Graph Sanitation with Application to Node Classification

Zhe Xu
zhexu3@illinois.edu
University of Illinois at Urbana-Champaign
USA

Boxin Du
boxindu2@illinois.edu
University of Illinois at Urbana-Champaign
USA

Hanghang Tong
htong@illinois.edu
University of Illinois at Urbana-Champaign
USA

ABSTRACT

The past decades have witnessed the prosperity of graph mining, with a multitude of sophisticated models and algorithms designed for various mining tasks, such as ranking, classification, clustering and anomaly detection. Generally speaking, the vast majority of the existing works aim to answer the following question, that is, *given a graph, what is the best way to mine it?*

In this paper, we introduce the *graph sanitation* problem, to answer an orthogonal question. That is, *given a mining task and an initial graph, what is the best way to improve the initially provided graph?* By learning a better graph as part of the input of the mining model, it is expected to benefit graph mining in a variety of settings, ranging from denoising, imputation to defense. We formulate the graph sanitation problem as a bilevel optimization problem, and further instantiate it by semi-supervised node classification, together with an effective solver named GASOLINE. Extensive experimental results demonstrate that the proposed method is (1) broadly applicable with respect to various graph neural network models and flexible graph modification strategies, (2) effective in improving the node classification accuracy on both the original and contaminated graphs in various perturbation scenarios. In particular, it brings up to 25% performance improvement over the existing robust graph neural network methods.

CCS CONCEPTS

• **Mathematics of computing** → **Graph algorithms**; • **Information systems** → **Data mining**.

KEYWORDS

graph sanitation, node classification, graph mining

ACM Reference Format:
Zhe Xu, Boxin Du, and Hanghang Tong. 2022. Graph Sanitation with Application to Node Classification. In *Proceedings of the ACM Web Conference 2022 (WWW '22), April 25–29, 2022, Virtual Event, Lyon, France.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3485447.3512180

1 INTRODUCTION

Graph mining has become the cornerstone in a wealth of real-world applications, such as social network mining [21, 58], brain connectivity analysis [1], computational epidemiology [25], and financial analysis [22, 47]. For the vast majority of existing works, they essentially aim to answer the following question, that is, *given* a graph, what is the best model and/or algorithm to mine it? To name a few, PageRank [36] and its variants [14, 17, 19, 41] measure the node importance and node proximity based on multiple weighted paths; spectral clustering [39] minimizes inter-cluster connectivity and maximizes the intra-cluster connectivity to partition nodes into different groups; graph neural networks (GNNs) [26, 27, 44, 51] learn representation of nodes by aggregating information from the neighborhood. In all these works and many more, they require a *given* graph, including its topology and/or the associated attribute information, as part of the input of the corresponding mining model.

Despite tremendous success, some fundamental questions largely remain open, e.g., where does the input graph come from at the first place? To what extent does the quality of the given graph impact the effectiveness of the corresponding graph mining model? In response, we introduce the *graph sanitation* problem, which aims to improve the initially provided graph for a given graph mining model, so as to maximally boost its performance. The rationality is as follows. In many existing graph mining works, the initially provided graph is typically constructed manually based on some heuristics. The graph construction is often treated as a preprocessing step, without the consideration of the specific mining task. What is more, the initially constructed graph could be subject to various forms of contamination, such as missing information, noise and even adversarial attacks. This suggests that there might be under-explored space for improving the mining performance by learning a 'better' graph as the input of the corresponding task.

There are a few lines of existing works for modifying graphs. For example, network imputation [18, 31] and knowledge graph completion [4, 32, 33, 50, 55] problems focus on restoring missing links in a partially observed graph; connectivity optimization [5] and computational immunization [6] problems manipulate the graph connectivity in a desired way by changing the underlying topology; robust GNNs [10, 20, 52] utilize empirical properties of a benign graph to remove or assign lower weights to the poisoned graph elements (e.g., contaminated edges).

The graph sanitation problem introduced in this paper is related to but bears subtle difference from these existing work in the following sense. Most, if not all, of these existing works for modifying graphs assume the initially provided graph is impaired or perturbed in a specific way, e.g., due to missing links, or noise, or adversarial attacks. Some existing works further impose certain assumptions on the specific graph modification algorithms, such as the low-rank assumption behind many network imputation methods, the types of attacks and/or the empirical properties of the benign graph (e.g., topology sparsity, feature smoothness) behind some robust GNNs.

WWW '22, April 25–29, 2022, Virtual Event, Lyon, France

ACM ISBN 978-1-4503-9096-5/22/04…$15.00
https://doi.org/10.1145/3485447.3512180

Data	With GaSoliNe?	GAT	SVD	RGCN
Cora	N	48.8±0.2	60.1±0.6	50.6±0.8
	Y	63.7±0.6	79.7±0.6	62.6±0.6
Citeseer	N	62.4±0.7	50.6±0.6	55.5±1.4
	Y	69.7±0.2	76.5±0.6	66.1±0.8
Polblogs	N	48.2±6.6	77.3±3.3	50.8±0.9
	Y	70.8±0.6	89.2±0.7	67.7±0.3

Table 1: Node classification accuracy (Mean±Std) boosting of existing defense methods on poisoned graphs (25% edges perturbed by metattack [65]) by the proposed GaSoliNe.

In contrast, the proposed graph sanitation problem does not make any such assumptions, but instead pursues a different design principle. That is, we aim to let the performance of the downstream data mining task, measured on a validation set, dictate how we should optimally modify the initially provided graph. This is crucial, as it not only ensures that the modified graph will directly and maximally improve the mining performance, but also lends itself to be applicable to a variety of graph mining tasks.

Formally, we formulate the graph sanitation problem as a generic bilevel optimization problem, where the lower-level optimization problem corresponds to the specific mining task and the upper-level optimization problem encodes the supervision to modify the provided graph and maximally improve the mining performance. Based on that, we instantiate such a bilevel optimization problem by semi-supervised node classification with GNNs, where the lower-level objective function represents the cross-entropy classification loss over the training data and the upper-level objective function represents the loss over validation data, using the mining model trained from the lower-level optimization problem. We propose an effective solver (GaSoliNe) which adopts an efficient approximation of hyper-gradient to guide the modification over the given graph. We carefully design the hyper-gradient aggregation mechanism to avoid potential bias from a specific dataset split by aggregating the hyper-gradient from different folds of data. GaSoliNe is versatile, and is equipped with multiple variants, such as discretized vs. continuous modification, modifying graph topology vs. feature. Comprehensive experiments demonstrate that (1) GaSoliNe is broadly applicable to benefit different downstream node classifiers together with flexible choices of variants and modification strategies, (2) GaSoliNe can significantly boost downstream classifiers on both the original and contaminated graphs in various perturbation scenarios and can work hand-in-hand with existing robust GNNs methods. For instance, in Table 1, the proposed GaSoliNe significantly boosts GAT [44], SVD [10], and RGCN [61].

In summary, our main contributions in this paper are as follows:

- **Problem Definition.** We introduce a novel graph sanitation problem, and formulate it as a bilevel programming. The proposed problem can be potentially applied to various graph mining models as long as they are differentiable w.r.t. the input graph.
- **Algorithmic Instantiation.** We instantiate the graph sanitation problem by semi-supervised node classification with GNNs. We further propose an effective and scalable solver named GaSoliNe with versatile variants.
- **Empirical Evaluations.** We perform extensive empirical studies on real-world datasets to demonstrate the effectiveness and the applicability of the proposed GaSoliNe algorithms.

2 GRAPH SANITATION PROBLEM

A - Notations. We use bold uppercase letters for matrices (e.g., $\mathbf{A}$), bold lowercase letters for column vectors (e.g., $\mathbf{u}$), lowercase letters for scalars (e.g., c), and calligraphic letters for sets (e.g., $\mathcal{T}$). $\mathbf{A}[i,j]$ represents the entry of matrix $\mathbf{A}$ at the i-th row and j-th column. $\mathbf{A}[i,:]$ and $\mathbf{A}[:,j]$ represent the i-th row and j-th column of matrix $\mathbf{A}$. $\mathbf{u}[i]$ denotes the i-th entry of vector $\mathbf{u}$. Prime denotes the transpose of matrices and vectors (e.g., $\mathbf{A}'$ is the transpose of $\mathbf{A}$). For the variables of the modified graphs, we set $\tilde{}$ over the corresponding variables of the original graphs (e.g., $\tilde{\mathbf{A}}$).

We represent an attributed graph as $G = \{\mathbf{A}, \mathbf{X}\}$, where $\mathbf{A} \in \mathbb{R}^{n\times n}$ is the adjacency matrix and $\mathbf{X} \in \mathbb{R}^{n\times d}$ is the feature matrix composed by d-dimensional feature vectors of n nodes. For *supervised graph mining models*, we first divide the node set into two disjoint subsets: labeled node set $\mathcal{Z}$ and test set $\mathcal{W}$, and then divide the labeled node set $\mathcal{Z}$ into two disjoint subsets: the training set $\mathcal{T}$ and the validation set $\mathcal{V}$. We use y and $\hat{y}$ with appropriate indexing to denote the ground truth supervision and the prediction result respectively. Take the classification task as an example, $y_{ij} = 1$ if node i belongs to class j and $y_{ij} = 0$ otherwise; $\hat{y}_{ij}$ is the predicted probability that node i belongs to class j. Furthermore, we use $\mathcal{Y}_{\texttt{train}}$ and $\mathcal{Y}_{\texttt{valid}}$ to denote the supervision information of all the nodes in the training set $\mathcal{T}$ and the validation set $\mathcal{V}$, respectively.

B - Optimization-Based Graph Mining Models. For many graph mining models, they can be formulated from the optimization perspective [23, 24] with a general goal to find an optimal solution θ^* so that a task-specific loss $\mathcal{L}(G, \theta, \mathcal{T}, \mathcal{Y}_{\texttt{train}})$ is minimized. Here, $\mathcal{T}$ and $\mathcal{Y}_{\texttt{train}}$ are the training set and the associated ground truth (e.g., class labels for the classification task), which would be absent for the unsupervised graph mining tasks (e.g., clustering, ranking). We give three concrete examples next.

Example #1: personalized PageRank [19] is a fundamental ranking model. When the adjacency matrix of the underlying graph is symmetrically normalized, the ranking vector $\mathbf{r}$ can be obtained as:

$$\mathbf{r}^* = \arg\min_{\mathbf{r}} \; q\mathbf{r}'(\mathbf{I} - \bar{\mathbf{A}})\mathbf{r} + (1-q)||\mathbf{r} - \mathbf{e}||^2, \tag{1}$$

where $\bar{\mathbf{A}}$ is the symmetrically normalized adjacency matrix; $q \in (0, 1]$ is the damping factor; $\mathbf{e}$ is the preference vector; the ranking vector $\mathbf{r}^*$ is the solution of the ranking model (i.e., $\theta^* = \mathbf{r}^*$).

