Imperceptible Adversarial Attacks on Discrete-Time Dynamic Graph Models

Anonymous Author(s)
Affiliation
Address
email

Abstract

Real-world graphs such as social networks, communication networks, and rating networks are constantly evolving over time. Many architectures have been developed to learn effective node representations using both graph structure and its dynamics. While the robustness of static graph models is well-studied, the vulnerability of the dynamic graph models to adversarial attacks is underexplored. In this work, we design a novel attack on discrete-time dynamic graph models where we desire to perturb the input graph sequence in a manner that preserves the structural evolution of the graph. To this end, we motivate a novel Time-Aware Perturbation (TAP) constraint, which ensures that perturbations introduced at each time step are restricted to only a small fraction of the number of changes in the graph since the previous time step. We present a theoretically-grounded Projected Gradient Descent approach for dynamic graphs to find the effective perturbations under the TAP constraint. Experiments on dynamic link prediction show that our approach is up to 4x more effective than the baseline methods for attacking these models under the novel constraint. Dyn-PGD also outperforms the existing baselines on the node classification task by up to 6x and is 2x more efficient in running time than the best-performing baseline.

1 Introduction

Graph Neural Networks (GNNs) have been shown to be vulnerable to adversarial perturbations [15, 2, 10, 39, 45, 23]. This has raised major concerns against their use in important industrial applications such as friend/product recommendation [43, 33, 36] and fraud detection [44, 14]. However, these advancements in designing attack and defense mechanisms have predominantly focused on GNN models for static, non-evolving graphs. In reality, the graph structure evolves with time as new interactions happen and new connections are formed [21, 18]. GNN models that incorporate the temporal information are shown to outperform their static counterparts in modeling dynamic networks on tasks such as predicting link existence in the future [16, 30, 34, 13, 7].

However, the vulnerability of dynamic graph models to adversarial perturbations is less studied. The design of adversarial attacks for dynamic graphs is challenging for two reasons — (1) Attacks must simultaneously optimize both the edge(s) to perturb and when to perturb them, and more importantly, (2) Attacks must preserve the original graph evolution after perturbation in order to be less detectable. In particular, attacks that disturb original graph evolution are not desired since they can be detected as anomalies by defense mechanisms, e.g. graph anomaly detection methods [1]. Therefore, it is crucial to formulate adversarial attacks over snapshots such that they do not significantly alter the original change in the graph structure.

In this work, we introduce a novel **T**ime-**A**ware **P**erturbation (TAP) constraint to formulate imperceptible attacks on discrete-time dynamic graphs. This constraint asserts that the number of

Submitted to 36th Conference on Neural Information Processing Systems (NeurIPS 2022). Do not distribute.

Method	Dynamic	White-box	Evasion	Targeted	Evolution-aware
PGD [42]		✓	✓	✓	
IG-JSMA [42]		✓	✓	✓	
Fan et al. [11]	✓		✓		
Dyn-Backdoor [8]	✓	✓			
TGA [9]	✓	✓	✓	✓	
Dyn-PGD (proposed)	✓	✓	✓	✓	✓

Table 1: Comparison of our attack with existing works on graph adversarial attacks.

modifications added at a timestep should only be a small fraction of the actual number of changes with respect to the preceding timestep. We theoretically show that perturbations made under TAP constraint preserves the rate of change both in the structural and the embedding spaces. To find effective attacks under this proposed constraint, we consider a targeted, white-box, and evasion setting. As noted in Table 1, prior works are unable to conduct attacks on dynamic graphs under this setting. Thus, we present a theoretically-grounded Projected Gradient Descent approach for dynamic graphs (Dyn-PGD) following the TAP constraint. Experiments on 4 datasets and 3 dynamic graph models (victim models) show that the proposed Dyn-PGD is up to 4x more effective than the baseline attacks on dynamic link prediction and node classification tasks. We further propose a new metric — Embedding Variability (EV), as a measure to assess the relative variability in the perturbed embedding space. Our contributions can be summarized as follows:

1. We introduce a novel Time-Aware Perturbation (TAP) constraint to make less detectable perturbations in discrete-time dynamic graphs.
2. We show theoretically that the proposed TAP constraint preserves the original rate of the structural change and embedding change in the dynamic graphs.
3. We present a theoretically-grounded PGD-based white-box attack to find effective imperceptible attacks on dynamic graphs under the novel TAP constraint.
4. We show that Dyn-PGD outperforms the baselines across 4 different datasets and 3 victim models on both dynamic link prediction and node classification tasks.

2 Related Work

Representation Learning for Dynamic Graphs. GNNs have been combined with sequential modeling architectures [16] to model dynamic graphs. For instance, discrete-time graphs have been modeled by using GNNs and RNNs together in a pipeline [28, 27] or an embedded manner [7, 30]. Attention-based models have also been proposed to jointly encode the graph structure and its dynamics [34]. For continuous-time graphs, both RNN [20, 37, 38, 24] and attention-based models [31, 40] have been proposed such that embeddings are updated in real time, upon an occurrence of a new event such as the addition/deletion of an edge.

Adversarial attacks on graphs. Static GNNs are known to be vulnerable to adversarial attacks in different settings [15]. White-box attacks are studied assuming complete knowledge of the underlying model [39, 42]. Limiting the model knowledge, gray-box [45] and black-box attacks [10] have also been proposed. In comparison, the literature on adversarial attacks for dynamic graphs is scarce. Time-aware Gradient Attack (TGA) [9] is a white-box evasion attack which greedily selects the perturbations across time under a budget constraint. In addition, attacks to poison training data [8] and black-box attacks using RL approaches [11] have also been proposed.

Imperceptible perturbations. The most common strategy to formulate imperceptible attacks on graphs is to bound the total number of perturbations. Various strategies, in addition to the budget constraint, have been developed to make the attacks imperceptible for static graphs. These include rewiring the perturbed edges [25, 26] , only perturbing the low degree nodes [23], and preserving the degree/feature distribution statistics [45]. In the case of dynamic graphs, perturbations must preserve the temporal flow to be imperceptible. Traditional anomaly detection algorithms flag an instance to be anomalous if distance between consecutive snapshots crosses a threshold [1]. In particular, Graph Edit Distance and Hamming distance between adjacency matrices have been used to monitor

communication networks [35, 5]. Neural approaches have looked at the consecutive change in the embedding space to detect anomalies without feature extraction [13, 6].

3 Methodology

Problem Let $\mathcal{G}_1, \mathcal{G}_2, \cdots, \mathcal{G}_T$ be the original graph snapshots and $\mathcal{G}'_1, \mathcal{G}'_2 \cdots, \mathcal{G}'_T$ be the corresponding perturbed snapshots. Note that $\mathcal{G}_i = (\boldsymbol{\mathcal{X}}_i, \boldsymbol{\mathcal{A}}_i)$ where $\boldsymbol{\mathcal{X}}_i, \boldsymbol{\mathcal{A}}_i$ are the node features and the adjacency matrix for snapshot i, respectively. Also, let $\mathcal{M}$ be a victim dynamic graph model that we want to attack and let $f_{\mathcal{M}}$ be a function that generates the corresponding node embeddings of $\mathcal{G}_t$ given $\mathcal{G}_{1:t-1}$. Let y_{task} be the actual labels for a given task (for dynamic link prediction, these correspond to binary labels representing link existence in the future snapshot).

