3 we report the final performance of the algorithms compared to the baselines. NOGE is able to outperform other methods on grids and the caveman data set and find the optimal strategy on trees. Somewhat surprisingly the NN feature seems to only help on the maze and tree data sets. Note that in ladder and tree, DFS's line is hidden as its performance matches that of NN.

Table 1: Basic statistics of the procedurally generated data sets (tr: training set, te: test set).

Dataset	Size	$ V _{min,max}$	$ E _{min,max}$
barabasi(tr)	400	(100, 199)	(384, 780)
barabasi(te)	100	(100, 199)	(384, 780)
ladder(tr)	80	(200, 398)	(298, 595)
ladder(te)	20	(220, 386)	(328, 577)
tree(tr)	4	(121, 1365)	(120, 1364)
tree(te)	2	(364, 1093)	(363, 1092)
grid(tr)	80	(64, 289)	(112, 544)
grid(te)	20	(72, 240)	(127, 449)
caveman(tr)	120	(60, 316)	(870, 12324)
caveman(te)	30	(70, 304)	(1190, 11400)
maze(tr)	400	(97, 251)	(96, 262)
maze(te)	100	(97, 255)	(96, 276)

Table 2: Basic statistics of the city road network data sets (tr: training set, te: test set).

Dataset	Size	$ V _{min,max}$	$ E _{min,max}$
MUC(tr)	1	(8559, 8559)	(12821, 12821)
MUC(te)	1	(5441, 5441)	(7772, 7772)
OXF(tr)	1	(2197, 2197)	(2561, 2561)
OXF(te)	1	(1185, 1185)	(1430, 1430)
SFO(tr)	1	(5691, 5691)	(9002, 9002)
SFO(te)	1	(3885, 3885)	(6579, 6579)

Table 3: Final exploration rate: Mean and standard deviation on the generated data sets.

Dataset	RANDOM	BFS	DFS	NN	NOGE	NOGE-NN
barabasi	0.3695 (0.0006)	0.4695 (0.0013)	0.5494 (0.0009)	<b>0.8179(0.0014)</b>	0.7214 (0.0663)	0.6970 (0.0497)
caveman	0.5664 (0.0050)	0.9526 (0.0025)	0.9778 (0.0006)	0.9827 (0.0015)	0.9817 (0.0029)	<b>0.9907(0.0031)</b>
grid	0.1461 (0.0037)	0.2264 (0.0039)	0.6272 (0.0041)	0.7670 (0.0028)	<b>0.8861(0.0162)</b>	<b>0.8373(0.0536)</b>
ladder	0.1531 (0.0226)	0.1691 (0.0341)	<b>0.7519(0.0010)</b>	<b>0.7530(0.0009)</b>	0.6046 (0.1208)	<b>0.6729(0.1114)</b>
maze	0.0688 (0.0027)	0.0626 (0.0025)	0.5266 (0.0050)	<b>0.5723(0.0033)</b>	0.4921 (0.0140)	0.5601 (0.0106)
tree	0.1242 (0.0011)	0.3397 (0.0002)	<b>0.5044(0.0002)</b>	<b>0.5044(0.0003)</b>	0.4403 (0.0272)	<b>0.5043(0.0004)</b>

