


RESEARCH

Open Access



# Selective network discovery via deep reinforcement learning on embedded spaces

Peter Morales<sup>1\*</sup> , Rajmonda Sulo Caceres<sup>1</sup> and Tina Eliassi-Rad<sup>2</sup>

\*Correspondence:  
pmorales@mit.edu

<sup>1</sup> MIT Lincoln Laboratory,  
Lexington, USA

Full list of author information  
is available at the end of the  
article

## Abstract

Complex networks are often either too large for full exploration, partially accessible, or partially observed. Downstream learning tasks on these incomplete networks can produce low quality results. In addition, reducing the incompleteness of the network can be costly and nontrivial. As a result, network discovery algorithms optimized for specific downstream learning tasks given resource collection constraints are of great interest. In this paper, we formulate the task-specific network discovery problem as a sequential decision-making problem. Our downstream task is selective harvesting, the optimal collection of vertices with a particular attribute. We propose a framework, called network actor critic (NAC), which learns a policy and notion of future reward in an offline setting via a deep reinforcement learning algorithm. The NAC paradigm utilizes a task-specific network embedding to reduce the state space complexity. A detailed comparative analysis of popular network embeddings is presented with respect to their role in supporting offline planning. Furthermore, a quantitative study is presented on various synthetic and real benchmarks using NAC and several baselines. We show that offline models of reward and network discovery policies lead to significantly improved performance when compared to competitive online discovery algorithms. Finally, we outline learning regimes where planning is critical in addressing sparse and changing reward signals.

**Keywords:** Incomplete networks, Reinforcement learning, Network embedding

## Introduction

Complex networks are critical to many applications such as those in the social, cyber, and bio domains. We commonly have access to partially observed data. The challenge is to discover enough of the complex network so that we can perform a learning task well. The network discovery step is especially critical in the case where the learning task has the characteristics of the “needle in a haystack” problem. If the discovery process is not carefully tuned, the noise introduced, almost always, overwhelms the signal. This presents an optimization problem: how should we grow an incomplete network to achieve a learning objective on the network, while at the same time minimize the cost of observing new data?

In this work, we view the network discovery problem from a decision-theoretic lens, where notions of utility and resource cost are naturally defined and jointly leveraged in

a sequential, closed-loop manner. In particular, we will leverage Reinforcement Learning (RL) and its mathematical formalism, Markov Decision Processes (MDP): a general decision-theoretic model that allows us to treat network discovery as an interactive, sequential learning and planning problem. MDP approaches have been successfully used in many other application settings (Mnih et al. 2015; Heess et al. 2017; Silver et al. 2017). However, the use of decision-theoretic approaches in the context of discovery of complex networks is novel and presents very interesting research opportunities. In particular, it requires learning effective models of reward that can capture properties of network structure at various topological scales and learning contexts. The network science community has defined many such topological and task quality metrics; but, to-date, they have not been leveraged in the context of guiding the process of network discovery. We consider the task of selective harvesting on graphs (Murai et al. 2017), where the learning objective is to maximize the collection of nodes of a particular type, under budget constraints. We make the following contributions:

- We introduce a deep RL framework for task-driven discovery of incomplete networks. This formulation allows us to train models of environment dynamics and reward offline.
- We show that, for a variety of complex learning scenarios, the added feature of learning from closely related scenarios leads to substantial performance improvements relative to existing online discovery methods.
- We show that network embedding can play an important role in the convergence properties of the RL algorithm. It does so by imposing structure on the network state space and prioritizing navigation over this space.
- Among a class of embedding algorithms, we identify Pagerank (PPR) as a suitable network embedding algorithm for the selective harvesting task. Our combined approach of PPR embedding and offline planning achieves substantial reductions in training time.
- Leveraging several evaluation metrics, we delineate learning regimes where embedding alone stops being effective and planning is required.
- Our approach is able to generalize well to unseen real network topologies and new downstream tasks. Specifically, we show that policies discovered by training on synthetically generated networks translate well to detection of anomalous nodes in real-world networks.

### **Related work**

Our learning task falls under the category of finding the largest number of a particular type of node under budget constraints. The node type can be specified by node attributes (for example, males on a social network), or they can be determined by node's participation in a particular class of behavior (for example, accounts that belong to dense bipartite subgraphs in a communication network). Unlike the problem setting in Wang et al. (2013), we do not assume access to the full topology of the network and therefore have to perform the learning task with partial information.

Discovering incomplete networks with limited resources has received a lot of attention in recent literature. The primary learning objective in these works is to increase the visibility of the network topology by increasing the number of discovered nodes (LaRock et al. 2018, 2020; Soundarajan et al. 2015, 2016), increasing the number of discovered nodes of a given type (Murai et al. 2017), or increasing network coverage (Avrachenkov et al. 2014). Our problem setting is the closest to Murai et al. (2017). However, while Murai et al. (2017) leverages supervised learning to infer discovery heuristics, our approach leverages an MDP formulation of RL to estimate offline models of network discovery strategies (a.k.a. policy) and node utility (a.k.a. reward) that are network state-aware. More specifically, our approach explicitly connects the utility of a discovery choice to the network state when that choice was made. We will illustrate in later sections that learning state-dependent discovery strategies, allows our approach to stay robust in learning scenarios where nodes of interest are sparsely observed. In LaRock et al. (2018, 2020), they frame their network discovery task as an MDP, but they only consider online training of their policy. The online-only training, we will show, suffers from tunnel vision and is not able to generalize well.

Reinforcement learning for tasks on complex networks is a relatively new perspective. Work in Ho et al. (2015) and Goindani and Neville (2019) leverages RL to engineer diffusion processes in networks assumed to be fully observed, while authors in Mofrad et al. (2019) focus on the problem of graph partitioning. You et al. (2018) leverage RL to generate novel molecular graphs with desired domain-specified properties. There are connections to our problem setting. The graph generation is approached as an MDP, in a similar fashion to our network discovery problem, by iteratively expanding a seed graph via defined actions. There are, however, some important differences with our work. Their definition of reward and environment dynamics is tailored to the biochemical domain and molecular design application, which is characterized by a comparatively small state space and fixed-sized action space.

In our problem setting, the agent deals with a much larger network space and a larger and variable-sized action space. To address this increased complexity, we introduce a network embedding step that enables a more efficient navigation of the decision space. Our notion of reward is also more general, in that we do not utilize domain-specific properties to guide the learning process. Since we cannot rely on added information provided by the domain-specific properties (e.g., biochemistry), we had to carefully model ways of introducing topological diversity in our policy training data. Others (Dai et al. 2017; Mittal et al. 2019) have leveraged deep RL techniques to learn a class of greedy optimization heuristics on fully observed networks. We summarize key differences between our method and related methods in Table 1.

### **Problem definition**

We start with the assumption that a network contains a target subnetwork representing a set of relevant nodes. The decision-making agent can initially observe only part of the original network  $G_0 = (N_0, E_0)$ , where some of its nodes have their relevance status  $C_0$  revealed; 0 representing non-target nodes and 1 representing target nodes.

