Hybrid Attack Graph Generation with Graph Convolutional Deep-Q Learning

Sam Donald, Rounak Meyur, Sumit Purohit
Pacific Northwest National Laboratory
Richland, Washington, USA
{Sam.Donald,Rounak.Meyur,Sumit.Purohit}@pnnl.gov

ABSTRACT

Effective risk mitigation for cyber-physical energy systems (CPES), such as power grids, requires preemptive knowledge of likely adversarial attack sequences. However, the scarcity of documented attack sequences hinders this process. We propose a data-driven Graph Convolutional Deep-Q Network (GCDQ) to address this lack of data through generating hybrid attack graphs (HAGs) - a graphical representations of CPS attack sequences. By leveraging limited real-world observations from the MITRE ATT&CK knowledge base, our GCDQ model synthesizes realistic graphs with the targeted attribute of minimum detectability via reinforcement learning. This generative model is the first step in creating a tool to substantially boost the attack sequence dataset and enhance the performance of CPS defense-related tasks by providing insights into likely attack sequences with given attributes.

CCS CONCEPTS

• **Security and privacy** → *Malware and its mitigation*; • **Computing methodologies** → *Sequential decision making*.

KEYWORDS

graph generation, cyber attack modeling, attribute targeting

ACM Reference Format:
Sam Donald, Rounak Meyur, Sumit Purohit . 2023. Hybrid Attack Graph Generation with Graph Convolutional Deep-Q Learning. In *3rd Workshop on AI-Enabled Cybersecurity Analytics - KDD 2023.* ACM, New York, NY, USA, 7 pages.

1 INTRODUCTION

The ever increasing complexity of cyber attacks poses a continual challenge to agents seeking to defend cyber physical energy system (CPES) infrastructure. In order to better understand and prepare for attacks, cybersecurity professionals typically rely on databases of prior documented attacks, such as those described by the MITRE ATT&CK (Adversarial Tactics, Techniques, and Common Knowledge) framework. While detailed, these resources are limited in number as data is typically restricted to publicly available intelligence and incident reports [14]. Due to the inability of defenders to adequately prepare, unforeseen CPES attacks result in an increased chance of severe impact and as such are attractive to attackers including nation-states and organized crime groups.

Hybrid attack graphs (HAGs) provide a representation of possible attack sequences for a CPES, and incorporate the technique and tactic categories used by the MITRE ATT&CK framework. Tactics represent the high-level goals of an attack, and comprise of techniques describing actions undertaken. These tactics are ordered to represent successive goals typically undertaken to reach the terminal tactic describing impacts such as system manipulation or data destruction [2]. HAGs describe a CPES through a set of possible MITRE ATT&CK techniques, represented as nodes, along with edges denoting their ordering within the attack sequence. A single attack is represented as a linear sequence of techniques, while a collection of attacks is represented as a graph describing potential attack paths [4].

Given the limited set of existing HAGs, we aim to generate realistic synthetic HAGs with user defined traits, such as low detection rates. In doing so we will bolster the existing dataset to improve data-hungry downstream downstream models and provide a robust tool for cyber analysts and applications capable of leveraging attack sequences with specific traits. Deep learning-based generative models have emerged as a promising approach to bolster limited data sets. By training models to produce synthetic data from the distribution underlying the limited source data, the effective data set size can be increased. This approach is particularly useful when data is scarce or difficult to collect, and is an ideal candidate for the limited HAG dataset and our problem statement.

2 RELATED WORK

In [1], the authors generate molecular graph structures using recurrent neural networks (RNNs). A character-level RNN is used to generate a string representation of molecules, which are post-processed into to molecular graphs. While simple to implement, this approach is significantly constrained by representing graphs as a 1D sequence, allowing for both invalid sequences and preventing the expansion partial graphs. This model is extended by [18], who introduce GRNN, a two stage LSTM based model capable of generating graphs through iterative next node and edge predictions conditioned by an existing graph. A 2-dimensional adjacency matrix representation is used, and as such the GRNN is capable of extending partial graphs, however it does not posses a method of generating node features.

More recently, [10] uses Graph Recurrent Attention Networks (GRANs), combining the strengths of graph convolutional neural

Sam Donald performed this research while employed as a Masters Intern at PNNL. They can be reached at Samdonald@vt.edu.

networks (GCNs) and attention mechanisms to captures long-range dependencies during the graph generation process. Similar to the GRNN model, GRAN does not generate node features, yet outperforms it due to inclusion of GNN layers allowing for more expressive latent representations.