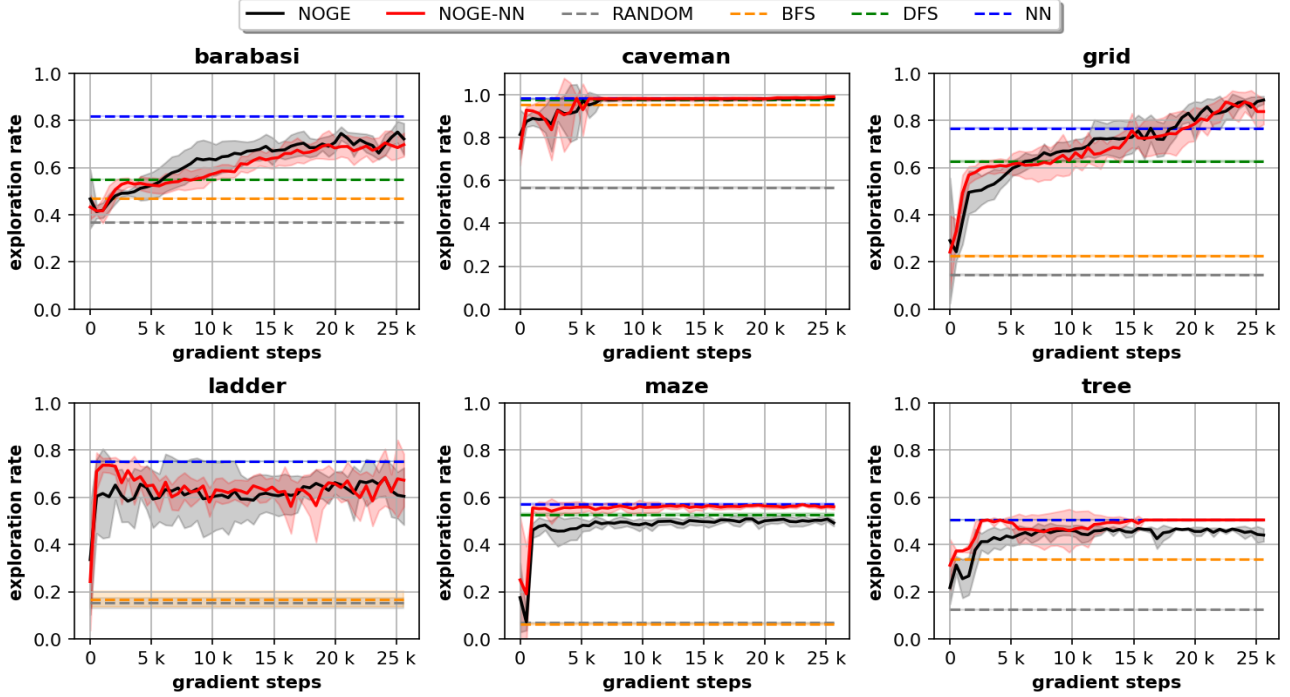


Figure 4: Exploration rate over gradient steps for the six procedurally generated data sets.

### 5.3 City Road Networks

To examine its capabilities, we also evaluate NOGE, on three real road networks. We use openly available drivable road networks from OpenStreetMap (Haklay and Weber, 2008). We explore three cities with diverse networks: Munich (MUC), Oxford (OXF) and San Francisco (SFO), shown in Figure 7.

In Table 2, we show basic statistics for these data sets. We constructed a training set and test set for each city by cutting each graph in two components and removing any edges connecting them. The cut was defined by the diagonal of each city’s 2D bounding box. The larger component (approx. 60% of the nodes) was used for training and the smaller one for testing.

In Figure 5 we show the test performance of NOGE over 40000 training steps and in Table 4 we report the final performance. Nearest Neighbor performs clearly

better in San Francisco and Munich. NOGE-NN is able to match and surpass DFS, as the second best method. In Oxford, NOGE-NN is within a standard deviation from the best performance by NN. In these graphs the NN feature clearly improves performance.

## 6 CONCLUSION

In this work, we presented NOGE, a learning-based algorithm for exploring graphs online. First, we formulated an appropriate memory-augmented Markov Decision Process. Second, we proposed a neural architecture that can handle the growing graph as input and the dynamic frontier as output. Third, we devised a node feature space that can represent greedy methods as *options* (Precup et al., 1998). Finally, we showed experimentally that NOGE is competitive to well known classical graph exploration algorithms in terms of the exploration rate of unseen graphs.

Table 4: Final exploration rate: Mean and standard deviation on the city road networks data sets.

Dataset	RANDOM	BFS	DFS	NN	NOGE	NOGE-NN
MUC	0.0674 (0.0003)	0.1961 (0.0007)	0.7644 (0.0053)	<b>0.8314(0.0091)</b>	0.6458 (0.0441)	0.7814 (0.0386)
OXF	0.0624 (0.0009)	0.1608 (0.0019)	0.6012 (0.0048)	<b>0.6422(0.0037)</b>	0.4695 (0.0136)	<b>0.6328(0.0141)</b>
SFO	0.0726 (0.0015)	0.2007 (0.0033)	0.8252 (0.0073)	<b>0.9017(0.0064)</b>	0.7541 (0.0679)	0.8289 (0.0456)