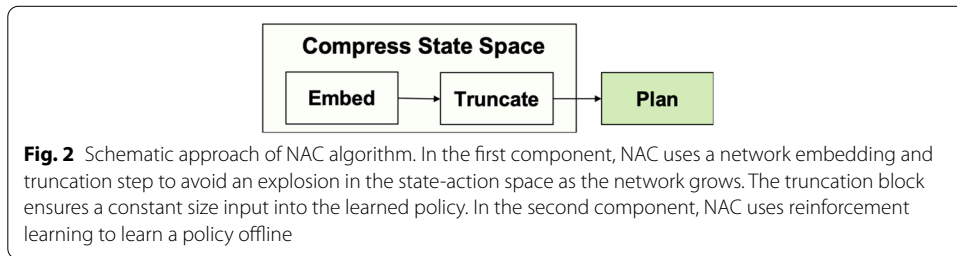
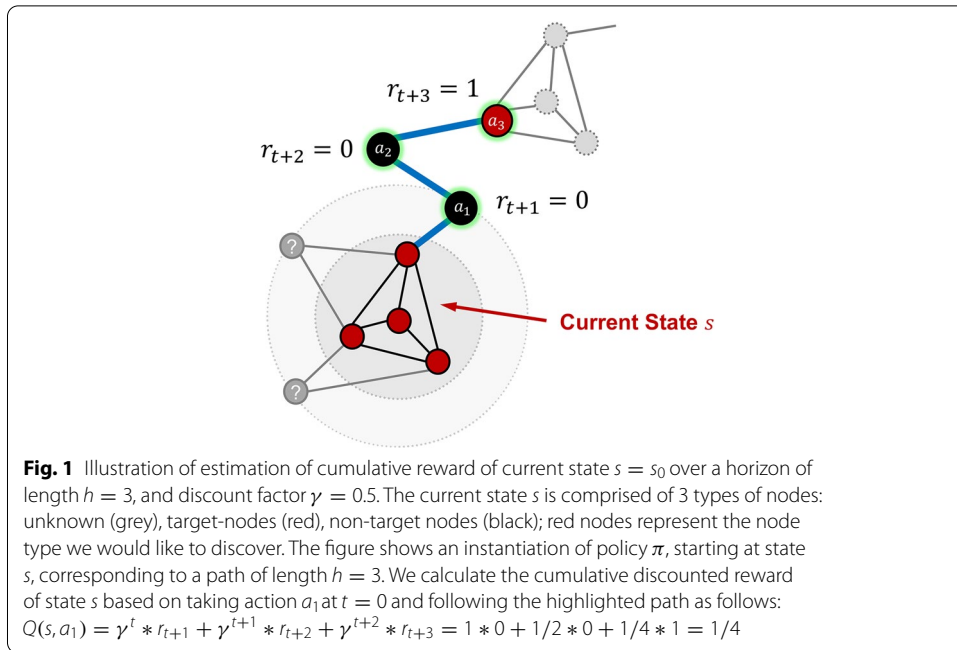
The agent has a pre-specified way by which it can interact with the partially observed network: at each step, it is allowed to query the label of one selected observed node

**Table 1** Comparison of NAC features with related methods

Method	State space	Action space	Observability	Learning goal	Learning framework	Policy training	State embedding
NOL (LaRock et al. 2018)	Large	Dynamic	Partial	Vertex property	MDP	Online	No
D <sup>3</sup> TS (Murai et al. 2017)	Large	Dynamic	Partial	Vertex property	Supervised	Online	No
GCPN (You et al. 2018)	Small	Fixed	Full	Graph property	MDP	Offline on given dataset	No
NAC	Large	Dynamic	Partial	Vertex property	MDP	Offline and online on designed dataset	Yes

whose label is unknown. We call this type of node a *boundary* node. The environment responds by revealing the label of the queried node as well as its neighboring nodes (but not their labels). An immediate reward is given if the selected (i.e., queried) node is a target node. The agent’s overall objective is then to strategically grow the original network over a sequence of steps, so that it can discover as many relevant nodes as possible under the query budget constraints. This problem may be stated as a Markov Decision Process (MDP). An MDP is defined by the tuple  $\langle \mathcal{S}, \mathcal{A}, T, r, \gamma \rangle$ :

- *State space*: Let  $\{M^i\}$  be a set of random graph models, each defining a set of network instances  $\{G^i\}$  defined over a set of nodes  $V$ . We define the state space as  $\mathcal{S} = \bigcup_{M^i} \{s_t = G_t^i\}$ , the set of partially-observed network instances  $G_t^i = \{V_t^i, E_t^i\}$  discovered at intermediate time-steps  $t$ , where  $V_t^i \subseteq V$  and  $E_t^i \subseteq E$ . Each  $v \in V_t^i$  has a label  $C(v) \in \{0, 1, *\}$ , representing non-target, target and unobserved node states, respectively.  $E_t^i$  is the set of edges induced by  $V_t^i$ . The state space includes the initial state  $G_0^i$  that contains at least one target node, as well as the terminal state which is the fully observed network instance  $G^i$ .
- *Action space*: For each network instance  $G^i$ , we define the action space as  $\mathcal{A} = \{A_t\}$ , where  $A_t = \{a = v\}$  is the set of boundary nodes  $v$ , observed at time-step  $t : \{v \in V_t, C(v) = *\}$ .
- *Transition function*:  $T(s_t, a_t, s_{t+1}) = P(s_{t+1}|s_t, a_t)$  encodes the transition probability from state  $s_t$  to  $s_{t+1}$  given that action  $a_t$  was taken. Let  $v$  be the selected node by action  $a_t$ . Then  $s_{t+1} = s_t \cup C(v) \cup N_v \cup E_{N_v}$ , where  $N_v$  are the neighbors of node  $v$  and  $E_{N_v}$  are all the edges incident to  $N_v$ . For each network instance  $G^i$ , the transition function is deterministic:  $T(s_t, a_t, s_{t+1}) = 1$ .
- *Reward function*:  $r(s_t, a_t)$  returns the reward gained by executing action  $a_t$  in state  $s_t$  and is defined as:  $r(s_t, a_t) = 1$  if  $C(a_t) = 1$ . The total cumulative, action-specific reward, also referenced as the action-value function  $Q$ , is defined as,

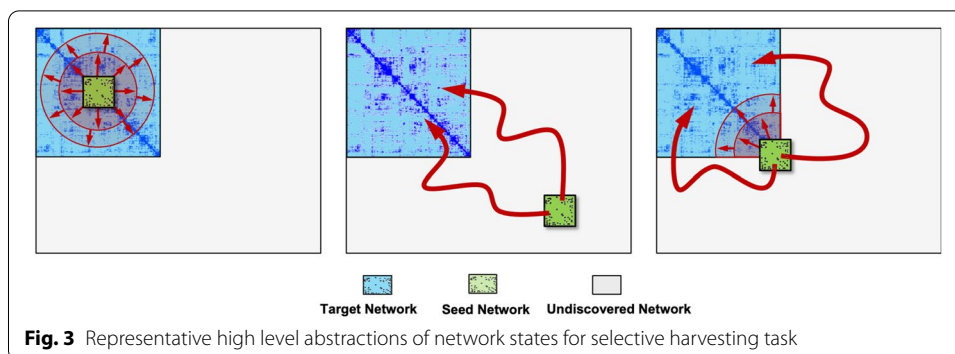


$$Q(s, a) = \left[ \sum_{t=0}^h \gamma^t r_{t+1} | s, a \right], \tag{1}$$

with  $\gamma \in [0, 1]$  representing a discount factor that captures the utility of exploring future graph states and  $h$  is the horizon length. Figure 1 gives a simple illustration of how this cumulative reward is computed over a network topology. In the next section, we describe in detail our deep reinforcement learning algorithm.

### Network actor critic (NAC) algorithm

Our network discovery algorithm has two main components as illustrated in Fig. 2. The first component, “Compress State Space”, is concerned with effective ways of representing the large network state space so that policy learning can happen efficiently relative to our selective harvesting task. The second component, “Plan”, utilizes the reinforcement learning framework and offline training to learn task-driven discovery strategies. We discuss both components in detail in the rest of this section.



### Compression of network state space

Training an effective network discovery agent implies exploration over an extremely large network space. However, not all observations contribute to learning better discovery policies. In fact, for the task of selective harvesting, we can identify three representative, higher-level abstractions of the network states illustrated in Fig. 3.

In the first canonical case, discovery starts within the region of interest, where many of the relevant nodes we need to discover are nearby. In networks whose states are similar to this canonical case, the optimal discovery agent would follow localized paths and primarily exploit rather than explore new regions. In the second canonical case, discovery starts outside the region of interest and the agent has to now explore longer, deeper paths in order to reach the target region. Finally, there is a hybrid canonical case, where discovery can start in the boundary of the target region and the agent has to more carefully decide when to exploit and when to explore.