A unique Reinforcement Learning (RL) method for graph generation is proposed by [17] though their Graph Convolutional Policy Network (GCPN). The GCPN model combines a GCN for node embeddings and a multilayer perception (MLP) based generator to produce graphs maximizing a user defined reward function. Similar to the GRAN model, an intermediate graph and set of proposal nodes are embedded, with the node embeddings used to generate an action containing two nodes between which a new edge is added. Associated with each action is a reward based on molecule solubility and other features of interest, along with a discriminator loss from a classifier trained under a GAN framework, with this reward used to perform policy gradient optimization. Of the models described, the GCPN allows for maximum flexibility though a user defined reward function targeting specific attributes, along with its ability to node features. While the GCPN outperforms prior state-of-the-art methods, it is noted that both a large training dataset of 250,000 molecules was used along with multiple pre-training steps.

3 PROPOSED METHODS

3.1 Hybrid Attack Graph Dataset

Training HAGs are sourced from 620 documented software instances within the MITRE ATT&CK enterprise database, with each software described by a set of techniques it is capable of performing. As these techniques are provided in the form of an unordered set, an ordering scheme must be imposed to construct a HAG. This is done through grouping techniques into their corresponding tactic, ordering these groups by their tactic ID, and adding edges between techniques in adjacent tactics. As the resultant HAGs from this process are exceptionally dense, sparse HAGs are extracted by a random walk algorithm with backtracking to simplify the task of graph generation. This random walk algorithm is outlined within appendix A.1, with figs. 1 and 2 displaying a sample dense and sparse HAG from the MITRE ATT&CK enterprise database respectively, along with their MITRE ATT&CK technique codes. Dense graphs will be reconstructed from multiple generated sparse HAGs through post-processing steps described in future work.

3.2 Graph Convolutional Deep-Q Learning

We propose a Graph Convolution Deep-Q Learning model (GCDQ), inspired by the GCPN model described by [17]. Our model uses Deep-Q learning (DQL) as opposed to policy gradient (PG) along with a modified action generation procedure. These changes were done to leverage the benefits of DQL, specifically its training stability, ability for off-policy training, and sample efficiency [15], with these features allowing for future models to be trained in an online environment with minimal interactions.

3.2.1 Deep Q-Learning. The goal of Q-learning is to find an optimal Q function, providing the long term value of an action for a given state. For a policy π the true value of action a given state s is defined by eq. (1), where a discount factor $\gamma \in [0, 1]$ is used to balance immediate and delayed rewards, and R_n is the reward at step n [15]. A parameterized version of Q-learning can then be implemented through a neural network [8], with the process defined by eqs. (1) to (3).

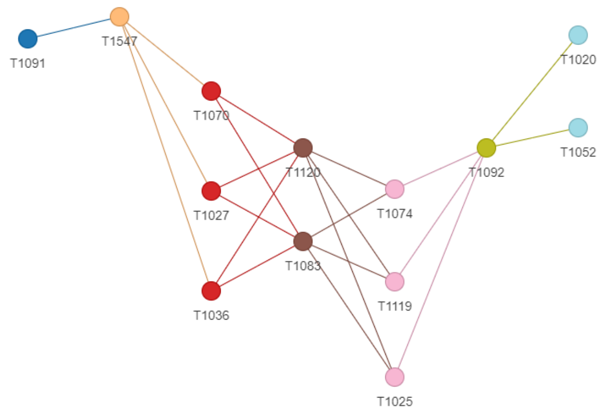

Figure 1: Sample Dense HAG

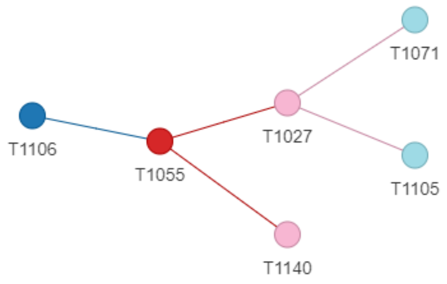

Figure 2: Sample Sparse HAG

$$Q_\pi(s,a) = \mathbb{E}(R_1 + \gamma R_2 + \gamma^2 R_3 + \ldots \quad |S_0 = s, A_0 = a, \pi) \tag{1}$$

$$Target = R_{t+1} + \gamma \max_{a'} Q_{local}(S_{t+1}, a') \tag{2}$$

$$Loss = MSE(Target, Q_{local}(S_t, a_t)) \tag{3}$$

In equation (2), $Target$ is the ideal future discounted reward that our Q-function neural network (Q_{local}) aims to approximate, and is the core concept of reinforcement learning whereby cumulative discounted future rewards are maximized. The max term highlights this principle, implying the optimal action a' leads to a state with the highest possible future rewards. While traditional tabular Q-Learning methods ensure convergence to the optimal Q-function, Q^*, under certain conditions like infinite exploration

[15], these guarantees may not apply when using function approximators such as neural networks due to the curse of dimensionality. However with careful design and hyperparameter tuning similar results can be reliably achieved in practical implementations [8]. Equation (3) then defines the network loss as the Mean Squared Error (MSE) between the Target and the current Q-value estimate of the state-action pair at time t, $Q_{local}(S_t, a_t)$. This loss quantifies disagreement between our Q-function's current estimate and the Target, which we aim to minimize through usage of the Adam optimizer by adjusting the parameters of Q_{local}, θ_{local}.

