Learning policies for Social network discovery with Reinforcement learning

Harshavardhan Kamarthi[*], Priyesh Vijayan[*†], Bryan Wilder[‡], Balaraman Ravindran[*†] and Milind Tambe[§]
[*]Dept. of Computer Science and Engineering, Indian Institute of Technology Madras
[†]Robert Bosch Centre for Data Science and AI, Indian Institute of Technology Madras
[‡]Center for Artificial Intelligence in Society, University of Southern California
[§]Center for Research on Computation and Society, Harvard University
Email: {harshavardhan864.hk, bryan.wilder0, milind.tambe11}@gmail.com, {priyesh, ravi}@cse.iitm.ac.in,

***Abstract*—A serious challenge when finding influential actors in real-world social networks is the lack of knowledge about the structure of the underlying network. Current state-of-the-art methods rely on hand-crafted sampling algorithms; these methods sample nodes and their neighbours in a carefully constructed order and choose opinion leaders from this discovered network to maximize influence spread in the (unknown) complete network.**

In this work, we propose a reinforcement learning framework for network discovery that automatically learns useful node and graph representations that encode important structural properties of the network. At training time, the method identifies portions of the network such that the nodes selected from this sampled subgraph can effectively influence nodes in the complete network. The realization of such transferable network structure based adaptable policies is attributed to the meticulous design of the framework that encodes relevant node and graph signatures driven by an appropriate reward scheme. We experiment with real-world social networks from four different domains and show that the policies learned by our RL agent provide a 10-36% improvement over the current state-of-the-art method.

***Index Terms*—Influence Maximization, Reinforcement Learning, Social Networks, Network Representation Learning**

I. Introduction

Social network interventions are used across a wide variety of domains to disseminate information or inspire changes in behavior; application areas range from substance abuse [20], to microfinance adoption [1], to HIV prevention [24, 22]. Such processes are computationally modelled via the *influence maximization* problem, where the goal is to select a subset of $\mathcal{A}$ nodes from the network to spread a message, such that the number of people eventually reached is maximized. Several algorithmic approaches have been proposed for influence maximization [9, 8, 3, 13], mostly to scale up to large networks.

However, real-world applications of influence maximization are often limited by the high cost of collecting network data. In many domains, for instance, those arising in public health, a successful intervention requires information about the face-to-face interactions amongst the members of a population. This information is typically gathered via in-person surveys; conducting such surveys requires substantial effort on the part of the organization deploying an intervention. We are motivated in particular by the problem of using influence maximization for HIV prevention among homeless youth, where algorithms have been successfully piloted in real-world settings [24, 22]. In the HIV prevention domain, gathering the social network of the youth who frequent a given homeless centre requires a week or more of effort on the part of social workers, which is not feasible for a typical community agency.

Accordingly, an important direction for algorithm development is to create methods which *subsample* the population by surveying only a small subset of nodes to obtain network information. Each node that is surveyed reveals its neighbours, and the goal is to carefully select the nodes to be surveyed to choose an influential set of seed nodes. Previous works have developed approaches for this network discovery problem [23, 22]. However, existing algorithms are entirely hand-designed, typically aiming to exploit a specific property of graphs in the target domain such as community structure or the friendship paradox [5].

We propose an alternative framework which automatically learns the structural properties of available social networks and leverages it to learn effective policies for selecting which nodes to query via reinforcement learning. Our approach exploits the fact that we typically have access to historical network data from similar populations (for instance, when deciding which youths to survey for an HIV prevention intervention, we can train using networks gathered at other centres). By leveraging structural information from such datasets, our approach learns more nuanced policies which can both be fine-tuned better to a particular distribution of graphs and which can adapt more precisely over the course of the surveying process itself (since the trained policy is a function of a graph discovered so far). Even if training networks are not available from the specific domain of interest, we show that comparable performance can also be obtained by training on standard social network datasets or datasets generated synthetically using community structure information and transferring the learned policy unchanged to the target setting.

The main contributions of our work can be summarized as follows:

1) We formulate the process of network discovery for influence maximization as a Markov Decision Process.
2) We propose a neural network architecture and a training algorithm that uses Deep Q learning at its core to learn important graph properties from training dataset that in turn helps it to query nodes efficiently for network

discovery at deployment. To the best of our knowledge, this is the first work to use deep reinforcement learning for network discovery to aid influence propagation in unknown networks.

3) We address the challenges involved in designing appropriate graph based state representations and node based action representations that are amenable for deep reinforcement learning methods. Our RL algorithm can deal with an arbitrary number of actions that can vary depending on the state. We also made training on multiple graphs possible by designing an appropriately scaled reward scheme.
4) We achieve improved influence spread at deployment by using an RL agent that is trained on multiple similar real-world networks from the same domain. We further show that in the absence of a large number of similar networks, we can additionally leverage numerous synthetic networks generated with similar community structures to maximize the performance further.
5) We experimentally evaluate our RL based network discovery algorithm on social networks from four different domains. Our model outperforms existing algorithms by a substantial margin of 10-35% improvement over the previous state-of-the-art approaches.

II. Preliminaries

A. Reinforcement Learning

In the reinforcement learning setting an agent has to learn to make decisions in an unknown environment in order to maximize a reward signal. The agent's decision affects the subsequent *state* of the environment which further affects the reward signals for all actions in the next time-step.

We describe the task as a Markov Decision Process (MDP). A MDP $\mathcal{M}$ is a tuple $< \mathbb{S}, \mathbb{A}, R, T, p_o, \gamma >$ where $\mathbb{S}$ is the set of possible states, $\mathbb{A}$ is the set of actions, $R : \mathbb{S} \times \mathbb{A} \to \mathbb{R}$ is the reward function. $R(s,a)$ is the average of the reward received by an agent on taking action a when it is at state s. $T(s,a,s')$ is the transition function that gives the probability that agent goes to state s' on taking action a when it is at state s. p_o is the initial state probability distribution. Let R_t be the reward received at timestep t on taking action a_t in state s_t. The agent seeks to maximize the return $G = \sum_{k=0}^{\infty} \gamma^k R_k$ where $\gamma \in [0,1]$ is called the discount factor.

A *policy*, π is a mapping of states to probability distributions over actions. Formally $\pi(a|s)$ is the probability of selecting action a at state s. The *value function* $V_\pi(s)$ of state s over a policy π is the expected return when the agent starts at state s and acts according to π.

$$V_\pi(s) = \mathbb{E}_{a \sim \pi(.|s)}[R(s,a) + \mathbb{E}_{s' \sim T(s,a,.)}[V_\pi(s')]]$$

The *state-value function* $Q_\pi(s,a)$ of state-action pair (s,a) is the expected return when the agent starts at state s, chooses action a and acts according to π thereafter.

$$Q_\pi(s,a) = R(s,a) + \mathbb{E}_{s' \sim T(s,a,.)}[V_\pi(s')]$$

An optimal policy π^* is a policy such that

$$v_{\pi*}(s) = \max_\pi v_\pi(s)$$

An iterative algorithm to converge to Q_{π^*} is *Q-learning* [21] where we perform the following update at each step:

$$Q(s_t,a_t) \leftarrow Q(s_t,a_t) + \alpha\delta_t$$

where $\delta_t = [R_t + \gamma \max_a Q(s_{t+1},a) - Q(s_t,a_t)]$ is called the *TD-error*.

B. Deep Q learning Networks

When the action set $\mathbb{S}$ is large, it is infeasible to store value function and perform Q-learning updates for all possible states. Deep Q-networks(DQNs) learn to approximate the function $Q(s,a)$ using a neural network [14]. A Q-network with parameters θ receives state s and action a and outputs the estimated state-value $Q_\theta(s,a)$. A DQN algorithm has two networks, a source network $Q_\theta(s,a)$ and a target network $Q_{\theta'}(s,a)$. The parameters of the source network are copied to target network periodically. The DQN algorithm gathers experience via storing the tuples of the form (s_t, a_t, r_t, s_{t+1}) in a replay buffer D and periodically sampling from the buffer to update the neural network to minimize the loss via gradient updates using Backpropogation.

$$\mathbb{E}_{(s,a,r,s') \sim D}[(r + \gamma \max_{a'} Q_{\theta'}(s',a') - Q_\theta(s,a))^2] \quad (1)$$

C. Prioritized Experience Replay

In the original DQN algorithm, to update the network parameters, we sample experiences from replay buffer D at random. However, some experiences are more useful than others for efficient learning. Prioritized Experience replay [18] samples experiences with probability proportional to a function of TD-errors of the experiences.