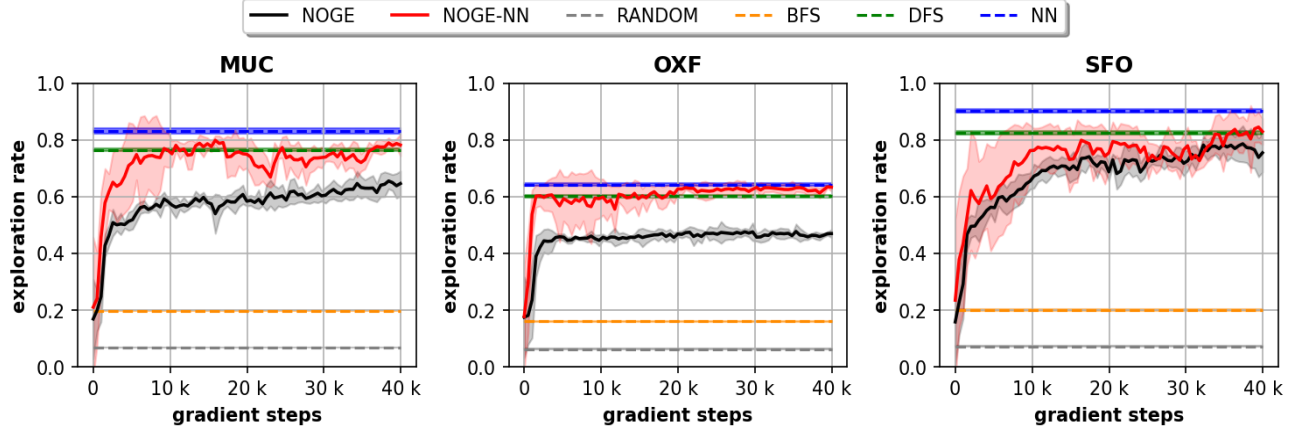


Figure 5: Exploration rate over gradient steps for the city road networks data sets.

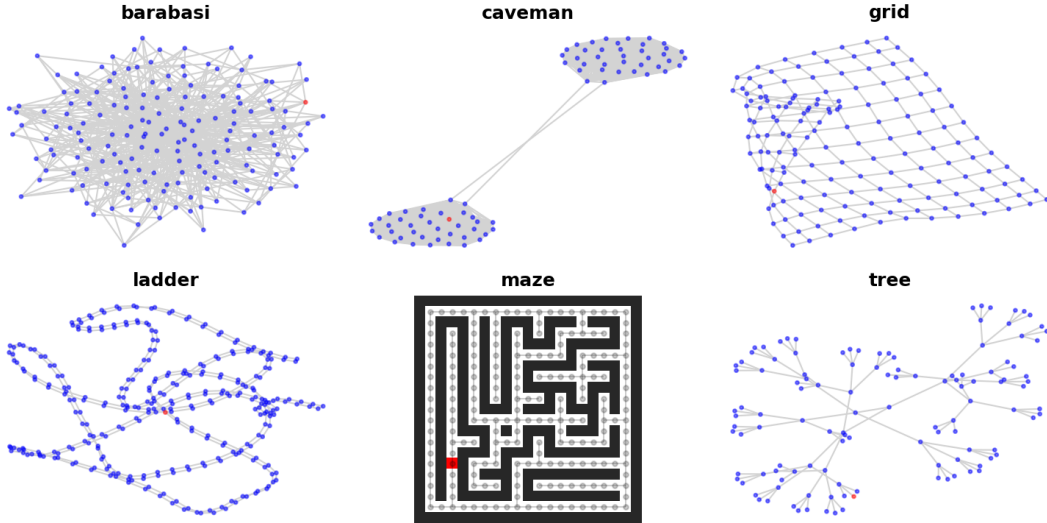


Figure 6: Samples from the procedurally generated data sets used in the experiments of Section 5.2.



Figure 7: The road city networks used for the experiments in Section 5.3.



