Certifiable Robustness of Graph Convolutional Networks under Structure Perturbations

Daniel Zügner and Stephan Günnemann
Technical University of Munich, Germany
{zuegnerd,guennemann}@in.tum.de

ABSTRACT

Recent works show that message-passing neural networks (MPNNs) can be fooled by adversarial attacks on both the node attributes and the graph structure. Since MPNNs are currently being rapidly adopted in real-world applications, it is thus crucial to improve their reliablility and robustness. While there has been progress on robustness certification of MPNNs under perturbation of the node attributes, no existing method can handle structural perturbations. These perturbations are especially challenging because they alter the *message passing scheme itself*. In this work we close this gap and propose the first method to certify robustness of Graph Convolutional Networks (GCNs) under perturbations of the graph structure. We show how this problem can be expressed as a jointly constrained bilinear program – a challenging, yet well-studied class of problems – and propose a novel branch-and-bound algorithm to obtain lower bounds on the global optimum. These lower bounds are significantly tighter and can certify up to twice as many nodes compared to a standard linear relaxation.

ACM Reference Format:
Daniel Zügner and Stephan Günnemann. 2020. Certifiable Robustness of Graph Convolutional Networks under Structure Perturbations. In *Proceedings of the 26th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '20), August 23–27, 2020, Virtual Event, CA, USA.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3394486.3403217

1 INTRODUCTION

Graph neural networks have recently enjoyed tremendous attention both in academia and practice. Graph neural networks are already deployed at scale at major tech companies (e.g. [15]). However, recent works (e.g., [2, 5, 17, 18]) show that graph neural networks are highly non-robust w.r.t. adversarial attacks. To date, there exists no effective defense strategy for the popular graph convolutional network (GCN) against perturbations of the graph structure. In this work, for the first time we close this gap and present a technique for certified robustness of GCN's predictions.

In robustness certification for 'traditional' data (i.e. images) it is sufficient to consider the data instances individually and determine whether we can certify a robust prediction. Robustness certification for graph neural networks is significantly more challenging since we have to take into account all nodes that influence the target node's prediction via message passing. The work [19] tackles such robustness certification of graph convolutional networks – however, they consider perturbations to the node features.

In this work we address for the first time the even more challenging setting where we assume the *graph structure* can be altered by an attacker. Specifically, we consider the realistic case where the attacker is allowed to insert new edges. This scenario reflects real-world conditions where new edges (corresponding to, e.g. links, likes, following, etc.) are cheap and easy to inject. A certificate issued by our method states that a node's prediction *could not have been altered* by edges potentially inserted by an attacker.

From a technical view, we solve the following three challenges:

(1) **Nonconvexity of the neural network**. Neural networks are (in general) nonconvex functions; therefore finding the optimal (i.e. worst-case) perturbation is intractable.
(2) **Discreteness of the data**. Since the adjacency matrix is binary, the number of possible perturbations grows exponentially with the perturbation budget; therfore enumerating and testing all admissible perturbations is intractable.
(3) **Modification of the message passing scheme**. While previous certificates deal with what happens when the *input* to the (static) MPNN is modified, we aim to certify robustness for cases when the graph structure – and therefore the way embeddings are propagated – changes. From the perspective of an individual node, the *computation graph itself* changes as the output is computed from a different set of nodes.

Indeed, **challenge (1)** is also encountered when certifying robustness in more traditional scenarios. For this aspect, we thus largely follow the convex relaxation of GCN presented by [19], which relies on a relaxation of the ReLU activation function introduced by [13].

Challenge (2), in contrast, requires special care: While [19] handles binary *attribute data* by performing a continuous relaxation, we will show in Section 3.2 that a standard continuous relaxation is not well-suited for perturbations of the *graph structure*. Instead, one of our contributions is to show how to bypass the problem altogether by working directly on the (continuous-valued) degree-normalized message passing matrix used by GCN. To this end we carefully construct *induced constraints* on the normalized message passing matrix from L_0 constraints on the (binary) adjacency matrix.