The probability of choosing a sample $e_i = (s_i, a_i, r_i, s'_i)$ is $P(i) = \frac{f(e_i)^\beta}{\sum_j f(e_j)^\beta}$ where $f(e_i)$=$|\delta_i|+\epsilon$ with δ_i as the TD-error corresponding to experience e_i and ϵ a small positive constant.

D. Graph representation

In our RL setup, the environment at timestep, t is the graph discovered at t. To obtain a rich graph (state) representation, we leverage the recent progress in Network Representation Learning with deep learning models to get efficient graph representations. Specifically, we use a neural network with permutation invariant Graph convolutional layers and Differentiable pooling layers [26] to obtain graph representations.

Differentiable Pooling (DiffPool) [26]
Given a graph, G with an adjacency matrix, $A \in \mathbb{R}^{n \times n}$ and a node feature matrix, $F \in \mathbb{R}^{n \times d}$ where n is the number of nodes and d is the number of features, Differentiable pooling (Diffpool) can be used to learn hierarchical representations of the graph, G in an end-end differentiable manner. It allows for learning hierarchical representations for graphs by iteratively coarsening the graph and learning representations for the coarsened graph at each stage. Diffpool can be used to map a graph to a single finite dimensional representation by

iteratively coarsening the input graph to a graph with a single node and extracting features for this new one node graph. In this work, we define a neural network that uses multiple Diffpools to pool the nodes and obtain a single graph level representation.

Let L be the number of coarsening steps (Diffpools) used to obtain a single graph with one node, i.e $A^{(L)} \in \mathbb{R}^{n_L \times n_L}$ where $n_L = 1$. At every step l of the coarsening process, let the input graph be defined by a weighted adjacency matrix, $A^{(l)} \in \mathbb{R}^{n_l \times n_l}$ and a feature matrix, $F^{(l)} \in \mathbb{R}^{n_l \times n_d}$. At the first step, the adjacency matrix and feature matrix corresponds to the input graph, G, i.e $A^{(0)} = A$ and $F^{(0)} = F$. The iterative coarsening of the graph is achieved by learning a soft cluster assignment, $C^{(l)} \in \mathbb{R}^{n_l \times n_{l+1}}$ at each stage l.

Diffpool leverages Graph Convolutional Networks (GCNs) [11] to obtain the cluster assignment matrix at each stage. GCNs are conventionally used to learn node representations for the task of node classification. A node representation summarizing a K-hop information can be obtained by stacking K layers of GCN. Similarly in our neural network, at every coarsening stage l before the Diffpool layer, K GCN layers are stacked to output a n_{l+1} dimensional cluster assignment for all the n_l nodes in the input graph. Each GCN layer before a Diffpool layer in the l^{th} coarsening layer of the neural network is defined as in Eqn: 2, where $\hat{L}(A^{(l)}) = \hat{D}^{(l)^{-\frac{1}{2}}} \hat{A}^{(l)} \hat{D}^{(l)^{-\frac{1}{2}}}$ is the renormalized Laplacian defined in [11] with weight matrix, $W_1^{(l)} \in \mathbb{R}^{n_l * n_{l+1}}$ and $\forall k > 1, W_k^{(l)} \in \mathbb{R}^{n_{l+1} * n_{l=1}}$. $\hat{D}^{(l)}$ is the degree matrix and $\hat{A}^{(l)}$ is the adjacency matrix defined for the l^{th} layer graph with a self-loop over all the nodes. The output of the last GCN layer, $h_K^{(l)}$ is made the cluster assignment matrix (pooling matrix), $C^{(l)}$ as in Eqn: 3.

$$h_k^{(l)} = ReLU(\hat{L}(A^{(l)}) h_{k-1}^{(l)} W_k^{(l)}) \tag{2}$$

$$C^{(l)} = h_K^{(l)} \tag{3}$$

This soft cluster assignment, $C^{(l)}$ is used to map the set of nodes from the current graph with an adjacency matrix, $A^{(l)}$ to a smaller graph with a substantially lower number of (clusters) nodes defined by an adjacency matrix, $A^{(l+1)}$ as in Eqn: 4. The new adjacency matrix, $A^{(l+1)}$ in essence is as a cluster similarity matrix.

$$A^{(l+1)} = C^{(l) \cdot T} A^{(l)} C^{(l)} \tag{4}$$

The feature matrix for the nodes in the newly defined graph with $A^{(l+1)}$ is computed as a weighted summation of the different node's features, $Z^{(l)}$ as per the node-cluster assignments, $C^{(l)}$, see Eqn: 5. $Z^{(l)}$ can be 0-hop node features, $F^{(l)}$ or a K-hop node features extracted with a K layer GCN that outputs d dimensional node features. In this work, we use two different GCN layers with $K = 2$ for learning both the cluster assignment matrix, $C^{(l)}$ and the node feature matrix, $Z^{(l)}$. The graph feature from the last layer, $F^{(L)}$ is used as the state representation in our RL algorithm.

$$F^{(l+1)} = C^{(l) \cdot T} Z^{(l)} \tag{5}$$

E. *Influence Model*

To model information diffusion over the network, we use the standard independent cascade model (ICM) [10], which is the most commonly used model in the literature. In the ICM, every node is either active or inactive. At the start of the process, every node is inactive except for the seed nodes, S. The process unfolds over a series of discrete time steps. At every step, each newly activated node attempts to activate each of its inactive neighbors. Each edge (u, v) is endowed with a propagation probability $p_{u,v}$, which gives the probability that u succeeds in influencing v. The process ends when there are no newly activated nodes. Our objective is to choose a limited budget of $|S|$ seed nodes such that the expected number of active nodes at the end of the process is maximized.

III. RELATED WORKS

A. *Influence Maximization and network discovery*

There is a large existing literature on various algorithmic aspects of influence maximization problems [10, 8, 3, 19, 13]. Almost all of this work builds on a greedy strategy for influence maximization, which iteratively adds the node with the largest marginal contribution to the objective until the budget is reached. The main challenge is computational: evaluating the influence spread requires simulating the process, and a great deal of effort has gone into finding efficient methods which reduce the expense on the number of simulations required.

Our work is situated along a more recent, largely orthogonal axis: developing algorithms which reduce the data collection requirements needed to deploy influence maximization in the real world. In many domains, a fundamental bottleneck to deploying influence maximization is the time and expense required to collect network data, even when sufficient computation is available [22]. Wilder et al. [23] introduced the exploratory influence maximization problem, where the goal is to sample nodes from the network to query for data such that an influential seed set can be selected. They propose an algorithm motivated by community structure and prove theoretical guarantees for graphs drawn from the stochastic block model. Later, [22] introduced a more practical algorithm called CHANGE based on the friendship paradox [5]. CHANGE repeatedly queries a random node and then a random neighbour of that node for data, running the greedy algorithm on the revealed subgraph. Our objective in this work is to move beyond these hand-designed heuristics, which target a specific structural property observed in real-world networks. To that goal, we aim to exploit the availability of previously gathered network datasets to automatically learn policies which can leverage non-obvious features of the network distribution.

B. *Representation learning for influence maximization*

The application of deep learning models for solving problems related to influence maximization is fairly recent. The models proposed in [15] use previous influence spread information to model diffusion probabilities between nodes and estimate the ability of nodes to cascade information through the *same* network. The algorithms proposed in [25] learn a mapping between active nodes and influenced nodes based on observation from the previous influence spread phenomenon. This approach doesn't require explicit network structure. However, unlike our approach, both these works rely on prior information about the network such as previous diffusion phenomenon. In real-world social networks, where the network structure is not provided, we don't have easy access to such information.

