DISMANTLE LARGE NETWORKS THROUGH DEEP REINFORCEMENT LEARNING

Changjun Fan
National University of Defense Technology
fanchangjun@nudt.edu.cn

Yizhou Sun
University of California, Los Angeles
yzsun@cs.ucla.edu

Li Zeng
National University of Defense Technology
crack521@163.com

Yang-Yu Liu
Harvard Medical School
yyl@channing.harvard.edu

Muhao Chen
University of California, Los Angeles
muhaochen@ucla.edu

Zhong Liu
National University of Defense Technology
phillipliu@263.net

ABSTRACT

Network dismantling aims to find an optimal set of nodes whose removal breaks down the network into components of subextensive size. It is one of the most important issues in network science, and has wide applications in the design of optimal strategies for network attack, information spreading, and immunization policies. Yet, due to its NP-hard nature, it cannot be solved by any exact algorithm with polynomial time complexity. Existing algorithms are mainly heuristic-driven, and are usually ad hoc and cannot be adapted to more complicated scenarios, such as the case where each node is associated with a different cost. Here we propose a novel solution based on deep reinforcement learning to efficiently solve the network dismantling problem. In particular, we reformalize the problem as a Markov Decision Process (MDP): an agent takes an action in terms of choosing a node to attack based on the current network state, with the goal to maximize the final reward. The policy is automatically learned by observing many simulations of network dismantling using reinforcement learning. Our learned agents, trained on small synthetic graphs, are able to outperform state-of-the-art benchmark methods in terms of both solution quality and running time. As the first practice to use deep reinforcement learning to solve network dismantling problem, our approach may inspire solutions to other network science problems.

1 INTRODUCTION

Networks are a common data structure to describe numerous types of interactive systems. In network science studies, network dismantling has been a longstanding problem, which seeks to remove an optimal (e.g., minimal) set of nodes to dismantle a large connected network into pieces with subextensive size. This problem has wide applications in different domains, some examples include (i) eliminating bacteria by disrupting their molecular structures (Kovács & Barabási, 2015); (ii) network immunization against epidemic spreading by vaccinating a few individuals (Pastor-Satorras & Vespignani, 2001); (iii) blocking rumor spreads by limiting some highly influential spreaders on social networks (Kempe et al., 2003); and (iv) controlling the spread of computer virus by guarding specific servers (Cohen et al., 2001).

The exact solution to this problem has been proven to be NP-hard (Kempe et al., 2003), which prohibits its practical solution for large-scale networks. Traditional approximation methods are often

based on some local or global structural node centrality measures, such as degree, or betweenness. More recently, some message passing algorithms (Braunstein et al., 2016) have been proposed. However, these methods either suffer from deteriorated performance, or require substantial problem-specific research. Usually, a method designed for one application scenario fails in another. We still lacks a generic and efficient solution that requires much less prior knowledge and is adaptive to different scenarios.

Recently, there has been some preliminary work on using deep architectures to resolve combinatorial problems (Khalil et al., 2017). Unfortunately, they either are not designed for graph-related problems, or fall short of tackling the network dismantling task. Inspired by the recent success of deep reinforcement learning (DRL) on solving intriguing Atari games and Go, we develop a novel DRL-based approach for network dismantling. The process of network dismantling can be naturally regarded as a Markov decision process, where at each step we choose to remove one node until the network is broken into isolated nodes or components with sub-extensive sizes. Our approach seeks to optimize this node removal sequence to obtain the optimal robustness score (Eq.1), which can be considered as the ultimate reward of node removal actions.

The proposed DRL-based network dismantling method successfully surpasses the state-of-the-art baselines, in terms of both effectiveness and efficiency. Furthermore, our method is capable of handling different variations of the network dismantling problem. For example, we can consider the robustness measures under the case where each node is associated with a different cost.

2 METHOD

We hereby formalize the network dismantling problem and introduce the method details.

2.1 NETWORK DISMANTLING PROBLEM

Formally, given a network or graph $G = (V, E)$ with node set V and edge set E, our objective is to design a node removal strategy, i.e., a sequence of nodes $(v_1, v_2, \ldots, v_N)$, to minimize the robustness score, defined as below:

$$R(v_1, v_2, \ldots, v_N) = \frac{1}{N} \sum_{m=1}^{N} s(v_1, v_2, \ldots, v_m) \tag{1}$$

where N is the total number of nodes in the graph, $v_i \in V$ denotes the ith node that is removed, $s(v_1, v_2, \ldots, v_m)$ is the fraction of nodes in the largest connected component after removing m nodes, and $1/N$ is the normalization factor to make the robustness score of networks with different sizes be comparable to each other. In case there is a removal cost associated with each node, we can design a variant of robustness score, R_cost, as follows

$$R\_cost(v_1, v_2, \ldots, v_N) = \sum_{m=1}^{N} s(v_1, v_2, \ldots, v_m) c(v_m) \tag{2}$$

where $c(v_m)$ denotes the normalized cost associated with node v_m and $\sum_{m=1}^{N} c(v_m) = 1$. Note that Eq.1 is a special case of Eq.2, where $c(v_m) = 1/N$.