Then, the objective of the attacker is to introduce structural perturbations $\mathbf{S}_t = \boldsymbol{\mathcal{A}}'_t - \boldsymbol{\mathcal{A}}_t$ at each timestep $t < T$ such that the model inference at timestep T for the target entities E_{tg} deteriorates. More formally, the attacker solves the following optimization problem:

$$\begin{aligned} \max_{\boldsymbol{\mathcal{A}}'_1, \boldsymbol{\mathcal{A}}'_2, \cdots, \boldsymbol{\mathcal{A}}'_{T-1}} \quad & \mathcal{L}_{\text{task}}\left(\widehat{y}_{\text{task}}(f_{\mathcal{M}}(\boldsymbol{\mathcal{A}}'_{1:T-1})), y_{\text{task}}, E_{tg}\right) \\ \text{such that} \quad & \mathcal{C}(\boldsymbol{\mathcal{A}}'_{1:T-1}) \text{ holds} \end{aligned} \tag{1}$$

for some constraint function $\mathcal{C}$ on the perturbed adjacency matrices $\boldsymbol{\mathcal{A}}'_t$ for each time t. Here, $\widehat{y}_{\text{task}}$ denotes the predicted labels for the given task and $\mathcal{L}_{\text{task}}$ is a task-specific loss, for example, a binary cross entropy (CE) loss for link prediction.

The constraint function $\mathcal{C}$ is designed to ensure imperceptibility of the adversarial perturbations. In the literature, a budget constraint has been widely used to enforce imperceptibility in graphs [10] and computer vision [12]. However, this constraint only bounds the total amount of perturbations that can be introduced by an attacker. When the input is dynamic, as in the case of dynamic graphs, the perturbations should be constrained in the context of how the input evolves. However, since the budget constraint completely ignores the graph dynamics, it could lead to a drastic change in the evolution trend of the graph, violating the imperceptibility requirement. For instance, with the budget constraint, all the perturbations can be made at a single time step, leading to an anomalous spike, which would be easily detected as a possible attack by graph anomaly detection methods for dynamic graphs [1, 35, 5]. Thus, a constraint is desired which would ensure that the perturbations do not disrupt the evolving trend of the dynamic graphs.

3.1 Time-Aware Perturbation (TAP) Constraint

The simplest measure to study evolution is to consider the change in the input between consecutive time steps. Thus, for a discrete-time input $\{\mathbf{x}_t\}$, this corresponds to considering the discrete-time differential norm at time t, given by $d\mathbf{x}_t = \|\mathbf{x}_t - \mathbf{x}_{t-1}\|$. Then, we propose

Proposition 1 *The number of perturbations introduced to input* $\mathbf{x}$ *at time step* t *must not be more than a fraction* ϵ *times the differential at* t*, i.e.* $TAP(\epsilon) := \|\mathbf{x}'_t - \mathbf{x}_t\| \leq \epsilon d\mathbf{x}_t \ \forall t$.

For the case of dynamic graphs, when the graph structure evolves (for example, in social networks and transaction networks [30, 34]), this constraint becomes $\|\boldsymbol{\mathcal{A}}'_t - \boldsymbol{\mathcal{A}}_t\|_1 \leq \epsilon d\boldsymbol{\mathcal{A}}_t$. Alternatively, a dynamic graph may also involve a temporally-evolving signal at each node [32, 22, 29], in which case, this constraint becomes $\|\boldsymbol{\mathcal{X}}'_t - \boldsymbol{\mathcal{X}}_t\|_1 \leq \epsilon d\boldsymbol{\mathcal{X}}_t$.

In this work, we focus on dynamic graph structures such that the constraint $\mathcal{C}$ (in Equation 1) for the perturbations is a TAP constraint. The optimization problem for the attacker, thus, becomes

$$\begin{aligned} \max_{\boldsymbol{\mathcal{A}}'_1, \boldsymbol{\mathcal{A}}'_2, \cdots, \boldsymbol{\mathcal{A}}'_{T-1}} \quad & \mathcal{L}_{\text{task}}\left(\widehat{y}_{\text{task}}(f_{\mathcal{M}}(\boldsymbol{\mathcal{A}}'_{1:T-1})), y_{\text{task}}, E_{tg}\right) \\ \text{such that } \forall t \in (1, T): & \frac{\|\boldsymbol{\mathcal{A}}'_t - \boldsymbol{\mathcal{A}}_t\|}{\|\boldsymbol{\mathcal{A}}_t - \boldsymbol{\mathcal{A}}_{t-1}\|} \leq \epsilon \\ & \|\boldsymbol{\mathcal{A}}'_1 - \boldsymbol{\mathcal{A}}_1\| \leq \varepsilon_1, \end{aligned} \tag{2}$$

where ϵ, ε_1 are given parameters for this optimization. We use $\|\cdot\|$ to denote the 1-norm of the matrix flattened into a vector, unless otherwise mentioned.

Implications. We show that TAP constraint has the following implications on the perturbations:

1. **Perturbations made under TAP constraint preserves the average rate of structural change.**
   Theorem 1 *Let* $\overline{d\mathcal{A}} = \frac{1}{T}\sum_t d\mathcal{A}_t, \overline{d\mathcal{A}'} = \frac{1}{T}\sum_t d\mathcal{A}'_t$. *Then,*
   $$\overline{d\mathcal{A}'} \leq \alpha\overline{d\mathcal{A}} + \beta, \tag{3}$$
   for some constants $\alpha, \beta \in \mathbb{R}_{\geq 0}$.
   Proof. By the definition of the perturbation matrix $\mathbf{S}_\tau$, $d\mathcal{A}'_t = \|\mathcal{A}'_t - \mathcal{A}'_{t-1}\| = \|(\mathcal{A}_t + \mathbf{S}_t) - (\mathcal{A}_{t-1} + \mathbf{S}_{t-1})\|$. Then, using triangle inequality, we get $d\mathcal{A}'_t \leq \|\mathcal{A}_t + \mathbf{S}_t\| + \|\mathcal{A}_{t-1} + \mathbf{S}_{t-1}\|$. Again using triangle inequality, $d\mathcal{A}'_t \leq \|\mathcal{A}_t\| + \|\mathbf{S}_t\| + \|\mathcal{A}_{t-1}\| + \|\mathbf{S}_{t-1}\|$. Now, since $\|\mathbf{S}_\tau\| \leq \epsilon d\mathcal{A}_t$ and $\|\mathcal{A}_\tau\|$ is a constant, we get $d\mathcal{A}'_t \leq \epsilon d\mathcal{A}_t + \epsilon d\mathcal{A}_{t-1} + C$, for some constant C.
   $\overline{d\mathcal{A}'} = \frac{1}{T}\sum_t d\mathcal{A}'_t \leq \frac{1}{T}\sum_t (\epsilon d\mathcal{A}_t + \epsilon d\mathcal{A}_{t-1} + C) \leq 2\epsilon\frac{1}{T}\sum_t d\mathcal{A}_t + C$. Hence, we get that $\overline{d\mathcal{A}'} \leq \alpha\overline{d\mathcal{A}} + \beta$, for some constants $\alpha, \beta \geq 0$. ■