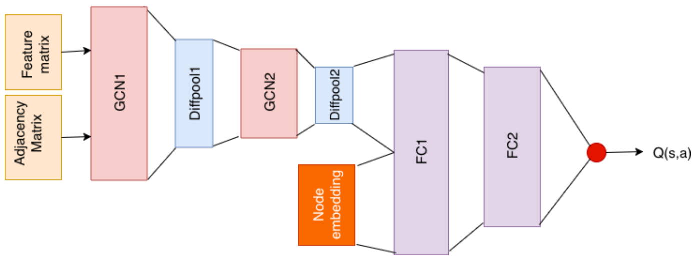


Figure 1: Model Architecture

IV. Problem Description

Let the entire unknown graph be $G^* = (V^*, E^*)$. Let $X \subseteq V^*$ denote a vertex set and let $G[X]$ denote a sub-graph of G^* induced by X. Let $V(G)$ be the vertex set of a graph G. Let $N_G(u)$ be neighbors of vertex u in a graph G. We abuse the notation, $N_G(X) = (\cup_{v \in X} N(X))/X$, i.e., all nodes which are neighbours of nodes in X except those in X.

Initially, we are given $|S|$ seed nodes and a budget of T queries. When we query a node, we discover the neighbours of the queried node. Let G_t be the sub-graph discovered after t queries with vertex set, $V_t = V(G_t)$. Let $G_0 = G[S \cup N_{G^*}(S)]$ with S as the initial seed set. During the $t+1$ query, we choose a node u_t from G_t and get $G_{t+1} = G[V_t \cup N_{G^*}(u_t)]$. Let $\mathcal{A} = \mathcal{O}(G)$ be final set of nodes chosen by an oracle, $\mathcal{O}$ to be *activated* for maximizing the influence spread in G. Let $I_G(\mathcal{A})$ be the expected number of influenced nodes in G on choosing $\mathcal{A}$ as the set of initial active nodes. The task is to find a sequence of queries $(u_0, u_1, \dots, u_{T-1})$ such that the discovered graph G_T is such that it maximizes the $I_{G^*}(\mathcal{O}(G_T))$, i.e, we need to discover a sub-graph G_T such that the nodes selected by $\mathcal{O}$ in G_T maximizes the number of nodes influenced in the entire graph, G^*.

V. Methodology

A. A Markov Decision Process Formulation

We formulate the sequential decision task formulated in Section IV as a MDP below:

State: At every time-step t, the current state is the discovered graph G_t.

Actions: Given a sub-graph G_t, we can query any of the nodes in G_t which are not yet queried. Thus, the action space is $V_t/\{S \cup_{i \leq t} u_i\}$ if $t > 0$ or is $N_{G^*}(S)$ if $t = 0$.

Rewards: The actual reward we get after T steps is the number of nodes influenced in the entire graph, G^* using the discovered graph, G_T which is $I_{G^*}(\mathcal{O}(G_T))$. We denote this reward, which the model receives at the end of an episode as R_i. However, the range of these values are highly dependent on size and structure of the influenced network. Therefore, we can't directly use this signal to train with multiple graphs simultaneously

When using multiple networks, the reward scheme should reflect the effectiveness of the policy across different networks of varying size and structure. We solve this problem by directly comparing the performance of the policy with the performance of the baseline, CHANGE at the end of each episode.

Further, we normalize this difference with respect to the difference between the performance of the baseline and the optimal performance of the training graph. This keeps the range of reward between 0 and 1 for all graphs and also helps improve the stability of DQN training [14].

Formally, we normalize the influence reward as:

$$R_s = \frac{I_{G^*}(\mathcal{O}(G_T)) - CHANGE(G^*)}{OPT(G^*) - CHANGE(G^*)} \tag{6}$$

where $CHANGE(G)$ is the average number of influenced nodes on sampling via CHANGE method (discussed in Section VI-B) and OPT is the number of influenced nodes when we select the active nodes given the knowledge of entire graph, i.e, $I_{G^*}(\mathcal{O}(G^*))$.

We use $CHANGE$ for bounding R_s in Equation 6 because of two reasons. Firstly, $CHANGE$ is a powerful method for graph discovery and dominates over other state-of-art sampling methods in terms of performance (see Appendix Table VII). Secondly, it is computationally inexpensive to simulate sampling according to $CHANGE$ on networks of large sizes. Hence we can pre-compute performance of $CHANGE$ for multiple training graphs before training the RL model.

We also add *step-rewards* at each step to help alleviate reward sparsity problem and encourage the agent to learn to find larger graphs.. The *step-reward* $R_{p,t}$ at time t is given as:

$$R_{p,t} = \frac{|V(G_t)| - |V(G_{t-1})|}{|V(G^*)|} \tag{7}$$

B. State and action representation

The MDP formulation presents us with challenges atypical of most reinforcement learning problems. A social network is a very structured object that can vary in size and complexity. We require a method to extract useful vector representations that encode the structural properties of the graph that are useful for the node discovery problem. A vector representation would also allows us to work seamlessly with deep reinforcement learning algorithms. The actions, which are nodes of the networks yet to be queried, need to be represented as vectors as well and the vectors need to encode structural information of the node in the context of the discovered network.

We use a Diffpool based neural architecture discussed in section: II-D to obtain graph representation, $F^{(L)}$. However, in the absence of node features, $F^{(0}$, using features learned jointly with the objective resulted in unstable training as the input was non-stationary. Hence, we opted for using pre-trained DeepWalk embeddings ($\Phi \in \mathbb{R}^*$) [16] learned for each G_t as the feature matrix, i.e $F_{(0)} = \Phi$. We also utilized Deepwalk embedding for the node representations. We found using embeddings for both nodes and graphs to help the RL model learn a generalized policy utilizing structural information from both.

Reinforcement learning problems usually have a fixed set of actions. Even when the action space is continuous, the range of action is known a priori and bounded. However, in this case, the action representations for nodes is a function of the state (discovered graph). Thus, we won't know the size of the action set to encode them apriori. Therefore, the network architecture of DQN in the original paper [14], which takes only the state representation as input and outputs action values can't be used. Instead, we input both state and action representation to the DQN and train it to predict the state-action value. This also allows the model to generalize to networks whose size may vary over time as well as run on different network datasets.

Our model (see figure 1) learns graph embeddings along with the policy using node and graph embeddings by back-propagating the TD error (Equation 1) to learn parameters of GCN layers also.

C. Model training and deployment

For training, we can use single or multiple graphs. Algorithm 1 summarizes the training steps. At the start of every episode, we sample a graph at random from available training graphs (Line 4). Every time we expand the discovered graph, for timestep t, we compute node embeddings for all vertices in G_t (Lines 9, 15). Deepwalk representation for node v, denoted as $\phi(v)$, is a d dimensional vector. Then we create the feature matrix $F_t \in \mathbb{R}^{|V_t| \times d}$ whose rows are the embeddings of nodes in V_t. F_t is fed along with adjacency A_t as input for GCN (Lines 9, 16). Thus, the state is represented by $S_t = (F_t, A_t)$. For each of the nodes v which are not queried yet, we get the state-value $Q(S_t, \phi(v_t))$ and choose the node v_t that has maximum estimated state-value to query next (Line 12). We receive the reward as discussed in Section V-A and observe the new graph G_{t+1} (Line 13). We also store the experience $(S_t, \phi(v_t), R_t, S_{t+1})$ in replay buffer. Since we use prioritized replay, we also compute the TD error which is used to determine the importance of the experience in training (as discussed in Section II-C) (Lines 18, 19). Since the rewards are scaled (Equation 6) we can combine experiences from different graphs to learn from all of them simultaneously. This is especially useful where we can augment available real-world networks with synthetic graphs that would provide more valuable experience to RL agent to generalize better.