Example #2: spectral clustering [39] is a classic graph clustering model aiming to minimizes the normalized cut between clusters:

$$\mathbf{u}^* = \arg\min_{\mathbf{u}} \; \mathbf{u}'\mathbf{L}\mathbf{u} \quad \text{s.t. } \mathbf{u}'\mathbf{D}\mathbf{u} = 1, \; \mathbf{D}\mathbf{u} \perp \mathbf{1}_{\mathbf{u}}, \tag{2}$$

where $\mathbf{L}$ is the Laplacian matrix, $\mathbf{D}$ is the diagonal degree matrix (i.e., $\mathbf{D}[i,i] = \sum_j \mathbf{A}[i,j]$), $\mathbf{1}_u$ is an all-one vector with the same size as $\mathbf{u}$; the model solution θ^* is the cluster indicator vector $\mathbf{u}^*$.

Example #3: node classification aims to construct a classification model based on the graph topology $\mathbf{A}$ and feature $\mathbf{X}$. A typical loss for node classification is cross-entropy (CE) over the training set:

$$\theta^* = \arg\min_{\theta} \; -\sum_{i\in\mathcal{T}} \sum_{j=1}^{c} y_{ij} \ln \hat{y}_{ij}, \tag{3}$$

where c is the number of classes, y_{ij} is the ground truth indicating if node i belongs to class j, $\mathcal{T}$ is the training set, $\hat{y}_{ij} = f(G,\theta)[i,j]$ is the predicted probability that node i belongs to class j by a classifier $f(G,\theta)$ parameterized by θ. For example, classifier $f(G,\theta)$ can be a GNN whose trained model parameters form the solution θ^*.

Mining Tasks	Personalized PageRank [2, 19, 30]	Spectral clustering [39, 48]	Semi-supervised node classification [9]
$\mathcal{L}_{\texttt{lower}}$	$\min_{\mathbf{r}}\ q\mathbf{r}'(\mathbf{I}-\bar{\mathbf{A}})\mathbf{r}+(1-q)\|\mathbf{r}-\mathbf{e}\|^2$	$\min_{\mathbf{u}}\ \mathbf{u}'\mathbf{L}\mathbf{u}\quad$ s.t. $\mathbf{u}'\mathbf{D}\mathbf{u}=1,\ \mathbf{D}\mathbf{u}\perp\mathbf{1}$	$\min_{\theta}\ -\sum_{i\in\mathcal{T}}\sum_{j=1}^{c} y_{ij}\ln\hat{y}_{ij}$
$\mathcal{L}_{\texttt{upper}}$	$\min_{\mathbf{A}}\ \sum_{x\in\mathcal{P},y\in\mathcal{N}}(1+\exp(\mathbf{r}^*[x]-\mathbf{r}^*[y])/w)^{-1}$	$\min_{\mathbf{A}}\ -\mathbf{u}^{*\prime}\mathbf{Q}\mathbf{u}^*$	$\min_{G}\ -\sum_{i\in\mathcal{V}}\sum_{j=1}^{c} y_{ij}\ln\hat{y}_{ij}$
$\mathcal{T}$	none	none	training set $\mathcal{T}$
$\mathcal{Y}_{\texttt{train}}$	none	none	labels of training set $\mathcal{Y}_{\texttt{train}}$
$\mathcal{V}$	positive node set $\mathcal{P}$, negative node set $\mathcal{N}$	'must-link' set $\mathcal{M}$ 'cannot-link' set C	validation set $\mathcal{V}$
$\mathcal{Y}_{\texttt{valid}}$	none	none	labels of validation set $\mathcal{Y}_{\texttt{valid}}$
Remarks	normalized adjacency matrix $\bar{\mathbf{A}}$, damping factor q, width parameter w, preference vector $\mathbf{e}$	Laplacian matrix $\mathbf{L}$, degree matrix $\mathbf{D}$, link constraints matrix $\mathbf{Q}$	number of classes c, predicted probability of node i to class j $\hat{y}_{ij}$, binary ground truth of node i to class j y_{ij}

Table 2: Instantiations of graph sanitation problem over various mining tasks

Remarks. Both the standard personalized PageRank and spectral clustering are unsupervised and therefore the training set $\mathcal{T}$ and its supervision $\mathcal{Y}_{\texttt{train}}$ are absent in the corresponding loss functions (i.e., Eq. (1) and (2), respectively). Nonetheless, both personalized PageRank and spectral clustering have been generalized to further incorporate some forms of supervision, as we will show next.

C - Graph Sanitation: Formulation and Instantiations. Given an initial graph G and an optimization-based graph mining model $\mathcal{L}(G,\theta,\mathcal{T},\mathcal{Y}_{\texttt{train}})$, we aim to learn a modified graph $\tilde{G}$ to boost the performance of the corresponding mining model and we name it as *graph sanitation problem*. The basic idea is to let the mining performance on a validation set $\mathcal{V}$ guide the modification process. Formally, the graph sanitation problem is defined as follows.

Problem 1. *Graph Sanitation Problem*

Given: *(1) a graph $G=\{\mathbf{A},\mathbf{X}\}$, (2) a mining task $\mathcal{L}(G,\theta,\mathcal{T},\mathcal{Y}_{\texttt{train}})$, (3) a validation set $\mathcal{V}$ and its supervision $\mathcal{Y}_{\texttt{valid}}$, and (4) the sanitation budget B;*

Find: *A modified graph $\tilde{G}=\{\tilde{\mathbf{A}},\tilde{\mathbf{X}}\}$ to boost the performance of input graph mining model.*

We formulate Problem 1 as a bilevel optimization problem:

$$\begin{aligned}\tilde{G}=\arg\min_{G}\ \ &\mathcal{L}_{\texttt{upper}}(G,\theta^*,\mathcal{V},\mathcal{Y}_{\texttt{valid}})\\ \text{s.t. }\theta^*=\arg\min\ &\mathcal{L}_{\texttt{lower}}(G,\theta,\mathcal{T},\mathcal{Y}_{\texttt{train}}),\ D(\tilde{G},G)\le B\end{aligned}\tag{4}$$

where the lower-level optimization is to train the model θ^* based on the training set $\mathcal{T}$; the upper-level optimization aims to optimize the performance of the trained model θ^* on the validation set $\mathcal{V}$, and there is no overlap between $\mathcal{T}$ and $\mathcal{V}$; the distance function D measures the distance between two graphs. In this paper, we instantiate $D(\tilde{G},G)$ as $||\tilde{\mathbf{A}}-\mathbf{A}||_{1,1}$ or $||\tilde{\mathbf{X}}-\mathbf{X}||_{1,1}$ based on the modification scenarios. Notice that the loss function at the upper level $\mathcal{L}_{\texttt{upper}}$ might be different from the one at the lower level $\mathcal{L}_{\texttt{lower}}$. For example, $\mathcal{L}_{\texttt{lower}}$ for both personalized PageRank (Eq. (1)) and spectral clustering (Eq. (2)) does not involve any supervision. However, $\mathcal{L}_{\texttt{upper}}$ for both models is designed to measure the performance on a validation set with supervision and therefore should be different from $\mathcal{L}_{\texttt{lower}}$. We elaborate this next.

The proposed bilevel optimization problem in Eq. (4) is quite general. In principle, it is applicable to *any* graph model with differentiable $\mathcal{L}_{\texttt{upper}}$ and $\mathcal{L}_{\texttt{lower}}$. We give its instantiations with the three aforementioned mining tasks and summarize them in Table 2.

Instantiation #1: supervised PageRank. The original personalized PageRank [19] has been generalized to encode pair-wised ranking preference [2, 30]. For graph sanitation with supervised PageRank, the training set and its supervision is absent, and the lower-level loss $\mathcal{L}_{\texttt{lower}}$ is given in Eq. (1). The validation set $\mathcal{V}$ is consisted of a positive node set $\mathcal{P}$ and a negative node set $\mathcal{N}$. The supervision of the upper-level problem is that ranking scores of nodes from $\mathcal{P}$ should be higher than that from $\mathcal{N}$, i.e., $\mathbf{r}[x]>\mathbf{r}[y],\forall x\in\mathcal{P},\forall y\in\mathcal{N}$. Several choices for the upper-level loss $\mathcal{L}_{\texttt{upper}}$ exist. For example, we can use Wilcoxon-Mann-Whitney loss [54]:

$$\min_{\mathbf{A}}\ \sum_{x\in\mathcal{P},y\in\mathcal{N}}\Big(1+\exp(\mathbf{r}^*[x]-\mathbf{r}^*[y])/w\Big)^{-1}\tag{5}$$

where w is the width parameter. It is worth-mentioning that Eq. (5) only modifies graph topology $\mathbf{A}$. Although Eq. (5) does not contain variable $\mathbf{A}$, $\mathbf{r}^*$ is determined by $\mathbf{A}$ through the lower-level problem.

Instantiation #2: supervised spectral clustering. A typical way to encode supervision in spectral clustering is via 'must-link' and 'cannot-link' [45, 48]. For graph sanitation with supervised spectral clustering, the training set together with its supervision is absent, and the lower-level loss $\mathcal{L}_{\texttt{lower}}$ is given in Eq. (2). The validation set $\mathcal{V}$ contains a 'must-link' set $\mathcal{M}$ and a 'cannot-link' set C. For the upper-level loss, the idea is to encourage nodes from must-link set $\mathcal{M}$ to be grouped in the same cluster and in the meanwhile push nodes from cannot-link set C to be in different clusters. To be specific, $\mathcal{L}_{\texttt{upper}}$ can be instantiated as follows.

$$\min_{\mathbf{A}}\quad -\mathbf{u}^{*\prime}\mathbf{Q}\mathbf{u}^*,\tag{6}$$

where $\mathbf{Q}$ encodes the 'must-link' and 'cannot-link', that is, $\mathbf{Q}[i,j]=1$ if $(i,j)\in\mathcal{M}$, $\mathbf{Q}[i,j]=-1$ if $(i,j)\in C$, and $\mathbf{Q}[i,j]=0$ otherwise. This instantiation only modifies the graph topology $\mathbf{A}$.

Instantiation #3: semi-supervised node classification. For graph sanitation with semi-supervised node classification, its lower-level optimization problem is given in Eq. (3). We have cross-entropy loss over validation set $\mathcal{V}$ as the upper-level problem:

$$\min_{G}\quad \mathcal{L}_{\texttt{CE}}(G,\theta^*,\mathcal{V},\mathcal{Y}_{\texttt{valid}})=-\sum_{i\in\mathcal{V}}\sum_{j=1}^{c} y_{ij}\ln\hat{y}_{ij}.\tag{7}$$

Notice that $\mathcal{T}\cap\mathcal{V}=\emptyset$. If both the topology $\mathbf{A}$ and node feature $\mathbf{X}$ are used for classification, then they both can be modified in this instantiation.

