Network discovery using Reinforcement Learning

Harshavardhan Kamarthi[*] **Priyesh Vijayan**[*†] **Bryan Wilder**[‡] **Balaraman Ravindran**[*†]

Milind Tambe[§]

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 to sample nodes in a carefully constructed order and choose opinion leaders from this discovered network to maximize influence spread in the complete network.

In this work, we propose a deep reinforcement learning framework graph neural network modules 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 learned policy can be directly applied on unseen graphs of similar domain. We experiment with real-world social networks and show that the policies learned by our RL agent provide a 7-23% improvement over the current state-of-the-art method.

1 Introduction

Real-world applications of influence maximization are often limited by the high cost of collecting network data. In particular, we are motivated in particular by the problem of using influence maximization for HIV prevention among homeless youth [Yad+18; Wil+18a] where gathering the social network of the youth who frequent a given homeless centre requires a week or more of effort. 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. Wilder et al. [Wil+18b] propose an algorithm motivated by community structure and prove theoretical guarantees for graphs drawn from the stochastic block model. Later, [Wil+18a] introduced a more practical algorithm called CHANGE based on the friendship paradox [Fel91]. It uses a simple yet powerful sampling method: *For each of the random seeds, we query one of its neighbors picked at random.*

We take a reinforcement learning approach to solve the network discovery problem and propose a graph-based neural network architecture to learn important graph properties from training dataset via Deep Q-learning algorithm that in turn helps it to query nodes efficiently for network discovery at deployment. By leveraging structural information from such datasets, our approach learns policies which can be deployed on unseen networks from similar domain.

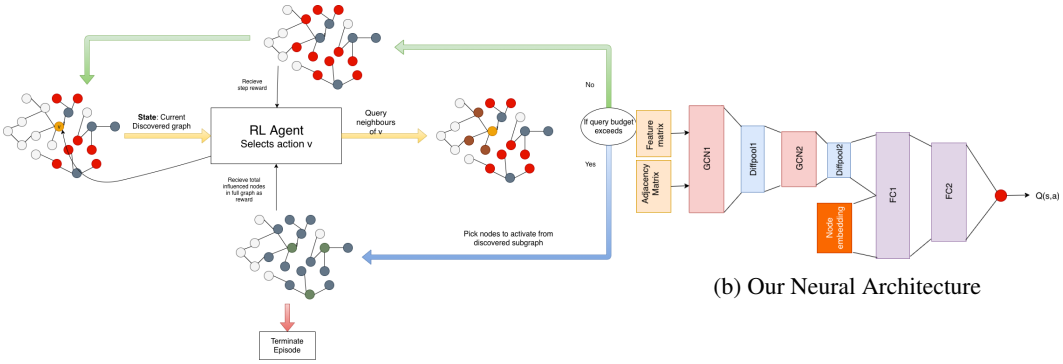
[*]Dept. of Computer Science and Engineering, Indian Institute of Technology Madras
[†]Robert Bosch Centre for Data Science and AI, Indian Institute of Technology Madras
[‡]Center for Artificial Intelligence in Society, University of Southern California
[§]Center for Research on Computation and Society, Harvard University

33rd Conference on Neural Information Processing Systems (NeurIPS 2019), Vancouver, Canada.

2 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 and $E(G)$ be edge set of G. Let $N_G(u)$ be neighbors of vertex u in a graph G, and $E(X, Y)$ be all edges such that it connect a node in X and a node in Y.

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 $G_0 = (S \cup N_{G^*}(S), E(S, N_{G^*}(S)))$ with S as the initial seed set. During the $t+1$ query, we choose a node u_t from G_t and observe $G_{t+1} = (V_t \cup N_{G^*}(u_t), E(G_t) \cup E(N_{G^*}(u_t), \{u_t\}))$. For any observed graph, we can use a standard influence maximization algorithm (which assumes that the graph is known) as an oracle to determine the best set of nodes to activate based on the available information. Let $\mathcal{A} = \mathcal{O}(G)$ be the output of this oracle on graph G, and 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 $(u_0, u_1, \ldots, u_{T-1})$ 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^*.