Challenge (3) – how to deal with changes to the message passing matrix – is completely unstudied and therefore our second main contribution. The difficulty lies in the fact that the message passing matrix (which becomes *variable*) is used at *each layer* – as opposed to the input features, which appear only in the first layer of the neural network. Thus, the ReLU relaxations of [13, 19] no longer lead to convex (linear) constraints but become nonconvex (details are discussed in Section 3.4). Indeed, we will show that the problem

KDD '20, August 23–27, 2020, Virtual Event, CA, USA

ACM ISBN 978-1-4503-7998-4/20/08…$15.00
https://doi.org/10.1145/3394486.3403217

can be expressed as a jointly constrained bilinear program – "one of the most persistently difficult and recurrent nonconvex problems in mathematical programming" [1]. To solve this challenging problem we propose a novel branch-and-bound algorithm that effectively exploits key insights to obtain lower bounds on the worst-case change in a node's prediction. If positive, these lower bounds serve as robustness certificates, stating that a node's prediction does not change under *any* admissible set of perturbations.

Related Work. GNNs are a fundamental part of the modern machine learning landscape and have been successfully used for a variety of tasks. However, GNNs are highly sensitive to small adversarial perturbations [5, 17, 18]. While some (heuristic) defenses exist [6, 14, 16], we can never assume attackers will not be able to break them.Robustness certificates, on the other hand, provide provable guarantees that no perturbation regarding a specific perturbation model will change the prediction of a sample. Only a few robustness certificates for graphs have been developed. While [19] can only handle perturbations to the node attributes, [3] is limited to a specific class of graph-models based on PageRank, not covering the highly important principle of graph convolutional networks. Our work closes this gap by providing the first robustness certificate for GCNs under structure perturbations. In concurrent work, [4] provide another certification method for structure perturbations.

2 PRELIMINARIES

We consider the task of (semi-supervised) node classification in a single large graph with D-dimensional node features. Let $G = (\boldsymbol{A}, \boldsymbol{X})$ be an attributed, unweighted graph, where $\boldsymbol{A} \in \{0,1\}^{N \times N}$ is the adjacency matrix and $\boldsymbol{X} \in \mathbb{R}^{N \times D}$ are the nodes' features. W.l.o.g. we assume the node-ids to be $\mathcal{V} = \{1, \ldots, N\}$. Given a subset $\mathcal{V}_L \subseteq \mathcal{V}$ of labeled nodes, with class labels from $C = \{1, 2, \ldots, K\}$, the goal is to assign each node in G to one class in C. In this work, we focus on node classification employing graph neural networks. In particular, we consider graph convolutional networks where the latent representations $\boldsymbol{H}^{(l)}$ at layer l are of the form

$$\boldsymbol{H}^{(l)} = \sigma^{(l)}\left(\hat{\boldsymbol{H}}^{(l)}\right), \text{ where} \tag{1}$$

$$\hat{\boldsymbol{H}}^{(l)} = \mathcal{T}(\boldsymbol{A})\, \boldsymbol{H}^{(l-1)} \boldsymbol{W}^{(l-1)} + \boldsymbol{b}^{(l-1)} \tag{2}$$

for $l = 2, ..., L$, where $\boldsymbol{H}^{(1)} = \boldsymbol{X}$ and activation functions given by

$$\sigma^{(L)}(\cdot) = \text{softmax}(\cdot), \qquad \sigma^{(l)}(\cdot) = \text{ReLU}(\cdot)$$

for $l = 2, ..., L-1$. Here, $\mathcal{T}(\cdot)$ is a transformation that is applied to the adjacency matrix in order to obtain the message passing matrix. This message passing matrix defines how the activations are propagated in the network. In GCN [8], for example,

$$\mathcal{T}(\boldsymbol{A}) = \boldsymbol{D}^{-\frac{1}{2}}(\boldsymbol{A} + \boldsymbol{I}_{N \times N})\boldsymbol{D}^{-\frac{1}{2}}, \text{ and } \boldsymbol{D}_{ii} = d_i = 1 + \sum\nolimits_j \boldsymbol{A}_{ij} \tag{3}$$

The $\boldsymbol{W}^{(l)}$ and $\boldsymbol{b}^{(l)}$ are the trainable weights of the graph neural network (summarized as θ), typically learned by minimizing the cross-entropy loss on the given labeled training nodes $\mathcal{V}_L$.