The baseline deep Q-network (DQN) model described by eqs. (2) and (3) uses the max operation to both select and evaluate actions, and commonly overestimates Q values [15]. Double Deep-Q networks (DDQNs) address this issue though utilizing two separate networks, Q_{target} and Q_{local}, to select and evaluate actions respectively, with the network weights of Q_{target}, θ_{target}, incrementally updated from Q_{local} every τ training steps via Polyak averaging [16][11]. Applying this DDQN framework alters the target equation eq. (2) to eq. (4), with the Polyak update process of Q_{target} described by eq. (5) using β as a parameter to control the information transfer rate.

$$Target = R_{t+1} + \gamma Q_{local}(S_{t+1}, \underset{a'}{\operatorname{argmax}}\, Q_{target}(S_{t+1}, a')) \tag{4}$$

$$\theta_{target} \leftarrow \beta\theta_{local} + (1-\beta)\theta_{target} \tag{5}$$

To train the Q_{local} network, trajectories comprising of an intermediate graph state, the chosen action, and associated reward, are generated, stored and sampled uniformly from a FIFO replay buffer. This allows for multiple training passes of generated data and boosts the convergence rate [11].

3.2.2 Model Architecture. At a high level, our GCDQ generator functions as follows: an intermediate, and initally empty, Hybrid Attack Graph (HAG) is associated with a set of potential techniques represented as an unconnected graph. Techniques within this extended graph are one-hot encoded, before being embedded by multiple GraphSAGE layers [7]. Next, we concatenate pairs of embeddings to form a proposed action embedding, denoted as $[X_{N_i}, X_{N_j}]$, representing the addition of an edge between nodes N_i and N_j.

Following this embedding step, we filter all invalid actions corresponding to a tactic ID increase of more than three. These filtered action embeddings are then passed through a Multilayer Perceptron (MLP), denoted by m, to produce a set of Q-values. We apply a rescaling operation to this Q-value distribution via a Boltzmann temperature coefficient τ. The Boltzmann exploration method applied, described by eq. (8), uses this temperature-adjusted distribution to stochastically select an action, and adds an exploration factor to our model's decision-making process [5]. Based on the selected action, an edge is then added between the associated nodes to expand the intermediate graph. This cycle is then repeated until the graph reaches the desired size. The generator network's equations are outlined in eqs. (6) and (7), and visually represented in fig. 3. Additional details related to network configurations and dimensions are available in appendix A.2.

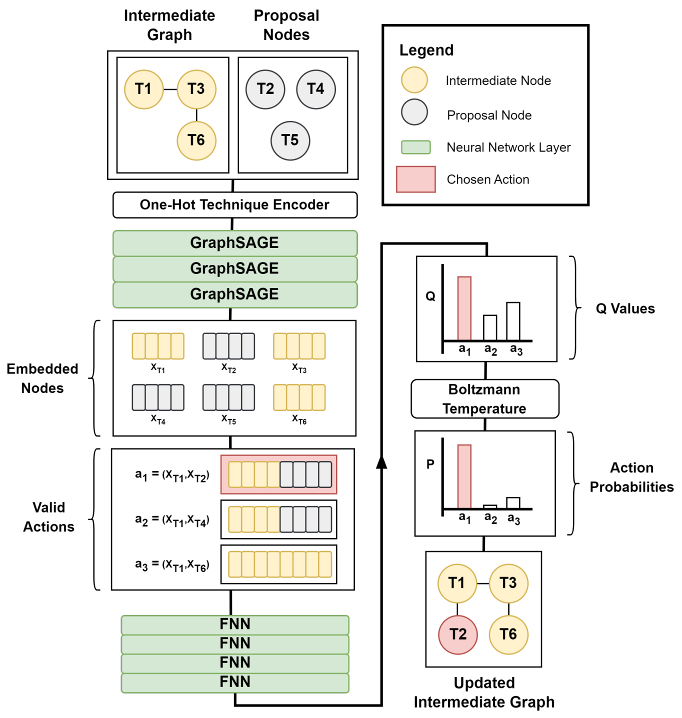

Figure 3: GCDQ Graph Generation Step

$$Q_{a=[N_i,N_j]} = Q(S_t, [N_i, N_j]) = m([X_{N_i}, X_{N_j}]) \tag{6}$$

$$a_t = [N_i, N_j] \sim \mathcal{B}(Q_A, \tau) \tag{7}$$

$$\mathcal{B}(Q_A, \tau) = \frac{\exp\left(\frac{Q(a_i)}{\tau}\right)}{\sum_{j=1}^{N}\exp\left(\frac{Q(a_j)}{\tau}\right)}, \quad i = 1, 2, \ldots, N \tag{8}$$