(a) Problem overview

(b) Our Neural Architecture

Figure 1: Problem setup and reinforcement learning model

3 Proposed Method

3.1 Information flow model

First we discuss the mechanics of information diffusion across the network. We assume that information flow is modelled by independent cascade model [KKT05]. We assign the same parameters to the model as done in [Wil+18c]. Hence, we fix the diffusion probabilities for all edges as 0.1. We fix the budget for the number of nodes to be activated at 10. Similar to setting in [Wil+18c], 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 ($|\mathcal{R}| = T$).

3.2 A Markov Decision Process Formulation

We formulate the sequential decision task of network discovery problem discussed in Section 2 as a MDP as follows.

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.

Using multiple graphs: 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 a baseline, the state of the art algorithm CHANGE [Wil+18a] and normalize this difference with respect to the difference between the performance of CHANGE and a lower-bound on optimal performance on the training graph.

$$R_s = \frac{I_{G^*}(\mathcal{O}(G_T)) - CHANGE(G^*)}{OPT(G^*) - CHANGE(G^*)} \tag{1}$$

where $CHANGE(G*)$ is the average number of influenced nodes when the graph is sampled using CHANGE [Wil+18a] and OPT is the number of influenced nodes when we select the active nodes given the knowledge of entire graph. 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:

$$R_{p,t} = \frac{|V(G_t)| - |V(G_{t-1})|}{|V(G^*)|} \tag{2}$$

3.3 Graph representation

In our RL setup, the state 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 [Yin+18] to obtain graph representations. In this work, we use the formulation in [KW16] for GCN layers. While GCNs refine node features by message passing and aggregating, differential pooling seeks to learn a global representation of the graph by aggregating node features in a hierarchical manner. Differentiable pooling (Diffpool) [Yin+18] learn hierarchical representations of the graph, G in an end-end differentiable manner 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.

3.4 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 use Graph convolutional layers and differentiable pooling to extract useful vector representations that learn to encode the structural properties of the graph. The actions, which are nodes of the networks yet to be queried, need to be represented as vectors as well.

We use a Diffpool based neural architecture shown in Figure 1b to obtain graph representation. We used DeepWalk embeddings [PAS14] as node features for input layer. We also utilized these embeddings for representing nodes as action input in Q-network.

3.5 Model training and deployment

For training, we can use single or multiple graphs. At start of an episode, we sample a graph from training set. At every time-step, we choose the node to query by feeding the graph representation of discovered graph G_t and node representations of unqueried nodes to select the node v_t with maximum predicted Q-value. On recieving the reward we store the experience in replay buffer and use it for gradient update of all weights in network including GCN and Diffpool layers. The deployment algorithm is similar to training algorithm except we freeze the weights of trained Q-network. The pseudocode of training algorithm is given in supplementary material.

3.6 Datasets

We evaluate the effectiveness of our proposed model on datasets from four different domains: 1) Rural Networks, 2) Retweet Networks, 3) Animal Interaction Networks, 4) Homeless Networks For each of the 3 families of networks, we divide them into train and test data as shown in Table 1a.

4 Results

The performance metric we used is *improve percent*. It is the percentage reduction with respect to the gap between OPT and CHANGE (our baseline). (This is same as the scaled reward for last step of episode (Equation 1)). The policies learned through reinforcement learning by our agent results in a significant increase in the number of nodes influenced as shown in Table 1b.

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

(a) Train and test split for different sets of networks

Network Family	improve %
Rural	23.76
Animal	26.6
Retweet	19.7
Homeless	7.91

(b) Summary of best results (Scores are averaged test networks for each class)

Table 1: Experiment tables

Next, 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.

We found that training with multiple networks gives better performance gains compared to average performance gains received by training on single networks. We also used synthetic graphs generated from extracting properties of training graphs(refer to supplementary for more details). Synthetic graphs can be useful substitutes when we don't have access to real-world social networks. They can be used for data-augmentation as well. We saw that the *improve percent* scores for models using only synthetic graphs are better than the average of the scores of models trained from individual networks. For Homeless, Retweet and Rural networks, data augmentation improved the performance over the model trained on only training graphs. The results are summarized in plots from Figure 2.