In order to map network states into canonical representations, we consider various network embedding approaches. Specifically, we look at popular walk-based algorithms (Grover and Leskovec 2016; Taher 2003; Murai et al. 2017) and matrix-factorization algorithms (Pearson 1901; Torres et al. 2020; Belkin and Niyogi 2003). The embedding step learns a new similarity function between nodes in a network. Since our downstream task is selective harvesting from a seed node, we reorder the rows of the original adjacency matrix based on the new learned distance from the seed node, with closer nodes being ranked higher. The reordering step makes sure the discovery algorithm observes a prioritized set of boundary nodes. For additional efficiency gains, we truncate the reordered adjacency matrix and only retain the network defined by the top  $k$  nodes.  $k$ , is a hyperparameter, which the user selects; it defines the supporting network for computing potential discovery trajectories and long-term reward. For training our algorithm, we found  $k = 256$  to perform well after incrementally lowering its value from the number of nodes in the network. In practice, larger values of  $k$  can be utilized during training, but they incur a higher computational costs at no substantial increase in algorithmic performance.

In “[Role of network embedding](#)” section, we present a detailed evaluation of various embedding algorithms and identify the role that they play in supporting the planning component of NAC. Among the embedding algorithms we study, we identify personalized Pagerank (PPR) (Taher 2003) as performing the best in supporting policy

learning for selective harvesting. For the rest of the paper, we assume that the planning agent only sees the compressed state representation, that is, the state that has gone through the sequence  $e$  of operations: embed, re-order and truncate:  $s_t = e(s_t)$ .

**Offline learning and policy optimization**

In our setting, learning of discovery strategies happens offline over a training set of possible discovery paths. “NAC performance results” section describes how we generate these paths.

Each path  $\tau_h$  represents an alternating sequence of discovered subnetworks and actions  $\{(s_0, a_0), (s_1, a_1), \dots, (s_h, a_h)\}$ , taken over  $h$  steps. Since in this setting we have access to the ground-truth node labels, we can map each discovery path to the corresponding cumulative reward value using Eq. 1. An illustration is given in Fig. 1.

Given the sampled trajectories, one of our learning objectives becomes to approximate the action-value function by minimizing the loss  $L_Q(\phi)$ ,

$$L_Q(\phi) = \|y_t - Q_\phi(x_t)\|_2^2. \tag{2}$$

We formulate this objective by taking the input tuples of discovered subnetworks  $s_t$ , boundary nodes  $a_t$  and corresponding cumulative reward values  $Q_t$ , such that  $\langle x_t = (s_t, a_t), y_t = Q_t \rangle$ .

The approximated reward function  $Q_\phi$  is subsequently used to estimate the policy function  $\pi_\theta(s) = P(a|s)$ , which defines the probability of selecting action  $a$  at state  $s$ . This is achieved by training a convolutional neural network with the network embedding as input and a softmax probability as output over the action space. Actions are selected via an *argmax* over the output probabilities. The parameters of the convolutional neural network are updated via gradient ascent and its objective function is defined in Eq. 5.

We estimate the advantage of choosing one node versus another at state  $s_t$ ,

$$\hat{A}_t = Q_\phi(s_t, a_t) - \sum_{a \in \mathcal{A}} Q_\phi(s_t, a). \tag{3}$$

This advantage is used to scale the policy gradient estimator, typically defined as,  $\hat{g}_t = \hat{\mathbb{E}}_t \left[ \hat{A}_t \nabla_\theta \log \pi_\theta \right]$ , where  $\hat{\mathbb{E}}_t[\cdot]$  is the empirical average over a finite batch of samples. We utilize a proximal policy optimization (PPO) method (Schulman et al. 2017) in order to compute this gradient. PPO methods are widely utilized for policy network optimization and have been demonstrated to achieve state of the art performance on graph tasks (You et al. 2018). The objective function utilized is defined in Eq. 4,

$$L_t^{CLIP}(\theta) = \hat{\mathbb{E}}_t \left[ \min \left( \frac{\pi_\theta}{\pi_{\theta_{old}}} \hat{A}_t, \text{clip} \left( \frac{\pi_\theta}{\pi_{\theta_{old}}}, 1 - \epsilon, 1 + \epsilon \right) \hat{A}_t \right) \right]. \tag{4}$$

Here,  $\epsilon$  is used to bound the objective function and help with convergence. The function  $\text{clip}(\cdot)$  keeps the ratio  $\frac{\pi_\theta}{\pi_{\theta_{old}}}$  within  $[1 - \epsilon, 1 + \epsilon]$  (Pascanu et al. 2013). Note that when the estimated expectation in Eq. 4 is expanded  $\frac{\pi_\theta}{\pi_{\theta_{old}}}$  becomes  $\frac{\pi_\theta(s,a)}{\pi_{\theta_{old}}(s,a)}$ , where  $\pi_\theta(s, a)$  is the probability of the current policy selecting action  $a$  when in state  $s$ ; and  $\pi_{\theta_{old}}(s,a)$  is probability of the previous policy selecting the same action  $a$  when in state  $s$ .



During offline training, we modify the objective in Eq. 4 to encourage exploration and reduce the number of required training epochs to converge to a solution. For Eq. 5,  $S(\pi_\theta(s_t))$  denotes the entropy of policy  $\pi_\theta$  over actions  $a$  in state  $s_t$ .  $c$  is a fixed constant that captures the weight of the entropy term,

$$L_t^{CLIP+S}(\theta) = L_t^{CLIP}(\theta) + cS(\pi_\theta(s_t)). \tag{5}$$

Both learning objectives (2) and (5) are jointly optimized via an actor critic training framework. This framework is detailed further below in the description of the *Network Actor Critic* (NAC) algorithm. To help with training times, multiple instantiations of agents are run in parallel. Collected  $\{s_t, a_t, Q_t\}$  values are gathered from each agent and are stored in a buffer  $\beta$  which is used to compute the losses for the value function and policy networks after a fixed number of iterations  $T$ .

---

**Algorithm 1:** Network Actor Critic (NAC)

---

```

1 set hyper-parameters: learning rate  $\lambda$ , embed function  $e$ , number of iterations  $T$ , exploration
  constant  $c$ , horizon length  $h$ , number of agents  $nAgents$ ;
2 initialize: parameters  $\theta_{old}$  of policy  $\pi$ , parameters  $\phi_{old}$  of value function  $Q$ , buffer  $\beta$ ;
3 for  $t = 1, 2, \dots, T$  do
4   for  $agent = 1, 2, \dots, nAgents$  do
5      $s_t \leftarrow e(G_{t,agent})$ ;
6      $a \sim \pi_{\theta_{old}}(s_t)$ ;
7      $r \leftarrow$  take action  $v = a$  and save reward  $r$ ;
8      $s_{t+1} = s_t \cup C(v) \cup N_v \cup E_{N_v}$ ;
9      $\beta \leftarrow$  save  $(s_t, a_t, r_t, s_{t+1})$  to buffer  $\beta$ ;
10     $s_t \leftarrow s_{t+1}$ ;
11  end
12  if  $t \bmod T$  is 0 then
13    Compute batch update tuples  $\{s_t, a_t, Q_t^T\}$  over horizon  $h$  using  $\beta$ ;
14    Batch update  $\phi$  via  $\nabla_\phi L^Q(\phi_{old})$  using Eq. 2;
15     $Q_{old} = Q$ ;
16    Compute  $\hat{A}_t$  using eq. (3);
17    Batch update  $\theta$  via  $\nabla_\theta L^{CLIP+S_c}(\theta_{old})$  using Eq. 5;
18     $\pi_{old} = \pi$ ;
19  end
20 end

```

---

**Training and network details**

The NAC algorithm is updated differently during offline training versus online evaluation. During offline training, the Adam optimizer (Kingma and Ba 2014) is used to update network parameters  $\theta$  and  $\phi$  for the policy and value function networks. In offline training, eight agents are run in parallel, carrying out the anomaly discovery task each on a unique network realization generated using one of the network models outlined in Table 2. Each network model is chosen with equal probability and their parameters are drawn uniformly at random from the parameter ranges defined in Table 2. Each independent agent contributes data to a common buffer, later used for learning the parameters of the value and policy functions. The use of multiple agents helps generate more training data faster. More agents will lead to further reductions of training time.