Remarks. If the initially given graph G is poisoned by adversarial attackers [64, 65], the graph sanitation problem with semi-supervised node classification can also be used as a defense strategy. However, it bears important difference from the existing robust GNNs [10, 20, 52] as it does not assume the given graph G is poisoned or any specific way by which it is poisoned. Therefore, graph sanitation problem in this scenario can boost the performance under a wide range of attacking scenarios (e.g., non-poisoned graphs, lightly-poisoned graphs, and heavily-poisoned graphs) and has the potential to work hand-in-hand with existing robust GNNs model. In the next section, we propose an effective algorithm to solve the graph sanitation problem with semi-supervised node classification.

3 PROPOSED ALGORITHMS: GASOLINE

We focus on graph sanitation problem in the context of semi-supervised node classification and propose an effective solver named GaSoliNe. The general workflow is as follows. First, we solve the lower-level problem (Eq. (3)) and obtain a solution θ^*. Then we compute the hyper-gradient of the upper-level loss function (Eq. (7)) w.r.t. the graph G to solve the upper-level optimization problem. Recall that a classifier $f(\theta)$ is needed to provide the predicted labels (in both the lower-level and upper-level problems) and we refer to this classifier as the *backbone classifier*. Finally we test the performance of another classifier over the modified graph on the test set $\mathcal{W}$ and this classifier is named as the *downstream classifier*. We will introduce our proposed solution GaSoliNe in four parts: (A) hyper-gradient computation, (B) hyper-gradient aggregation, (C) hyper-gradient-guided modification, and (D) low-rank speedup.

A - Hyper-Gradient Computation. Eq. (4) and its corresponding instantiations Eqs. (3)(7) fall into the family of bilevel optimization problem where the lower-level problem is to optimize θ via minimizing the loss over the training set $\{\mathcal{T}, \mathcal{Y}_{\texttt{train}}\}$ given G, and the upper-level problem is to optimize G via minimizing the loss over $\{\mathcal{V}, \mathcal{Y}_{\texttt{valid}}\}$. We compute gradient w.r.t. the upper-level problem and view the lower-level problem as a dynamic system:

$$\theta^{t+1} = \Theta^{t+1}(G, \theta^t, \mathcal{T}, \mathcal{Y}_{\texttt{train}}), \quad \theta^1 = \Theta^1(G, \mathcal{T}, \mathcal{Y}_{\texttt{train}}), \tag{8}$$

where Θ^1 is the initialization of θ and Θ^{t+1} ($t \neq 0$) is the updating formula which can be instantiated as an optimizer over the lower-level objective function on training set (Eq. (3)). For the hyper-gradient of the upper-level problem $\nabla_G \mathcal{L}$, we assume that the system converges in T iterations (i.e., $\theta^* = \theta^T$). Then we unroll the iterative solution of the lower-level problem to obtain the hyper-gradient $\nabla_G \mathcal{L}$ by the chain rule as follows [3] where $A_t = \nabla_{\theta^t} \theta^{t+1}$, $B_t = \nabla_G \theta^{t+1}$. For brevity, we abbreviate cross-entropy loss over the validation set $\mathcal{L}_{\texttt{CE}}(G, \theta^T, \mathcal{V}, \mathcal{Y}_{\texttt{valid}})$ as $\mathcal{L}_{\texttt{valid}}(\theta^T)$.

$$\nabla_G \mathcal{L} = \nabla_G \mathcal{L}_{\texttt{valid}}(\theta^T) + \sum_{t=0}^{T-2} B_t A_{t+1} \dots A_{T-1} \nabla_{\theta^T} \mathcal{L}_{\texttt{valid}}(\theta^T) \tag{9}$$

Our final goal is to improve the performance of converged downstream classifiers. Hence, T is set as a relatively large value (e.g., 200) to ensure the hyper-gradient is computed over a converged classifier. To balance the effectiveness and the efficiency, we adopt the truncated hyper-gradient [38] w.r.t. G and rewrite the second part of Eq. (9) as $\sum_{t=P}^{T-2} B_t A_{t+1} \dots A_{T-1} \nabla_{\theta^T} \mathcal{L}_{\texttt{valid}}(f_{\theta^T})$, where P denotes the truncating iteration. To further speed up, we adopt a first-order approximation [35, 65] and $\nabla_G \mathcal{L}$ can be computed as:

$$\nabla_G \mathcal{L} = \sum_{t=P}^{T} \nabla_G \mathcal{L}_{\texttt{valid}}(\theta^t). \tag{10}$$

where the updating trajectory of θ^t is the same as Eq. (8). If the initially-provided graph G is undirected, it indicates that $\mathbf{A} = \mathbf{A}'$. Hence, when we compute the hyper-gradient w.r.t. the undirected graph topology $\mathbf{A}$, we need to calibrate the partial derivative into the derivative [24] and update the hyper-gradient as follows:

$$\nabla_{\mathbf{A}} \mathcal{L} \leftarrow \nabla_{\mathbf{A}} \mathcal{L} + (\nabla_{\mathbf{A}} \mathcal{L})' - \text{diag}(\nabla_{\mathbf{A}} \mathcal{L}). \tag{11}$$

For the hyper-gradient w.r.t. feature $\mathbf{X}$ and directed graph topology $\mathbf{A}$ ($\mathbf{A} \neq \mathbf{A}'$), the above calibration process is not needed.

B - Hyper-Gradient Aggregation. To ensure the quality of graph sanitation without introducing bias from a specific dataset split, we adopt K-fold split with similar settings as cross-validation [7]. Specifically, during the training phase, we split all the labeled nodes $\mathcal{Z}$ into K folds and alternatively select one of them as $\mathcal{V}$ (with labels $\mathcal{Y}_{\texttt{valid}}$) and the others as $\mathcal{T}$ (with labels $\mathcal{Y}_{\texttt{train}}$). In total, there are K sets of training/validation splits. With the k-th dataset split, by Eq. (10), we obtain the hyper-gradient $\nabla_G^k \mathcal{L}$. For the hyper-gradient $\{\nabla_G^1 \mathcal{L}, \dots, \nabla_G^K \mathcal{L}\}$ from the K sets of training/validation split, we sum them up as the aggregated hyper-gradient: $\nabla_G = \sum_k \nabla_G^k \mathcal{L}$.

C - Hyper-Gradient-Guided Modification. To modify the graph based on ∇_G, we provide two variants, *discretized modification* and *continuous modification*. The discretized modification can work with binary inputs such as adjacency matrices of unweighted graphs and binary feature matrices. The continuous modification is suitable for both continuous and binary inputs. For the clarity of explanation, we replace the G with the adjacency matrix $\mathbf{A}$ as an example for the topology modification. It is straight-forward to generalize that to the feature modification with feature matrix $\mathbf{X}$.

The discretized modification is flipping B entries in $\mathbf{A}$ whose indices are also the the indices of the top-B entries in a hyper-gradient-based score matrix $\mathbf{S}$. Mathematically $\mathbf{S}$ is presented as:

$$\mathbf{S} = (-\nabla_{\mathbf{A}}) \circ (\mathbf{1} - 2\mathbf{A}), \tag{12}$$

where $\circ$ denotes Hadamard product, $\mathbf{1}$ is an all-one matrix. This score matrix is composed by 'preference' (i.e., $-\nabla_{\mathbf{A}}$) and 'modifiability' (i.e., $(\mathbf{1} - 2\mathbf{A})$). Only entries with both high 'preference' and 'modifiability' are assigned with high scores. For example, large positive $(-\nabla_{\mathbf{A}})[i, j]$ indicates strong preference of adding an edge between the i-th and j-th nodes based on the hyper-gradient and if the i-th and j-th nodes are not linked (i.e., $\mathbf{A}[i, j] = 0$), $(-\nabla_{\mathbf{A}})[i, j]$ and $(\mathbf{1} - 2\mathbf{A})[i, j]$ share the same sign which results in a large $\mathbf{S}[i, j]$.

The continuous modification is gradient descent with budget-adaptive learning rate:

$$\mathbf{A} \leftarrow \mathbf{A} - \frac{B}{\sum_{i,j} |\nabla_{\mathbf{A}}|[i, j]} \cdot \nabla_{\mathbf{A}}. \tag{13}$$

The learning rate is based on the ratio of the modification budget B to the sum of absolute values of the gradient matrix. In implementation, for both modification methods, we set the budget in every iteration as b and update the graph in multiple steps until the total budget B is used up. Algorithm 1 summarizes our methods. In addition, in our experiments, the B for topology and feature ($B_{\texttt{topo}}$ and $B_{\texttt{fea}}$) are set separately since the modification cost on different elements of a graph may not be comparable.

Remarks. LDS [13] formulates the structural learning problem under the bilevel optimization context. Here, we claim the differences and advantages: (1) LDS focuses on learning graph topology, but GaSoliNe can handle any graph components; (2) LDS formulates the topology as Bernoulli random variables, whose updating requires multiple samplings which are time-consuming, but GaSoliNe works in a deterministic and efficient way and also provides discrete solutions; (3) in the following section we will introduce a speed-up variant of GaSoliNe which shows great efficacy.