Equation (8) is the mathematical definition of the Boltzmann distribution. Here, $Q(a_i)$ is the Q-value of action a_i, calculated from eq. (6), and τ is a temperature parameter that controls the randomness of the action selection. Higher τ values result in actions being chosen from a distribution that is close to uniform, while lower τ values produce a more deterministic policy that tends to choose the action with the highest estimated Q-value, making the policy more greedy as it exploits the learnt Q-values [5].

Of note, to select the initial node of the intermediate graph, a zero vector is concatenated to all proposal nodes from the first six tactics. Node i is also restricted to the intermediate graph, while j may be from either the intermediate graph or proposal nodes.

3.2.3 Discriminator training. The loss of a trained discriminator $\mathcal{D}(s_t)$ is integrated into the reward function, ensuring generated graphs align with the reference training data. This discriminator operates within a sequential GAN framework, with the generator previously defined by eqs. (6) and (7). The discriminator consists of multiple graph convolutional layers for node embeddings [9], followed by a mean aggregator function, MLP, and a final sigmoid function, yielding a probability that a given graph s_t originates from the training dataset. For training, synthetic graphs produced from the current generator are used alongside an equivalent number of real graphs from the training set, with the model trained until validation loss falls below 0.45. During the inital training round, the discriminator learns to differentiate real HAGs from synthetic graphs generated via a random policy (represented as temperature $\tau = \infty$) as a generator has not been trained yet, with subsequent rounds using the previously trained generator.

3.2.4 Reward Function. For each generation step undertaken as described by fig. 3, an associated reward is calculated in accordance with eq. (9)

$$R_t = \frac{\alpha_1 \mathcal{D}(s_t) + \alpha_2(1 - \mathcal{I}(a))}{\sum_n \alpha_n} \tag{9}$$

where, $\mathcal{D}(s_t)$ is the discriminator loss for a given graph s_t and $\mathcal{I}(a) \in [0,1]$ denotes the defenders likelihood of identifying newly added techniques associated with action a. The values of α are used to balance reward components, and are normalized, such that the net reward $R_t \in [0,1]$. By optimizing the reward function actions will be taken to minimize the chance of detection (maximizing $1 - \mathcal{I}(a)$) while maintaining similarity to the available training data (maximizing $\mathcal{D}(s_t)$). This specific generic reward function is chosen to easily expand to incorporate additional metrics of interest based on the intermediate graph or chosen action.

4 EXPERIMENTS

To explore the GCDQ models ability to generate HAGs with targeted properties, we create an identification distribution spanning all techniques to simulate a CEPS with various defense measures. These values are generated by sampling detection rates from the uniform distribution [0, 1]. Training is then conducted over two sequential rounds of 2000 training epochs, each followed by boosting of the discriminator. Hyperparameters defined within appendix A were tested for each round, with the resultant generator reward convergence for round two displayed within fig. 4. Each trained generator was swept to determine its sensitivity to the Boltzmann temperature coefficient τ, as described within appendix B.1. This was done to ensure that during inference the learnt model was sufficiently exploited, while maintaining diversity within the generated graphs [5]. A temperature sweep plot for round two of training is provided within fig. 5, and displays the expected increasing exploitation and decreasing diversity as temperature decreases. Additional training plots are provided within appendix C.

4.1 Results

To validate the trained generators, synthetic graphs were produced at temperatures near the diversity inflection point as determined by the temperature sweep fig. 5. This point was chosen to balance exploitation of the learned reward function with with graph diversity [5][15]. Average reward components and validation metrics from the mean of pairwise comparison between 176 synthetic and validation graphs containing six nodes were then computed, with detection reward calculated as the product of all intermediate detection rewards and a graphs overall detection rate equal