Hyper-parameter searches were performed in a grid search manner for both offline and online values. During offline training, the hyper parameters used are:  $T = 32$ ,  $h = 4$ ,  $c = 0.2$ ,  $\epsilon = 0.1$ ,  $\gamma = 0.1$ , and learning rate  $\lambda = 1e-4$ . With the exception of  $h$ , these



**Table 2** Detailed list of parameter values used for synthetic networks

Model	Type	Parameters
SBM	Background	$k = [1, 10], p_i = [0.01, 0.4], r = [0.005, 0.25], i = 1 \dots k$
LFR	Background	$\tau_1 = [3, 2], \tau_2 = (1, 1.9), \mu = [0.1, 0.4], \langle d \rangle = [32, 256],$ $d_{\max} = [256, 2048], \min_c = [256, 1000], \max_c = [512, 2000]$
ER	Foreground	$n_f = \{30, 40, 80\}, k_f = \{1, 2, 4\}, p_f = [0.5, 1]$

Number of nodes is represented by  $N = 4000$ . SBM parameters are:  $k$  represents the number of communities,  $p_i$  the edge probability for within-community  $i$ ,  $r$  the across-community edge probability, such that  $p_i > r$ . LFR parameters are:  $\tau_1, \tau_2$  skewness parameters for degree and cluster size distributions respectively,  $\langle d \rangle$  represents the average network degree,  $d_{\min}, d_{\max}$  represent the min and max values of degree distribution,  $\min_c$  and  $\max_c$  represent the sizes of smallest and largest clusters, and finally  $n_f, k_f, p_f$  represent the size of the foreground subnetwork, number of foreground subnetworks and its edge probability, respectively

parameters generally did not affect the overall final performance of the network, but did alter the convergence rate. We found that going beyond  $h = 5$  reduced overall performance of the learned policy. For online evaluation, we used a single agent and parameters  $T = 1, h = 1, \gamma = 1, c = 0.0$ , and  $\lambda = 1e-3$ . The change in hyper-parameters, specifically removing the exploration components, reflects moving from the training setting which benefits from sufficient exploration of the search space, to a fully greedy online policy that favors exploitation. The policy and value function networks are both comprised of 3 convolutional layers with 64 hidden channels and a final fully connected layer. The value function is regressed using the loss function described in Eq. 2, while the policy network is trained using the objective function described in Eq. 5.

### NAC performance results

We evaluate our algorithm against several learning scenarios for both synthetic and realistic datasets. The NAC agent always starts the exploration from a seed subnetwork that contains 1 target node. Next we describe our datasets and baselines used for comparison.

#### Datasets

##### Synthetic datasets

We generate synthetic graphs by modeling *background* networks (i.e., networks that do not contain any target nodes), and *foreground* networks (i.e., networks that only contains target nodes). There are edges that connect the foreground and background nodes to each other.<sup>1</sup>

We use two models to generate samples of background networks. Stochastic Block Model (SBM) (Holland et al. 1983) is a commonly used random graph model, which allows us to model community structure as dense subgraphs sparsely connected with the rest of the network. Lancichinetti–Fortunato–Radicchi (LFR) model (Lancichinetti et al. 2008) is another frequently used random graph model, which allows us to simulate network samples with skewed degree distributions and skewed community sizes, and therefore is able to capture more realistic and complex properties of real networks. We use the Erdős–Renyi (ER) model (Holland et al. 1983) to simulate the foreground network. ER is a random graph model where nodes are connected with equal probability  $p_f$ . This

<sup>1</sup> In this section, we use the terms foreground and target (sub)networks interchangeably.

**Table 3** Characteristics of the real networks and corresponding target classes

Name	# Nodes	# Edges	Target type	Target size
Facebook politician	5908	41,729	Synthetic	80
Facebook TV shows	3892	17,262	Synthetic	80
Livejournal	≈ 4000 k	≈ 35,000 k	Real	≈ 1400

allows us to control the density of the foreground networks. Table 2 lists the parameter choices for all the aforementioned models.

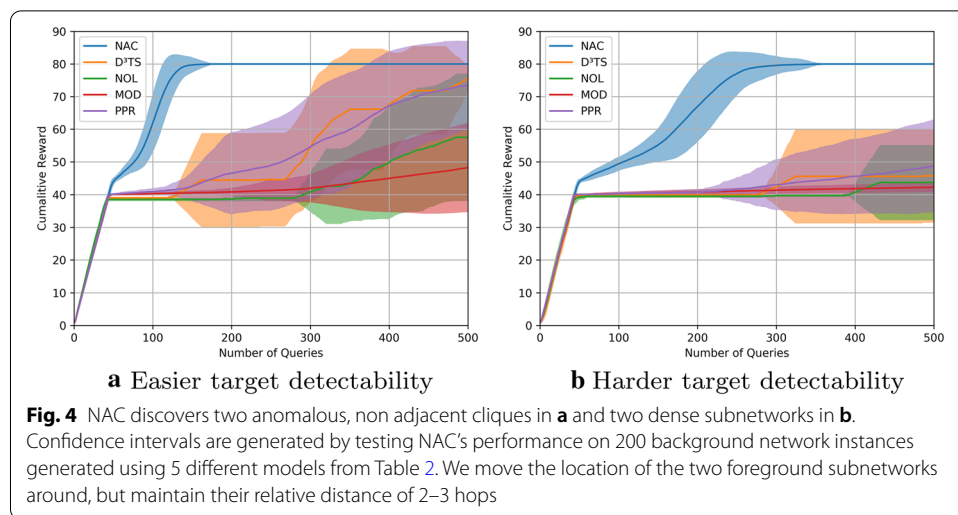
In order to create a background plus foreground network sample, we select a subset of the nodes from the background network that will represent the target nodes. We then simulate an ER subnetwork on these nodes and replace their background induced subnetwork with the ER subnetwork, while maintaining the edges from the target nodes to the rest of the background nodes.

### Real datasets

We analyzed two Facebook datasets (Rozemberczki et al. 2018) representing pages of different categories as nodes and mutual likes as edges. For both cases, we study the discovery of a target set of nodes, where we control how we generate and embed them in the background network. In particular, we embed a synthetic foreground subnetwork consisting of a denser (anomalous) ER graph with size  $n_f = 80$  and density  $p_f = 0.003$ . We also consider the Livejournal dataset (Murai et al. 2017). This dataset represents an online social network with users representing nodes, and their self-declared friendships representing edges. For each user, there is also information on the groups they have joined. Similar to Murai et al. (2017), we use one of the listed groups as the target class. The Livejournal dataset represents a departure from the two Facebook datasets, both in terms of its much larger size, but also because the target class does not represent an anomaly. Table 3 describes a few topological characteristics of the real networks described here, as well as details on their target classes.

### Baselines

We evaluate NAC by comparing its performance to two top-performing online network discovery approaches. The Network Online Learning (NOL) (LaRock et al. 2018, 2020) algorithm learns an online regression function that maximizes discovery of previously unobserved nodes for a given number of queries. We modify the objective of NOL to match our problem setting by requiring the discovery of previously unobserved nodes of a particular type. A second baseline we consider is the Diversity Dynamic Thompson Sampling ( $D^3TS$ ) (Murai et al. 2017) approach.  $D^3TS$  is a stochastic multi-armed bandit approach that leverages different node classifiers and Thompson sampling to diversify the selection of a boundary node. We also compare to a simple fixed node selection heuristic referenced in Murai et al. (2017) called Maximum Observed Degree (MOD). At every decision step, MOD selects the node with the highest number of observed neighbors that have the desired target label. Finally, we compare to the heuristic that at each step selects the node with the highest PPR network-embedding score.