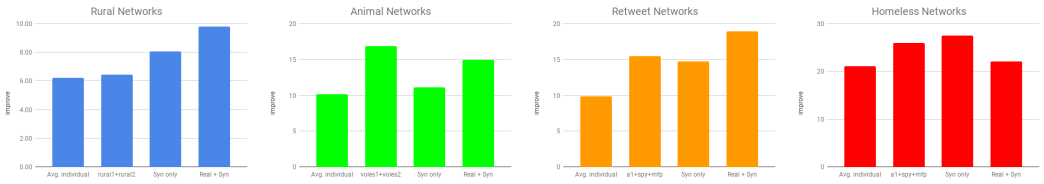


Figure 2: *improve percent* using different combinations of training datasets.(Avg. Individual: Average score from policies learned from one of the training graphs. Syn only: Only synthetic graphs for training. Real+Syn: Use both synthetic and actual train graphs)

Features of policies learnt Firstly, we observed that the DQN policies almost always discover larger graphs than CHANGE does. The DQN policy tends to pick nodes of higher *degree centrality* and *betweenness centrality with respect to the full graph* compared to CHANGE, especially during later stages of the episode (during steps 4 and 5).

5 Conclusion

We introduce a novel deep Q-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. An interesting direction for future work is to explore applications of our method to other network discovery problems to by altering the reward function.

References

[Ban+13] Abhijit Banerjee et al. "The diffusion of microfinance". In: *Science* 341.6144 (2013), p. 1236498.

[Blo+08] Vincent D Blondel et al. "Fast unfolding of communities in large networks". In: *Journal of statistical mechanics: theory and experiment* 2008.10 (2008), P10008.

[Dav+15] Stephen Davis et al. "Spatial analyses of wildlife contact networks". In: *Journal of the Royal Society Interface* 12.102 (2015), p. 20141004.

[Fel91] Scott L Feld. "Why your friends have more friends than you do". In: *American Journal of Sociology* 96.6 (1991), pp. 1464–1477.

[Han01] Robert A. Hanneman. "Introduction to Social Network Methods". In: 2001. Chap. 10.

[HLL83] Paul W Holland, Kathryn Blackmond Laskey, and Samuel Leinhardt. "Stochastic blockmodels: First steps". In: *Social networks* 5.2 (1983), pp. 109–137.

[KKT03] David Kempe, Jon Kleinberg, and Éva Tardos. "Maximizing the spread of influence through a social network". In: *KDD*. 2003, pp. 137–146.

[KKT05] 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.

[KW16] Thomas N Kipf and Max Welling. "Semi-supervised classification with graph convolutional networks". In: *arXiv preprint arXiv:1609.02907* (2016).

[Les+09] 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.

[PAS14] 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.

[RA15] 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`.

[Wil+18a] 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.

[Wil+18b] Bryan Wilder et al. "Maximizing influence in an unknown social network". In: *Thirty-Second AAAI Conference on Artificial Intelligence*. 2018.

[Wil+18c] Bryan Wilder et al. "Maximizing influence in an unknown social network". In: *AAAI*. 2018, pp. 4743–4750.

[Yad+18] 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.

[Yin+18] Zhitao Ying et al. "Hierarchical graph representation learning with differentiable pooling". In: *Advances in Neural Information Processing Systems*. 2018, pp. 4800–4810.

Supplementary Material

A Influence Model

To model information diffusion over the network, we use the standard independent cascade model (ICM) [KKT03], 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.

B Network Datasets

Here, we describe in detail, the datsets used in our work.

Rural Networks We used the networks gathered by [Ban+13] 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 relations such as health, finance, family, friendship, etc. For our experimental study, we considered four such networks (*rural1-4*).

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 [RA15] [5] 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.