2. **Perturbations made under TAP constraint preserves the rate of embedding change.**
   Theorem 2 *Let* $d\mathbf{Z}_t = \|\mathbf{Z}_t - \mathbf{Z}_{t-1}\|_1$. *Then, for some constants* $\gamma, \delta \in \mathbb{R}_{\geq 0}$,
   $$d\mathbf{Z}' \leq \gamma d\mathbf{Z} + \delta, \tag{4}$$
   Proof. Note that $\mathbf{Z}_t = f(\mathcal{A}_t; \mathcal{A}_{t-1}, \cdots, \mathcal{A}_1)$. Thus, we consider a stacked vector of flattened matrices $\mathbf{q}_{\leq\tau} = (\mathbf{q}_\tau, \mathbf{q}_{\tau-1}, \cdots, \mathbf{q}_1, \mathbf{0}, \mathbf{0}, \cdots, \mathbf{0})$, where $\mathbf{q}_i$ is the flattened vector of $\mathcal{A}_i$ and we prepend $(t-\tau)$ $\mathbf{0}$s to make all vectors $\mathbf{q}_{\leq i}$ of fixed dimension t. Then, by Cauchy's Mean Value Theorem in several variables, we have $\mathbf{Z}_t - \mathbf{Z}_{t-1} \leq \nabla f \cdot (\mathbf{q}_{\leq t} - \mathbf{q}_{\leq t-1})$, which gives us $\|\mathbf{Z}_t - \mathbf{Z}_{t-1}\| \leq \|\nabla f\| \, \|\mathbf{q}_{\leq t} - \mathbf{q}_{\leq t-1}\|$ by Cauchy-Schwarz inequality. We note that $\|\mathbf{q}_{\leq t} - \mathbf{q}_{\leq t-1}\|_1 = \|(\mathbf{q}_t - \mathbf{q}_{t-1}, \mathbf{q}_{t-1} - \mathbf{q}_{t-2}, \cdots, \mathbf{q}_2 - \mathbf{q}_1, \mathbf{q}_1\|_1 = \sum_t \|\mathbf{q}_t - \mathbf{q}_{t-1}\|_1 = \sum_t \|\mathbf{q}_t - \mathbf{q}_{t-1}\|_1 = T\overline{d\mathcal{A}}$. Thus, we have $\|\mathbf{Z}_t - \mathbf{Z}_{t-1}\|_F \leq C\overline{d\mathcal{A}}$ for some constant $C \geq 0$. Using Theorem 1, we get $d\mathbf{Z}' \leq C\overline{d\mathcal{A}'} \leq A\overline{d\mathcal{A}} + B$, for $A = C\alpha, B = C\beta$. By mean-value theorem, we also have $d\mathbf{Z}_t \geq \|\nabla f_{t-1}\| \, \|\mathbf{q}_{\leq t} - \mathbf{q}_{\leq t-1}\| \geq C_2\overline{d\mathcal{A}}$. Hence, $d\mathbf{Z}' \leq \gamma d\mathbf{Z} + \delta$ for some constants $\gamma, \delta \geq 0$. ■

3.2 Attack methods under TAP constraint

While the TAP constraint allows us to limit the effect of the perturbations on the graph's evolution, it is not clear how one can efficiently find perturbations that maximize a loss function under this constraint. To this end, we present two algorithms to solve the optimization problem of Equation 2.

Greedy. A greedy strategy can be adopted to find effective perturbations under our TAP constraint. In this approach, perturbations are selected in a greedy manner based on their gradient values with respect to the downstream loss. However, this does not scale well as one needs to find gradient values corresponding to all the perturbations, which would be $O(T|\mathcal{V}|)$, where $\mathcal{V}$ denotes the set of nodes. Thus, inspired by [9], we find the perturbations in two steps — first, we find the top-gradient perturbation at each time step and then, select the one that reduces the prediction probability the most. In particular, we greedily select the perturbations with the lowest probability such that TAP(ϵ) is not violated for any time-step. We defer the full algorithm to Appendix A.

Projected Gradient Descent for Dynamic Graphs (Dyn-PGD). Since the constrained optimization in Equation 2 has a general continuous objective, a greedy approach is only sub-optimal (even for a simpler convex objective) with no theoretical guarantees. A more standard approach to do optimization under a convex constraint is to use projected gradient descent (PGD) [3, 4]. Since our problem is in discrete-space, we first relax it into continuous space, find the solution using PGD and then, randomly round it to obtain a valid solution for the discrete problem. In particular, we relax the perturbation matrix $\mathbf{S}_t$ into a continuous vector $\mathbf{s}_t$ and show that a closed-form projection operator exists for the TAP(ϵ) constraint. Algorithm 1 demonstrates the steps involved in this approach (Dyn-PGD), following the result of Theorem 3.

Theorem 3 *Suppose* $\mathcal{S}$ *denotes the feasible perturbation space for the constraints* $\|\mathcal{A}'_t - \mathcal{A}_t\|/\|\mathcal{A}_t - \mathcal{A}_{t-1}\| \leq \epsilon$ *for all* $1 < t < T$ *and* $\|\mathcal{A}'_1 - \mathcal{A}_1\| \leq \varepsilon_1$. *Then, one can project a vector* $\mathbf{a}_t$ *onto* $\mathcal{S}$ *using the following projection operator:*
$$\Pi_{\mathcal{S}}(\mathbf{a}_t) = \begin{cases} P_{[0,1]}(\mathbf{a}_t - \mu_t) & \text{if } \exists \mu_t > 0: \ \mathbf{1}^T P_{[0,1]}(\mathbf{a}_t - \mu_t) = \varepsilon_t \\ P_{[0,1]}(\mathbf{a}_t) & \text{if } \mathbf{1}^T P_{[0,1]}(\mathbf{a}_t) \leq \varepsilon_t \end{cases} \tag{5}$$
where $\varepsilon_t = \epsilon d\mathcal{A}_t = \epsilon\|\mathcal{A}_t - \mathcal{A}_{t-1}\|$ *for* $t > 1$, *and* $P_{[0,1]}(x) = x$ *if* $x \in [0,1]$, 0 *if* $x < 0$, *and* 1 *if* $x > 1$.

Algorithm 1 Projected Gradient Descent for dynamic graphs

Require: TAP variables ε_t (from Theorem 3), Initial perturbation vector $\mathbf{s}^{(0)}$, Loss function $\mathcal{L}_{task}$, actual labels y_{task}, Target entities E_{tg}, Time steps T, Initial learning rate η_0, Iterations N

Ensure: Perturbation vector $\mathbf{s}^{(i)}$ preserves TAP(ϵ) at every time step t

1: **for** $i = 1$ to N **do**

2: **Gradient descent**: $\mathbf{a}^{(i)} = \mathbf{s}^{(i-1)} - \eta_i \nabla \mathcal{L}_{task}(\{\mathcal{G}_t \oplus \mathbf{s}_t^{(i-1)};\ \forall t\},\ y_{task},\ E_{tg})$

3: **Projection**: For all $t \in [1, T-1]$: $\mathbf{s}_t^{(i)} = \Pi_{\mathcal{S}}(\mathbf{a}_t^{(i)})$ according to Equation 5

4: $\mathbf{S}_t \sim \text{Bernoulli}(s_t^{(N)})$ such that $\mathbf{S}_t \leq \varepsilon_t$ for all t

Proof. Please see Appendix B for the proof. ■

Following [42], we use the bisection method [3] to solve the equation $\mathbf{1}^T P_{[0,1]}(\mathbf{a}_t - \mu_t) = \varepsilon_t$ in μ_t for $\mu_t \in [\min(\mathbf{a}_t - 1), \max(\mathbf{a}_t)]$. This converges in the logarithmic rate, i.e. it takes $O(\log_2[(\max(\mathbf{a}_t) - \min(\mathbf{a}_t - 1))/\xi])$ time for ξ-error tolerance.