After we train the DQN, we can deploy it on a new network using Algorithm 2. The deployment algorithm is similar to training algorithm in that we compute Deepwalk embeddings for nodes at each step and find the next node to query based on the estimated state-values of all nodes which are not queried yet. We just fix the parameters of the neural network and execute the network discovery policy.

Algorithm 1: Train Network

```
Input : Train Graphs $\mathcal{G} = \{G_1, G_2, \dots, G_K\}$, number of
        episodes $N$, Query budget $T$, number of random
        seeds $|S|$
1  Initialize DQN $Q_\theta$ and target DQN $Q_{\theta'}$;
2  Initialize Prioritized Replay Buffer $B$;
3  for episode = 1 to $N$ do
4      Choose a graph $G$ from $\mathcal{G}$;
5      Select random nodes from $G$ as $S$;
6      $V_0 = S \cup N_G(S)$;
7      Initial graph is $G_0 = G[V_0]$;
8      Compute Deepwalk node embeddings for $G_0$ as $\phi$;
9      Get feature matrix $F_0$, adjacency matrix $A_0$ for $G_0$
         as $S_0 = (F_0, A_0)$;
10     $X \leftarrow N(S)$;
11     for $t$ = 0 to $T-1$ do
12         With probability $\epsilon$ select a random node $v_t$ from
             $X$ else select node $v_t \leftarrow \underset{v \in X}{argmax} Q_\theta(S_t, \phi(v))$;
13         Query node $v_t$ and observe new graph $G_{t+1}$;
14         Set $R \leftarrow R_{p,t}$ (Eqn. 7).;
15         If $t = T-1$, $R \leftarrow I_{G^*}(\mathcal{O}(G_T)) + R_{p,t}$;
16         Compute scaled influence reward $R_t$ (Eqn. 6);
17         Compute Deepwalk node embeddings for $G_{t+1}$
             as $\phi$;
18         Get feature matrix $F_t$, adjacency matrix $A_{t+1}$ for
             $G_{t+1}$ as $S_{t+1} = (F_{t+1}, A_{t+1})$;
19         $X \leftarrow$ nodes not yet queried in $G_{t+1}$;
20         $\delta_t =$
             $R_t + \gamma \max_{v \in X} Q_\theta(S_{t+1}, \phi(v)) - Q_\theta(S_t, \phi(v_t))$;
21         Add $(S_t, \phi(v_t), R_t, S_{t+1})$ to replay buffer $D$ with
             TD-error $\delta_t$ to calculate sample priority;
22         Sample from $B$ and update $Q_\theta$;
23     end
24     Update target network $Q_{\theta'}$ with parameters of $Q_\theta$
         from time to time;
25 end
```

VI. Experiment setup

Here, we provide the following details concerning our experimentation setup: (i) the datasets used, (ii) the baselines compared, (iii) evaluation metric (iv) model hyper-parameters and (v) synthetic graph generation methods.

A. Datasets

We evaluate the effectiveness of our proposed model on datasets from four different domains. The network statistics of each for these family of networks are shown in Appendix Table VII.

a) Rural Networks: We used the networks gathered by [1] to the study diffusion of micro-finance in Indian rural households. Different household networks correspond to different rural regions. Each of these networks models a household in a particular region as a node and connects them by an edge if they were related by a set of possible

Algorithm 2: Deploy Network

```
Input : Trained network $Q_\theta$, Graph to be deployed on
        $G$, initial random seeds $S$, query budget $T$
1  $V_0 = S \cup N_G(S)$;
2  Get the initial graph $G[V_0]$;
3  Compute Deepwalk node embeddings for $G_0$ as $\phi$;
4  Get feature matrix $F_0$, adjacency matrix $A_0$ for $G_0$ as
     $S_0 = (F_0, A_0)$;
5  $X \leftarrow N(S)$;
6  for $t$ = 0 to $T - 1$ do
7      Select note $v_t \leftarrow \underset{v \in X}{argmax} Q_\theta(S_t, \phi(v))$;
8      Query node $v_t$ and observe new graph $G_{t+1}$;
9      Compute Deepwalk node embeddings for $G_{t+1}$ as $\phi$;
10     Get feature matrix $F_t$, adjacency matrix $A_{t+1}$ for
         $G_{t+1}$ as $S_{t+1} = (F_{t+1}, A_{t+1})$;
11     $X \leftarrow$ nodes not yet queried in $G_{t+1}$;
12 end
13 $\mathcal{A} \leftarrow \mathcal{O}(G_T)$;
14 Activate nodes in $\mathcal{A}$ to start the influence process;
```

relations such as health, finance, family, friendship, etc. For our experimental study, we considered four such networks (*rural1-4*).

b) Retweet Networks: These are information flow networks extracted from the Twitter social network. In these networks, each node is a Twitter user, and two users are connected in a graph if one of the users retweets the tweets of the other. We considered four such retweet networks from the online network dataset repository [17] [1] viz: *occupy, copen, israel, damascus*. Each of these retweet networks is related to a specific hashtag based information flow. *occupy* network is related to hashtags concerning the famous "Occupy Wall Street" movement, *copen* is related to mentions about UN conference held in Copenhagen. *israel* and *damascus* are concerned about tweets with political hashtags that are related to the country Israel and the city of Damascus.

c) Animal Interaction Networks: These networks are a part of the wildlife contact networks collected by [4] at different sessions. They specifically studied the physical interactions between Voles and created a contact network. In these contact networks, the animals (Voles) are modeled as nodes, and there is an edge between them if the animals were caught together in one of the traps laid out in the study. We use four of these contact networks for our experiments (*voles1-4*).

d) Homeless Networks: We collected homeless networks from various HIV intervention campaigns organized for homeless youth in Los Angeles. These networks are gathered from previous intervention campaigns [22, 23]. We considered ten of these networks for our experiments: *a1, b1, cg1, node4, mfp2, mfp3, spy2, spy3*.

[1]http://networkrepository.com/

B. Sampling methods

We describe below four sampling based graph discovery baselines which are used in the real world [23, 20]. CHANGE and RANDOM-GREEDY discover the network first by querying and then select nodes from the discovered network to be activated for spreading influence. SNOWBALL and RECOMMEND select node to activate during discovery. Let $\mathcal{A}$ be the final set of nodes chosen to be *activated*.

1) *RANDOM-GREEDY*: Along with the given initial $|S|$ seeds we query another T nodes at random. Then from the subgraph made up of queried nodes, initial seed nodes and their neighbors we use $\mathcal{O}$ to obtain $\mathcal{A}$, nodes to be activated.
2) *RECOMMEND*: We query a node at random and its neighbors and then add the neighbor with the maximum degree to $\mathcal{A}$. We do this until we exhaust the total budget of $2T$ queries. If we don't have sufficient nodes in $\mathcal{A}$ then we get the other nodes from $\mathcal{O}$ from discovered subgraph.
3) *SNOWBALL*: We start by querying a node at random and its neighbors and then adding the neighbor with the maximum degree to $\mathcal{A}$. Then we again query neighbours of the node newly added to $\mathcal{A}$ and add the neighbor with the maximum degree to $\mathcal{A}$. In case we have already queried all neighbors, we again start with querying another random node. Similar to *RECOMMEND*, we do this till we exhaust out query budget. Then we choose the rest of the nodes from the greedy algorithm $\mathcal{O}$ on the discovered graph.
4) *CHANGE* [22]: This is a recent method that was used for effective HIV intervention campaign. It uses a simple yet powerful sampling method: *For each of the random seeds, we query one of its neighbors picked at random.* The model is inspired by *friendship paradox* which states that the expected degree of a random node's neighbor is larger than the expected degree of a random node. Again we use $\mathcal{O}$ to get nodes for $\mathcal{A}$.

We show the influence score for each of these methods in Appendix Table VII. CHANGE outperforms other methods in most cases. Therefore, we choose it as our baseline.

C. Environment parameters

We assume that information flow is modelled by independent cascade model [9] discussed in Section II-E. We fix the diffusion probabilities for all edges as 0.1. We also fix the maximum number of nodes that can be activated after network discovery as 10. Before we start network discovery, we are given 5 random seed nodes $\mathcal{R}$ and their neighbourhood is revealed. We have $T = 5$ queries to discover the graph G_T using which we find the 10 nodes to activate. Thus, the number of queries is equal to the number of random seeds ($|\mathcal{R}| = T$).