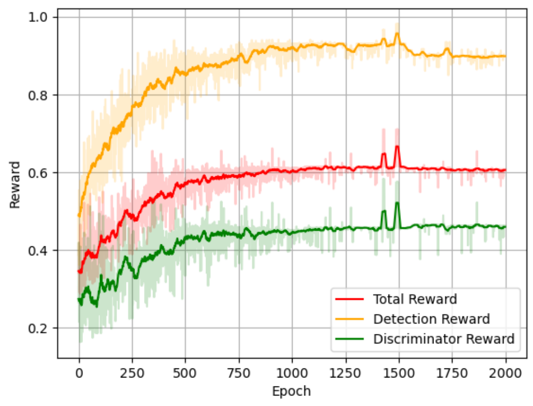

Figure 4: Round 2 reward convergence

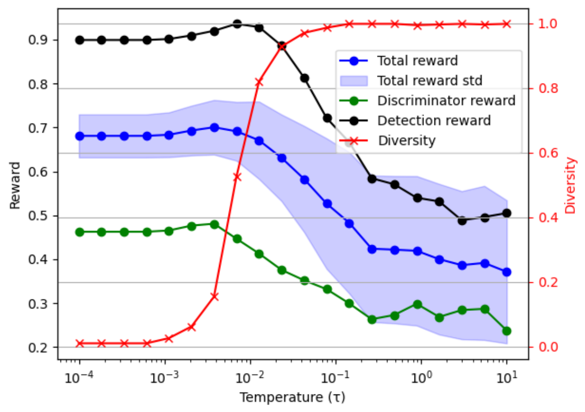

Figure 5: Round 2 Boltzmann temperature sweep

to (1 − detection score). To quantify graph similarity we employ a two-pronged approach, examining node features and assessing overall graph structure. Node feature comparison are done through Graph Edit Distance (GED) [13] and Jaccard similarity [3], with GED measuring the minimum number of operations required to morph one graph into another, while Jaccard similarity calculates the overlap of node features. When considering graph structures, we use the Frobenius norm [6] and the Laplacian Spectral Distance (LSD) [12]. The Frobenius norm quantifies similarity based on the norm of the difference in two graphs adjacency matrices, while LSD measures structural resemblance via eigenvalues of the Laplacian matrix. By coupling these methods with graph diversity, measuring the proportion of unique graphs generated, we are able to accurately identify trends in graph similarity. The process of calculating each metrics is described fully in appendix B. To confirm that the generator had not simply learned to produce the training graphs, 1000 graphs were produced for each configuration within table 1, with no graphs matching any within the training set.

Table 1: Validation results

				Rewards		Node Features		Graph Structure	
Round	Temperature	Diversity	Detection Rate	Detection	Discriminator	GED	Jaccard	Frobenius	LSD
0	∞	1.000	0.939	0.061	0.331	5.323	0.0219	1.660	0.777
1	0.025	0.977	0.305	0.695	0.544	5.700	0.0405	2.063	1.226
1	0.010	0.432	0.127	0.873	0.620	5.559	0.0384	1.656	0.736
1	0.005	0.080	0.105	0.895	0.624	5.596	0.0385	1.536	0.587
2	0.025	1.000	0.375	0.625	0.339	5.566	0.0372	1.934	1.061
2	0.010	0.966	0.184	0.816	0.384	4.902	0.0434	2.066	1.187
2	0.005	0.511	0.188	0.812	0.438	4.436	0.0457	2.065	1.168

4.2 Discussion

The GCDQ models ability to learn both the detection and discriminator components of the reward functions is demonstrated by table 1. It achieves a detection reward of 0.873 and a discriminator reward of 0.62 for $\tau = 0.01$ during the first training round, with a diversity of 0.432. This is a significant improvement from round zero, with the detection and discriminator reward increasing by 0.812 and 0.289. With respect to decreasing temperature, node features metrics for round one remain relatively stable, whereas the graph structure metrics improve significantly. These results suggest that the initial discriminator primarily exploits structural information as opposed to features for classification, and aligns with the expectations of randomly generated graphs containing uniformly distributed nodes. As the temperature decreases and the model further exploits the learned reward function, generated graphs become more structurally similar to the validation set, as observed by the decreasing Frobenus norm and LSD values, as this strategy effectively deceives a structural based graph discriminator to receive maximum reward.

A shift in trends occurs during the second round of training, after the discriminator is fine-tuned to distinguish training graphs from synthetic graphs produced by the newly trained generator. As shown by fig. 4, detection and discriminator rewards continue to improve during training, however node feature metrics as opposed to structural ones now improve as temperature decreases and the learnt model is exploited. This is evidenced by a decreasing GED and increasing Jaccard similarity score, while the Frobenius Norm and LSD remain stable. These trends imply that the fine-tuned discriminator now relies on node features to differentiate synthetic and training graphs, and is possibly due to the round one generators bias towards a subset of techniques with low detection rates. Furthermore, we note the slight decrease in overall detection rate between round one and two, as the generator incorporates a larger set of techniques to fool the feature focused discriminator.