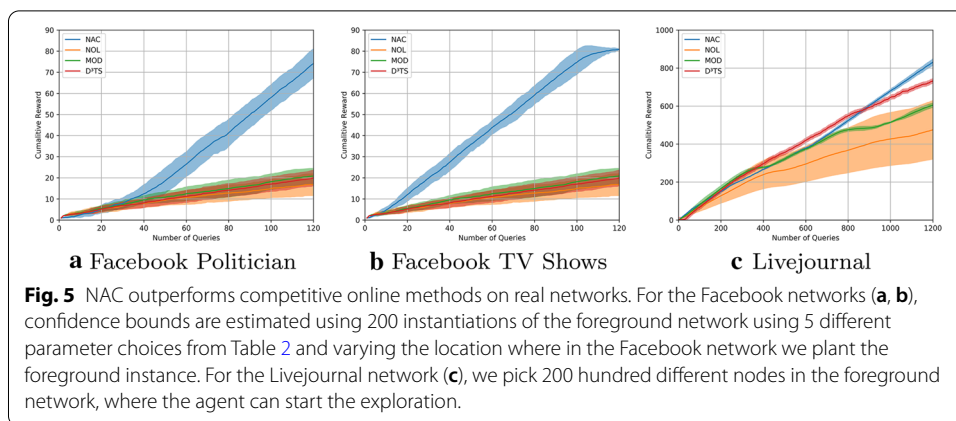


### Learning scenarios

In the first learning scenario, the goal is to detect a set of distributed anomalous nodes. They are represented by two cliques, each containing 40 nodes, that are 2 to 3 hops away from each other. The training instances are networks generated by the SBM model, while the test cases are network instances generated by the LFR model. In this scenario, the discovery agent has to figure out (1) how to value longer exploration paths over the cost of including nodes not in target set, and (2) how to adjust to topological differences between training and testing instances. In Fig. 4a, we consider a test case where detectability of the two cliques with complete network information is relatively easy (average background density around the cliques is comparatively low). We observe that all the methods are able to find the first clique, yet all the baselines struggle once they enter the region where no clique nodes are present. The baselines eventually find some of the second clique’s nodes, but they are unable to fully retrieve the entire second clique. NAC is able to leverage estimation of long-term reward and access to the offline policy to fully recover both cliques. Furthermore, NAC is able to generalize to the more complex LFR topology.

In Fig. 4b, we consider a much harder case. The foreground networks are two disjoint denser subnetworks, 2–3 hops away, each with density 0.2 in a background of density 0.05. Even though these foreground networks are denser than the background network, they are still much sparser than the cliques embedded in the first learning scenario. In fact, their relative density parameters are close to the undetectability bound (Nadakuditi and Newman 2012). In this case, neither of the baselines learns how to recover the second foreground network. NAC goes through a longer exploration phase, but eventually learns how to grow the network to identify the second foreground network.

In Fig. 5, we illustrate how our model trained on synthetic background networks generalizes to realistic background topologies. For this scenario, we trained with instances from both the LFR and SBM models. We observe that NAC generalizes very well to the Facebook network topologies and is able to fully discover the target nodes (Fig. 5a, b). Note the substantial performance improvement when we compare to the other network discovery baselines. The Livejournal network in Fig. 5c presents a much more complex



discovery challenge both in terms of the size and realism the network (see Table 3 for details). In particular, the target network is not synthetically generated as was the case with the Facebook networks, and we do not know the underlying process that generated this target network. Note that NAC is able to overcome the most competitive baseline  $D^3TS$  at about iteration 900, when it has seen a bit more than a third of the target nodes (600 nodes). After this point, NAC is able to recover the remaining target nodes at a substantially faster rate than all the baselines.

**Role of network embedding**

In this section, we systematically explore the role of the embedding algorithm in supporting better network discovery for selective harvesting. We consider two broad classes of embedding methods: walk-based methods (Grover and Leskovec 2016; Murai et al. 2017; Taher 2003), and matrix factorization methods (Pearson 1901; Torres et al. 2020; Belkin and Niyogi 2003).

As introduced earlier, Maximum Observed Degree (MOD) (Murai et al. 2017) is a heuristic embedding which ranks nodes by the number of edges shared with a target node. Personalized Page Rank (PPR) (Taher 2003) is a random-walk method which ranks nodes by their estimated random-walk distance to an observed target nodes. We used a damping parameter  $\alpha = 0.8$ . Node2vec (Grover and Leskovec 2016) is a deep-walk based method which attempts to learn a neighborhood preserving representation for each node in a given network instance. We used the following Node2vec parameters: number of random walks = 5, length of each random walk = 40, and embedding dimension = 64. We ranked embedded nodes by estimating the Euclidean distance between each node and the observed target nodes. For Principal Component Analysis (PCA) (Pearson 1901), we compute the eigen-decomposition of the input adjacency matrix. We estimate the node ranking by looking at the average Euclidean distance between a node and observed target nodes.

Laplace Eigenmap Embedding (Eigenmap) (Belkin and Niyogi 2003) is a low-dimensional graph representation based on spectral properties of the Laplacian matrix of a graph. In this embedding, we represent vertices using the eigenvector corresponding to the second smallest eigenvalue. Rank is estimated by looking at the absolute value of the

dot product of embedding vectors as described in Torres et al. (2020). The embedding dimension was set to 64.

Laplace Eigenmap Embedding (Eigenmap) (Torres et al. 2020) is a low-dimensional graph representation based on geometric properties of the Laplacian matrix of a graph. Unlike eigenmap, GLEE represents vertices using the eigenvector corresponding to the largest eigenvalue. Ranking is estimated in the same way as Eigenmap. The embedding dimension was set to 64.

**Embedding evaluation metrics**

Our evaluation of the embedding algorithm is in the context of its support to NAC’s policy learning component. An effective RL agent for selective harvesting would benefit from state approximations that reflect canonical states for this task (illustrated in Fig. 3). This may imply differing embedding objectives than if we analyze network embedding algorithms as standalone solutions. To this effect, we consider the following metrics for evaluating the role of the embedding algorithm.

**Consistent embedding**

Ideally, we would like the embedding algorithm to place probed and unprobed target nodes near each other. This property implies that NAC will have a higher chance of visiting target nodes earlier than background nodes. To capture the consistency property of the embedding algorithm  $e(\cdot)$ , we measure, at every discovery step  $t$ , the accuracy of the embedding algorithm in recovering the top  $k$  target nodes:

$$Accuracy_t(e(G_t)) = \frac{\text{\#top k target nodes identified by embedding}}{\text{\#true target nodes}}. \tag{6}$$

**Compressability of state-action space**

In an ideal RL setting, the highest reward value  $Q(s, a)$  for a given action space will be highly concentrated over the best action option. We can conceptualize this scenario using a Gaussian distribution with mean represented by the reward value of the best action and minimal variance. We favor node embeddings that concentrate favorable actions—e.g., tightly clustering target nodes in the border set.

This entropy minimization concept is illustrated by Fig. 7. In 7a the entropy for  $Q(s = G_t, a = u)$  is higher than Fig. 7b causing the policy  $\pi(a = u | s = G_t)$  to have higher variance and lower probability of successfully selecting the “best” node.

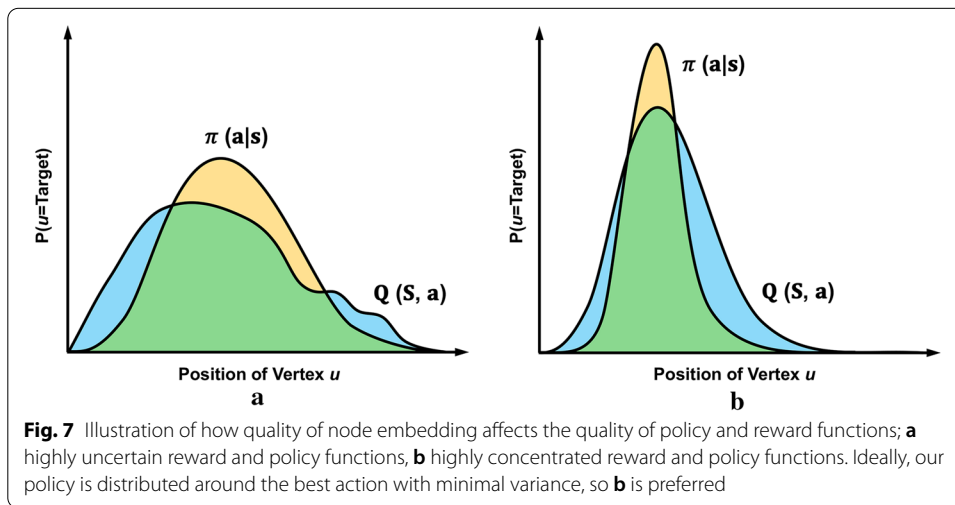
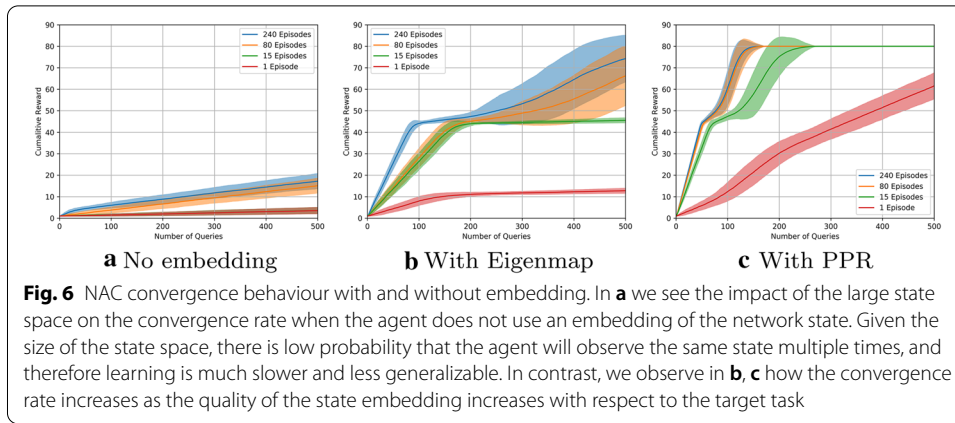
To measure how the embedding algorithm supports this entropy minimization principle, we look at the variance over node rankings in the embedding space for each target cluster and compute the entropy as follows,