Animal Interaction Networks These networks are a part of the wildlife contact networks collected by [Dav+15] 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*).

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 [Wil+18a; Wil+18b]. We considered ten of these networks for our experiments: *a1, b1, cg1, node4, mfp2, mfp3, spy2, spy3*.

[5] `http://networkrepository.com/`

Graph	**Nodes**	**Edges**	**Average degree**	Average betweenness	Louvian modularity
damascus	3052	3869	2.53	0.00135	0.784
israel	3698	4165	2.25	0.0016	0.87
rural3	203	410	4.04	0.0165	0.677
rural4	204	672	6.59	0.014	0.496
voles3	1686	4623	5.48	0.003	0.786
voles4	1218	3592	5.89	0.048	0.773
mfp2	182	263	2.89	0.018	0.765
mfp3	233	368	3.16	0.01	0.748
spy2	117	234	4	0.024	0.677
spy3	118	237	4.017	0.187	0.685
b1	188	375	3.98	0.015	0.626
node4	95	123	2.58	0.02	0.768

Table 2: Some properties of networks

C Training Algorithm

Algorithm 1: Train Network

Input : Train Graphs $\mathcal{G} = \{G_1, G_2, \ldots, G_K\}$, number of episodes N, Query budget T, number of random seeds $|S|$

Initialize DQN Q_θ and target DQN $Q_{\theta'}$;
Initialize Prioritized Replay Buffer B;
for *episode = 1 to* N **do**
  Choose a graph G from $\mathcal{G}$;
  Select random nodes from G as S;
  $V_0 = S \cup N_G(S)$;
  Initial graph is $G_0 = G[V_0]$;
  Compute Deepwalk node embeddings for G_0 as ϕ;
  Get feature matrix F_0, adjacency matrix A_0 for G_0 as $S_0 = (F_0, A_0)$;
  $X \leftarrow N(S)$;
  for *t = 0 to* $T - 1$ **do**
    With probability ϵ select a random node v_t from X else select node $v_t \leftarrow \underset{v \in X}{argmax} Q_\theta(S_t, \phi(v))$;
    Query node v_t and observe new graph G_{t+1};
    Set $R \leftarrow R_{p,t}$ (Eqn. 2).;
    If $t = T - 1$, $R \leftarrow I_{G^*}(\mathcal{O}(G_T)) + R_{p,t}$;
    Compute scaled influence reward R_t (Eqn. 1);
    Compute Deepwalk node embeddings for G_{t+1} as ϕ;
    Get feature matrix F_t, adjacency matrix A_{t+1} for G_{t+1} as $S_{t+1} = (F_{t+1}, A_{t+1})$;
    $X \leftarrow$ nodes not yet queried in G_{t+1};
    Add $(S_t, \phi(v_t), R_t, S_{t+1})$ to replay buffer D;
    Sample from B and update Q_θ;
  end
  Update target network $Q_{\theta'}$ with parameters of Q_θ from time to time;
end

D Deployment algorithm

Algorithm 2: Deploy Network

Input : Trained network Q_θ, Graph to be deployed on G, initial random seeds S, query budget T

$V_0 = S \cup N_G(S)$;
Get the initial graph $G[V_0]$;
Compute Deepwalk node embeddings for G_0 as ϕ;
Get feature matrix F_0, adjacency matrix A_0 for G_0 as $S_0 = (F_0, A_0)$;
$X \leftarrow N(S)$;
for *t = 0 to* $T - 1$ **do**
    Select note $v_t \leftarrow \underset{v \in X}{argmax} Q_\theta(S_t, \phi(v))$;
    Query node v_t and observe new graph G_{t+1};
    Compute Deepwalk node embeddings for G_{t+1} as ϕ;
    Get feature matrix F_t, adjacency matrix A_{t+1} for G_{t+1} as $S_{t+1} = (F_{t+1}, A_{t+1})$;
    $X \leftarrow$ nodes not yet queried in G_{t+1};
end
$\mathcal{A} \leftarrow \mathcal{O}(G_T)$;
Activate nodes in $\mathcal{A}$ to start the influence process;

E Other sampling methods

We describe below other sampling methods previously used for network discovery problem.