The output $\boldsymbol{H}^{(L)}_{vc}$ is the probability of assigning node v to class c, and we denote the logits (before applying softmax) as $\hat{\boldsymbol{H}}^{(L)}_{v} =: f^{v}_{\theta}(\boldsymbol{X}, \mathcal{T}(\boldsymbol{A}))$ indicating the dependency on $\boldsymbol{X}$, $\boldsymbol{A}$, and θ.

Notations: We denote with $\mathcal{N}_t$ the 1-hop neighborhood of a node t, i.e. all neighbors of t and the node t itself (regarding the original, unperturbed graph). Given a matrix $\boldsymbol{X}$, we denote its positive part with $[\boldsymbol{X}]_+ = \max(\boldsymbol{X}, 0)$ where the max is applied entry-wise. Similarly, the negative part is $[\boldsymbol{X}]_- = -\min(\boldsymbol{X}, 0)$, which are non-negative numbers. We denote with $h^{(l)}$ the dimensionality of the latent space in layer l, i.e. $\boldsymbol{H}^{(l)} \in \mathbb{R}^{N \times h^{(l)}}$.

3 ROBUSTNESS CERTIFICATION FOR GRAPH STRUCTURE PERTURBATIONS

3.1 Problem Definition

To derive our robustness certificates, we first set up the optimization problem we aim to solve. We assume that we are provided with an already trained GCN with weights θ as well as a (potentially corrupted) graph structure in form of an adjacency matrix $\boldsymbol{A}$. The true (unperturbed) graph structure $\boldsymbol{A}^*$ is not known, but $\boldsymbol{A}$ is assumed to be reachable from $\boldsymbol{A}^*$ via an admissible set of perturbations.

We aim to certify robustness for a single target node t at a time (called the *target* node). Such certificate guarantees that the prediction made for node t *cannot have been altered* by the attacker. In practice, predictions of nodes for which robustness cannot be certified can be ignored or sent to an expert for verification.

Formally we aim to solve:

Problem 1. *Given a graph G with adjacency matrix $\boldsymbol{A}$, a target node t, and a GCN with parameters θ. Let y^* denote the class of node t (ground truth or predicted). The worst case margin between classes y^* and y achievable under some set $\mathcal{A}(\boldsymbol{A})$ of admissible perturbations to the graph structure is given by*

$$m^t(y^*, y) := \underset{\tilde{\boldsymbol{A}}}{\text{minimize}} \; f^t_\theta(\boldsymbol{X}, \mathcal{T}(\tilde{\boldsymbol{A}}))_{y^*} - f^t_\theta(\boldsymbol{X}, \mathcal{T}(\tilde{\boldsymbol{A}}))_y \tag{4}$$
$$\text{subject to } \tilde{\boldsymbol{A}} \in \mathcal{A}(\boldsymbol{A})$$

If $m^t(y^*, y) > 0$ for all classes $y \neq y^*$, the neural network is certifiably robust w.r.t. node t and $\mathcal{A}$.

If the minimum obtained from Eq. (4) is positive, it means that there exists *no* adversarial example (within our defined admissible perturbations) that leads to the classifier changing its prediction to the other class y – i.e. the logit of class y^* is always larger than the one of y. Note that since we are interested in certifying an individual target node t's prediction, we omit t in the following when it is clear from the context.

Roadmap: Solving the above optimization problem is hard due to the three challenges mentioned in the introduction. Nevertheless, as we will show in the following, we can find *lower bounds* on the minimum of the original problem by (i) optimizing over the continuous-valued message passing matrix $\mathcal{T}(\boldsymbol{A})$ instead of the binary adjacency matrix $\boldsymbol{A}$; (ii) performing relaxations of the activation functions in the neural network; (iii) expressing the problem as a jointly-constrained bilinear program and proposing a novel branch-and-bound algorithm. Finally, if the lower bound is positive, the classifier is robust w.r.t. admissible perturbations.