5 CONCLUSION AND FUTURE WORK

In this paper, we presented a Graph Convolutional Deep-Q Learning model capable of generating realistic HAGs while minimizing a manufactured technique detection rate. By analysing trends in graph similarity metrics we highlighted the model's capability to align synthetic graphs to a reference training dataset, based on both node features and overall graph structure. This analysis also unveiled limitations in the current training methodology, largely tied to the sequential training of the discriminator. This training scheme led to the generator exploiting different graph features across training rounds, initially prioritizing graph structure and then transitioning to node features. To mitigate this, we propose a continuous training scheme, similar to typical GAN, whereby the discriminator and generator are trained concurrently. This strategy is anticipated to smooth the abrupt transition between training rounds, and produce HAGs with a more balanced alignment to the training data. We also suggest modifying the discriminator architecture to explicitly account for both graph structure and node features.

A limitation of the current approach is that the expressiveness of the generated HAGs is restricted, due to their alignment with a restricted training set of historical HAGs. While the generated HAGs describe a significantly larger attack surface than that of the training data, there remains a gap between the generated HAGs and the larger attack surface of all likely HAGs. This gap means that any additional defence mechanisms incorporated will not be fully equipped to defend against all conceivable attacks, but only those that are encapsulated by the generated HAGs. This limitation can be minimized by broadening the size and diversity of the training HAG dataset.

Future work will focus on various down-sampling methods for dense graphs, more sophisticated targeted attributes such as energy expenditure based on real world data, determining the GCDQ model's sensitivity to larger and more diverse graphs, and validating the post-processing step whereby multiple sparse HAGs are combined. This comprehensive approach will further improve our understanding of the GCDQ models capabilities, and create a robust tool capable of generating diverse HAGs with user defined traits.

ACKNOWLEDGMENTS

The research described in this paper is part of the Resilience Through Data-Driven, Intelligently Designed Control (RD2C) Initiative at Pacific Northwest National Laboratory. It was conducted under the Laboratory Directed Research and Development Program at PNNL, a multiprogram national laboratory operated by Battelle for the U.S. Department of Energy.

We also thank the anonymous reviewers for their insightful feedback which was incorporated into this work.

REFERENCES

[1] Esben Jannik Bjerrum and Richard Threlfall. [n. d.]. Molecular Generation with Recurrent Neural Networks (RNNs). https://doi.org/10.48550/arXiv.1705.04612 arXiv:1705.04612 [cs, q-bio]

[2] The MITRE Corporation. [n. d.]. MITRE ATT&CK Framework. https://attack.mitre.org/ Last accessed February 2023.

[3] Corinna Coupette and Jilles Vreeken. 2021. Graph Similarity Description. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery &amp Data Mining*. ACM. https://doi.org/10.1145/3447548.3467257

[4] Ashutosh Dutta, Sumit Purohit, Arnab Bhattacharya, and Oceane Bel. 2022. Cyber Attack Sequences Generation for Electric Power Grid. In *2022 10th Workshop on Modelling and Simulation of Cyber-Physical Energy Systems (MSCPES)*. IEEE, 1–6.

[5] Mathias Edman and Neil Dhir. 2019. Boltzmann Exploration Expectation-Maximisation. arXiv:1912.08869 [stat.ML]

[6] Timo Gervens and Martin Grohe. 2022. Graph Similarity Based on Matrix Norms. arXiv:2207.00090 [cs.DM]

[7] William L. Hamilton, Rex Ying, and Jure Leskovec. 2018. Inductive Representation Learning on Large Graphs. arXiv:1706.02216 [cs.SI]

[8] Matteo Hessel, Joseph Modayil, Hado van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, and David Silver. 2017. Rainbow: Combining Improvements in Deep Reinforcement Learning. arXiv:1710.02298 [cs.AI]

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

[10] Renjie Liao, Yujia Li, Yang Song, Shenlong Wang, Charlie Nash, William L. Hamilton, David Duvenaud, Raquel Urtasun, and Richard S. Zemel. [n. d.]. Efficient Graph Generation with Graph Recurrent Attention Networks. https://doi.org/10.48550/arXiv.1910.00760 arXiv:1910.00760 [cs, stat]

[11] Timothy P Lillicrap and et al. 2015. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971* (2015).