D. Performance metrics

We employ the efficient greedy algorithm [9], which we denote as the oracle $\mathcal{O}$, on the discovered graph to pick the nodes to activate. OPT is the influence score when we have the entire network for $\mathcal{O}$ to choose from. This value can be represented as $I_{G^*}(\mathcal{O}(G^*))$ We use the following performance metrics to validate our models:

- *influence score* or *influence reward*: We deploy our model on graphs from the test set and consider the *average number of nodes influenced* over 100 runs as the performance metric. We use the terms *influence score* and *influence reward* interchangeably to to refer to this metric.
- *increase percent*: percentage increase over CHANGE baseline
- *improve percent*: percentage reduction in the difference between OPT and influence of baseline. This is same as the scaled reward used in training (Equation 6).

increase percent and *improve percent* are useful to aggregate performance over multiple networks since the range of values for actual *influence scores* for each network vary depending on network size and structure.

E. Simulation experiments setup

For each of the 3 families of networks mentioned in Section VI-A, we divide them into train and test data as shown in Table I.

Network category	**Train networks**	**Test Networks**
Rural	rural1,rural2	rural3,rural4
Animal	animals1,animals2	animals3, animals4
Retweet	copen, occupy	assad, isreal,obama,damascus
Homeless	a1,spy,mfp	b1,cg1,node4, mfp2,mfp3,spy2,spy3

Table I: Train and test split for different sets of networks

We use Algorithm 1 to train models on networks from training set and deploy them on the networks from test set for each network family.

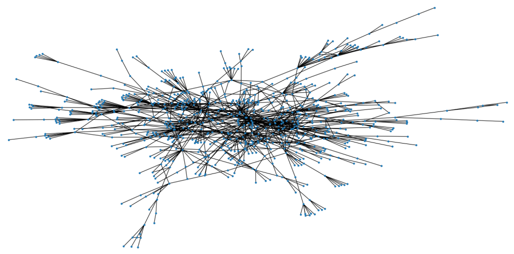
F. Synthetic graph generation

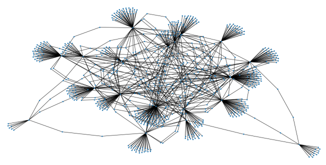
When we don't have actual networks for training we can still use known structural properties of the family of networks we are dealing with to synthetically generate graphs for training. Even if we have a good set of training graphs, we can also use the synthetic graphs generation as a data augmentation technique. Then, the synthetic graphs emulate small perturbations to training graphs and provide our training algorithm with a richer set of experience.

We discuss a simple graph generation technique based on the assumption that our social networks have similar structures to graphs generated by **stochastic block models**. Real-world social networks have densely connected components called *communities* [12]. The nodes of the same community are tightly connected and nodes of different communities are less frequently connected by an edge. Stochastic Block Models(SBMs), which originated in sociology [7], can generate graphs that emulate such structural properties. The nodes of a graph are divided into communities $\{C_1, C_2, \ldots, C_k\}$. We add an edge between two nodes of the same community with probability p_{in} and we add an edge between two nodes of different community with probability p_{out}.

Given a training graph, we now wish to estimate community sizes $\{C_1, C_2, \ldots, C_k\}$ and edge probabilities p_{in}, p_{out} to generate a graph with similar community based properties. First, we use the Louvain community detection algorithm [2] to partition the graph into communities. We find the maximum likelihood estimate for p_{in} and p_{out} based on the number of edges that connect nodes of same community and number of edges that connect nodes of different communities in the training graph. Then, we construct SBM using the calculated parameters to generate synthetic graphs.

However, Retweet networks don't usually resemble the stochastic block model structure. Rather, in each of the communities detected by Louvain Algorithm, we observe that all nodes in the community are connected to one or two nodes only (see Figure 2). Hence, to generate synthetic graphs similar to retweet networks we tweak the SBM generation procedure. For nodes in each community, we choose a single node in the community and connect all other nodes to that node. We call this graph generation model as *Stochastic Star Model*(SSM).


(a) copen


(b) Generated by SSM

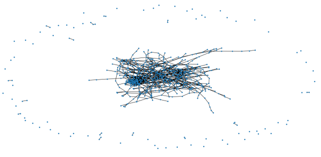

(c) Generated by SBM(Most nodes are isolated)

Figure 2: Difference in graphs generated by SBM and SSM for retweet networks.

VII. Results

The policies learned through reinforcement learning by our agent results in a significant increase in the number of nodes influenced. Table II, which shows the scores of *best* models from all our experiments (Appendix Tables VIII, X) for each graph averaged across graphs of same family, clearly demonstrates the effectiveness of learned policies.

Network Family	**increase %**	**improve %**
Rural	10.54	23.76
Animal	36.03	26.6
Retweet	33.87	19.7
Homeless	21.03	7.91

Table II: Summary of best results (Scores are averaged test networks for each class)

In the following subsections, we discuss multiple ways we can improve robustness of training in the face of uncertainties such as choosing the right training graph or overcoming lack of actual real-world network to train on. First, we show that simultaneously training with multiple networks is on average better than using a single network. Then, we show the effectiveness of synthetic graphs both as an effective substitute for real networks and as a valuable method for data augmentation.

a) Individual network vs Multiple networks for training: When we have networks from similar domain to the ones we are attempting to influence, we can directly use these networks to discover efficient policies for sampling from the unknown network. One way to do this is to pick one of the train networks to train on. However, since our approach uses scaled rewards (Equation 6) we can gather information from multiple networks simultaneously.

Train\Test	rural3	rural4
Avg. individual	**13.14**	14.86
rural1+rural4	10.26	**18.71**

Train\Test	voles3	voles4
Avg. individual	13.07	29.05
voles1+voles2	**14.84**	**35.21**

Train\Test	damascus	israel
Avg. individual	11.58	8.16
copen+occupy	**15.54**	**15.37**

Train\test	b1	cg1	node4	mfp2	mfp3	spy2	spy3
Avg. individual	4.76	12.51	25.75	11.74	7.18	**21.40**	31.61
a1+mfp+spy	**14.29**	**22.11**	**35.12**	**12.92**	**18.10**	20.76	**35.77**

Table III: Comparing average *improve percent* using individual network with that for training with multiple networks

As shown in table III, for most cases training with multiple networks gives better performance gains compared to average performance gains received by training on single networks.

In practice, we may not know a priori which of the train networks is most suitable for a particular setting. Since we typically have only shot at deployment, training on multiple graphs would help alleviate this problem.

b) Synthetic graphs can be valuable: As discussed in Section VI-F, synthetic graphs can be useful substitutes when we don't have access to real-world social networks. Since synthetic graphs are cheaper substitutes to collecting more real-world data, they can be used along with available networks for training.

We performed two different experiments to validate the usefulness of synthetic graphs: 1) Use only the synthetic graphs generated for training, 2) Use synthetic graphs along with training dataset. During training, we use synthetic graphs with probability 0.5 for an episode. Otherwise, we sample from the training dataset. Note that for Retweet networks we used SSM discussed in Section VI-F and we used SBMs for other families of networks as models for graph generation. We compare *improve percent* for different families of graphs using these two settings along with other settings in figure 3.