## References

- Bahdanau, D., Cho, K., and Bengio, Y. (2015). Neural machine translation by jointly learning to align and translate. In Bengio, Y. and LeCun, Y., editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*.
- Battaglia, P. W., Hamrick, J. B., Bapst, V., Sanchez-Gonzalez, A., Zambaldi, V., Malinowski, M., Tacchetti, A., Raposo, D., Santoro, A., Faulkner, R., et al. (2018). Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*.
- Dai, H., Li, Y., Wang, C., Singh, R., Huang, P., and Kohli, P. (2019). **Learning transferable graph exploration**. In Wallach, H. M., Larochelle, H., Beygelzimer, A., d’Alché-Buc, F., Fox, E. B., and Garnett, R., editors, *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 2514–2525.
- Das, S., Dereniowski, D., and Karousatou, C. (2015). Collaborative exploration by energy-constrained mobile robots. In *International Colloquium on Structural Information and Communication Complexity*, pages 357–369. Springer.
- Deng, X. and Papadimitriou, C. H. (1999). Exploring an unknown graph. *Journal of Graph Theory*, 32(3):265–297.
- Dereniowski, D., Disser, Y., Kosowski, A., Pajkak, D., and Uznański, P. (2013). **Fast collaborative graph exploration**. In *International Colloquium on Automata, Languages, and Programming*, pages 520–532. Springer.
- Dobrev, S., Kráľovič, R., and Markou, E. (2012). Online graph exploration with advice. In *International Colloquium on Structural Information and Communication Complexity*, pages 267–278. Springer.
- Dosovitskiy, A. and Koltun, V. (2017). **Learning to act by predicting the future**. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net.
- Duncan, C. A., Kobourov, S. G., and Kumar, V. A. (2006). Optimal constrained graph exploration. *ACM Transactions on Algorithms (TALG)*, 2(3):380–402.
- Edwards, H. and Storkey, A. J. (2017). Towards a neural statistician. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net.
- Grover, A. and Leskovec, J. (2016). node2vec: Scalable feature learning for networks. In Krishnapuram, B., Shah, M., Smola, A. J., Aggarwal, C. C., Shen, D., and Rastogi, R., editors, *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, August 13-17, 2016*, pages 855–864. ACM.
- Guan, M. G. (1962). Graphic programming using odd or even points chinese mathematics. *V ol*, 1(2):73–2.
- Haklay, M. and Weber, P. (2008). Openstreetmap: User-generated street maps. *IEEE Pervasive Computing*, 7(4):12–18.
- Higashikawa, Y., Katoh, N., Langerman, S., and Tanigawa, S.-i. (2014). Online graph exploration algorithms for cycles and trees by multiple searchers. *Journal of Combinatorial Optimization*, 28(2):480–495.
- Kalyanasundaram, B. and Pruhs, K. R. (1994). Constructing competitive tours from local information. *Theoretical Computer Science*, 130(1):125–138.
- Kingma, D. P. and Ba, J. (2015). Adam: A method for stochastic optimization. In Bengio, Y. and LeCun, Y., editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*.
- Kipf, T. N. and Welling, M. (2017). Semi-supervised classification with graph convolutional networks. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net.
- Lee, J., Lee, Y., Kim, J., Kosiorek, A. R., Choi, S., and Teh, Y. W. (2019). Set transformer: A framework for attention-based permutation-invariant neural networks. In Chaudhuri, K. and Salakhutdinov, R., editors, *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, volume 97 of *Proceedings of Machine Learning Research*, pages 3744–3753. PMLR.
- Megow, N., Mehlhorn, K., and Schweitzer, P. (2012). Online graph exploration: New results on old and new algorithms. *Theoretical Computer Science*, 463:62–72.
- Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., and Dean, J. (2013). Distributed representations of words and phrases and their compositionality. In Burges, C. J. C., Bottou, L., Ghahramani, Z., and Weinberger, K. Q., editors, *Advances in Neural Information Processing Systems 26: 27th Annual Conference on Neural Information Processing Systems*