[12] Chang Liu and Jianping Li. 2021. Distance signless Laplacian spectral radius and perfect matching in graphs and bipartite graphs. arXiv:2104.01288 [math.CO]

[13] Sushovan Majhi and Carola Wenk. 2022. Distance Measures for Geometric Graphs. arXiv:2209.12869 [cs.CG]

[14] MITRE Corporation. Accessed: 2023. MITRE ATT&CK Database. https://attack.mitre.org/matrices/enterprise/.

[15] Richard S. Sutton and Andrew G. Barto. 1998. *Reinforcement Learning: An Introduction*. MIT Press. http://www.cs.ualberta.ca/~sutton/book/the-book.html

[16] Hado van Hasselt, Arthur Guez, and David Silver. 2015. Deep Reinforcement Learning with Double Q-learning. arXiv:1509.06461 [cs.LG]

[17] Jiaxuan You, Bowen Liu, Zhitao Ying, Vijay Pande, and Jure Leskovec. 2018. Graph convolutional policy network for goal-directed molecular graph generation. *Advances in neural information processing systems* 31 (2018).

[18] Jiaxuan You, Rex Ying, Xiang Ren, William Hamilton, and Jure Leskovec. 2018. Graphrnn: Generating realistic graphs with deep auto-regressive models. In *International conference on machine learning*. PMLR, 5708–5717.

A ADDITIONAL EXPERIMENTAL DETAILS

A.1 Sparse HAG Extraction

In order a extract a sparse HAG from a dense HAG, we apply a simple random walk algorithm with back tracking. A random node is first selected from the lowest tactic ID present in the dense HAG, N_1. This node is then added to the sparse HAG and becomes the source node N_s. With probability P we then select a node connected to N_s with an increasing tactic ID from the dense HAG and add it to the sparse HAG, this node N_2 then becomes the new source node N_s. Else, with probability $1-P$, we move the source node to a random node within the existing sparse HAG. This process continues until a sparse HAG containing 6 unique nodes is produced. In all experiments, a value of $P = 0.25$ was used. Example sparse and dense HAGs are displayed within figs. 1 and 2.

A.2 Generator Model Architecture

The GNN portion of the generator model incorporates three SAGEConv layers, each using a LeakyReLU activation function with negative slope of 0.1. The input dimensions are 208, 32, 32 respectively, outputting a embedding vector of size 32. The inital input of 208 corresponds to the node encoding dimension, based on the concatenation of the one hot encoded node techniques (193) and tactics (14) used to represent a given technique.

The MLP portion of the generator, mapping pairs of embedded nodes to action Q values, comprises of 5 fully connected layers of 64, 128, 128, 128, 128 units. The LeakyReLU activation function is again used with negative slope of 0.1. A final fully connected layer is used produce a singular Q value. All weights are initialized via a Xavier uniform scheme, with biases initialized to zero.

A.3 Discriminator Model Architecture

The discriminator model uses two GCNLayers and ReLU activation, with input dimensions 208 and 32, and outputs a embedding vector of size 32. Global mean pooling across dimension zero produces a single vector of size 32, which is mapped to a classification probability via a fully connected layer and sigmoid function. This classification probability denotes the probability that the graph is sourced from the real training data. All weights are initialized via a Xavier uniform scheme, with biases initialized to zero.

A.4 Hyperparameters

A.4.1 Fixed Hyperparameters. In all experiments, the discount factor was set to $\gamma = 0.9$, the generator was trained for 2000 epochs, batch size = 8, graphs were generated up to a maximum of 6 nodes, polyak averaging coefficient $\beta = 0.01$, and the replay buffer size was set to 2500. Upon initialization, 500 generation steps were conducted to populate the replay buffer. Following initialization, 5 graphs of maximum size 6 were then repeatedly generated and stored within the buffer followed by 2 rounds of DQN training via uniform replay buffer sampling with replacement. The Boltzmann temperature was decayed during training, from an inital value of $\tau_0 = 2$ to τ_f according to equation eq. (10). Decay point = $\frac{2}{3}$ was used to determine the point at which temperature decays to its minimum value τ_f, corresponding to epoch 1333.

$$\tau = \max\left[\tau_0 \times \left(\frac{\tau_f}{\tau_0}\right)^{\frac{\text{episode}}{2000\times\text{decay point}}}, \tau_f\right] \tag{10}$$

The discriminator is trained with a constant learning rate of $\alpha = 6e-4$ until a training and validation binary cross entropy loss of 0.4 and 0.45 are reached respectively (averaged over 10 sequential epochs). The the Adam optimizer was used for training both the generator and discriminator. When retraining the discriminator, a temperature value τ must be chosen to generate synthetic graphs. This value was chosen through inspection of the associated Boltzmann temperature plots fig. 5, with a temperature value chosen that corresponded with the inflection point of the model diversity. For all experiments, a value of $\tau = 0.025$ was chosen as it balanced exploitation of the learnt reward function with model diversity.