First, we see that the *improve percent* scores for models using only synthetic graphs are better than the average of

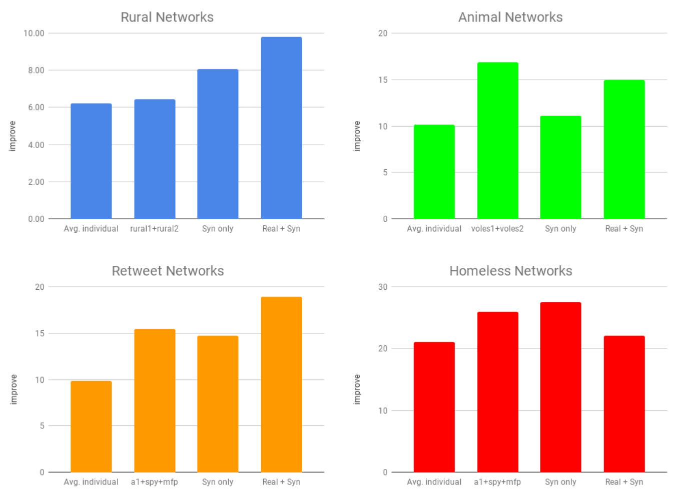


Figure 3: Variation of average *improve percent* using synthetic data. Syn only: Only synthetic graphs for training. Real+Syn: Use both synthetic and actual train graphs

the scores of models trained from individual networks. This indicated that training on synthetic graph alone is a more robust method than training on individual real networks. For Homeless, Retweet and Rural networks, data augmentation improved the performance on the model trained on only training graphs. In the case of Homeless networks using only synthetic graphs give the highest score on average. This may indicate that the SBM model generates graphs that model some of the Homeless networks very well.

These experiments suggest that using synthetic graphs when we don't have access to real graph datasets can be very effective. However, we need to be careful in selecting the model for generating synthetic graphs. For instance, using SBM to generate graphs for Retweet network gives drastically poor scores compared to other DQN models as shown in table IV.

Train graphs	damascus	israel
CHANGE	95.24	30.6
copen+occupy	110.8	43.6
SSM graphs	116.7	37.4
Real + SSM graphs	119.3	42.3
SBM Graphs	98.4	32.7
Real + SBM Graphs	104.2	41.9

Table IV: Comaprison of *influence score* for synthetic graphs generated by SBMs and SSMs on retweet networks.

VIII. Insights on Policy Learnt

We observe some of the characteristics of the policy learnt by the DQN models and investigate some interpretable quirks of the policies that lead to improvement over CHANGE.

A. Size of the discovered graph

Appendix Tables IX and XI shows the size of the discovered sub-graph using different DQN policies (averaged over 100 runs). In general, we note that the DQN policies almost always discover larger graphs than CHANGE though there is no correlation between influence scores and size of discovered graphs by different DQN policies. Therefore, we further explore the properties of the nodes selected by the policy in the next subsection.

B. Observations on node selection

We observed two recurring events, labelled *O1* and *O2*, during deployment on test networks:
O1: The next node to be selected from current sub-graph has *minimum* degree in the sub-graph.
O2: The next node selected in time step t is from set of nodes discovered only in previous step $t-1$.

We found that almost all the time, if *O2* occurs, *O1* also does. We summarize the frequency of both observations in Table V.

Graph	O1	O2
rural3	0.88	0.5
rural4	0.76	0.31
voles3	0.66	0.32
voles4	0.85	0.31
damascus	0.86	0.44
israel	0.87	0.28
b1	0.93	0.44
spy2	0.83	0.58

Table V: Fraction of queries conforming to observations *O1* and *O2*

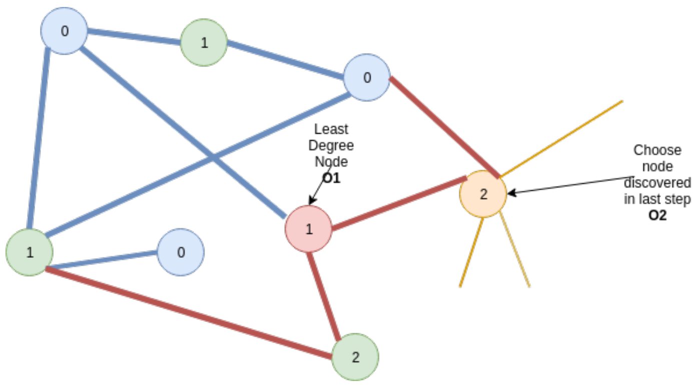


Figure 4: A toy example demonstrating observations *O1* and *O2*

a) Heuristics based on observations: To verify that the behaviours $O1$ and *O2* were beneficial for our task, we devised two heuristics.
H1: *At each step query only from the nodes with* minimum *degree in current sub-graph*
H2: *At each step query only from the nodes with minimum degree from the set of node discovered in the* previous *step. If no new nodes are discovered in the previous step, choose any node not queried in the discovered graph.*

We break all ties by choosing uniformly at random. The performance of the heuristics is summarized in Table VI.

Graph	CHANGE	H1	H2	Best-DQN
rural3	17.4	17.36	17.3	18.75
rural4	32.4	32.39	32.6	35.7
voles3	33.7	38.7	39.6	45.8
voles4	58.9	61.9	72.8	80.2
damascus	95.2	93.7	104.9	119.3
israel	30.6	30.37	34.2	43.6
b1	19.1	19.0	19.4	20.2
spy2	16.01	15.877	16.4	17.4

Table VI: Comparisons of scores of heuristics with baselines and best of DQN models for each graph

We observe that H2 outperforms CHANGE for Animals, Homeless and Retweet networks whereas H1 performs similar to CHANGE in all networks. However, the DQN models still perform much better than heuristics. This indicates that the model learns more complex patterns than the simple heuristics we designed.

b) Properties of selected nodes: To further investigate why the heuristics and our DQN policy performs better, we look at **degree centrality** and **betweenness centrality** of the nodes queried in the true underlying graph, (including the nodes and edges not discovered yet).

We call that betweenness and degree centrality of a node computed on the true graph as its *true betweenness centrality* and *true degree centrality* respectively.

Picking nodes with high true degree centrality allows access to a larger number of nodes during discovery. Betweenness centrality is an important measure of centrality of nodes in transportation systems, biological networks and social networks [6]. For network discovery, nodes of high true betweenness centrality could act as a bridge between different strongly connected communities of nodes for further exploration. In relation to influence maximization, nodes with true high betweenness centrality can allow the flow of information between parts of the network which would otherwise be hard to access.

In particular, we study three networks: *b1, rural4* and *israel*. We compare the true degree centrality and true betweenness centrality of queried nodes using CHANGE, DQN model and the heuristics discussed above.

As we can see from Figure 5, the DQN model can recognize nodes with high full degree centrality and full betweenness centrality. For graphs *b1* and *israel*, H2 also picks nodes with higher full betweenness centrality than CHANGE but we don't see much difference in degree of nodes picked with respect that picked by CHANGE. Perhaps the reason for why both H1 and H2 didn't outperform CHANGE on *rural4* and why H1 didn't outperform CHANGE on *israel* is related to this trend of picking lower full betweenness centrality nodes.

For *b1* and *israel* we further investigate how full degree and betweenness centrality vary across timesteps for H1, H2 and Best DQN model (see Figure 6).

We observe that on average, DQN model finds nodes of high full betweenness centrality and degree centrality, especially in the last query. This may indicate that DQN has leveraged the Deepwalk embeddings as well as the graph embeddings

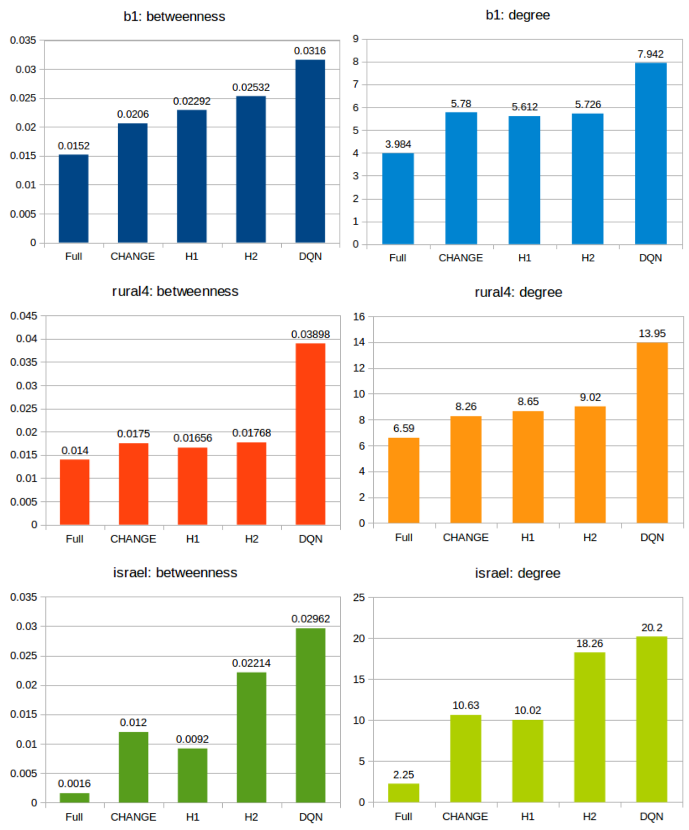


Figure 5: Average full betweenness (R) and degree(L) centrality of Full graph, nodes queried by CHANGE, H1, H2 and Best DQN