$$H_{\mathcal{B}_t}(e) = 0.5[1 + \log(2\pi \text{Var}[e(\mathcal{B}_t)])], \tag{7}$$

where  $\mathcal{B}_t$  is the set of target nodes in the border set at time step  $t$ .

**Robustness to increasing signal complexity**

A good embedding algorithm allows the discovery agent to stay robust as the strength of the signal deteriorates and its complexity increases. We consider two



parameters: the strength of background class and the strength of target class and vary them to explore both regions of high and low signal-to-noise ratio (SNR). To capture robustness, we examine sensitivity to target and background model parameters of Area Under the Curve (AUC), the aggregated accuracy metric over discovery time-steps  $t$ ,

$$AUC = \sum_t Accuracy(e(G_t)). \tag{8}$$

**Learning convergence time**

A useful embedding algorithm reduces the number of episodes required to learn effective discovery policies. The embedding algorithm does this by mapping network states to fewer canonical representations that aid policy learning. Here we estimate improvements in NAC’s convergence rates without and with access to the embedding steps. Results are shown in Fig. 6.

### Data generation for embedding analysis

We consider the following learning setting for our embedding analysis: the target class is a set of disjoint dense subgraphs within a background network. A variety of background and anomaly (target) densities are tested. For each learning step, the incomplete graph is embedded and a ranking for all observed nodes is computed and scored. An optimal policy is defined as navigating each step of the selective-harvesting task in the minimal number of steps. We utilize ground truth to navigate the graph optimally. For illustration, in the setting of two anomalous subnetworks with 40 nodes each, separated by 2-hops, a perfect traversal is 81 steps long.

In our experiments, we consider the following parameters: each anomalous subnetwork has 40 vertices, and the background network consists of 2000 vertices. We use stochastic block model to generate background instances at various densities. Each background instance contains two communities with intra-community edge probability  $p_1 = p_2 = 0.25$  and inter-community edge probabilities  $r$  in the range  $\{0.01, 0.025, 0.05, 0.075, 0.1\}$ . We use the ER model to generate anomalous subnetworks with edge probabilities  $p_t$  in the range  $\{0.25, 0.5, 0.75, 1\}$ . For each unique set of parameters, we generate 10 graph instances leading to a total of 200 graph instances.

### Empirical analysis of embedding algorithms

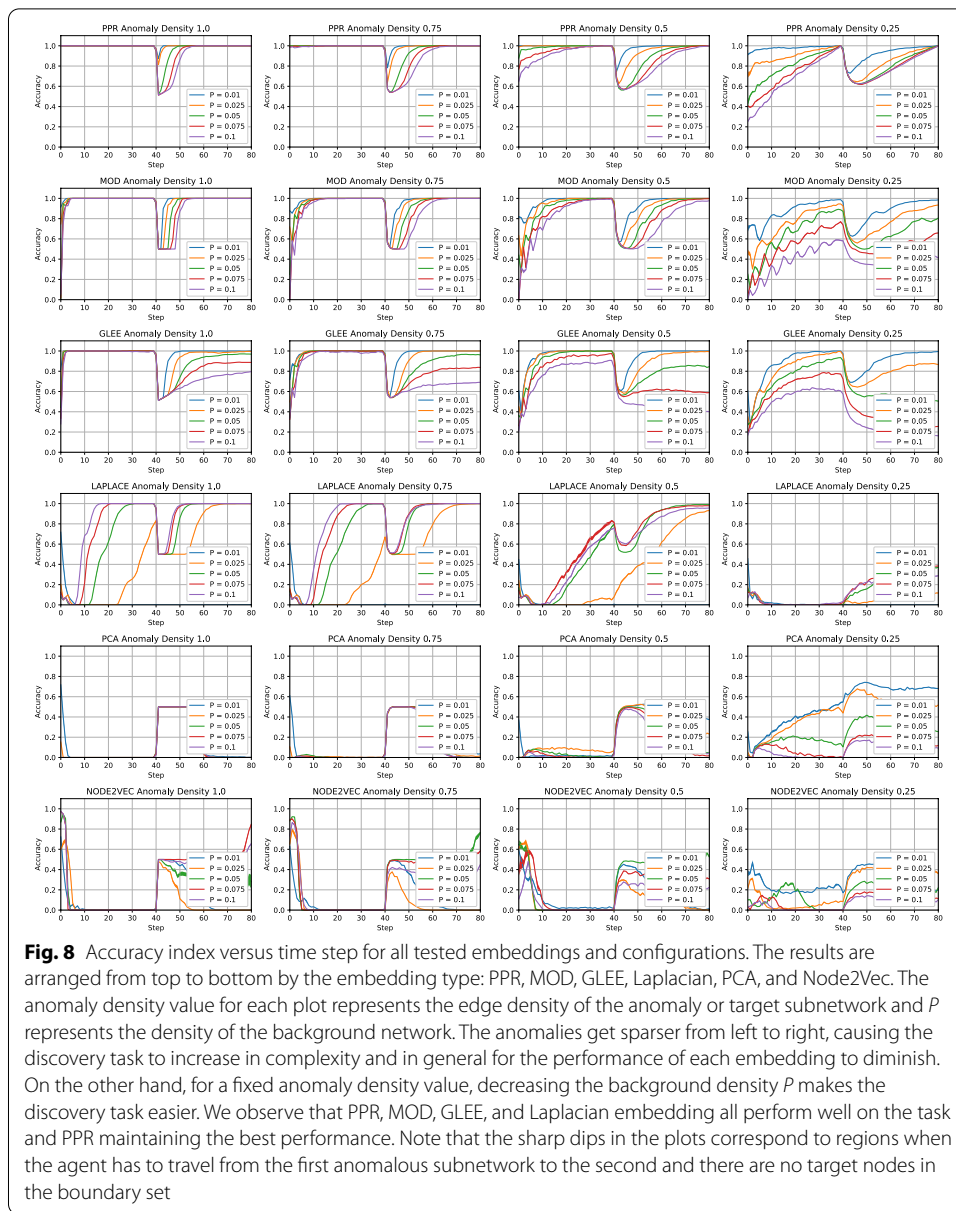
We summarize our empirical evaluation of a few embedding algorithms.

*Consistent embedding* is analyzed in Fig. 8. Within the two target subnetworks, we observe that walk-based methods, PPR and MOD, do a fairly good job in prioritizing target nodes for subsequent selection. PPR is much more robust as the strength of the anomalous subnetwork weakens relative to the background network. Node2vec, by contrast, struggles in the same regions, though it seems to do slightly better once the agent discovers the second subnetwork. It is possible that increasing the dimensionality of the feature vectors would lead to improved performance, however this method is computationally intensive as we consider embeddings over many learning iterations and many graph instances. Overall, across all the embeddings, we observe a strong drop in performance when transitioning between exploitation and exploration regimes. We observe similar embedding sensitivity to decreasing levels of SNR. These observations highlight the role of offline policy learning in recognizing and adapting to changing discovery regimes and sparse task-related signals.