2013. *Proceedings of a meeting held December 5-8, 2013, Lake Tahoe, Nevada, United States*, pages 3111–3119.
- Miyazaki, S., Morimoto, N., and Okabe, Y. (2009). The online graph exploration problem on restricted graphs. *IEICE transactions on information and systems*, 92(9):1620–1627.
- Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. (2015). Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533.
- Perozzi, B., Al-Rfou, R., and Skiena, S. (2014). Deepwalk: online learning of social representations. In Macskassy, S. A., Perlich, C., Leskovec, J., Wang, W., and Ghani, R., editors, *The 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '14, New York, NY, USA - August 24 - 27, 2014*, pages 701–710. ACM.
- Precup, D., Sutton, R. S., and Singh, S. (1998). Theoretical results on reinforcement learning with temporally abstract options. In *European conference on machine learning*, pages 382–393. Springer.
- Rosenkrantz, D. J., Stearns, R. E., and Lewis, II, P. M. (1977). An analysis of several heuristics for the traveling salesman problem. *SIAM journal on computing*, 6(3):563–581.
- Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. (2017). Attention is all you need. In Guyon, I., von Luxburg, U., Bengio, S., Wallach, H. M., Fergus, R., Vishwanathan, S. V. N., and Garnett, R., editors, *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 5998–6008.
- Velickovic, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., and Bengio, Y. (2018). Graph attention networks. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net.
- Vinyals, O., Fortunato, M., and Jaitly, N. (2015). Pointer networks. In Cortes, C., Lawrence, N. D., Lee, D. D., Sugiyama, M., and Garnett, R., editors, *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, pages 2692–2700.
- Yamauchi, B. (1997). A frontier-based approach for autonomous exploration. In *Proceedings 1997 IEEE International Symposium on Computational Intelligence in Robotics and Automation CIRA'97. 'Towards New Computational Principles for Robotics and Automation'*, pages 146–151. IEEE.
- Zaheer, M., Kottur, S., Ravanbakhsh, S., Póczos, B., Salakhutdinov, R., and Smola, A. J. (2017). Deep sets. In Guyon, I., von Luxburg, U., Bengio, S., Wallach, H. M., Fergus, R., Vishwanathan, S. V. N., and Garnett, R., editors, *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 3391–3401.
- Zhang, M. and Chen, Y. (2018). Link prediction based on graph neural networks. In Bengio, S., Wallach, H. M., Larochelle, H., Grauman, K., Cesa-Bianchi, N., and Garnett, R., editors, *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, pages 5171–5181.

## A IMPLEMENTATION DETAILS

### A.1 Hyperparameters

In Table 5 we show the hyperparameters used for the experiments on the procedurally generated data sets. The only differences in the experiments for the city road networks are the number of training steps which was set to 40000 and the hidden layer width of the neural network (see next subsection). We elaborate on hyperparameters, the usage of which may not be immediately clear:

**Node History** It is common in deep reinforcement learning to replace the input observation  $\mathbf{x}_t$  with a stack of the last  $k$  observations  $[\mathbf{x}_{t-k+1}, \mathbf{x}_{t-k+2}, \dots, \mathbf{x}_t]$ , particularly when the observations are images. This gives the agent a sense of the environment dynamics. We found that using a stack of the last 2 feature vectors for each node also improves performance in graph exploration, as it gives a sense of direction.

**Feature Range** As a preprocessing step, shifting input features to the  $[-0.5, 0.5]$  range speeds up learning.

**Target Normalization** As a postprocessing step, target normalization also aids the learning process. We scaled targets  $\mathbf{y}$  by the standard deviation of measurements collected during random exploration, as described by Dosovitskiy and Koltun (2017).

**Evaluation Episodes** For evaluation, we sampled 50 graphs from the test set and fixed one source node per graph. If the test set contained less than 50 graphs, we sampled 50 source nodes uniformly from all test graphs.

**$\varepsilon$ -greedy Policy** As described in our training algorithm, we used an  $\varepsilon$ -greedy policy to collect experiences, namely a random frontier node was selected to be visited with probability  $\varepsilon$  and a node was selected by the network’s policy with probability  $1 - \varepsilon$ . We linearly interpolated  $\varepsilon$  from 1 to 0.15 over the course of training. During testing, the greedy policy ( $\varepsilon = 0$ ) was used.

### A.2 Network Architecture

The architecture of our network, used for the procedurally generated graphs, is shown in Table 6. The same architecture was used for the city road networks except that all layers - apart for input and output - are wider by a factor of two. The input dimension for the graph neural network (GNN) was 3 for NOGE and 4

Table 5: Hyperparameters used in experiments on procedurally generated graphs.

Parameter	Value
test set ratio	0.2
max. episode steps ( $T_{max}$ )	500
node history	2
feature range	$[-0.5, 0.5]$
target normalization	True
training steps	25600
evaluation episodes	50
env. steps per tr. step	32
tr. steps per evaluation	512
replay buffer size ( $ \mathcal{R} $ )	20000
$\varepsilon_{max}$	1
$\varepsilon_{min}$	0.15
temporal coefficients ( $\mathbf{g}$ )	$[0, 0, 0, \frac{1}{4}, \frac{1}{4}, \frac{1}{2}, \frac{1}{2}, 1]$
minibatch size (B)	32
learning rate	0.0001

for NOGE-NN. In all networks we use the ReLU non-linearity after all layers except for the output layer of the row-wise feed-forward (rFF) network.