4 Experimental Setup

Datasets. We use 3 datasets for dynamic link prediction — `Radoslaw`[1], `UCI` [1], and `Reddit`[2]. `Radoslaw` and `UCI` are email communication networks, where two users are connected if they have an email communication at a given time. `Reddit` is a hyperlink network with directed connections between subreddits if there is a hyperlink from one to the other [19]. For node classification, we use one publicly-available dataset, `DBLP-5` [41]. This is a co-author network with node attributes as word2vec representations of the author's papers. There are 5 node labels representing the different fields that the authors belong to. Please refer to Appendix C for more details regarding the datasets.

Attack Methods. We consider 4 attack methods to find adversarial perturbations in our setting.

1. **Dyn-PGD**: Algorithm 1 shows the proposed approach of projected gradient descent with a valid projection operator for the TAP constraint.
2. **GREEDY**: Greedily selects the perturbation with the highest gradient value of the loss.
3. **DEGREE**: Flips the edges (adds or deletes if already there) attached with the highest degree nodes in the graph at each time step while making at most $\epsilon d \mathcal{A}_t$ perturbations.
4. **RANDOM**: Randomly flip (add or delete) at most $\epsilon d \mathcal{A}_t$ edges at each time step t.

Victim Models. We test the performance of the above on 3 discrete-time dynamic graph models.

1. **GC-LSTM** [7] embeds GCN into an LSTM to encode the sequence of graphs.
2. **EVOLVEGCN** [30] uses a recurrent model (RNN-LSTM) to evolve the weights of a GCN. We use the EVOLVEGCN-O version for our experiments.
3. **DYSAT** [34] is an attention-based architecture utilizing joint structural and temporal self-attention.

Metrics. We use the relative drop, as defined below, to evaluate the efficacy of the attack methods.

$$\text{Rel. Drop (\%)} = \frac{\text{Perturbed performance} - \text{Original performance}}{\text{Original performance}} \times 100, \tag{6}$$

where performance is evaluated using ROC-AUC for dynamic link prediction and using Accuracy for node classification.

In order to evaluate the detectability of the attack methods, we propose a novel metric **Embedding Variability (EV)** to compare the consecutive embedding difference for the perturbed graph and that for the original graph. Consecutive embedding difference has been used to identify anomalies in the data [13]. Here, we measure how the range of this difference changes due to the perturbation. In particular, we consider

$$EV(\mathbf{Z}, \mathbf{Z}') := \left| 1 - \frac{\max_\tau d\mathbf{Z}'_\tau - \min_\tau d\mathbf{Z}'_\tau}{\max_\tau d\mathbf{Z}_\tau - \min_\tau d\mathbf{Z}_\tau} \right| \tag{7}$$

[1] http://konect.cc/networks/

[2] https://snap.stanford.edu/data/soc-RedditHyperlinks.html

This measures the relative variability of the consecutive change in the embedding space. For the attacks to be less detectable, this metric should be close to 0. Note that we do not directly optimize for this metric in our constraint.

5 Results

We compare the performance of different attack methods for the 3 victim models on link prediction and node classification tasks. For all the experiments, we vary ϵ from 0 to 1 and fix $\varepsilon_1 = \min_{t>1} \varepsilon_t := \epsilon d \boldsymbol{\mathcal{A}}_t$.

5.1 Dynamic Link Prediction

In this section, we show the attack performance on the task of dynamic link prediction [34, 30]. The task here is to predict whether a link (u, v) will appear or not at the future timestep. The objective of the attacker, thus, is to introduce perturbations in the past time steps to make the model mispredict link's existence in future. We test the victim models on the final snapshot for a set of target links. We consider 3 different sets of 100 positive and 100 negative random targets and show the mean relative ROC-AUC drop with error bars.

Figure 1 shows the performance of different attack methods on this task across different datasets and models. Dyn-PGD outperforms the other baselines in all cases, except in GC-LSTM model trained on `Reddit`. Moreover, Dyn-PGD is able to drop the AUC by upto 4 times the baselines and lead to $\sim 100\%$ drop in the AUC, completely flipping the prediction. We also note that Dyn-PGD often has a continuously decreasing slope and its performance saturates much later than the other baselines. The second best baseline is often GREEDY but in many cases, it is only as good as random. One can also note that EVOLVEGCN shows a larger drop than the other 2 models across all datasets. This may pertain to the lower model complexity of EVOLVEGCN compared to others.

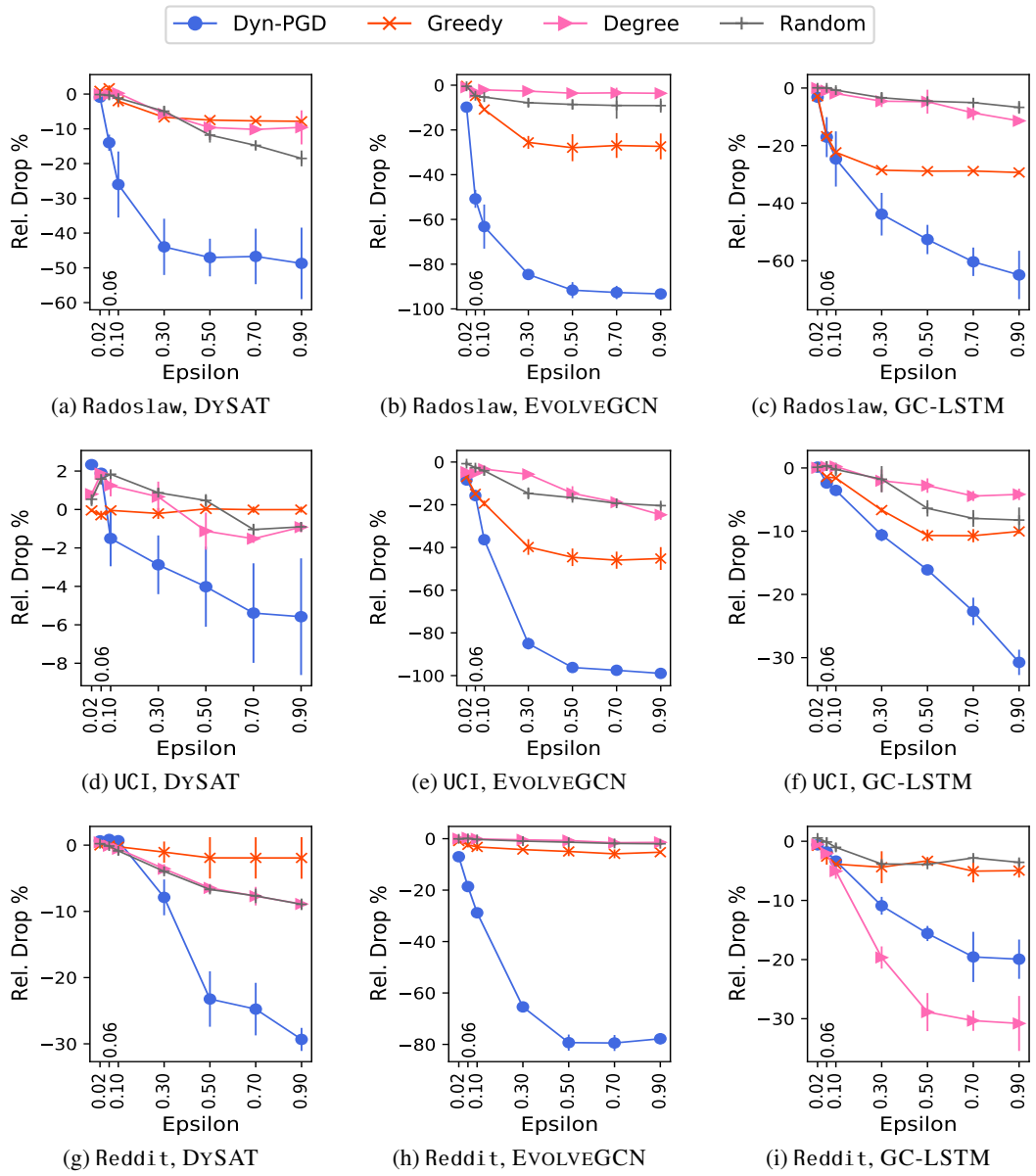
Detectability of the attack methods: Here, we use the EV metric, as defined in Section 4, to assess how detectable the attack methods are under our constraint. Table 2 compares the attack performance of the best method Dyn-PGD at $\epsilon = 0.5$ and the corresponding variability in the embeddings, as given by Equation 7. One can note that E is smaller than 0.5 in all but one case and that Dyn-PGD is able to cause up to 90% drop in performance while changing the evolution by only a factor of 0.22, on average. This shows that TAP allows for undetectable yet effective attacks.