D - Speedup and Scale-up. The core operation of our proposed GaSoliNe is to compute hyper-gradient w.r.t. the graph components (i.e., $\mathbf{A}$ and $\mathbf{X}$) which leads into a gradient matrix (e.g. $\nabla_{\mathbf{A}} \mathcal{L}$). In many real-world scenarios (e.g., malfunctions of certain nodes, targeted adversarial attacks), perturbations are often around a small set of nodes, which leads to low-rank perturbation matrices. Hence,

Algorithm 1: GaSoliNe

Input : a graph G, the labeled nodes $\mathcal{Z}$, total budget and budget in every step B and b, number of fold K, truncating and converging iterations T and P;
Output : the modified graph $\tilde{G}$;

initialization: split the labeled nodes $\mathcal{Z}$ and their labels into K folds: $\mathcal{Z} = \{\mathcal{Z}^1, \ldots, \mathcal{Z}^K\}$, $\mathcal{Y}_{\texttt{labeled}} = \{\mathcal{Y}^1, \ldots, \mathcal{Y}^K\}$; $\tilde{G} \leftarrow G$; cumulative budget $\delta \leftarrow 0$;
while $\delta < B$ **do**
  for k=1 *to* K **do**
    $\mathcal{V} \leftarrow \mathcal{Z}^k, \mathcal{T} \leftarrow \mathcal{Z} \backslash \mathcal{Z}^k, \mathcal{Y}_{\texttt{valid}} \leftarrow \mathcal{Y}^k$, $\mathcal{Y}_{\texttt{train}} \leftarrow \mathcal{Y}_{\texttt{labeled}} \backslash \mathcal{Y}^k$, $\nabla^k_{\tilde{G}} \mathcal{L} \leftarrow 0$;
    for $t = 1$ *to* T **do**
      update θ^t to θ^{t+1} by Eq. (8);
      if $t > P$ **then**
        compute $\nabla^k_{\tilde{G}} \mathcal{L}_{\texttt{valid}}$ given $\{\tilde{G}, \theta^{t+1}, \mathcal{V}, \mathcal{Y}_{\texttt{valid}}\}$;
        $\nabla^k_{\tilde{G}} \mathcal{L} \leftarrow \nabla^k_{\tilde{G}} \mathcal{L} + \nabla^k_{\tilde{G}} \mathcal{L}_{\texttt{valid}}$
      end
    end
  end
  calibrate $\{\nabla^k_{\tilde{\mathbf{A}}} \mathcal{L}\}$ by Eq. (11) (if needed);
  sum $\{\nabla^k_{\tilde{G}} \mathcal{L}\}$ into $\nabla_{\tilde{G}}$, $\delta \leftarrow \delta + b$;
  update $\tilde{G}$ based on the guide of score matrix $\mathbf{S}$ by Eq. (12) (discretized modification) or by Eq. (13) (continuous modification) with budget b
end

for topology modification, we propose to decompose the incremental matrix (i.e., $\Delta\mathbf{A}$) into its low-rank representation (i.e., $\Delta\mathbf{A} = \mathbf{U}\mathbf{V}'$), and compute the hyper-gradient with respect to the low-rank matrices instead, which can significantly speedup and scale up the computation. Recall that the low-rank assumption is only held for the incremental matrix, but for the modified graph (i.e., $\tilde{\mathbf{A}}$), it is not limited to be low-rank. Mathematically, the low-rank modification can be represented as:

$$\tilde{\mathbf{A}} = \mathbf{A} + \Delta\mathbf{A} = \mathbf{A} + \mathbf{U}\mathbf{V}', \tag{14}$$

where $\mathbf{U}, \mathbf{V} \in \mathbb{R}^{n \times r}$, and r is the rank of $\Delta\mathbf{A}$. Hence, by substituting $\mathbf{A}$ with $\mathbf{A} + \mathbf{U}\mathbf{V}'$ in Eq. (7) (i.e., $G = \{\mathbf{A} + \mathbf{U}\mathbf{V}', \mathbf{X}\}$) and changing the optimization variable from $\mathbf{A}$ into $\mathbf{U}$ and $\mathbf{V}$, we can obtain hyper-gradient with respect to $\mathbf{U}$ and $\mathbf{V}$ (i.e., $\nabla_{\mathbf{U}} \mathcal{L}$ and $\nabla_{\mathbf{V}} \mathcal{L}$) in the same manner as Eq.(10). By aggregating the hyper-gradients from different training/validation splits as we introduced in Sec. 3-B, we obtain aggregated hyper-gradients $\nabla\mathbf{U}$ and $\nabla\mathbf{V}$. Any gradient descent-based method can then be used to update $\mathbf{U}$ and $\mathbf{V}$.

In this way, we can significantly reduce the time and space complexity, which is summarized in the following lemma. Notice that n, m, d are number of nodes, number of edges and feature dimension, respectively and we have $d \ll n$ and $m \ll n^2$. As a comparison, the time complexity of computing $\nabla_{\mathbf{A}} \mathcal{L}$ is $O(n^2 d)$ and the space complexity of computing $\nabla_{\mathbf{A}} \mathcal{L}$ is $O(n^2)$. Hence, this low-rank method is much more efficient in both time and space.

Lemma 1. *For computing $\nabla_{\mathbf{U}} \mathcal{L}$ and $\nabla_{\mathbf{V}} \mathcal{L}$, the time complexity is $O(nd^2 + md)$ and the space complexity is $O(m + nd)$.*

Proof. See Appendix. □

4 EXPERIMENTS

Here, we design experiments to answer the following questions:

- **RQ1** How applicable is the proposed GaSoliNe with respect to different backbone/downstream classifiers, as well as different modification strategies?
- **RQ2** How effective is the proposed GaSoliNe for initial graphs under various forms of perturbation? To what extent does the proposed GaSoliNe strengthen the existing robust GNNs?
- **RQ3** How efficient and effective is the low-rank GaSoliNe?

Remark. A case study about the behaviour of GaSoliNe and hyper-parameter sensitivity studies are provided in Appendix for space.

4.1 Experiment Setups

We evaluate the proposed GaSoliNe on Cora, Citeseer, and Polbolgs datasets [26, 64, 65]. Since the Polblogs dataset does not contain node features, an identity matrix is used as the node feature matrix. All the datasets are undirected unweighted graphs and we experiment on the largest connected component of every dataset.

In order to set fair modification budgets across different datasets, the modification budget on adjacency matrix $B_{\texttt{topo}}$ is defined as $B_{\texttt{topo}} = m \times \texttt{modification rate}_{\texttt{topo}}$ and the budget on feature matrix $B_{\texttt{fea}}$ is defined as $B_{\texttt{fea}} = n \times d \times \texttt{modification rate}_{\texttt{fea}}$. where m, n, and d are the number of edges, nodes, and node features. In all the experiments, $\texttt{modification rate}_{\texttt{topo}} = 0.1$ and $\texttt{modification rate}_{\texttt{fea}} = 0.001$. Detailed hyper-parameter settings are attached in Appendix. We report the mean ± std accuracy over 10 repetitions as the evaluation metric.

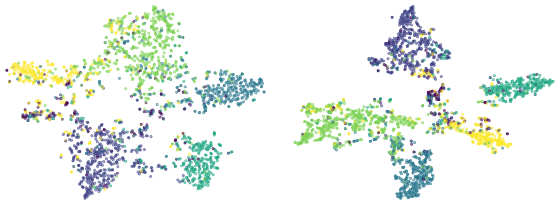
4.2 Applicability of GaSoliNe

The proposed GaSoliNe trains a *backbone classifier* in the lower-level problem and uses the trained backbone classifier to modify the initially-provided graph and improve the performance of the *downstream classifier* on the test nodes. In addition, GaSoliNe is capable of modifying both the graph topology (i.e., $\mathbf{A}$) and feature (i.e., $\mathbf{X}$) in both the discretized and continuous fashion. To verify that, we select three classic GNNs-based node classifiers, including GCN [26], SGC [51], and APPNP [27] as the backbone classifiers and the downstream classifiers. The detailed experiment procedure is as follows. First, we modify the given graph using proposed GaSoliNe algorithm with 4 modification strategies (i.e., modifying topology or node feature with discretized or continuous modification). Each variant is implemented with 3 backbone classifiers so that in total there are 12 sets of GaSoliNe settings. Second, for every of the 12 modified graphs, we evaluate 3 downstream classifiers and report the result (mean±std Acc). For this subsection, the initially provided graph is Citeseer [26].

Experimental results are reported in Table 3 where 'DT', 'CT', 'DF', and 'CF' denote 'discretized topology', 'continuous topology', 'discretized feature', and 'continuous feature' modifications, respectively. The second row of Table 3 shows the results on the initially-provided graph and the other rows denote the results on modified graphs with different settings. We use • to indicate that the improvement of the result is statistically significant compared with results on the initially-provided graph with a p-value< 0.01, and we use ∘

Variant	Backbone	GCN	SGC	APPNP
None	None	72.2±0.5	72.8±0.2	71.8±0.4
DT	GCN	74.7±0.3●	74.8±0.1●	75.4±0.2●
	SGC	74.7±0.4●	75.2±0.2●	75.6±0.3●
	APPNP	74.6±0.3●	74.6±0.1●	75.4±0.4●
DF	GCN	72.4±0.3○	72.7±0.2○	72.8±0.4●
	SGC	73.3±0.5●	73.4±0.2●	73.6±0.4●
	APPNP	72.6±0.3○	72.9±0.1○	73.6±0.4●
CT	GCN	73.1±0.4●	73.6±0.1●	74.8±0.2●
	SGC	73.0±0.3●	73.5±0.2●	74.4±0.3●
	APPNP	72.8±0.5○	73.4±0.1●	74.4±0.9●
CF	GCN	72.7±0.4○	73.6±0.1●	73.8±0.3●
	SGC	72.9±0.4●	73.6±0.4●	73.8±0.4●
	APPNP	73.0±0.3●	73.6±0.2●	73.9±0.3●

Table 3: Effectiveness of GASOLINE under multiple variants (Mean±Std Accuracy). The first and second columns denote the modification strategies and backbone classifiers adopted by GASOLINE respectively. The remaining columns show the performance of various downstream classifiers. • indicates significant improvement compared with results on the original graph (values at the second row) with a p-value<0.01 and ○ indicates no statistically significant improvement.



(a) Original **(b) After modification**

Figure 1: Visualization of node embeddings from original Citeseer graph (a) and modified Citeseer graph by GASOLINE (b). Best viewed in color.

to indicate no statistically significant improvement. We have the following observations. First, in the vast majority cases, the proposed GASOLINE is able to statistically significantly improve the accuracy of the downstream classifier over the initially-provide graph, for every combination of the modification strategy (discretized vs. continuous) and the modification target (topology vs. feature). Second, the graphs modified by GASOLINE with different backbone classifier can benefit different downstream classifiers, which demonstrates great transferability and broad applicability.

We further provide visualization of node embeddings before and after modification. We present the visualizations of initial Citeseer graph and the modified Citeseer graph from GASOLINE DT variant with SGC [51] backbone classifier. The detailed procedure is that a GCN [26] is trained on the training set $\mathcal{T}$ of given initial/modified graphs and hidden representations are used as node embeddings. Then, t-SNE [43] maps the node embeddings into two-dimensional ones for visualization. Figure 1 shows the visualization results from the original and the modified Citeseer graphs. Clearly, the node embeddings from modified graph are more discriminative than the embeddings from the original graph. In specific, the clusters are more cohesive and there is less overlap between clusters in the modified graphs (i.e., Figures 1b) compared with those on the original graph (i.e., Figure 1a). It further demonstrates that even we have no knowledge about the downstream classifiers (in this case the backbone classifier and downstream classifier are different), the proposed GASOLINE can still improve the graph quality to benefit downstream classifiers.