### A.3 Replay Buffer for Graphs

To use the replay buffer for training, we need to be able to sample graph observations  $G_t$  from any time step in an episode. To do that, for each episode we store the discovered graph  $G_T = (V_T, E_T)$  at the end of the episode and two arrays: an array of node counts and an array of edge counts, indicating the size of the graph at each time step. To be able to reconstruct the frontier at an arbitrary time step  $t$ , we need to store two integers per node  $v$ : the time of discovery  $t_{dis}(v)$  and the time of visit  $t_{vis}(v)$ . Then the frontier  $F_t$  at any time step  $t$  is:

$$F_t = \{v \in V_t : t \geq t_{dis}(v) \wedge t < t_{vis}(v)\} \quad (16)$$

Table 6: The network architecture.

module	input dimension	output dimension
GNN	3 or 4	32
	32	64
MLP	1	64
	64	64
rFF	256	128
	128	8

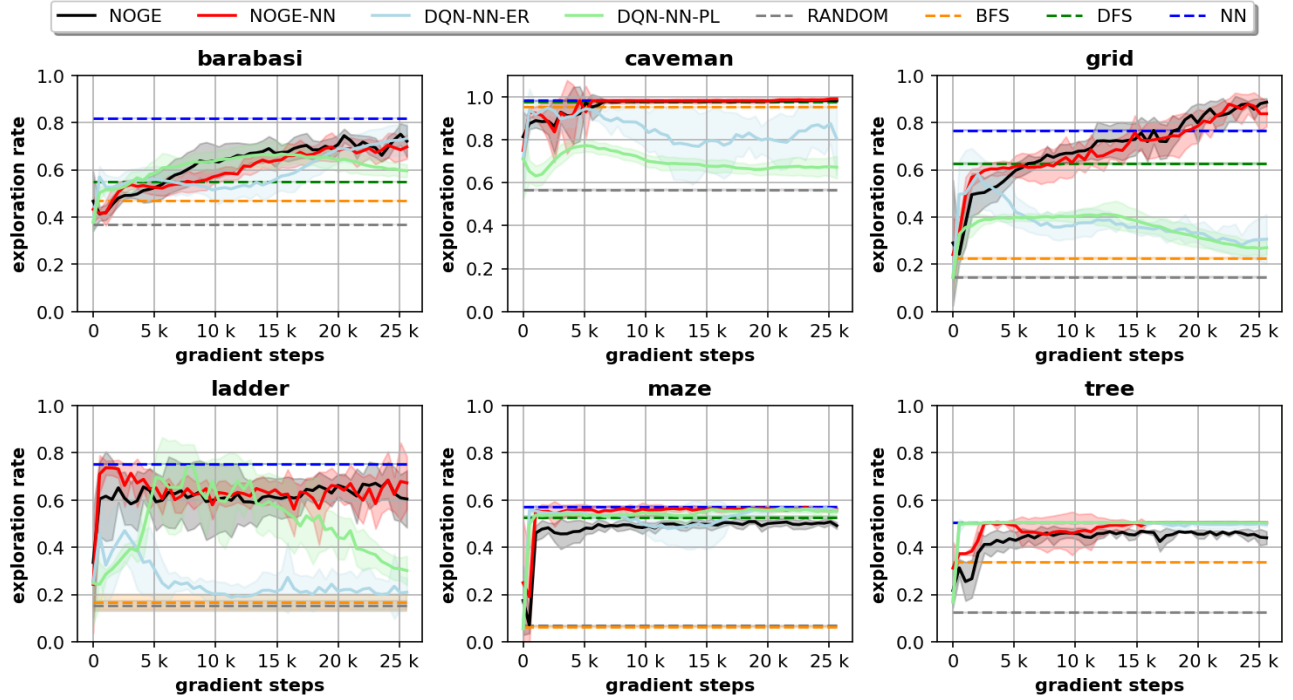


Figure 8: Exploration rate over gradient steps for the six procedurally generated data sets.

## B COMPARISON TO DQN

We originally tried using standard RL algorithms but found them to be unstable compared to DFP. In Figure 8, we additionally show the test exploration rate of a DQN (Mnih et al., 2015) during training on the procedurally generated data sets. The curves show mean and standard deviation over 5 random seeds. Except for the original path length reward function (PL), described in the paper, we also tried using the exploration rate difference between two time steps (ER). In both cases the DQN had access to the NN feature. We can see that in 3 out of 6 data sets, the DQN struggles and degrades to solutions much worse than those found by NOGE.