5.2 Node Classification

In this section, we compare the attack performance on the task of semi-supervised node classification [30]. In this task, the objective is to predict node labels of a set of nodes while knowing the labels of the remaining nodes. In our experiments, we consider a $60 - 20 - 20$ split on the data. The objective of the attacker is to introduce perturbations at each snapshot under the TAP constraint such that the CE loss for the target node is maximized.

Dataset	Model	Rel. Drop %	EV
`Radoslaw`	DYSAT	47.03 (5.42)	0.04 (0.18)
	EVOLVEGCN	91.61 (3.58)	0.45 (1.29)
	GC-LSTM	52.63 (5.09)	0.18 (1.16)
`UCI`	DYSAT	4.02 (2.08)	0.14 (0.25)
	EVOLVEGCN	96.21 (0.17)	0.22 (0.35)
	GC-LSTM	16.12 (0.75)	0.35 (0.47)
`Reddit`	DYSAT	23.24 (4.18)	0.14 (0.45)
	EVOLVEGCN	79.31 (3.13)	0.81 (6.37)
	GC-LSTM	15.59 (1.28)	0.36 (0.59)

Table 2: Comparison of attack performance and detectability (refer Section 4) for Dyn-PGD at $\epsilon = 0.5$. Mean values are noted with standard deviations in the parentheses.



(a) Radoslaw, DySAT (b) Radoslaw, EvolveGCN (c) Radoslaw, GC-LSTM

(d) UCI, DySAT (e) UCI, EvolveGCN (f) UCI, GC-LSTM

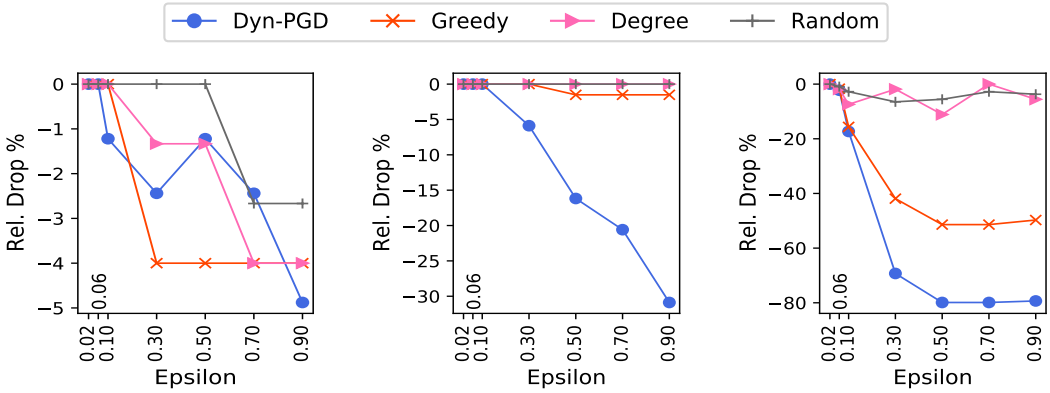
(g) Reddit, DySAT (h) Reddit, EvolveGCN (i) Reddit, GC-LSTM

Figure 1: Attack performance on dynamic link prediction task across datasets and models.

Figure 2 shows the effect of structural perturbations on node classification task by different attack strategies for the 3 models. Due to sparsity of edges in DBLP-5, randomly chosen targets only allow for a very small number of perturbations under the TAP constraint. Therefore, we consider the performance on 50 top-degree nodes for each class. Results show that Dyn-PGD outperforms the baselines in all models except DySAT, where all attacks perform almost equally. In particular, Dyn-PGD causes a 30% drop in EvolveGCN while the baselines only lead to a drop of 5%.

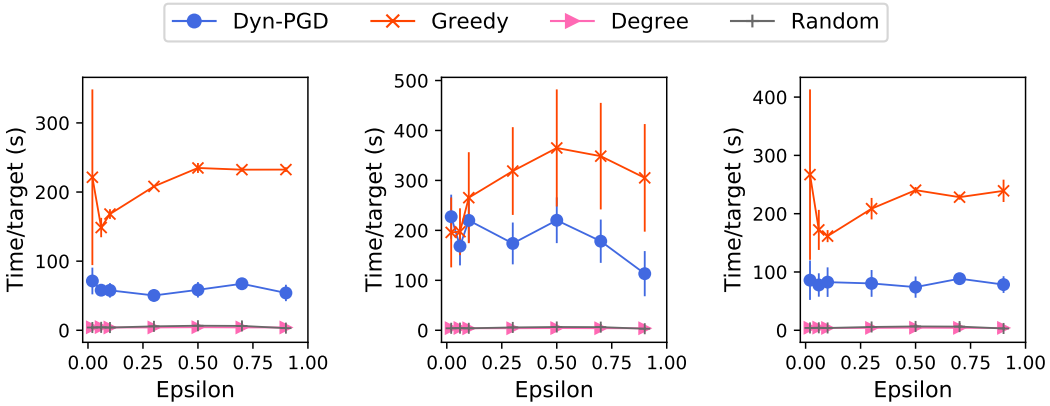
5.3 Running Time

Figure 3 compares the running time per target for different attack methods on the largest dataset, Reddit. The times are averaged over 200 targets from 3 different seeds and error bars note the standard deviation. Greedy is the most expensive method in terms of time and scales almost linearly with ϵ (capped at 300 s). Dyn-PGD takes around half the time than Greedy and remains constant with increase in ϵ.



(a) DBLP-5, DySAT (b) DBLP-5, EvolveGCN (c) DBLP-5, GC-LSTM

Figure 2: Attack performance on node classification task for top degree nodes across models.



(b) Reddit, DySAT (c) Reddit, EvolveGCN (d) Reddit, GC-LSTM

Figure 3: Running time of different attack methods on the largest dataset (Reddit)

6 Conclusion

Our work has shown that dynamic graph models can be attacked in an imperceptible manner. We introduce a novel Time-Aware Perturbation (TAP) constraint to devise imperceptible perturbations in discrete-time dynamic graphs, preserving the graph evolution within a factor. We hope that our work serves as a first step towards opening exciting research avenues on studying attacks and defense mechanisms for both discrete and continuous-time dynamic graphs. Some limitations of our current exposition can be noted. First, the proposed method Dyn-PGD is not memory-efficient as it stores a perturbation vector corresponding to each perturbation Second, randomly rounding the solution to discrete space may lead to suboptimal perturbations. Future work can study more effective and efficient methods to attack dynamic graphs under TAP constraint.