*Compressability of state-action space* is illustrated in Fig. 9. Similar to the accuracy metric, PPR and MOD appear to do the best job of quickly collapsing to a set of node positions as enough target nodes are collected. Again, node2vec appears to be a poor choice, but does exhibit some compressability in the higher SNR cases. The graph (a.k.a. matrix) factorization approaches appear to follow the expected trend of degrading in performance with a reduction of SNR.

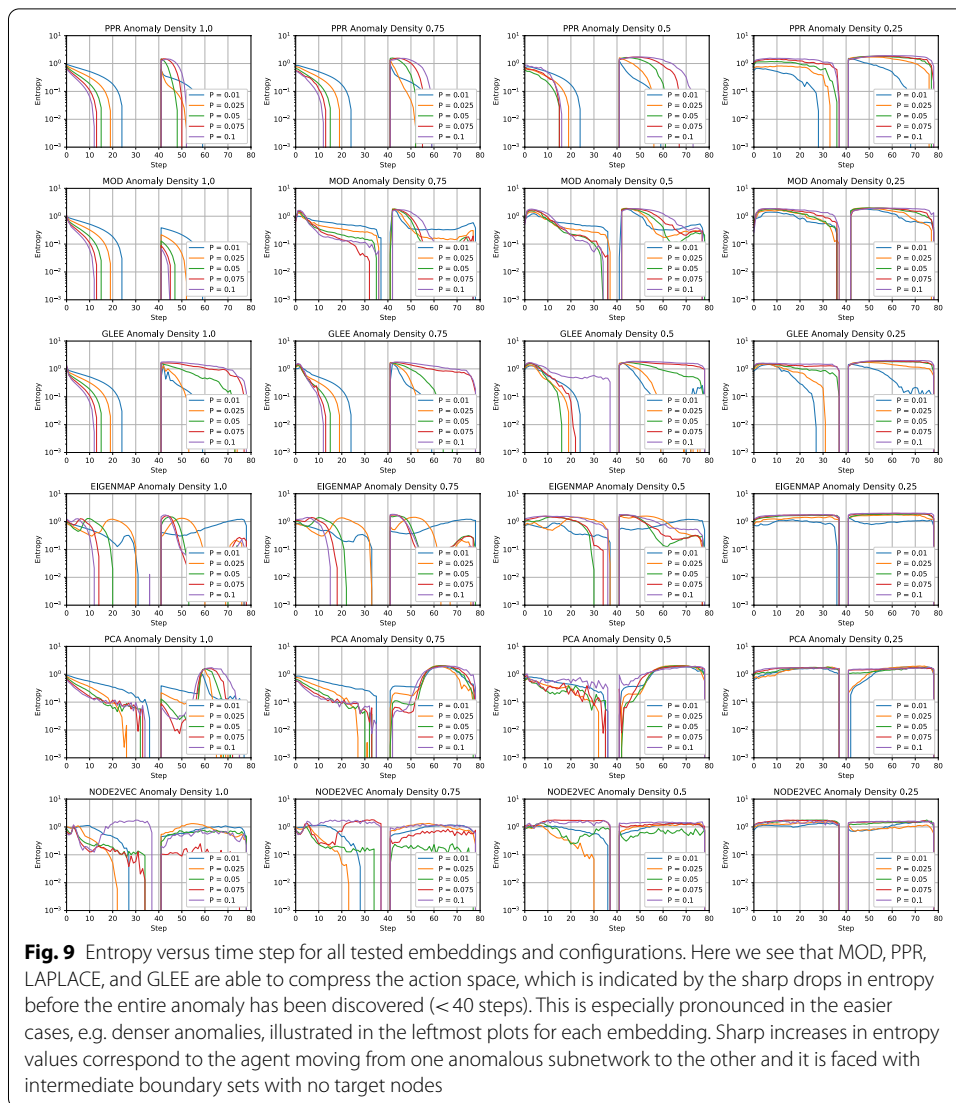
*Robustness to increasing signal complexity* is demonstrated by Fig. 10, which represents the integrated accuracy over the entire selective harvesting task. The same trends discussed in the accuracy section are illustrated here. This figure delineates, at an aggregate level, the network topology characteristics where simple embedding





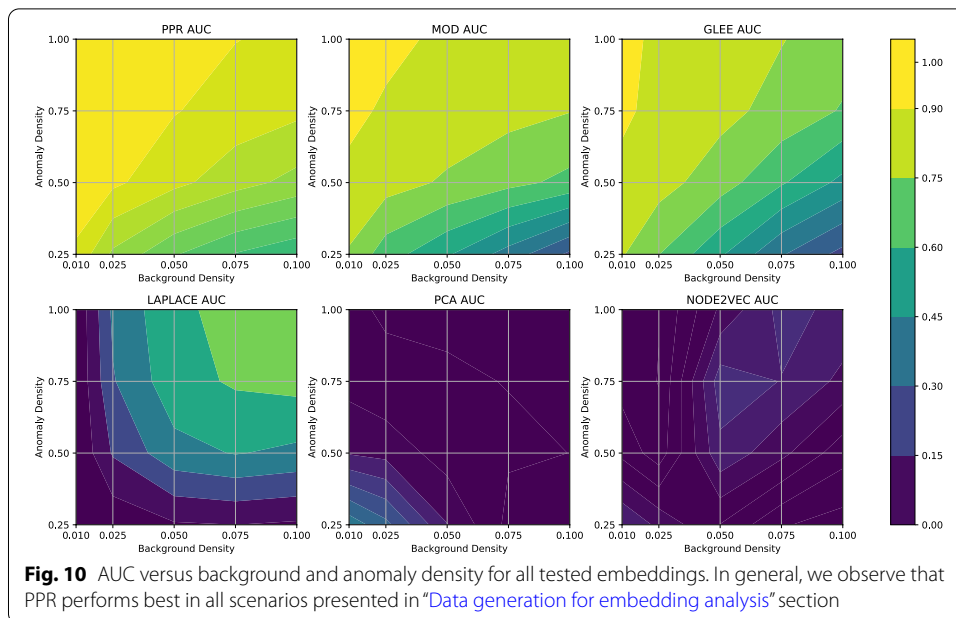
heuristics are sufficient to support effective selective harvesting (lighter color regions) and those topology characteristics where offline planning is required.

*Learning convergence time* is analyzed in Fig. 6, which shows the performance of each embedding paired with policy learning after  $N$  episodes of training. We demonstrate in Fig. 6a that without embedding, the convergence time is likely to be very long and requires a high capacity network. We show representative embeddings from the walk-based and factorization approaches in Fig. 6b, c; and observe that they converge in a consistent way to their entropy and accuracy scores. We illustrate this by analyzing the test case described in Fig. 4a, but the behavior is consistent for all the different test cases considered. Overall we observe that embedding quality directly impacts



convergence time and ultimately the ability of the discovery algorithm to achieve the downstream task objective with budget and resource constraints.

Across the various evaluation metrics and learning regimes, we consistently observe PPR outperforming other embedding algorithms in best augmenting discovery policy learning for selective harvesting. The success of PPR across the various evaluation metrics could be explained by the shared characteristics between the selective harvesting task and the PPR algorithm. Both algorithms rely on the concept of exploring local, relatively dense neighborhoods from a seed node. The same rationale can explain the relative success of the MOD heuristic, though MOD does not have the randomness feature that allows PPR to handle sparser distributions of target nodes. The rest of the embedding approaches lack the seed-centric embedding property and therefore never match the overall performance of PPR. Our hypothesis, however, is that consideration of alternative downstream tasks, might imply a different ranking of suitable embedding methods.



### Conclusions and future work

We introduced NAC, a deep RL framework for task-driven discovery of incomplete networks. NAC learns offline models of reward and network-discovery policies based on synthetically generated training data. NAC is able to learn effective strategies for the task of selective harvesting, especially for learning scenarios where the target class is relatively small and difficult to discriminate. We show that NAC strategies transfer well to unseen and more complex network topologies including real networks. We analyze various network embedding algorithms as mechanisms for supporting fast navigation through the large network state space. Across several metrics of evaluation, we identify personalized Pagerank as a robust network embedding strategy that best supports planning for the task of selective harvesting. We leave analysis of alternative downstream tasks and their respective suitable network embedding for future work.