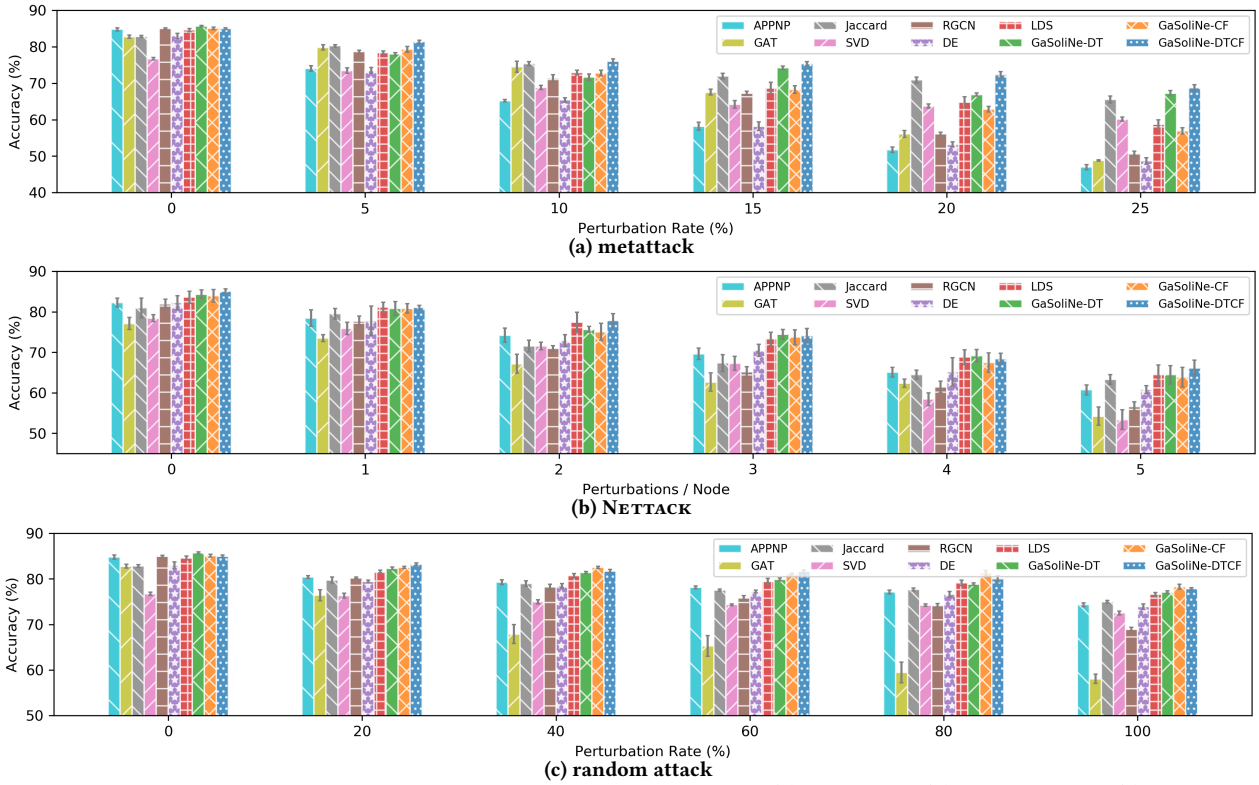
4.3 Effectiveness of GASOLINE

As we point out in Sec. 1, the defects of the initially-provided graph can be due to various reasons. In this subsection, we evaluate the effectiveness of the proposed GASOLINE by (A) the comparison with baseline methods on various poisoned/noisy graphs and (B) integrating with existing robust GNNs methods. The attack methods we adopt are as follows: (1) Random Attack randomly flips entries of benign adjacency matrices with different `perturbation rate`; (2) NETTACK [64] attacks a set of target nodes with different `perturbations/node`; (3) metattack [65] poisons the performance of node classifiers by perturbing the overall benign graph topology with different `perturbation rate`.

A - Comparison with baseline methods. We compare GASOLINE with the following baseline methods: APPNP [27], GAT [44], Jaccard [52], SVD [10], RGCN [61], DE [37], and LDS [13]. Recall that we feed all the graph modification-based methods (Jaccard, SVD, DE, LDS, GASOLINE) with the exactly same downstream classifier (APPNP) for a fair comparison.

We set 3 variants of GASOLINE to compare with the above baselines. To be specific, we refer to (1) GASOLINE with discretized modification on topology as GASOLINE-DT, (2) GASOLINE with continuous modification on feature as GASOLINE-CF, and (3) GASOLINE with discretized modification on topology and continuous modification on feature as GASOLINE-DTCF. All these GASOLINE variants use APPNP [27] as both the backbone classifier and the downstream classifier. We test various perturbation rates (i.e., `perturbation rate` of metattack from 5% to 25% with a step of 5%, `perturbation rate` of random attack from 20% to 100% with a step of 20%, and `perturbations/node` of NETTACK from 1 to 5) to attack the Cora [26] dataset and report the accuracy (mean±std) in Figure 2. From experiment results we observe that: (1) with the increase of adversarial perturbation, the performance of all methods drops, which is consistent with our intuition; (2) variants of GASOLINE consistently outperform the baselines under various adversarial/noisy scenarios; and (3) the proposed GASOLINE even improves over the original, benign graphs (i.e., 0 `perturbation rate` and 0 `perturbations/node`).

An interesting question is, if the initially-provided graph is heavily poisoned/noisy, to what extent is the proposed GASOLINE still effective? To answer this question, we study the performance of GASOLINE and other baseline methods on heavily-poisoned graphs (100% `perturbation rate` of random attack, 25% `perturbation rate` of metattack, and 5 `perturbations/node` of NETTACK). The detailed experiment results are presented in Table 4. In most cases, GASOLINE can obtain competitive or even better performance against baseline methods. On the Polblogs graph, GASOLINE does not perform as well as in the other two datasets. This is because, (1) the Polblogs graph does not have node feature which weakens the effectiveness of modification from GASOLINE and (2) the Polblogs graph has strong low-rank structure, which can be further verified in Sec. 4.4. As flexible solutions, in the next subsection, we study that if GASOLINE can work together with other graph defense methods.



(a) metattack

(b) NETTACK

(c) random attack

Figure 2: Performance comparison between baselines and GASOLINE under (a) metattack, (b) NETTACK, and (c) random attack with various perturbation rates. Best viewed in color.

Attack	Data	APPNP	GAT	Jaccard	SVD	RGCN	DE	LDS	G-DT	G-CF	G-DTCF
metattack	Cora	47.0±0.7	48.8±0.2	65.4±0.9	60.3±0.8	50.6±0.8	48.7±0.9	58.7±1.3	67.3±0.7	57.0±0.9	**68.8±0.9**
	Citeseer	49.4±2.2	62.4±0.7	57.1±1.0	49.5±0.8	55.5±1.4	50.1±2.3	58.2±2.3	**63.5±1.5**	58.4±1.5	62.2±1.0
	Polblogs	58.4±3.6	48.2±6.6	N/A	**79.1±2.4**	50.8±0.9	56.4±6.3	63.7±5.7	65.0±0.7	55.0±4.1	64.7±1.4
Nettack	Cora	60.7±1.2	54.2±2.3	63.7±1.4	52.9±2.8	56.5±1.1	60.8±1.0	64.5±2.4	64.5±2.2	63.9±2.4	**66.1±1.9**
	Citeseer	68.3±6.8	61.9±4.4	72.5±3.3	50.2±6.6	56.4±1.5	63.3±4.7	71.0±3.3	71.6±3.9	69.4±4.8	**74.3±1.6**
	Polblogs	90.5±1.0	91.1±0.7	N/A	**93.6±1.2**	93.1±0.2	89.1±2.4	91.1±1.8	92.3±1.6	90.3±0.7	92.4±1.7
random attack	Cora	74.3±0.4	58.1±1.0	75.1±0.5	72.6±0.3	68.9±0.4	73.9±0.6	76.6±0.4	77.1±0.3	**78.3±0.5**	77.8±0.2
	Citeseer	69.8±0.6	60.8±1.6	69.7±0.5	66.7±0.4	65.7±0.2	69.4±0.5	72.3±0.4	**73.8±0.2**	72.3±0.4	73.4±0.5
	Polblogs	74.7±2.8	**84.5±1.0**	N/A	83.3±2.8	81.7±0.9	75.9±1.4	73.2±2.8	73.4±4.1	77.1±1.6	77.6±2.9

Table 4: Comparison with baselines on heavily poisoned datasets (Mean±Std Accuracy). Some results are not applicable since Jaccard requires node features which are absent on Polblogs graph. G denotes GASOLINE for short.

B - Incorporating with graph defense strategies. GASOLINE does not make any assumption about the property of the defects of the initially-provided graph. We further evaluate if GASOLINE can boost the performance of both model-based and data-based defense strategies under the heavily-poisoned settings. We use a data-based baseline SVD [10], a model-based baseline RGCN [61], and another strong baseline GAT [44] to integrate with GASOLINE since they have shown competitive performance from Table 4 and Figure 2. The detailed procedure is that for model-based methods (GAT and RGCN), GASOLINE modifies the graph first, and then the baselines are implemented on the modified graphs to report the final results. For the data-based method (SVD), the baseline is implemented to preprocess graphs first, and then we modify graphs again by GASOLINE, and finally run the downstream classifiers (APPNP) on the twice-modified graphs. In this task, GASOLINE-DTCF is adopted.

In order to heavily poison the graphs, we use metattack [65] with `perturbation rate` as 25% to attack the benign graphs. We report the results in Table 1 and observe that after integrating with GASOLINE, performance of all the defense methods further improves significantly with a p-value<0.01.

4.4 Efficacy of Low-Rank GASOLINE

To answer RQ3, we first compare the performance of APPNP [27] on two modified graphs from the low-rank GASOLINE (denoted as GASOLINE-LR) and the original GASOLINE, respectively. Specifically, for the original GASOLINE, we adopt its variant with continuous modification towards the network topology. Due to the space limitation, we only show the results given graphs perturbed by metattack [65] in Table 5. We observe that in most settings on both the

Cora and Citeseer datasets, the GaSoliNe-LR can obtain promising performance against the original GaSoliNe. Surprisingly, on the Polblogs dataset, the GaSoliNe-LR shows great advantages over the original GaSoliNe. One possible explanation is that Polblogs dataset is inherently low-rank (which can be corroborated by Table 4 where SVD [10] obtains strong performance) and GaSoliNe-LR learns a low-rank incremental matrix which amplifies the advantage further.

To verify the efficiency of proposed GaSoliNe-LR, we generate a set of synthetic graphs with different number of nodes n. The wall-clock time for computing hyper-gradient is presented in Figure 3. Clearly, the GaSoliNe-LR is much more efficient compared with the original GaSoliNe especially when the network size is large.