2.2 THE PROPOSED FRAMEWORK

Figure 1 (Appendix B.5) illustrates two iterations of the dismantling process with the proposed framework, which consists of three main components: (i) The *Network embedding part* learns nodes' representations, in which the nodes' structural information and auxiliary information (like removal cost) are encoded; (ii) The *Q-learning part* learns Q value for each node, which represents the predicted accumulative future rewards after selecting this node. (iii) The *Greedy selection part* selects the node with the highest Q value each step until the terminal state. In the following, we describe each part in detail.

Network Embedding Part: We refer to this part as an encoding process, which aims to effectively represent the current state or graph s and the current action or node a. The ideal representations should capture the graph structural information (and auxiliary information, like removal costs, if available), and most importantly, the relative position of node a in graph s. Traditional hand-crafted features, such as degree distribution, motif counts, etc., are incapable of describing such complex graph information. Here we utilize the graph embedding method, more specifically, the graph neural

network, to preserve the structural information of a graph into a low dimensional embedding space. This technique recursively aggregates node features according to the input graph topology, and finally computes a low dimensional feature embedding for each node (action) after a few rounds of recursion, given the current network (state). For more encoding details, see Algorithm 1 (Appendix A). Note that differing from previous methods that simply use sum or mean pooling over all nodes to obtain the whole graph embedding, we add a virtual node that connects (one-way connection) all nodes in the graph, to represent the current state s, , and update the same round of propagations. The output embedding of the virtual node is used for the state representation.

Q-Learning Part: We refer to this part as a decoding process. Q-learning part aims to learn a mapping the from state-action pair (s, a) to a scalar value R, for which traditionally a nonlinear function is utilized, such as SVM or MLPs. Here we parameterize the Q function with a MLP employed with ReLU activation. More specifically,

$$Q(s, a) = W_5^T ReLU(z_a^T \cdot z_s) W_4 \tag{3}$$

Here $W_4 \in R^{p\times 1}, W_5 \in R^{p\times 1}$, are weight parameters, z_s and $z_a \in R^{1\times p}$ are the output embeddings after K-th iteration in the network embedding part, for state and action respectively. Some recent related work (Khalil et al., 2017) usually concatenate these two embeddings before feeding them into the MLP, we argue that the concatenation does not take full advantage of the relation between state s and action a. Since $[u,v] \cdot \left[\begin{smallmatrix} a \\ s \end{smallmatrix}\right] = ua + vs$, given the same state s, different actions' Q values share the same second term vs, and only differentiate between the first one. This means Q values are determined by the action's embedding only. This is apparently unreasonable. Considering that, we apply the outer product operation on state embedding and action embedding, to make each node's Q values be fully influenced by the cross product of these two items, based on which, we apply the MLP to map to a scalar value. For more efficient computation, we use an architecture in which there is a separate output unit for each possible action, and feed only the state embeddings as the input, so as to compute Q values for all possible actions under a given state with only a single forward pass through the neural networks.

Greedy Selection Part: Once we have Q values for all possible actions for a given state, we greedily remove the node with the highest Q value. For tied ones, we randomly chose one among them. Then we will obtain a new graph state, and continue to choose nodes based on the updated Q values. This process is repeated until we break up the graph into components of expected sub-extensive sizes.

2.3 Training Algorithm

There are multiple weight parameters, i.e. Θ_{Embed}, Θ_Q in the encoding and decoding process respectively, we update them by minimizing the following loss function:

$$\begin{aligned} Loss(\Theta_Q, \Theta_{Embed}) = & \underbrace{\mathrm{E}_{(s_t,a_t,r_t,s_{t+n})\sim U(D)}[(r_t + \gamma max_{a'} \hat{Q}(s_{t+n}, a'; \hat{\Theta}_Q) - Q(s_t, a_t; \Theta_Q))^2]}_{\text{Q-Learning Loss}} \\ & + \alpha \underbrace{\sum_{m,n=1}^{N} s_{m,n} ||(y_m - y_n); \Theta_{Embed}||_2^2}_{\text{Network Embedding Loss}} \end{aligned} \tag{4}$$