Our approach opens up many interesting venues for future research. The effectiveness and convergence of our algorithm relies on being trained on sufficiently representative training data. It is valuable to further explore and quantify the limits of transferability of synthetically generated training sets. Our current framework is flexible enough to incorporate additional discovery strategies generated from other methods, as part of the offline training process. This feature can lead to more efficient discovery strategies, but we leave that careful analysis for future work. Additionally, for convenience and processing speed, we chose to encode our policy with a standard convolutional neural network. Understanding the impact of utilizing alternative neural network designs, such as graph convolutional networks, is an interesting future research direction. Finally, the NAC framework is general enough to support discovery for other network learning tasks. It is valuable to explore how a different learning objective changes the training, convergence, and generalizability requirements.

### Acknowledgements

We thank Timothy LaRock and Timothy Sakharov for help with the NOL and embedding experiments, respectively. We also thank Ayan Chatterjee, Eric "Nick" Generous, and Kendra "V" Lange for feedback on this manuscript.

### Authors' contributions

Peter Morales (PM), Rajmonda S. Caceres (RSC) and Tina Eliassi-Rad (TER) jointly formulated the research direction. PM and RSC jointly designed methodology, experimental setup as well as wrote the manuscript. PM implemented and ran all experiments presented. PM, RSC, TER provided analysis and interpretation of experimental results. TER provided revisions of the manuscript.

### Funding

This material is based upon work supported by the Under Secretary of Defense for Research and Engineering under Air Force Contract No. FA8702-15-D-0001. Any opinions, findings, conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the Under Secretary of Defense for Research and Engineering. Distribution Statement A. Approved for public release. Distribution is unlimited.

### Availability of data and materials

Data utilized for synthetic experiments are generated using the NetworkX library and the parameters outlined in Table 2. Real datasets utilized are publicly available and are referenced in "Datasets" section. Experiment code will be shared upon request. All authors read and approved the final manuscript.

### Declarations

#### Competing interests

The authors of this paper are unaware of any existing competing interests.

#### Author details

<sup>1</sup> MIT Lincoln Laboratory, Lexington, USA. <sup>2</sup> Northeastern University, Boston, USA.

Received: 24 October 2020 Accepted: 19 February 2021

Published online: 20 March 2021

### References

- Avrachenkov K, Basu P, Neglia G, Ribeiro B, Towsley D (2014) Pay few, influence most: online myopic network covering. In: IEEE conference on computer communications workshops, pp 813–818
- Belkin M, Niyogi P (2003) Laplacian eigenmaps for dimensionality reduction and data representation. *Neural Comput* 15:1373–1396
- Dai H, Khalil EB, Zhang Y, Dilikina B, Song L (2017) Learning combinatorial optimization algorithms over graphs. In: Proceedings of the 31st international conference on neural information processing systems, pp 6351–6361
- Goindani M, Neville J (2019) Social reinforcement learning to combat fake news spread. UAI
- Grover A, Leskovec J (2016) node2vec scalable feature learning for networks. In: 22nd ACM SIGKDD international conference on knowledge discovery and data mining
- Heess N, Dhruva TB, Sriram S, Lemmon J, Merel J, Wayne G, Tassa Y, Erez T, Wang Z, Eslami SMA, Riedmiller MA, Silver D (2017) Emergence of locomotion behaviours in rich environments. CoRR [arXiv:1707.02286](https://arxiv.org/abs/1707.02286)
- Ho C, Kochenderfer MJ, Mehta V, Caceres RS (2015) Control of epidemics on graphs. In: 54th IEEE conference on decision and control, pp 4202–4207
- Holland PW, Laskey KB, Leinhardt S (1983) Stochastic blockmodels: first steps. *Soc Netw* 5(2):109–137
- Kingma DP, Ba J (2014) Adam: a method for stochastic optimization. In: 3rd international conference on learning representations
- Lancichinetti A, Fortunato S, Radicchi F (2008) Benchmark graphs for testing community detection algorithms. *Phys Rev E* 78:046110
- LaRock T, Sakharov T, Bhadra S, Eliassi-Rad T (2018) Reducing network incompleteness through online learning a feasibility study. In: The 14th international workshop on mining and learning with graphs
- LaRock T, Sakharov T, Bhadra S, Eliassi-Rad T (2020) Understanding the limitations of network online learning. *Appl Netw Sci* 5(1):60
- Mittal A, Dhawan A, Medya S, Ranu S, Singh AK (2019) Learning heuristics over large graphs via deep reinforcement learning. CoRR [arXiv:1903.03332](https://arxiv.org/abs/1903.03332)
- Mnih V, Kavukcuoglu DS, Rusu AA, Veness J, Bellemare MG, Graves A, Riedmiller M, Fidjeland AK, Ostrovski G, Petersen S, Beattie C, Sadik A, Antonoglou I, King H, Kumaran D, Wierstra D, Legg S, Hassabis D (2015) Human-level control through deep reinforcement learning. *Nature* 518:529–533
- Mofrad MH, Melhem R, Hammoud M (2019) Partitioning graphs for the cloud using reinforcement learning. CoRR [arXiv:1907.06768](https://arxiv.org/abs/1907.06768)
- Murai F, Rennó D, Ribeiro B, Pappa GL, Towsley DF, Gile K (2017) Selective harvesting over networks. *Data Min Knowl Disc* 32(1):187–217
- Nadakuditi RR, Newman MEJ (2012) Graph spectra and the detectability of community structure in networks. CoRR [arXiv:1205.1813](https://arxiv.org/abs/1205.1813)
- Pascanu R, Mikolov T, Bengio Y (2013) On the difficulty of training recurrent neural networks. In: International conference on machine learning, pp III-1310–III-1318
- Pearson K (1901) On lines and planes of closest fit to systems of points in space. *Philos Mag* 2:559–572

- Rozemberczki B, Davies R, Sarkar R, Sutton CA (2018) GEMSEC: graph embedding with self clustering. CoRR [arXiv:1802.03997](https://arxiv.org/abs/1802.03997)
- Schulman J, Wolski F, Dhariwal P, Radford A, Klimov O (2017) Proximal policy optimization algorithms. CoRR [arXiv:1707.06347](https://arxiv.org/abs/1707.06347)
- Silver D, Schrittwieser J, Simonyan K, Ioannis A, Ioannis HA, Arthur G, Arthur HT, Baker L, Lai M, Bolton A, Chen Y, Lillicrap T, Hui F, Sifre L, van den Driessche G, Graepel T, Hassabis D (2017) Mastering the game of Go without human knowledge. *Nature* 550:354–359
- Soundarajan S, Eliassi-Rad T, Gallagher B, Pinar A (2015) MaxOutProbe an algorithm for increasing the size of partially observed networks. CoRR [arXiv:1511.06463](https://arxiv.org/abs/1511.06463)
- Soundarajan S, Eliassi-Rad T, Gallagher B, Pinar A (2016) MaxReach reducing network incompleteness through node probes. In: ASONAM, pp 152–157
- Taher H (2003) Haveliwala: topic-sensitive pagerank: a context-sensitive ranking algorithm for web search. *IEEE Trans Knowl Data Eng* 15(4):784–796
- Torres L, Chan KS, Eliassi-Rad T (2020) GLEE: geometric Laplacian eigenmap embedding. *J Complex Netw* 8(2):cnaa007
- Wang X, Garnett R, Schneider J (2013) Active search on graphs. In: Proceedings of the 19th ACM SIGKDD international conference on knowledge discovery and data mining
- You J, Liu B, Ying R, Pande V, Leskovec J (2018) Graph convolutional policy network for goal-directed molecular graph generation. In: Proceedings of the 32nd international conference on neural information processing systems, pp 6412–6422

### Publisher's Note

Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

**Submit your manuscript to a SpringerOpen<sup>®</sup> journal and benefit from:**

- ▶ Convenient online submission
- ▶ Rigorous peer review
- ▶ Open access: articles freely available online
- ▶ High visibility within the field
- ▶ Retaining the copyright to your article

---

Submit your next manuscript at ▶ [springeropen.com](https://www.springeropen.com)

---