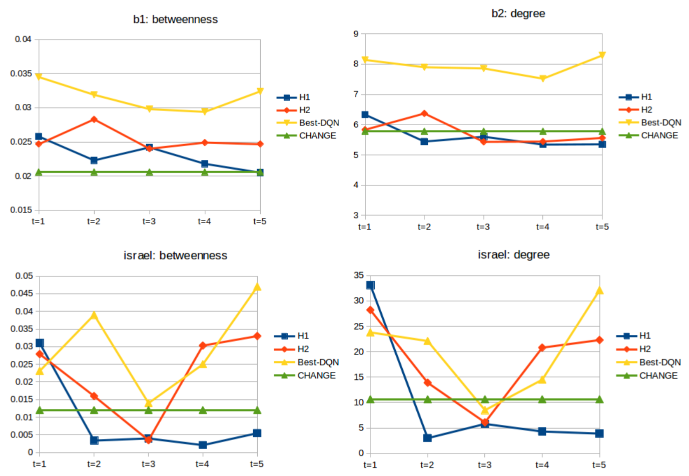


Figure 6: Average full betweenness(L) and degree(R) centrality across timesteps for H1,H2 and DQN. (We have added corresponding CHANGE values for reference)

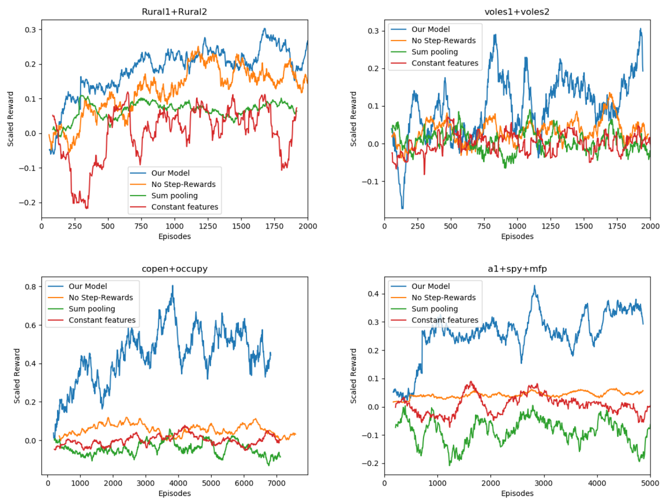


Figure 7: Reward curves for variants of models with one of the component discussed in Section IX removed

learned during training to find complex higher-order patterns in the graphs that enable it to find such nodes.

IX. Ablation Studies

From the discussion is Section V-B, we note that the important neural architecture components that were essential in encoding the network structure and node properties are Deepwalk embeddings and Differential pooling layers. We also added step-rewards (see Section V-A) to encourage the agent to discover larger graphs.

In this section, we investigate the importance of each of these components for training the model. We consider three variants of our model. In the first variant, we don't provide step rewards. In the second variant, we use sum pooling rather that differential pooling layer. In the third variant, we provide constant numbers in the range $[0, 1]$ as node features instead of using Deepwalk embeddings.

We show the reward curves in figure 7 for three of our experiments where we trained on all of the training networks for each network family. We note that in most cases the reward curve doesn't increase for architectures except our model. We conclude that in most settings all three components are vital to learning a stable policy.

X. Discussion and Conclusion

We are the first to have proposed a deep learning based method to leverage structural properties of the available social networks to learn effective policies for the network discovery problem for influence maximization on undiscovered social networks using deep Q-learning based approach.

Even if we don't have real networks to train on, we can leverage information about structural properties of the networks and create synthetic graphs that can serve as a robust alternative to training on real-world graphs or can be used along with training graphs as data augmentation technique. We showed that our trained models outperform current state-of-the-art algorithms on social networks across different domains. We observed 10-36% improvement over CHANGE (a state-of-art sampling algorithm).

Our Q-learning based model uses the influence score on the entire network as the reward signal to learn the policy. One interesting direction of research is to explore other problems that involve discovery that we can apply our model to by tweaking the reward function.

We have discussed some high-frequency behaviors common to all the policies discovered form different networks and discovered heuristics that perform better than CHANGE for some family of networks (See Section VIII). We have also observed the graph embeddings learned by our models was used to pick nodes with high betweenness centrality with respect to the entire network, which was key to discovering important portions of the social network.

References

[1] Abhijit Banerjee et al. "The diffusion of microfinance". In: *Science* 341.6144 (2013), p. 1236498.

[2] Vincent D Blondel et al. "Fast unfolding of communities in large networks". In: *Journal of statistical mechanics: theory and experiment* 2008.10 (2008), P10008.

[3] Wei Chen, Chi Wang, and Yajun Wang. "Scalable influence maximization for prevalent viral marketing in large-scale social networks". In: *KDD*. 2010, pp. 1029–1038.

[4] Stephen Davis et al. "Spatial analyses of wildlife contact networks". In: *Journal of the Royal Society Interface* 12.102 (2015), p. 20141004.

[5] Scott L Feld. "Why your friends have more friends than you do". In: *American Journal of Sociology* 96.6 (1991), pp. 1464–1477.

[6] Robert A. Hanneman. "Introduction to Social Network Methods". In: 2001. Chap. 10.

[7] Paul W Holland, Kathryn Blackmond Laskey, and Samuel Leinhardt. "Stochastic blockmodels: First steps". In: *Social networks* 5.2 (1983), pp. 109–137.

[8] Kyomin Jung, Wooram Heo, and Wei Chen. "Irie: Scalable and robust influence maximization in social networks". In: *ICDM*. 2012, pp. 918–923.

[9] David Kempe, Jon Kleinberg, and Éva Tardos. "Influential nodes in a diffusion model for social networks". In: *International Colloquium on Automata, Languages, and Programming*. Springer. 2005, pp. 1127–1138.

[10] David Kempe, Jon Kleinberg, and Éva Tardos. "Maximizing the spread of influence through a social network". In: *KDD*. 2003, pp. 137–146.

[11] Thomas N Kipf and Max Welling. "Semi-supervised classification with graph convolutional networks". In: *arXiv preprint arXiv:1609.02907* (2016).

[12] Jure Leskovec et al. "Community structure in large networks: Natural cluster sizes and the absence of large well-defined clusters". In: *Internet Mathematics* 6.1 (2009), pp. 29–123.

[13] Yuchen Li et al. "Influence maximization on social graphs: A survey". In: *IEEE Transactions on Knowledge and Data Engineering* 30.10 (2018), pp. 1852–1872.

[14] Volodymyr Mnih et al. "Human-level control through deep reinforcement learning". In: *Nature* 518.7540 (2015), p. 529.

[15] George Panagopoulos, Michalis Vazirgiannis, and Fragkiskos D Malliaros. "Influence Maximization via Representation Learning". In: *arXiv preprint arXiv:1904.08804* (2019).

[16] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. "Deepwalk: Online learning of social representations". In: *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM. 2014, pp. 701–710.

[17] Ryan A. Rossi and Nesreen K. Ahmed. "The Network Data Repository with Interactive Graph Analytics and Visualization". In: *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*. 2015. URL: http://networkrepository.com.