Potential negative societal impacts. Our work has demonstrated the vulnerability of dynamic graph models under a practical TAP constraint. This means that these models are not suitable for deployment in environments where adversarial attacks under a TAP constraint are realistic. If these models are deployed in such a setting, adversarial attacks proposed in this work can be used by an adversary to hamper the predictions. However, it must be noted that none of the models used in this work are known to be deployed in the real-world. Secondly, TAP constraint with a large ϵ may not be realistic in many applications. Thus, our study has an overall positive impact on the society as it allows practitioners to test the robustness of their models under a practical constraint before deploying in vulnerable environments. Although our attacks can be adopted by adversaries for negative use, it is important to put it out in the community so that the model designers are made aware of these attacks that are specifically designed to evade detection.

References

[1] Leman Akoglu, Hanghang Tong, and Danai Koutra. Graph based anomaly detection and description: a survey. *Data mining and knowledge discovery*, 29(3):626–688, 2015.

[2] Aleksandar Bojchevski and Stephan Günnemann. Adversarial attacks on node embeddings via graph poisoning. In *International Conference on Machine Learning*, pages 695–704. PMLR, 2019.

[3] Stephen Boyd, Stephen P Boyd, and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

[4] Sébastien Bubeck et al. Convex optimization: Algorithms and complexity. *Foundations and Trends® in Machine Learning*, 8(3-4):231–357, 2015.

[5] Horst Bunke, Peter J Dickinson, Miro Kraetzl, and Walter D Wallis. *A graph-theoretic approach to enterprise network dynamics*, volume 24. Springer Science & Business Media, 2007.

[6] Lei Cai, Zhengzhang Chen, Chen Luo, Jiaping Gui, Jingchao Ni, Ding Li, and Haifeng Chen. Structural temporal graph neural networks for anomaly detection in dynamic graphs. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*, pages 3747–3756, 2021.

[7] Jinyin Chen, Xueke Wang, and Xuanheng Xu. Gc-lstm: Graph convolution embedded lstm for dynamic link prediction. *arXiv preprint arXiv:1812.04206*, 2018.

[8] Jinyin Chen, Haiyang Xiong, Haibin Zheng, Jian Zhang, Guodong Jiang, and Yi Liu. Dyn-backdoor: Backdoor attack on dynamic link prediction. *arXiv preprint arXiv:2110.03875*, 2021.

[9] Jinyin Chen, Jian Zhang, Zhi Chen, Min Du, and Qi Xuan. Time-aware gradient attack on dynamic network link prediction. *IEEE Transactions on Knowledge and Data Engineering*, 2021.

[10] Hanjun Dai, Hui Li, Tian Tian, Xin Huang, Lin Wang, Jun Zhu, and Le Song. Adversarial attack on graph structured data. In *International conference on machine learning*, pages 1115–1124. PMLR, 2018.

[11] Houxiang Fan, Binghui Wang, Pan Zhou, Ang Li, Meng Pang, Zichuan Xu, Cai Fu, Hai Li, and Yiran Chen. Reinforcement learning-based black-box evasion attacks to link prediction in dynamic graphs. *arXiv preprint arXiv:2009.00163*, 2020.

[12] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.

[13] Palash Goyal, Nitin Kamra, Xinran He, and Yan Liu. Dyngem: Deep embedding method for dynamic graphs. *arXiv preprint arXiv:1805.11273*, 2018.

[14] Bryan Hooi, Kijung Shin, Hyun Ah Song, Alex Beutel, Neil Shah, and Christos Faloutsos. Graph-based fraud detection in the face of camouflage. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 11(4):1–26, 2017.

[15] Wei Jin, Yaxin Li, Han Xu, Yiqi Wang, Shuiwang Ji, Charu Aggarwal, and Jiliang Tang. Adversarial attacks and defenses on graphs: A review, a tool and empirical studies. *arXiv preprint arXiv:2003.00653*, 2020.

[16] Seyed Mehran Kazemi, Rishab Goel, Kshitij Jain, Ivan Kobyzev, Akshay Sethi, Peter Forsyth, and Pascal Poupart. Representation learning for dynamic graphs: A survey. *J. Mach. Learn. Res.*, 21(70):1–73, 2020.

[17] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[18] Gueorgi Kossinets and Duncan J Watts. Empirical analysis of an evolving social network. *science*, 311(5757):88–90, 2006.

[19] Srijan Kumar, William L Hamilton, Jure Leskovec, and Dan Jurafsky. Community interaction and conflict on the web. In *Proceedings of the 2018 World Wide Web Conference on World Wide Web*, pages 933–943. International World Wide Web Conferences Steering Committee, 2018.

[20] Srijan Kumar, Xikun Zhang, and Jure Leskovec. Predicting dynamic embedding trajectory in temporal interaction networks. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 1269–1278, 2019.

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

[22] Yaguang Li, Rose Yu, Cyrus Shahabi, and Yan Liu. Diffusion convolutional recurrent neural network: Data-driven traffic forecasting. *arXiv preprint arXiv:1707.01926*, 2017.

[23] Jiaqi Ma, Shuangrui Ding, and Qiaozhu Mei. Towards more practical adversarial attacks on graph neural networks. *Advances in neural information processing systems*, 33:4756–4766, 2020.

[24] Yao Ma, Ziyi Guo, Zhaocun Ren, Jiliang Tang, and Dawei Yin. Streaming graph neural networks. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 719–728, 2020.

[25] Yao Ma, Suhang Wang, Tyler Derr, Lingfei Wu, and Jiliang Tang. Attacking graph convolutional networks via rewiring. *arXiv preprint arXiv:1906.03750*, 2019.

[26] Yao Ma, Suhang Wang, Tyler Derr, Lingfei Wu, and Jiliang Tang. Graph adversarial attack via rewiring. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, pages 1161–1169, 2021.

[27] Franco Manessi, Alessandro Rozza, and Mario Manzo. Dynamic graph convolutional networks. *Pattern Recognition*, 97:107000, 2020.

[28] Apurva Narayan and Peter HO'N Roe. Learning graph dynamics using deep neural networks. *IFAC-PapersOnLine*, 51(2):433–438, 2018.

[29] George Panagopoulos, Giannis Nikolentzos, and Michalis Vazirgiannis. Transfer Graph Neural Networks for Pandemic Forecasting. In *Proceedings of the 35th AAAI Conference on Artificial Intelligence*, 2021.

[30] Aldo Pareja, Giacomo Domeniconi, Jie Chen, Tengfei Ma, Toyotaro Suzumura, Hiroki Kanezashi, Tim Kaler, Tao Schardl, and Charles Leiserson. Evolvegcn: Evolving graph convolutional networks for dynamic graphs. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 5363–5370, 2020.

[31] Emanuele Rossi, Ben Chamberlain, Fabrizio Frasca, Davide Eynard, Federico Monti, and Michael Bronstein. Temporal graph networks for deep learning on dynamic graphs. *arXiv preprint arXiv:2006.10637*, 2020.

[32] Benedek Rozemberczki, Paul Scherer, Oliver Kiss, Rik Sarkar, and Tamas Ferenci. Chickenpox cases in hungary: a benchmark dataset for spatiotemporal signal processing with graph neural networks. *arXiv preprint arXiv:2102.08100*, 2021.

[33] Aravind Sankar, Yozen Liu, Jun Yu, and Neil Shah. Graph neural networks for friend ranking in large-scale social platforms. In *Proceedings of the Web Conference 2021*, pages 2535–2546, 2021.