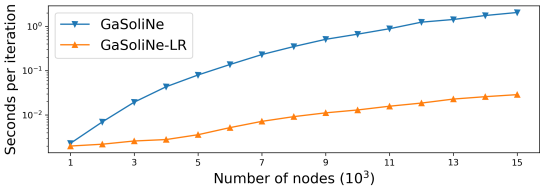


Figure 3: Efficiency comparison between GaSoliNe and GaSoliNe-LR

5 RELATED WORK

A - Graph Modification Most existing works on graph modification assume the initially provided graph is impaired or perturbed in a specific way. Network imputation problems restore missing links in a partially observed graph. For example, Liben-Nowell and Kleinberg [31] study the property of node topology proximity measures; Huisman [18] handles missing data in exponential random graphs. Besides, knowledge graph completion is to predict missing links between entities. The representative works include TransE [4], TransH [50], ComplEx [42] and many more. In another line of research, network enhancement and denoising problems delete irrelevant links for the given graphs. Arts such as NE [46], E-net [53], Mask-GVAE [29] study this problem under various scenarios. For network connectivity analysis, Chen et al. [5, 6] modify the underlying topology to manipulate the graph connectivity. Another relevant line is adversarial defense, which is a response for the attack on graph mining models. Wu et al. [52] propose to delete edges connecting two dissimilar nodes; Entezari et al. [10] adopt the low-rank approximation of given graphs to retain the performance of downstream GCNs; Jin et al. [20] merge the topology sparsity and feature smoothness into the optimization goal. In addition, supervised PageRank [2, 30] and constrained spectral clustering [48] also encode extra supervision to guide the modification of graphs. Recently many graph structural learning arts [15, 49, 59] emerge such as neural sparsification [34, 60], augmentation-based contrastive learning [16, 40, 56, 57, 63], and many more. Zhu et al. [62] provide a comprehensive survey about advanced arts on that.

B - Bilevel Optimization Bilevel optimization problem is a powerful mathematical tool with broad applications. For instance, Finn et al. [11] formulate the learning to initialization problem in the bilevel optimization context; Li et al. [28] propose a bilevel optimization-based poisoning attack method for factorization-based systems.

Data	Ptb Rate (%)	APPNP	GaSoliNe	GaSoliNe-LR
Cora	0	84.0±0.4	85.2±0.2	84.4±0.3
	5	74.1±0.7	77.4±0.5	75.0±0.3
	10	65.2±0.4	70.8±0.5	67.9±0.9
	15	58.2±1.1	67.1±0.8	65.3±0.8
	20	51.7±0.7	62.5±0.5	60.2±1.2
	25	47.0±0.7	57.3±0.6	57.1±0.5
Citeseer	0	71.8±0.4	74.7±0.2	73.4±0.2
	5	67.6±0.9	69.6±0.7	68.2±0.8
	10	61.8±0.8	66.3±1.0	63.9±0.4
	15	54.1±0.8	59.3±1.1	56.8±1.1
	20	51.0±1.2	56.5±0.9	55.3±0.9
	25	49.4±2.2	57.7±1.8	56.5±0.8
Polblogs	0	94.1±0.6	95.3±0.6	95.7±0.3
	5	70.1±0.6	73.8±0.9	93.4±0.3
	10	69.8±0.8	72.8±0.4	90.8±0.2
	15	67.5±0.5	70.1±1.2	88.7±0.3
	20	64.1±0.9	68.5±1.0	88.0±0.3
	25	57.0±3.6	64.8±2.1	89.9±0.5

Table 5: Effectiveness comparison between GaSoliNe and GaSoliNe-LR

There are effective solutions such as the forward and reverse gradients [12], truncated back-propagation [38] and so on. In addition, Colson et al. [8] provide a detailed review about this topic. The most related arts to our work are [7] and [13]. Both of them aim to modify (or generate from scratch) the given data in a bilevel optimization context. The former studies a data debugging problem under the collaborative filtering scenario whose lower-level problem has a closed-form solution. The latter models every edge with a Bernoulli random variable. As a comparison, the lower-level problem of graph sanitation may not necessarily have a closed-form solution and we modify the initially provided graphs deterministically with versatile variants and broader applications.

6 CONCLUSION

In this paper, we introduce the graph sanitation problem, which aims to improve an initially-provided graph for a given graph mining model. We formulate the graph sanitation problem as a bilevel optimization problem and show that it can be instantiated by a variety of graph mining models such as supervised PageRank, supervised clustering and node classification. We further propose an effective solver named GaSoliNe for the graph sanitation problem with semi-supervised node classification. GaSoliNe adopts an efficient approximation of hyper-gradient to guide the modification over the initially-provided graph. GaSoliNe is versatile, and equipped with multiple variants. The extensive experimental evaluations demonstrate the broad applicability and effectiveness of the proposed GaSoliNe.

ACKNOWLEDGMENTS

This work is supported by NSF (1947135 and 2134079), DARPA (HR001121C0165), ARO (W911NF2110088), and DHS (17STQAC00001-03-03). The content of the information in this document does not necessarily reflect the position or the policy of the Government, and no official endorsement should be inferred. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation here on.

REFERENCES

[1] Basak Alper, Benjamin Bach, Nathalie Henry Riche, Tobias Isenberg, and Jean-Daniel Fekete. 2013. Weighted graph comparison techniques for brain connectivity analysis. In *SIGCHI*. 483–492.

[2] Lars Backstrom and Jure Leskovec. 2011. Supervised random walks: predicting and recommending links in social networks. In *WSDM*. 635–644.

[3] Atılım Günes Baydin, Barak A Pearlmutter, Alexey Andreyevich Radul, and Jeffrey Mark Siskind. 2017. Automatic differentiation in machine learning: a survey. *The Journal of Machine Learning Research* 18, 1 (2017), 5595–5637.

[4] Antoine Bordes, Nicolas Usunier, Alberto Garcia-Duran, Jason Weston, and Oksana Yakhnenko. 2013. Translating embeddings for modeling multi-relational data. In *NIPS*. 1–9.

[5] Chen Chen, Ruiyue Peng, Lei Ying, and Hanghang Tong. 2018. Network connectivity optimization: Fundamental limits and effective algorithms. In *SIGKDD*. 1167–1176.

[6] Chen Chen, Hanghang Tong, B Aditya Prakash, Charalampos E Tsourakakis, Tina Eliassi-Rad, Christos Faloutsos, and Duen Horng Chau. 2015. Node immunization on large graphs: Theory and algorithms. *IEEE Transactions on Knowledge and Data Engineering* 28, 1 (2015), 113–126.

[7] Long Chen, Yuan Yao, Feng Xu, Miao Xu, and Hanghang Tong. 2020. Trading Personalization for Accuracy: Data Debugging in Collaborative Filtering. *NeurIPS* 33 (2020).

[8] Benoît Colson, Patrice Marcotte, and Gilles Savard. 2007. An overview of bilevel optimization. *Annals of operations research* 153, 1 (2007), 235–256.

[9] Boxin Du, Changhe Yuan, Robert Barton, Tal Neiman, and Hanghang Tong. 2021. Hypergraph Pre-training with Graph Neural Networks. *arXiv preprint arXiv:2105.10862* (2021).

[10] Negin Entezari, Saba A Al-Sayouri, Amirali Darvishzadeh, and Evangelos E Papalexakis. 2020. All You Need Is Low (Rank) Defending Against Adversarial Attacks on Graphs. In *WSDM*. 169–177.

[11] Chelsea Finn, Pieter Abbeel, and Sergey Levine. 2017. Model-agnostic meta-learning for fast adaptation of deep networks. In *ICML*. PMLR, 1126–1135.

[12] Luca Franceschi, Michele Donini, Paolo Frasconi, and Massimiliano Pontil. 2017. Forward and reverse gradient-based hyperparameter optimization. In *ICML*. PMLR, 1165–1173.

[13] Luca Franceschi, Mathias Niepert, Massimiliano Pontil, and Xiao He. 2019. Learning discrete structures for graph neural networks. In *ICML*. PMLR, 1972–1982.

[14] Zoltan Gyongyi, Hector Garcia-Molina, and Jan Pedersen. 2004. Combating web spam with trustrank. In *VLDB*.

[15] Jonathan Halcrow, Alexandru Mosoi, Sam Ruth, and Bryan Perozzi. 2020. Grale: designing networks for graph learning. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 2523–2532.

[16] Kaveh Hassani and Amir Hosein Khasahmadi. 2020. Contrastive multi-view representation learning on graphs. In *ICML*.

[17] Taher H Haveliwala. 2003. Topic-sensitive pagerank: A context-sensitive ranking algorithm for web search. *IEEE transactions on knowledge and data engineering* 15, 4 (2003), 784–796.

[18] Mark Huisman. 2009. Imputation of missing network data: Some simple procedures. *Journal of Social Structure* 10, 1 (2009), 1–29.

[19] Glen Jeh and Jennifer Widom. 2003. Scaling personalized web search. In *WWW*. 271–279.

[20] Wei Jin, Yao Ma, Xiaorui Liu, Xianfeng Tang, Suhang Wang, and Jiliang Tang. 2020. Graph Structure Learning for Robust Graph Neural Networks. In *SIGKDD*. ACM, 66–74.

[21] Baoyu Jing, Chanyoung Park, and Hanghang Tong. 2021. Hdmi: High-order deep multiplex infomax. In *Proceedings of the Web Conference 2021*. 2414–2424.

[22] Baoyu Jing, Hanghang Tong, and Yada Zhu. 2021. Network of Tensor Time Series. In *Proceedings of the Web Conference 2021*. 2425–2437.

[23] Jian Kang, Jingrui He, Ross Maciejewski, and Hanghang Tong. 2020. InFoRM: Individual Fairness on Graph Mining. In *SIGKDD*. 379–389.

[24] Jian Kang and Hanghang Tong. 2019. N2n: Network derivative mining. In *CIKM*. 861–870.

[25] Matt J Keeling and Ken TD Eames. 2005. Networks and epidemic models. *Journal of the Royal Society Interface* 2, 4 (2005), 295–307.

[26] Thomas N. Kipf and Max Welling. 2017. Semi-Supervised Classification with Graph Convolutional Networks. In *ICLR*.

[27] Johannes Klicpera, Aleksandar Bojchevski, and Stephan Günnemann. 2018. Predict then Propagate: Graph Neural Networks meet Personalized PageRank. In *ICLR*.

[28] Bo Li, Yining Wang, Aarti Singh, and Yevgeniy Vorobeychik. 2016. Data poisoning attacks on factorization-based collaborative filtering. In *NIPS*. 1893–1901.

[29] Jia Li, Mengzhou Liu, Honglei Zhang, Pengyun Wang, Yong Wen, Lujia Pan, and Hong Cheng. 2021. Mask-GVAE: Blind Denoising Graphs via Partition. In *TheWebConf*. 3688–3698.

[30] Liangyue Li, Yuan Yao, Jie Tang, Wei Fan, and Hanghang Tong. 2016. QUINT: on query-specific optimal networks. In *SIGKDD*. 985–994.

[31] David Liben-Nowell and Jon Kleinberg. 2007. The link-prediction problem for social networks. *Journal of the American society for information science and technology* 58, 7 (2007), 1019–1031.

[32] Lihui Liu, Boxin Du, Yi Ren Fung, Heng Ji, Jiejun Xu, and Hanghang Tong. 2021. KompaRe: A Knowledge Graph Comparative Reasoning System. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*. 3308–3318.