A.4.2 Searched Hyperparameters. For each round of generator training, the following hyperparameters were tested via a grid search. Learning rate $\alpha = [0.01, 0.003, 0.001, 0.0003, 0.0001]$, minimum Boltzmann temperature $\tau_f = [0.01, 0.005, 0.0025, 0.001]$, reward identification alpha $\alpha_2 = [1, 0.5, 0.25]$. Models were then evaluated based on the validation metrics used within table 1, with the models corresponding with maximum detection and discriminator reward displayed within the body of the paper.

B EVALUATION METRICS

B.1 Boltzmann Temperature Sweep

For a given trained generator model, 1000 graphs containing between two and six nodes were generated for various temperatures. The individual reward components were then averaged, and plotted along with the graph diversity score for their corresponding temperature. Graph diversity was calculated through the proportion of unique graphs within the set of 1000 generations.

B.2 Graph Edit Distance

Graph Edit Distance (GED) measures dissimilarity between two graphs by calculating the number of operations required to transform one graph into another. These operations include node and edge insertions, deletions, or substitutions and is approximated by the NetworkX library `optimize_graph_edit_distance` function.

B.3 Jaccard Similarity Score

The Jaccard Similarity score is calculated based on the node features, and is done so through dividing the size of the node feature intersection set by the size of the node feature union set.

B.4 Laplacian Spectral Distance

The Laplacian Spectral Distance (LSD) calculates the eigenvalues of the Laplacian matrices of the two graphs and then measures the distance between them. Practically this is done through first computing the Laplacian of the adjacency matrices via the `laplacian_matrix` function from the NetworkX library. The euclidean distance of each eigenvalue pair from the two graphs is then calculated and averaged, resulting in an average spectral distance. The intuition behind this method is that two graphs with similar structures will also have a similar Laplacian spectra and small spectral distance.

B.5 Frobenius Norm

The Frobenius norm is commonly used to compare two matrices, and is defined as the square root of the sum of the absolute squares of its elements. To compare two adjacency matrices, they are first subtracted to form a difference matrix, with the Frobenius norm then applied to give a distance metric between the two graphs. Of note the Frobenius norm is sensitive to arbitrary node ordering within an adjacency matrix, therefore all possible arrangements are evaluated to find the minimum configuration. For a graph containing 6 nodes, this results in $6! = 720$ calculations.

C SUPPLEMENTARY RESULTS

Additional training results displaying generator reward and loss for round one, along with the generator loss for round two are provided within figs. 6 to 8

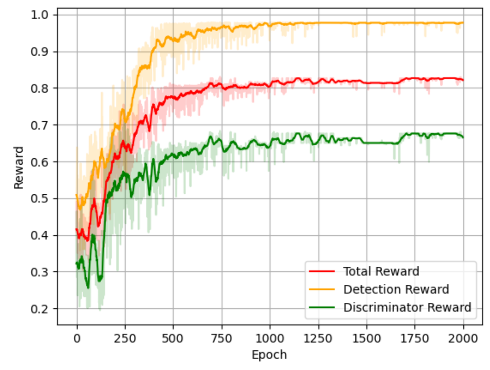

Figure 6: Reward convergence (round 1)

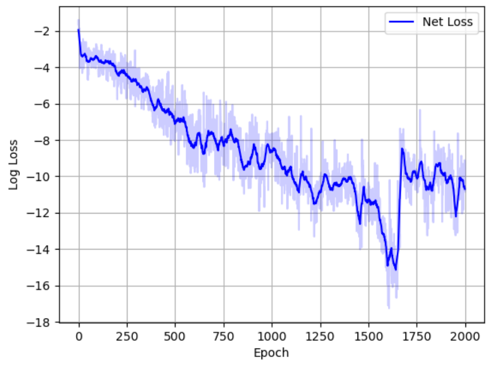

Figure 7: DQN Loss (round 1)

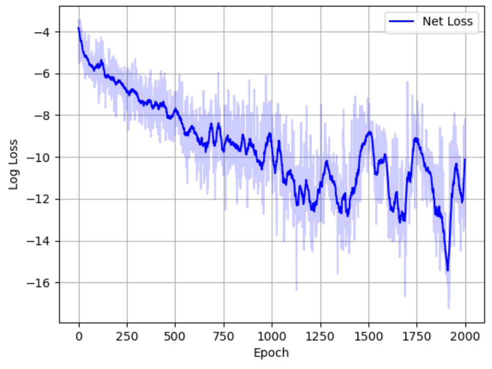

Figure 8: DQN Loss (round 2)