3.2 Optimization over the graph structure

In this section we address **Challenge (2)** – how to efficiently optimize over the discrete graph structure – from the introduction.

First, it is important to set reasonable constraints to the perturbations the attacker can perform such that resulting certificates reflect realistic attacks. For discrete data (such as the graph structure $\boldsymbol{A}$), a natural norm to measure 'distance' is the *number* of perturbed elements, as measured by the non-convex L_0 norm. Here we translate the setup of [19], who introduce both local and global L_0 constraints on the node attributes, to graph structure perturbations.

More precisely, we allow the attacker to insert at most $\boldsymbol{q}_i$ edges to any node i, and at most $Q \in \mathbb{N}$ edges across the whole graph. Formally, any admissible perturbed adjacency matrix $\tilde{\boldsymbol{A}}$ must be in:

$$\begin{aligned}\mathcal{A}(\boldsymbol{A}) = \Big\{ \tilde{\boldsymbol{A}} \in \{0,1\}^{N\times N} \mid \tilde{\boldsymbol{A}}_{ij} \leq \boldsymbol{A}_{ij} \wedge \tilde{\boldsymbol{A}} = \tilde{\boldsymbol{A}}^T \\ \wedge \|\tilde{\boldsymbol{A}} - \boldsymbol{A}\|_0 \leq 2Q \\ \wedge \ \|\tilde{\boldsymbol{A}}_i - \boldsymbol{A}_i\|_0 \leq \boldsymbol{q}_i \ \forall 1 \leq i \leq N \Big\}.\end{aligned} \tag{5}$$

Recall that we assume an attacker has potentially *inserted* edges to the (unknown) original graph structure $\boldsymbol{A}^*$ to produce $\boldsymbol{A}$. This means that $\boldsymbol{A}^*$ must be reachable from $\boldsymbol{A}$ by *removing* edges. Hence in Eq. (5) we have the constraints $\tilde{\boldsymbol{A}}_{ij} \leq \boldsymbol{A}_{ij}$ in the definition of the admissible perturbations on $\boldsymbol{A}$. Note also that we consider only undirected graphs, hence the constraint $\tilde{\boldsymbol{A}} = \tilde{\boldsymbol{A}}^T$ and $2Q$ in the constraint, even though in principle our certification procedure also applies to directed graphs. We further assume $\boldsymbol{q}_i$ to be smaller then node i's degree, in order to prevent singleton nodes.

One traditional approach (e.g. pursued by [19]) to optimize over a discrete variable is to perform a continuous relaxation, i.e. constrain the entries to be in $\tilde{\boldsymbol{A}}_{ij} \in [0,1]$. However, recall from Eq. (3) that GCN uses a degree-normalized message passing matrix

$$\mathcal{T}(\boldsymbol{A})_{ij} = \begin{cases} \frac{1}{\sqrt{d_i d_j}} & \text{if } \boldsymbol{A}_{ij} = 1 \vee i = j \\ 0 & \text{else} \end{cases}. \tag{6}$$

The terms $\frac{1}{\sqrt{d_i d_j}} = \frac{1}{\sqrt{(1+\sum_k \boldsymbol{A}_{ik})\cdot(1+\sum_k \boldsymbol{A}_{jk})}}$ contain nonconvex quadratic terms of the form $\boldsymbol{A}_{ij}\boldsymbol{A}_{kl}$, and therefore any optimization problem involving $\tilde{\boldsymbol{A}}$ in the objective function is not convex in the variables $\tilde{\boldsymbol{A}}_{ij}$. Thus, a simple continuous relaxation does not lead to a tractable optimization problem.