[33] Lihui Liu, Boxin Du, Heng Ji, ChengXiang Zhai, and Hanghang Tong. 2021. Neural-Answering Logical Queries on Knowledge Graphs. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*. 1087–1097.

[34] Dongsheng Luo, Wei Cheng, Wenchao Yu, Bo Zong, Jingchao Ni, Haifeng Chen, and Xiang Zhang. 2021. Learning to drop: Robust graph neural network via topological denoising. In *WSDM*.

[35] Alex Nichol, Joshua Achiam, and John Schulman. 2018. On first-order meta-learning algorithms. *arXiv preprint arXiv:1803.02999* (2018).

[36] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. 1999. *The PageRank citation ranking: Bringing order to the web.* Technical Report. Stanford InfoLab.

[37] Yu Rong, Wenbing Huang, Tingyang Xu, and Junzhou Huang. 2019. DropEdge: Towards Deep Graph Convolutional Networks on Node Classification. In *ICLR*.

[38] Amirreza Shaban, Ching-An Cheng, Nathan Hatch, and Byron Boots. 2019. Truncated back-propagation for bilevel optimization. In *AISTATS*. PMLR, 1723–1732.

[39] Jianbo Shi and Jitendra Malik. 2000. Normalized cuts and image segmentation. *IEEE Transactions on pattern analysis and machine intelligence* 22, 8 (2000), 888–905.

[40] Susheel Suresh, Pan Li, Cong Hao, and Jennifer Neville. 2021. Adversarial Graph Augmentation to Improve Graph Contrastive Learning. *arXiv preprint arXiv:2106.05819* (2021).

[41] Hanghang Tong, Christos Faloutsos, and Jia-Yu Pan. 2006. Fast random walk with restart and its applications. In *ICDM*. IEEE, 613–622.

[42] Théo Trouillon, Johannes Welbl, Sebastian Riedel, Éric Gaussier, and Guillaume Bouchard. 2016. Complex embeddings for simple link prediction. In *ICML*. PMLR, 2071–2080.

[43] Laurens Van Der Maaten. 2014. Accelerating t-SNE using tree-based algorithms. *The Journal of Machine Learning Research* 15, 1 (2014), 3221–3245.

[44] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. 2018. Graph Attention Networks. In *ICLR*.

[45] Kiri Wagstaff and Claire Cardie. 2000. Clustering with instance-level constraints. *AAAI/IAAI* 1097 (2000), 577–584.

[46] Bo Wang, Armin Pourshafeie, Marinka Zitnik, Junjie Zhu, Carlos D Bustamante, Serafim Batzoglou, and Jure Leskovec. 2018. Network enhancement as a general method to denoise weighted biological networks. *Nature communications* 9, 1 (2018), 1–8.

[47] Daixin Wang, Jianbin Lin, Peng Cui, Quanhui Jia, Zhen Wang, Yanming Fang, Quan Yu, Jun Zhou, Shuang Yang, and Yuan Qi. 2019. A semi-supervised graph attentive network for financial fraud detection. In *ICDM*. IEEE, 598–607.

[48] Xiang Wang and Ian Davidson. 2010. Flexible constrained spectral clustering. In *SIGKDD*. 563–572.

[49] Yiwei Wang, Wei Wang, Yuxuan Liang, Yujun Cai, Juncheng Liu, and Bryan Hooi. 2020. Nodeaug: Semi-supervised node classification with data augmentation. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 207–217.

[50] Zhen Wang, Jianwen Zhang, Jianlin Feng, and Zheng Chen. 2014. Knowledge graph embedding by translating on hyperplanes. In *AAAI*, Vol. 28.

[51] Felix Wu, Amauri Souza, Tianyi Zhang, Christopher Fifty, Tao Yu, and Kilian Weinberger. 2019. Simplifying Graph Convolutional Networks. In *ICML*. 6861–6871.

[52] Huijun Wu, Chen Wang, Yuriy Tyshetskiy, Andrew Docherty, Kai Lu, and Liming Zhu. 2019. Adversarial examples for graph data: deep insights into attack and defense. In *IJCAI*. AAAI Press, 4816–4823.

[53] Jiarong Xu, Yang Yang, Chunping Wang, Zongtao Liu, Jing Zhang, Lei Chen, and Jiangang Lu. 2020. Robust Network Enhancement from Flawed Networks. *IEEE Transactions on Knowledge and Data Engineering* (2020).

[54] Lian Yan, Robert Dodier, Michael C Mozer, and Richard Wolniewicz. 2003. Optimizing classifier performance via an approximation to the Wilcoxon-Mann-Whitney statistic. In *ICML*. 848–855.

[55] Yuchen Yan, Lihui Liu, Yikun Ban, Baoyu Jing, and Hanghang Tong. 2021. Dynamic Knowledge Graph Alignment. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 35. 4564–4572.

[56] Yuning You, Tianlong Chen, Yang Shen, and Zhangyang Wang. 2021. Graph Contrastive Learning Automated. In *ICML*, Marina Meila and Tong Zhang (Eds.).

[57] Yuning You, Tianlong Chen, Yongduo Sui, Ting Chen, Zhangyang Wang, and Yang Shen. 2020. Graph contrastive learning with augmentations. *NeurIPS* (2020).

[58] Reza Zafarani, Mohammad Ali Abbasi, and Huan Liu. 2014. *Social media mining: an introduction.* Cambridge University Press.

[59] Tong Zhao, Yozen Liu, Leonardo Neves, Oliver Woodford, Meng Jiang, and Neil Shah. 2021. Data Augmentation for Graph Neural Networks. In *Proceedings of*

the AAAI Conference on Artificial Intelligence, Vol. 35. 11015–11023.
[60] Cheng Zheng, Bo Zong, Wei Cheng, Dongjin Song, Jingchao Ni, Wenchao Yu, Haifeng Chen, and Wei Wang. 2020. Robust graph representation learning via neural sparsification. In *ICML*.
[61] Dingyuan Zhu, Ziwei Zhang, Peng Cui, and Wenwu Zhu. 2019. Robust graph convolutional networks against adversarial attacks. In *SIGKDD*. 1399–1407.
[62] Yanqiao Zhu, Weizhi Xu, Jinghao Zhang, Qiang Liu, Shu Wu, and Liang Wang. 2021. Deep Graph Structure Learning for Robust Representations: A Survey. *arXiv preprint arXiv:2103.03036* (2021).
[63] Yanqiao Zhu, Yichen Xu, Feng Yu, Qiang Liu, Shu Wu, and Liang Wang. 2021. Graph contrastive learning with adaptive augmentation. In *The Web Conference*.
[64] Daniel Zügner, Amir Akbarnejad, and Stephan Günnemann. 2018. Adversarial attacks on neural networks for graph data. In *SIGKDD*. 2847–2856.
[65] Daniel Zügner and Stephan Günnemann. 2019. Adversarial Attacks on Graph Neural Networks via Meta Learning. In *ICLR*.

A REPRODUCIBILITY

The code is available at https://github.com/pricexu/GASOLINE. Three public graph datasets, Cora[1], Citeseer[1], and Polbolgs [2], are used in our paper with following detailed statistics in Table 6

Data	Nodes	Edges	Classes	Features
Cora	2,485	5,069	7	1,433
Citeseer	2,110	3,668	6	3,703
Polblogs	1,222	16,714	2	N/A

Table 6: Statistics of Datasets

A.1 Hyper-Parameter Settings

We summarize the hyper-parameter settings of models implemented in our experiments including *baseline methods*, *backbone* and *downstream classifiers* of GASOLINE:

- **GCN [26]/GAT [44]/RGCN [61]:** We follow the default settings of publicly available implementation of GCN[1], GAT[3], RGCN[4].
- **SGC [51]:** The implementation of SGC is the same as GCN [26] but we remove the activation function of the hidden layers.
- **APPNP [27]:** We follow the recommended hyper-parameter settings of APPNP [27] in the original paper.
- **Jaccard [52]:** We search the edge removing threshold of Jaccard similarity from $\{0.01, 0.03, 0.05, 0.1, 0.3, 0.5\}$ and report the best.
- **SVD [10]:** We search the rank of SVD from $\{10, 30, 50, 100, 200, 300\}$ and report the best results from the above settings.
- **DE [37]:** We search the dropout rate of DE over existing edges from $\{0.05, 0.1, 0.15, 0.2, 0.25\}$ and report the best results from the above settings.
- **LDS [13]:** We implement LDS with the same modification budget as GASOLINE-CT.

The detailed settings of GASOLINE are as follows: (1) for all the modification strategies (discretized vs. continuous and topology vs. feature), the modification budget towards topology $B_{\texttt{topo}}$ and the modification budget towards feature $B_{\texttt{fea}}$ are introduced in the Section 4.1. Specifically, the $\texttt{modification rate}_{\texttt{topo}} = 0.1$ and $\texttt{modification rate}_{\texttt{fea}} = 0.001$. We modify the graph in 10 steps so the budget b in every modification step is $\lfloor \frac{B}{10} \rfloor$ (i.e., $b_{\texttt{topo}} = \lfloor \frac{B_{\texttt{topo}}}{10} \rfloor$ and $b_{\texttt{fea}} = \lfloor \frac{B_{\texttt{fea}}}{10} \rfloor$). (2) the settings of backbone classifiers and downstream classifiers (GCN [26], SGC [51], APPNP [27]) used in our experiments follow the aforementioned settings. (3) the number of iterations for the optimization of lower-level problem T is set as 200 and the truncating iteration P is set as 196. The number of folds K is set as 8.

For the attacking methods, their attacking perturbation rates are introduced in the Section 4.3, and here we present the detailed implementation of them. (1) We follow the publicly-available implementation[5] of metattack [65] and adopt the 'Meta-Self' variant to attack the provided graphs; (2) we follow [20] to select nodes with degree larger than 10 as the target nodes and implement NETTACK with the publicly-available implementation[6]; (3) we implement random attack by symmetrically flipping $B_{\texttt{attack}}$ entries of the adjacency matrix $\mathbf{A}$ of provided graphs.

[1] https://github.com/tkipf/gcn
[2] https://github.com/ChandlerBang/Pro-GNN
[3] https://github.com/PetarV-/GAT
[4] https://github.com/ZW-ZHANG/RobustGCN
[5] https://github.com/danielzuegner/gnn-meta-attack
[6] https://github.com/danielzuegner/nettack

B PROOF OF LEMMA 1

Our complexity analysis is mainly based on the propagation formula of matrix multiplication-based GNNs and only focuses on a single-layered GNN with first order approximation of the hyper-gradient. However, it can be easily generalized into a wide range of scenarios (e.g., multi-layered GNNs) with similar analysis and conclusions. For brevity, we only analyze the complexity of computing $\nabla_{\mathbf{U}}\mathcal{L}$. The analysis w.r.t. computing $\nabla_{\mathbf{V}}\mathcal{L}$ is similar and we omit here.