Eq. 4 consists of two losses: (i) The Q-learning loss minimizes the difference between predicted Q values and target Q values; (ii) The network embedding loss preserves the original network structures in the embedding space. α is a positive hyper-parameter that weighs between these two losses. For the Q-learning loss, we randomly sample mini_batch experiences $(s_t, a_t, r_t, s_{t+n}) \sim U(D)$ from the memory buffer D, note that here we use n-step Q-learning updates (Khalil et al., 2017), which waits n steps so as to collect a more accurate estimation of the future rewards. γ is the discount factor that determines the importance of future rewards. $\hat{\Theta}_Q$ is parameter of the target network, which are only updated with Θ_Q every C steps, and are fixed between the individual updates. For the network embedding loss, N is the number of graph nodes, $s_{m,n}$ denotes whether node m and n are adjacent, and y_i is the embedding vector of node i.

Complexity Analysis. Since most real-world networks are sparsely connected, the adjacency matrix in our setting is regarded as a sparse matrix. The total time complexity of the model in the test phase

is $O(k|E|t)$, where k is the number of iterations of neighbor aggregations and usually set as a small constant, $|E|$ is the number of edges, and t is the number of greedy steps (in the worst case, equals to the number of nodes). If we choose to remove a finite fraction of nodes at each adaptive step, t here actually can be regarded as a constant. We find in our experiments that the final performance is practically unaffected by the removal of up to 1% of nodes each time compared to the one-by-one removal. As a result, our method tends to be directly proportional to the number of edges only, which is highly scalable to those very large networks.

3 Experiments and Results

We use the agent trained from synthetic graphs, and test it on eight real-world networks with various types against the state-of-the-art baselines. Detailed descriptions of test networks and baselines are presented in Appendix B.2 and B.3. The robustness score and running time for different methods are shown in Table 1 and 2. It is obvious that our trained agent GraphDQN has surpassed the state-of-the-art baselines on most datasets, in terms of both effectiveness and efficiency and our efficiency advantages are more obvious on very large networks.

Robust Score / Test Data / Method	crime	digg	Email-Enron	p2p-Gnutella31	soc-Epinions	facebook-links	com-youtube	flickr-links
CI	0.1243	0.0866	0.0445	0.1174	0.0506	0.2695	**0.0225**	NA
MinSum	0.1383	0.0952	0.0477	0.1173	0.0604	0.2725	0.0257	0.0610
BPD	0.1395	0.0952	0.0618	0.1172	0.0748	0.2835	0.0299	0.0841
CoreHD	0.1133	0.0868	0.0514	0.1197	0.0546	0.2747	0.0244	0.0622
GND	0.1381	0.1066	0.0456	0.1257	0.0537	0.2742	0.0232	NA
GraphDQN	**0.1099**	**0.0866**	**0.0430**	**0.1110**	**0.0499**	**0.2684**	0.0237	**0.0546**

Table 1: Comparison of robustnness score for different methods

Time/s / Test Data / Method	crime	digg	Email-Enron	p2p-Gnutella31	soc-Epinions	facebook-links	com-youtube	flickr-links
CI	**0.01**	10.99	70.74	**9.38**	668.43	2723.23	7056.41	>5d
MinSum	1.75	135.57	333.34	247.96	743.68	1414.58	7265.14	81536.70
BPD	0.30	31.00	86.00	41.00	254.00	839.00	462.00	17942.00
CoreHD	0.09	11.52	36.20	11.87	142.92	214.83	2018.88	36837.26
GND	0.11	55.01	35.70	386.31	233.14	4614.54	52246.20	>5d
GraphDQN	0.30	**8.21**	**12.75**	13.34	**22.57**	**73.38**	**281.22**	**989.66**

Table 2: Comparison of running time for different methods

To handle the scenario where each node comes with a different removal cost, we train the GraphDQN_cost agent under the same framework as GraphDQN, by only changing the reward to minimize the weighted robustness score (Eq.2). To test its performance, we add a removal cost on each node that is proportional to its degree, on the networks used in the experiments above. Table 3 and 4 compares the weighted robustness score and running time for different methods. We can see that GraphDQN_cost is still very superior to all baselines both effectively and efficiently. Especially in terms of effectiveness, our model is significantly better than others. We also illustrate the robustness curve, which is plotted with horizontal axis being the fraction of removed nodes, and vertical axis being the fraction of nodes in the remaining largest components, as in Figure 2 and 3 (Appendix B.5). Note that the robustness score can be viewed as an estimation of the area under these curves.