[34] Aravind Sankar, Yanhong Wu, Liang Gou, Wei Zhang, and Hao Yang. Dysat: Deep neural representation learning on dynamic graphs via self-attention networks. In *Proceedings of the 13th International Conference on Web Search and Data Mining*, pages 519–527, 2020.

[35] Peter Shoubridge, Miro Kraetzl, WAL Wallis, and Horst Bunke. Detection of abnormal change in a time series of graphs. *Journal of Interconnection Networks*, 3(01n02):85–101, 2002.

[36] Xianfeng Tang, Yozen Liu, Neil Shah, Xiaolin Shi, Prasenjit Mitra, and Suhang Wang. Knowing your fate: Friendship, action and temporal explanations for user engagement prediction on social apps. In *Proceedings of the 26th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 2269–2279, 2020.

[37] Rakshit Trivedi, Hanjun Dai, Yichen Wang, and Le Song. Know-evolve: Deep temporal reasoning for dynamic knowledge graphs. In *international conference on machine learning*, pages 3462–3471. PMLR, 2017.

[38] Rakshit Trivedi, Mehrdad Farajtabar, Prasenjeet Biswal, and Hongyuan Zha. Dyrep: Learning representations over dynamic graphs. In *International conference on learning representations*, 2019.

[39] Huijun Wu, Chen Wang, Yuriy Tyshetskiy, Andrew Docherty, Kai Lu, and Liming Zhu. Adversarial examples for graph data: Deep insights into attack and defense. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*, pages 4816–4823. International Joint Conferences on Artificial Intelligence Organization, 7 2019.

[40] Da Xu, Chuanwei Ruan, Evren Korpeoglu, Sushant Kumar, and Kannan Achan. Inductive representation learning on temporal graphs. *arXiv preprint arXiv:2002.07962*, 2020.

[41] Dongkuan Xu, Wei Cheng, Dongsheng Luo, Yameng Gu, Xiao Liu, Jingchao Ni, Bo Zong, Haifeng Chen, and Xiang Zhang. Adaptive neural network for node classification in dynamic networks. In *2019 IEEE International Conference on Data Mining (ICDM)*, pages 1402–1407. IEEE, 2019.

[42] Kaidi Xu, Hongge Chen, Sijia Liu, Pin-Yu Chen, Tsui-Wei Weng, Mingyi Hong, and Xue Lin. Topology attack and defense for graph neural networks: An optimization perspective. *arXiv preprint arXiv:1906.04214*, 2019.

[43] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L Hamilton, and Jure Leskovec. Graph convolutional neural networks for web-scale recommender systems. In *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 974–983, 2018.

[44] Tong Zhao, Bo Ni, Wenhao Yu, Zhichun Guo, Neil Shah, and Meng Jiang. Action sequence augmentation for early graph-based anomaly detection. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*, pages 2668–2678, 2021.

[45] Daniel Zügner, Amir Akbarnejad, and Stephan Günnemann. Adversarial attacks on neural networks for graph data. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 2847–2856, 2018.

Checklist

The checklist follows the references. Please read the checklist guidelines carefully for information on how to answer these questions. For each question, change the default **[TODO]** to [Yes] , [No] , or [N/A] . You are strongly encouraged to include a **justification to your answer**, either by referencing the appropriate section of your paper or providing a brief inline description. For example:

- Did you include the license to the code and datasets? [Yes] See Section **??**.
- Did you include the license to the code and datasets? [No] The code and the data are proprietary.
- Did you include the license to the code and datasets? [N/A]

Please do not modify the questions and only use the provided macros for your answers. Note that the Checklist section does not count towards the page limit. In your paper, please delete this instructions block and only keep the Checklist section heading above along with the questions/answers below.

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes] See Section 6

(c) Did you discuss any potential negative societal impacts of your work? [Yes] See Section 6
(d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]

2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [Yes] See Section 3
   (b) Did you include complete proofs of all theoretical results? [Yes] See Section 3 and Appendix B
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] See footnote 7 on page 15.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] See Section 4 and Appendix D
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] See Figures 1 and Table 2
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] See Appendix D
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes]
   (b) Did you mention the license of the assets? [N/A]
   (c) Did you include any new assets either in the supplemental material or as a URL? [Yes] See footnotes on Page 5
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

Appendix

A Greedy approach

Algorithm 2 Greedy algorithm

Require: TAP variables ε_t (from Equation 5), Initial perturbation vector $\mathbf{s}^{(0)}$, Loss function $\mathcal{L}_{task}$, Probability function $p_{\mathcal{M}}$ predicting link existence probability, actual labels y_{task}, Target entities E_{tg}, Time steps T.

For all t: $\mathcal{G}'_t \leftarrow \mathcal{G}_t$. Attack history $\mathbf{S} \leftarrow \phi$.
while True **do**
  for t=1 to T **do**
    $\mathbf{s}_t \leftarrow \phi$
    for n_{tg} in E_{tg} **do**
      grads$[v] \leftarrow \partial\mathcal{L}_{task}(\mathcal{G}'_{1:T-1})/\partial(n_{tg}, v)$ for v in $\mathcal{V}$.
      Pick the first v in the descending order of **grads** such that $(n_{tg}, v) \notin \mathbf{S}$.
      $\mathbf{s}_t$.append((n_{tg}, v)).
    probs.append($p_{\mathcal{M}}(\mathcal{G}'_t \oplus \mathbf{s}_t)$)
  Pick $\mathbf{s}_\tau$ in the descending order of **probs** such that $\|\mathbf{S}_t \oplus \mathbf{s}_\tau\| \le \varepsilon_t$.
  if no τ found **then**
    break
  $\mathcal{G}'_t \leftarrow \mathcal{G}_t \oplus \mathbf{s}_t$.
  $\mathbf{S}_t$.append($\mathbf{s}_t$).

B Proof of Theorem 3

Suppose $\mathcal{S}$ denotes the feasible perturbation space for the constraints $\|\boldsymbol{\mathcal{A}}'_t - \boldsymbol{\mathcal{A}}_t\|/\|\boldsymbol{\mathcal{A}}_t - \boldsymbol{\mathcal{A}}_{t-1}\| \le \epsilon$ for all $1 < t < T$ and $\|\boldsymbol{\mathcal{A}}'_1 - \boldsymbol{\mathcal{A}}_1\| \le \varepsilon_1$. Then, one can project a vector $\mathbf{a}_t$ onto $\mathcal{S}$ using the following projection operator:

$$\Pi_{\mathcal{S}}(\mathbf{a}_t) = \begin{cases} P_{[0,1]}(\mathbf{a}_t - \mu_t) & \text{if} \exists \mu_t > 0: \ \mathbf{1}^T P_{[0,1]}(\mathbf{a}_t - \mu_t) = \varepsilon_t \\ P_{[0,1]}(\mathbf{a}_t) & \text{if } \mathbf{1}^T P_{[0,1]}(\mathbf{a}_t) \le \varepsilon_t \end{cases} \tag{8}$$

where $\varepsilon_t = \epsilon d\boldsymbol{\mathcal{A}}_t = \epsilon\|\boldsymbol{\mathcal{A}}_t - \boldsymbol{\mathcal{A}}_{t-1}\|$ for $t > 1$, and $P_{[0,1]}(x) = x$ if $x \in [0,1]$, 0 if $x < 0$, and 1 if $x > 1$.