Proof. For typical matrix multiplication-based GNNs (e.g., [26]), their propagation formula can be represented as $\mathbf{X} \leftarrow \sigma(\mathbf{AXW})$ (or even simplified by removing the nonlinear activation function σ and feature transformation matrix $\mathbf{W}$ between several layers [27, 51]). If we do not consider the gradient across the model parameter (i.e., $\mathbf{W}$) updating trajectory (i.e., first order approximation), and assume that our GNN contains only one layer, the hyper-gradient with respect to the vector $\mathbf{U}$ can be computed as follows,

$$\nabla_{\mathbf{U}}\mathcal{L} = [\frac{\partial \mathcal{L}}{\partial \sigma((\mathbf{A}+\mathbf{UV}')\mathbf{XW})} \circ \sigma'((\mathbf{A}+\mathbf{UV}')\mathbf{XW})]\mathbf{W}'\mathbf{X}'\mathbf{V}. \quad (15)$$

The computation of $(\mathbf{A}+\mathbf{UV}')\mathbf{XW}$ can be rewritten as $\mathbf{AXW} + \mathbf{UV}'\mathbf{XW}$. Note that $\mathbf{A}$ is a sparse matrix and the space cost is $O(m)$ for computing $\mathbf{AXW}$. The space cost is $O(nd)$ for $\mathbf{UV}'\mathbf{XW}$. For $\mathbf{W}'\mathbf{X}'\mathbf{V}$ the space cost is $O(nd)$. Put everything together the space cost for computing $\nabla_{\mathbf{U}}\mathcal{L}$ is $O(m+nd)$.

The time complexity of the part within $[\cdot]$ in Eq.(15) is $O(nd^2 + md)$. The time complexity of computing $\mathbf{W}'\mathbf{X}'\mathbf{V}$ is $O(ndr)$. The time complexity about the multiplication between $[\cdot]$ and $\mathbf{W}'\mathbf{X}'\mathbf{V}$ is $O(ndr)$. Hence, put everything together the total time complexity for computing $\nabla_{\mathbf{U}}\mathcal{L}$ is $O(nd^2 + md)$ given $r \ll d$. □

C CASE STUDY ABOUT THE BEHAVIOUR OF GASOLINE

Here, we further study the potential reasons behind the success of GASOLINE. To this end, we conduct a case study whose core idea is to label malicious modifications (from adversaries) and test if GASOLINE is able to detect them. The specific procedure is that we utilize different kinds of attackers (i.e., metattack [65], NETTACK [64], and random attack) to modify the graph structure of a *benign* graph G (with adjacency matrix $\mathbf{A}$) into a *poisoned* graph $G_{\texttt{adv}}$ (with adjacency matrix $\mathbf{A}_{\texttt{adv}}$). Then, we utilize the score matrix $\mathbf{S}$ from Eq. (12) to assign a score to every entry of the poisoned adjacency matrix $\mathbf{A}_{\texttt{adv}}$. As we mentioned in Section 3, the higher score an entry obtains, the more likely GASOLINE will modify it. We compute the average score of three groups of entries from $\mathbf{A}_{\texttt{adv}}$: the poisoned entries after adding/deleting perturbations from adversaries, the benign existing edges without perturbation, and the benign non-existing edges without perturbation. Remark that both the benign graphs and the poisoned graphs are unweighted and we define following auxiliary matrices. $\mathbf{A}_{\texttt{diff}} = |\mathbf{A}_{\texttt{adv}} - \mathbf{A}|$ is a difference matrix whose entries with value 1 indicate poisoned entries. $\mathbf{A}_{\texttt{benign-E}} = \mathbf{A} \circ (\mathbf{1} - \mathbf{A}_{\texttt{diff}})$ is a benign edge indicator matrix whose entries with value 1 indicate the benign existing edges without perturbation. $\circ$ indicates element-wise multiplication. $\mathbf{A}_{\texttt{benign-NE}} = (\mathbf{1} - \mathbf{A}) \circ (\mathbf{1} - \mathbf{A}_{\texttt{diff}})$ is a benign non-existing edge indicator matrix whose entries with value 1 indicate the benign non-existing edges without perturbation. Based on that, we have the following three statistics:

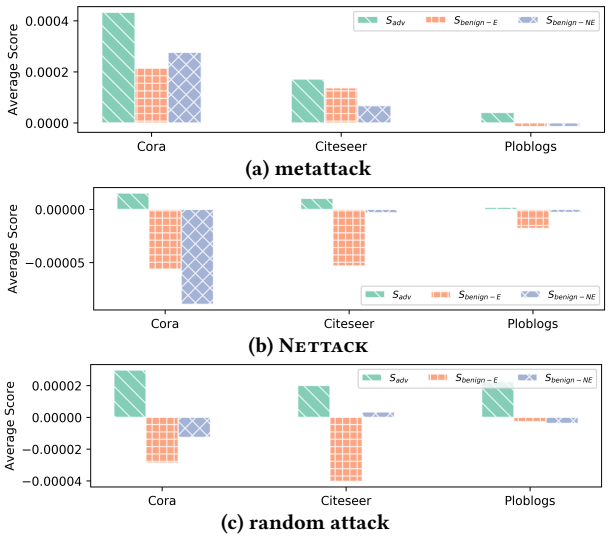


Figure 4: Score of various entries under metattack (a), NETTACK (b), and random attack (c). Best viewed in color.

$$S_{\text{adv}} = \frac{\sum_{i,j}(\mathbf{S} \circ \mathbf{A}_{\texttt{diff}})[i,j]}{\sum_{i,j}\mathbf{A}_{\texttt{diff}}[i,j]},$$

$$S_{\texttt{benign-E}} = \frac{\sum_{i,j}(\mathbf{S} \circ \mathbf{A}_{\texttt{benign-E}})[i,j]}{\sum_{i,j}\mathbf{A}_{\texttt{benign-E}}[i,j]},$$

$$S_{\texttt{benign-NE}} = \frac{\sum_{i,j}(\mathbf{S} \circ \mathbf{A}_{\texttt{benign-NE}})[i,j]}{\sum_{i,j}\mathbf{A}_{\texttt{benign-NE}}[i,j]},$$

which denote the average score obtained by poisoned entries, benign existing edges, and benign non-existing edges.

Detailed results are presented in Figure 4. We observe that GASOLINE tends to modify poisoned entries more (with higher scores) than to modify benign unperturbed entries in the adjacency matrix of poisoned graphs, which is consistent with our expectation and enables the algorithm to partially recover the benign graphs and to boost the performance of downstream classifiers.

D EFFECT OF MODIFICATION BUDGET

In this section we study the relationships between the budget of GASOLINE and the corresponding performance of the downstream classifier. Here, we instantiate two variants of GASOLINE: discretized modification towards topology (GASOLINE-DT) and continuous modification towards feature (GASOLINE-CF). The provided graph is Cora [26] which is heavily-poisoned by metattack [65] with `perturbation rate` = 25% (i.e., B). The perturbation budget per modification step b is set to be $\frac{B}{10}$. Both the backbone classifier and the downstream classifier of GASOLINE are the APPNP [27] models with the aforementioned settings. From Figure 5 we observe that with the increase of the budget (`modification` $\texttt{rate}_{\texttt{topo}}$ and `modification` $\texttt{rate}_{\texttt{fea}}$), GASOLINE enjoys great potential to further improve the performance of the downstream classifiers. At the same time, 'economic' choices are strong enough to benefit downstream classifiers so we set `modification` $\texttt{rate}_{\texttt{topo}}$ as 0.1 and `modification` $\texttt{rate}_{\texttt{fea}}$ as 0.001 throughout our experiment settings.

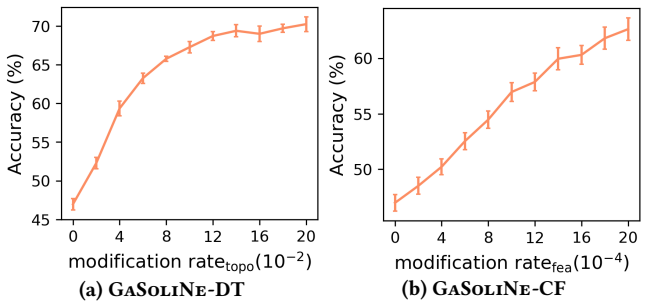


Figure 5: Performance of downstream classifier vs. the modification budget of GASOLINE-DT (a) and GASOLINE-CF (b)

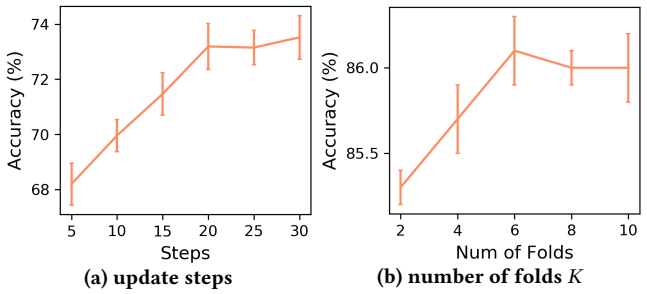


Figure 6: Performance of GASOLINE-DTCF vs. the update steps (a) and the number of folds K (b).

E EFFECTS OF MODIFICATION STEPS AND NUMBER OF FOLDS

As we claimed in the main content, in implementation we set the budget in every iteration as b and update the given graph multiple iterations till we run out of total budget B. Hence, the update steps equals to $\lceil \frac{B}{b} \rceil$. Intuitively less budget per iteration can provide finer update towards the given graphs. To validate that we test the performance of an instantiation of GASOLINE with discretized modification towards topology and continuous modification towards feature (GASOLINE-DTCF) on the Cora [26] graph which is poisoned by metattack [65] with `perturbation rate` = 25%. Both the backbone classifier and the downstream classifier of GASOLINE are the APPNP [27] with the aforementioned settings. From Figure 6a we observe that with more update steps downstream classifiers can get better performance. However, when the number of steps is larger than 20, the improvement of performance is minor.

In addition, the number of training/validation split fold K is another important hyper-parameter in our model. Intuitively larger K leads into better usage of the given data. To study the relationships between K and the corresponding performance of the downstream classifier, we implement GASOLINE-DTCF on the original Cora graph to verify that. Note that the `modification` $\texttt{rate}_{\texttt{topo}} = 0.1$, `modification` $\texttt{rate}_{\texttt{fea}} = 0.001$, and the number of modification steps is set as 10. From Figure 6b we observe that performance of the downstream classifier is improved with the increase of the number of folds. However, such performance gaining stops when $K = 6$. Hence, $K = 6$ is enough to make full use of the given graph by GASOLINE.