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* [Wil+18a]: 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 $\mathcal{A}$.

The performance of above methods are shown in Table 3.

F Synthetic Graph Generation

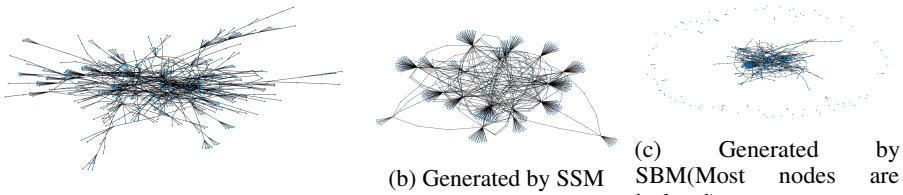
We discuss a simple graph generation technique based on the assumption that our social networks have similar structures to graphs generated by **stochastic block models**. Real-world social networks have densely connected components called *communities* [Les+09]. The nodes of the same community are tightly connected and nodes of different communities are less frequently connected by an edge. Stochastic Block Models(SBMs), which originated in sociology ([HLL83]), 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}.

Graph	OPT	CHANGE	RANDOM-GREEDY	Snowball	Recommend
damascus	195.4	95.24	**98.45**	55.85	51.24
israel	115.2	30.6	**30.9**	21.0	22.63
rural3	25.2	**17.4**	17.1	14.48	15.11
rural4	45.4	**31.5**	30.9	14.68	15.25
voles3	110.6	**33.7**	32.38	31.64	33.89
voles4	115.7	**58.9**	56.5	41.72	45.93
mfp2	20.56	14.6	**14.8**	12.69	13.14
mfp3	23.45	16.5	**16.6**	14.97	15.31
spy2	21.26	**16.01**	15.27	14.15	15.13
spy3	21.71	**16.09**	15.88	14.97	15.60
b1	24.7	**19.1**	17.4	15.66	16.11
node4	15.84	12.85	**13.2**	11.40	11.59
cg1	17.02	**14.17**	13.9	11.86	12.16

Table 3: Influence Scores using different state-of-art sampling methods

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 ([Blo+08]) 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 3). 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 3: Difference in graphs generated by SBM and SSM for retweet networks.

We need to be careful to choose models that closely resemble properties of test networks. Table 4 shows the large difference in performance when using SBM and SSM models for training on retweet networks.

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

G Insights on policy learnt: details

G.1 Size of discovered graphs

Tables 5 and 6 provide results on the size of discovered graph (number of vertices and edges) at end of an episode.

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	3.5,82.7	80.7,97.0	copen	372.0,370.2	153.8,149.1
rural4	2.7,44.5	65.3,76.7	voles2	52.4,82.8	2.8,113.2	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	12.5,209.7

Table 5: 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	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	9.6,40.1	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**	9.5,44.2	38.1,42.9
Synth	40.9,42.5	4.8,34.5	9.8,30.3	8.6,37.5	35.1,35.3	**39.2,46.4**	35.9,40.8

Table 6: 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)

G.2 Observations on nodes selected

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 7.

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 7: Fraction of queries conforming to observations *O1* and *O2*

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 8.

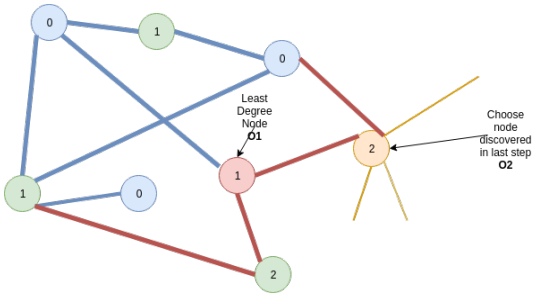


Figure 4: A toy example demonstrating observations *O1* and *O2*

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 8: 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.

G.3 Properties of nodes selected by the policy

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.

In particular, we study three networks: *b1* 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.