Proof. Let the perturbation vector be $\mathbf{s} = [\mathbf{s}_1, \mathbf{s}_2, \cdots, \mathbf{s}_{T-1}]^T$. Then, for all $t \in [2, T-1]$, since $\mathbf{s}_t = \boldsymbol{\mathcal{A}}_t - \boldsymbol{\mathcal{A}}_{t-1}$, our constraint becomes $\|\mathbf{s}_t\| \le \epsilon d\boldsymbol{\mathcal{A}}_t \le \epsilon \ \|\boldsymbol{\mathcal{A}}_t - \boldsymbol{\mathcal{A}}_{t-1}\| =: \varepsilon_t$, which reduces to $\mathbf{1}^T\mathbf{s}_t \le \varepsilon_t$. For $t = 1$, we have $\|\boldsymbol{\mathcal{A}}'_1(\mathcal{V}_1) - \boldsymbol{\mathcal{A}}_1(\mathcal{V}_1)\| \le \varepsilon_1$, which also becomes $\mathbf{1}^T\mathbf{s}_1 \le \varepsilon_1$. Hence, we have the constraint $\mathbf{1}^T\mathbf{s}_t \le \varepsilon_t$ for all t.

By definition, then, projection operator must be $\Pi_{\mathcal{S}}(\mathbf{a}) = \arg\min_{\mathbf{s}\in\mathcal{S}} \frac{1}{2}\|\mathbf{s} - \mathbf{a}\|^2$, where $\mathcal{S} = \{\mathbf{s} \in [0,1]^n \mid \mathbf{1}^T\mathbf{s}_t \le \varepsilon_t \ \ \forall t\}$. This reduces to the following optimization problem:

$$\Pi_{\mathcal{S}}(\mathbf{a}) = \arg\min_{\mathbf{s}\in\mathcal{S}} \frac{1}{2} \sum_{t\in[1,T]} \|\mathbf{s}_t - \mathbf{a}_t\|^2 + \mathcal{I}_{[0,1]}(\mathbf{s}_t), \tag{9}$$
$$\text{such that} \quad \forall t \in [1,T] : \mathbf{1}^T\mathbf{s}_t \le \varepsilon_t$$

where $\mathcal{I}_{[0,1]}(x) = 0$ if $x \in [0,1]^n$ and ∞ otherwise.

This can be solved by the Lagrangian method. We note that the Lagrangian function of the above optimization problem is $\mathcal{L}(\mathbf{s}, \mathbf{a}, \mu) = \sum_{t\in[1,T]} \left(\frac{1}{2}\|\mathbf{s}_t - \mathbf{a}_t\|^2 + \mathcal{I}_{[0,1]}(\mathbf{s}_t) + \mu_t(\mathbf{1}^T\mathbf{s}_t - \varepsilon_t)\right)$.

$\partial\mathcal{L}/\partial\mathbf{s}_t = 0 \implies \mathbf{s}_t = \mathbf{a}_t - \mu_t$. However, if $s_{t,i} < 0$ or $s_{t,i} > 1$ for any i, then $\mathcal{L} = \infty$. Thus, the minimizer to the above function is $\mathbf{s}^*_t = P_{[0,1]}(\mathbf{a}_t - \mu_t)$, where $P_{[0,1]}(x) = x$ if $x \in [0,1]$, 0 if $x < 0$ and 1 if $x > 1$. In addition, the solution must satisfy the following KKT conditions $\forall t$:

$(\mathbf{1})\ \mu_t(\mathbf{1}^T\mathbf{s}_t^* - \varepsilon_t) = 0, \quad (\mathbf{2})\ \mu_t \geq 0, \quad (\mathbf{3})\ \mathbf{1}^T\mathbf{s}_t^* \leq \varepsilon_t.$

If $\mu_t > 0$, then we must have $\mathbf{1}^T P_{[0,1]}(\mathbf{a}_t - \mu_t) = \varepsilon_t$. Otherwise if $\mu_t = 0$, then $\mathbf{1}^T P_{[0,1]}(\mathbf{a}_t) \leq \varepsilon_t$.

■

C Data statistics and pre-processing

Table 3 shows the statistics of the datasets used in this paper. DBLP-5 is the node classification dataset with 5 labels while the others are dynamic link prediction datasets with varying sizes. The datasets span over differing periods of time. We split each of them into finite number of timesteps to keep a large enough number of time steps while maintaining a realistic period of splitting. Radoslaw is split using a 3-week period in 13 snapshots while the 13 snapshots in UCI denote a 2-week period. The Reddit dataset is spanned over 3 years, thus we use a 2-month split to obtain the 20 snapshots. We use the publicly available pre-processed data for DBLP-5 [41].

For datasets with no node features, i.e., Radoslaw, UCI, and Reddit, we use uniformly random features with dimension 10. The pre-processed DBLP-5 has 100 node features for each node.

	# Nodes	# Edges	# Time-steps	# Labels
Radoslaw	167	22K	13	-
UCI	1.9K	24K	13	-
Reddit	35K	715K	20	-
DBLP-5	6.6K	43K	10	5

Table 3: Description of the datasets.

D Additional details on experimental setup

D.1 Setup

We consider a targeted setting with single targets, that are selected using either a random sampling or a degree-biased sampling. Each target is attacked one-by-one and the total performance of an attacker is measured using either an ROC-AUC or an Accuracy over the set of sampled targets.

D.2 Hyperparameters

For Dyn-PGD optimization, we used ADAM optimizer [17] with the initial learning rate of 10. The initial perturbation vector $\mathbf{s}^{(0)}$ was initialized with all ones, thus, giving each perturbation an equal chance at the start. The algorithm was run for 50 iterations. We use Binary Cross Entropy loss for training dynamic link prediction and Weighted Cross Entropy for node classification. Further, we stop the greedy search if the time taken exceeds 300 s, which is at least 3 times that of Dyn-PGD.

D.3 Victim Model training

Table 4 shows the performance on the test set of the victim models on different datasets. The performance is evaluated using ROC-AUC for the dynamic link prediction and using Accuracy for the node classification tasks. The test set for dynamic link prediction task denotes the edges and non-edges in the final snapshot, while for node classification, we use a 20% held-out set of nodes as the test set for noting the performance and attacking.

D.4 Implementation

We use the TorchGeometric-Temporal [3] implementation of EvolveGCN and GC-LSTM to train these models. For DySAT, we use the pytorch implementation [4]. We adapt the official code of

[3] https://pytorch-geometric.readthedocs.io/en/latest/index.html#

[4] https://github.com/FeiGSSS/DySAT_pytorch

Dataset	Model	Perf.
Radoslaw	DySAT	0.743
	EvolveGCN	0.742
	GC-LSTM	0.813
UCI	DySAT	0.952
	EvolveGCN	0.873
	GC-LSTM	0.968
Reddit	DySAT	0.947
	EvolveGCN	0.939
	GC-LSTM	0.941
DBLP-5	DySAT	0.699
	EvolveGCN	0.687
	GC-LSTM	0.695

Table 4: Performance (Perf.) on test set for different datasets and models. For Radoslaw, UCI, Reddit, we use ROC-AUC as the performance metric and for DBLP-5, Accuracy is used.

TGA[5] to implement the greedy approach. For Dyn-PGD implementation, we adapt the DeepRobust [6] implementation for dynamic graphs under TAP constraint. Full code is provided in the link[7].

[5]https://github.com/jianz94/tga

[6]https://github.com/wenqifan03/RobustTorch

[7]https://anonymous.4open.science/r/dyn-pgd-685F/