Weighted Score / Test Data / Method	crime	digg	Email-Enron	p2p-Gnutella31	soc-Epinions	facebook-links	com-youtube	flickr-links
CI	0.3103	0.4332	0.3942	0.3833	0.5422	0.6791	0.3677	NA
MinSum	0.3066	0.4160	0.3794	0.3662	0.5204	0.6089	0.3750	0.6336
BPD	0.3325	0.4057	0.3875	0.3798	0.5130	0.5933	0.3823	0.5823
CoreHD	0.2886	0.4292	0.4026	0.3840	0.5457	0.6816	0.3701	0.7114
GND	**0.2879**	0.2502	0.1824	0.4067	0.1355	0.3732	0.2962	0.1830
GraphDQN_cost	0.2901	**0.2018**	**0.1599**	**0.2620**	**0.1054**	**0.1881**	**0.1737**	**0.0667**

Table 3: Comparison of weigthed robustnness score for different methods

4 Conclusion

In this work, we design a generic deep learning framework to tackle network dismantling problems. To the best of our knowledge, this is the first effort of using deep learning techniques to address this problem. We empirically prove that our models consistently outperform state-of-the-art baselines on

Time/s \ Test Data / Method	crime	digg	Email-Enron	p2p-Gnutella31	soc-Epinions	facebook-links	com-youtube	flickr-links
CI	**0.01**	**10.99**	70.74	**9.38**	668.43	2723.23	7056.41	>5d
MinSum	1.75	135.57	333.34	247.96	743.68	1414.58	7265.14	81536.70
BPD	1.00	19.00	54.00	41.00	166.00	553.00	**507.00**	20110.00
CoreHD	0.09	11.52	**36.20**	11.87	**142.92**	**214.83**	2018.88	36837.26
GND	0.11	42.93	62.08	144.79	382.11	354.97	61288.20	174363.00
GraphDQN_cost	0.59	40.22	40.41	50.84	143.37	555.83	2044.89	**7734.63**

Table 4: Comparison of running time for different methods on weighted networks

various real-world networks, in terms of both solution quality and running time. Since we incorporate the 'end-to-end' learning framework, our approach requires minor prior knowledge, and hence can be applied to a wide range of real-world network dismantling problems.

5 ACKNOWLEDGEMENTS

The authors are grateful to the anonymous referees for their insightful suggestions and comments. And thanks to the financial support of China Scholarship Council (CSC).

REFERENCES

Alfredo Braunstein, Luca DallAsta, Guilhem Semerjian, and Lenka Zdeborová. Network dismantling. *Proceedings of the National Academy of Sciences*, 113(44):12368–12373, 2016.

Reuven Cohen, Keren Erez, Daniel Ben-Avraham, and Shlomo Havlin. Breakdown of the internet under intentional attack. *Physical review letters*, 86(16):3682, 2001.

David Kempe, Jon Kleinberg, and Éva Tardos. Maximizing the spread of influence through a social network. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 137–146. ACM, 2003.

Elias Khalil, Hanjun Dai, Yuyu Zhang, Bistra Dilkina, and Le Song. Learning combinatorial optimization algorithms over graphs. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (eds.), *Advances in Neural Information Processing Systems 30*, pp. 6348–6358. 2017.

István A Kovács and Albert-László Barabási. Network science: Destruction perfected. *Nature*, 524 (7563):38, 2015.

Jérôme Kunegis. Konect: the koblenz network collection. In *Proceedings of the 22nd International Conference on World Wide Web*, pp. 1343–1350. ACM, 2013.

Jure Leskovec, Jon Kleinberg, and Christos Faloutsos. Graph evolution: Densification and shrinking diameters. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 1(1):2, 2007.

Jure Leskovec, Kevin J Lang, Anirban Dasgupta, and Michael W Mahoney. Community structure in large networks: Natural cluster sizes and the absence of large well-defined clusters. *Internet Mathematics*, 6(1):29–123, 2009.

Flaviano Morone and Hernán A Makse. Influence maximization in complex networks through optimal percolation. *Nature*, 524(7563):65, 2015.

Salomon Mugisha and Hai-Jun Zhou. Identifying optimal targets of network attack by belief propagation. *Physical Review E*, 94(1):012305, 2016.

Romualdo Pastor-Satorras and Alessandro Vespignani. Epidemic spreading in scale-free networks. *Physical review letters*, 86(14):3200, 2001.

Xiao-Long Ren, Niels Gleinig, Dirk Helbing, and Nino Antulov-Fantulin. Generalized network dismantling. *Proceedings of the National Academy of Sciences*, pp. 201806108, 2019.

Lenka Zdeborová, Pan Zhang, and Hai-Jun Zhou. Fast and simple decycling and dismantling of networks. *Scientific reports*, 6:37954, 2016.