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. 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 ([Han01]). 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.

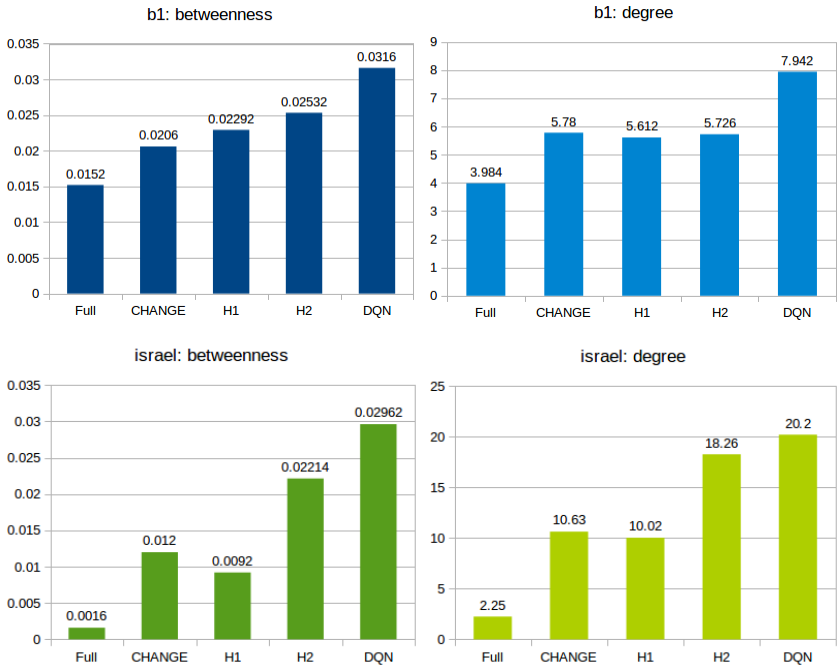


Figure 5: Average full betweenness (R) and degree(L) centrality of Full graph, nodes queried by CHANGE, H1, H2 and Best DQN

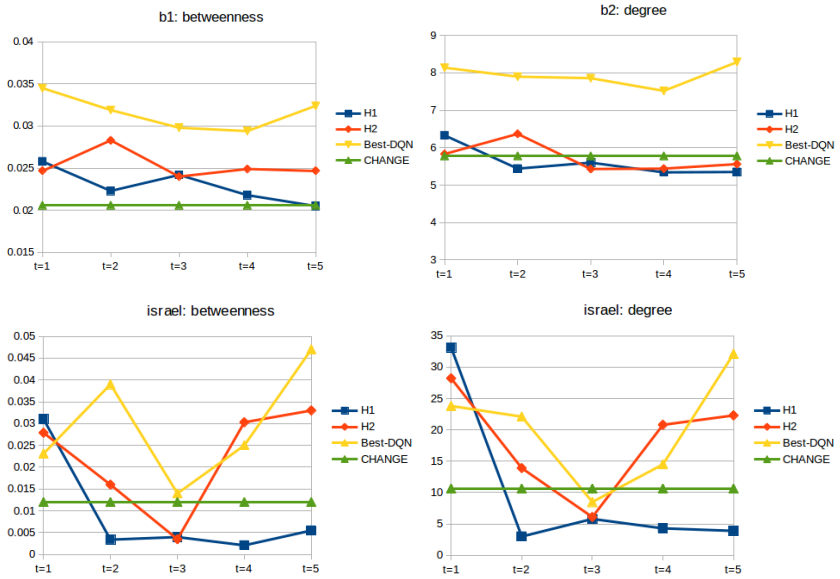


Figure 6: Average full betweenness(L) and degree(R) centrality across timesteps for H1,H2 and DQN. (We have added corresponding CHANGE values for reference)

H Influence scores for all experiments

Tables 9 and 10 show the influence scores for all experiments performed.

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 9: *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	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 10: *influence scores* of all experiments on Homeless Networks (starred* values were not statistically significant using t-test with $p < 0.01$)