[18] Tom Schaul et al. "Prioritized Experience Replay". In: *CoRR* abs/1511.05952 (2016).

[19] Youze Tang, Xiaokui Xiao, and Yanchen Shi. "Influence maximization: Near-optimal time complexity meets practical efficiency". In: *SIGMOD*. ACM. 2014, pp. 75–86.

[20] Thomas W Valente and Patchareeya Pumpuang. "Identifying opinion leaders to promote behavior change". In: *Health Education & Behavior* 34.6 (2007), pp. 881–896.

[21] Christopher JCH Watkins and Peter Dayan. "Q-learning". In: *Machine learning* 8.3-4 (1992), pp. 279–292.

[22] Bryan Wilder et al. "End-to-end influence maximization in the field". In: *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*. International Foundation for Autonomous Agents and Multiagent Systems. 2018, pp. 1414–1422.

[23] Bryan Wilder et al. "Maximizing influence in an unknown social network". In: *Thirty-Second AAAI Conference on Artificial Intelligence*. 2018.

[24] Amulya Yadav et al. "Bridging the Gap Between Theory and Practice in Influence Maximization: Raising Awareness about HIV among Homeless Youth." In: *IJCAI*. 2018, pp. 5399–5403.

[25] Bo Yan et al. "On the Maximization of Influence Over an Unknown Social Network". In: *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems*. International Foundation for Autonomous Agents and Multiagent Systems. 2019, pp. 2279–2281.

[26] Zhitao Ying et al. "Hierarchical graph representation learning with differentiable pooling". In: *Advances in Neural Information Processing Systems*. 2018, pp. 4800–4810.

Graph	Nodes	Edges	Average degree	Average betweenness	Louvian modularity	OPT	CHANGE	RANDOM-GREEDY	Snowball	Recommend
damascus	3052	3869	2.53	0.00135	0.784	195.4	95.24+-22.4	**98.45+-46**	55.85+-21.9	51.24+-25.7
israel	3698	4165	2.25	0.0016	0.87	115.2	30.6+-6.1	**30.9+-11.5**	21.0+-8.3	22.63+-11.0
rural3	203	410	4.04	0.0165	0.677	25.2	**17.4+-2.7**	17.1+-1.5	14.48+-1.3	15.11+-1.3
rural4	204	672	6.59	0.01	0.496	45.4	**31.5+-1.2**	30.9+-2.8	14.68+-1.5	15.25+-1.6
voles3	1686	4623	5.48	0.003	0.786	110.6	**33.7+-9.1**	32.38+-1.3	31.64+-8.1	33.89+-9.5
voles4	1218	3592	5.89	0.048	0.773	115.7	**58.9+-18.2**	56.5+-19.2	41.72+-22.3	45.93+-23.4
mfp2	182	263	2.89	0.018	0.765	20.56	14.6+-1.1	**14.8+-1.0**	12.69+-1.6	13.14+-1.5
mfp3	233	368	3.16	0.01	0.748	23.45	16.5+-1.3	**16.6+-1.4**	14.97+-1.8	15.31+-1.96
spy2	117	234	4	0.024	0.677	21.26	**16.01+-1.2**	15.27+-1.5	14.15+-1.9	15.13+-1.84
spy3	118	237	4.017	0.187	0.685	21.71	**16.09+-1.5**	15.88+-0.96	14.97+-2.1	15.60+-2.3
b1	188	375	3.98	0.015	0.626	24.7	**19.1+-1.4**	17.4+-1.4	15.66+-1.5	16.11+-1.92
node4	95	123	2.58	0.02	0.768	15.84	12.85+-1.4	**13.2+-0.7**	11.40+-0.45	11.59+-0.58
cg1	127	174	2.74	0.0121	0.801	17.02	**14.17+-0.7**	13.9+-0.96	11.86+-1.0	12.16+-1.0

Table VII: Some properties of networks and baselines' scores

Train\Test	rural3	rural4		voles3	voles4		damascus	israel
CHANGE	17.4	31.5	CHANGE	33.7	58.9	CHANGE	95.24	30.6
rural1	**18.95**	33.1	voles1	42.2	75.3	copen	107.48	36.3
rural4	18.1*	35.2	voles2	45.3	75.5	occupy	106.2	38.7
rural1+rural2	18.2	34.1	voles1+voles2	45.11	78.9	copen+occupy	110.8	**43.6**
Syn only	18.72	34.2	Syn only	45.2	77.3	Syn only	116.7	37.4
Real + Syn	18.49	**35.7**	Real + Syn	**45.8**	**80.2**	Real + Syn	**119.3**	42.3

Table VIII: *influence scores* of all experiments on Rural, Animal and Retweet networks(starred* values were not statistically significant using t-test with $p < 0.01$)

Train\Test	rural3	rural4	Train\Test	voles3	voles4		damascus	israel
CHANGE	34.0, 40.2	51.3, 63.1	CHANGE	48.4,52.2	71.2,82.4	CHANGE	355.9,351	149.3,145.1
rural1	**39.8,42**	64.5,77.6	voles1	73.5,82.7	80.7,97.0	copen	372.0,370.2	153.8,149.1
rural4	42.7,44.5	65.3,76.7	voles2	52.4,82.8	92.8,113.2	occupy	373.6,372.65	159.8,155.0
rural1+rural2	38.2,40.5	68.6,84.1	voles1+voles2	70.7,76.1	83.6,95.5	copen+occupy	389.9,384.6	**181.1,168.8**
Syn only	38.4,49.3	67.5,79.8	Syn only	72.2,74.1	88.5,104.2	Syn only	410.7, 409.5	173.5,168.7
Real + Syn	38.9,40.4	**75.1,85.5**	Real + Syn	**71.4,77.4**	**90.0,109.1**	Real + Syn	**420.5,425.5**	212.5,209.7

Table IX: Size of discovered graph. (No. of vertices, No. of edges). Entries with maximum influence scores are bold and entries with largest no. of vertices are underlined)

Train\Test	b1	cg1	node4	mfp2	mfp3	spy2	spy3
CHANGE	19.1	14.17	12.85	14.6	16.5	16.01	16.09
a1	19.6	14.62	13.63	15.2	17.1	17	**18.2**
spy	19.2*	14.53	13.67	15.3	16.5*	18	17.6
mfp	19.3*	14.43	13.56	15.4	17.4	19	17.8
Train	19.9	14.8	**13.9**	15.37	17.76	20	18.1
Train+Synth	**20.2**	**14.98**	13.83	**15.95**	**17.7**	21	18
Synth	19.4*	14.77	13.7	15.6	17.52	**22**	17.7

Table X: *influence scores* of all experiments on Homeless Networks (starred* values were not statistically significant using t-test with $p < 0.01$)

Train\Test	b1	cg1	node4	mfp2	mfp3	spy2	spy3
CHANGE	39.28,40.4	26.45,26,6	23.84,23.7	28.3,28.1	33.1,33.2	32.8,40.6	34.7,42.1
a1	38.7,40.3	30.23,28.9	28.5,28.1	34.9,35.0	36.3,36.8	37.1,45.3	**40.3,51.8**
spy	43.7,44.5	31.0,30.5	29.1,27.6	36.2,36.8	34.8,35.6	36.8,46.6	36.8,46.2
mfp	41.2,44.2	31.5,29.5	28.9,28.2	33.9,34.2	37.5,38.8	39.1,46.7	36.7,45.6
Train	43.1,42.7	30.2,39.4	**26.2,25.8**	33.8,32.7	39.6,40.1	38.2,41.0	37.2,43.7
Train+Synth	**44.5,47,3**	**29.5,33.1**	27.2,27.8	**36.2,38.9**	**36.1,38.2**	39.5,44.2	38.1,42.9
Synth	40.9,42.5	34.8,34.5	29.8,30.3	38.6,37.5	35.1,35.3	**39.2,46.4**	35.9,40.8

Table XI: Size of discovered graph. (No. of vertices, No. of edges). Entries with maximum influence scores are bold and entries with largest no. of vertices are underlined)