As an alternative we propose to optimize over the (continuous-valued) degree-normalized message-passing matrix (i.e. the matrix after having applied $\mathcal{T}$) instead of the (binary) adjacency matrix $\tilde{\boldsymbol{A}}$. We denote the variable corresponding to the message-passing matrix by $\hat{\boldsymbol{A}}$. This has the benefits of avoiding to optimize over a discrete variable as well as bypassing the nonconvex degree-normalization procedure of GCN. That is, we replace Eq. 4 by

$$\hat{m}^t(y^*, y) := \underset{\hat{\boldsymbol{A}}}{\text{minimize}} \ f_\theta^t(\boldsymbol{X}, \hat{\boldsymbol{A}})_{y^*} - f_\theta^t(\boldsymbol{X}, \hat{\boldsymbol{A}})_y \tag{7}$$
$$\text{subject to } \hat{\boldsymbol{A}} \in \hat{\mathcal{A}}(\boldsymbol{A}).$$

Note that the neural network f_θ^t now directly takes as input the degree-normalized message-passing matrix instead of the $\mathcal{T}(\boldsymbol{A})$, i.e. we have absorbed the degree-normalization procedure into the optimization problem. However, we now have to carefully design the set of admissible degree-normalized message passing matrices $\hat{\mathcal{A}}(\boldsymbol{A})$ based on the perturbation budget and the graph at hand.

It is crucial that $\hat{\mathcal{A}}(\boldsymbol{A}) \supseteq \mathcal{T}(\mathcal{A}(\boldsymbol{A}))$. Here, $\mathcal{T}(\mathcal{A}(\boldsymbol{A}))$ denotes the set of all degree-normalized message-passing matrices that are produced from adjacency matrices in $\mathcal{A}(\boldsymbol{A})$. In other words, *any* degree-normalized matrix that could be produced by first performing discrete perturbations to $\boldsymbol{A}$ and then degree-normalizing the resulting binary matrix $\tilde{\boldsymbol{A}}$ must be included in this new set of admissible matrices. In this case it follows that $\hat{m}^t(y^*, y) \leq m^t(y^*, y)$, i.e. the optimum of our modified optimization problem in Eq. (7) is a lower bound on the original problem's optimal value. Thus if Eq. (7) leads to a positive value, we have certificate.

Clearly, instantiating $\hat{\mathcal{A}}(\boldsymbol{A})$ with all matrices $\hat{\boldsymbol{A}} \in [0,1]^{N\times N}$ fulfills this property; however, it leads to a very loose approximation. Thus, we would also like $\hat{\mathcal{A}} \setminus \mathcal{T}(\mathcal{A})$ to be as small as possible, meaning that any relaxations leading to $\hat{\mathcal{A}}$ are as tight as possible. In the following we derive *induced constraints* that give rise to a valid and tight set of admissible message passing matrices $\hat{\mathcal{A}}$ and are convex, thus enabling tractable optimization.

3.2.1 Induced constraints on $\hat{\mathcal{A}}$. In this section we construct a set of constraints on the message-passing matrix $\hat{\boldsymbol{A}}$ that are induced by the local and global budgets on the change on the adjacency matrix (see Eq. (5)) as well as the degree-normalization procedure of obtaining the message-passing matrix in GCN (Eq. (3)). These constraints reflect necessary conditions every matrix from $\mathcal{T}(\mathcal{A})$ fulfills; thus our search space over $\hat{\boldsymbol{A}}$ can be restricted to a smaller domain. To enable efficient optimization, we further require these constraints to be linear (resp., can be reformulated to a linear constraint) as well as are efficient to compute.

In the following we derive the constraints in detail since they are one core contribution of our work. The eager reader might directly jump to Equation (11) for the final result.

For convenience we denote with r_i the number of edges which have been removed from the node i for an arbitrary perturbed matrix $\tilde{\boldsymbol{A}} \in \mathcal{A}$. Note that $0 \leq r_i \leq \boldsymbol{q}_i$. We further introduce $\mathcal{D}_i \subset \mathcal{N}_i$ as the set of edges being removed from node i (recall that $\mathcal{N}_i$ are the neighbors in the original graph including i).

Element-wise bounds. We can bound every element of $\hat{\boldsymbol{A}}$ individually by $\boldsymbol{L}_{ij} \leq \hat{\boldsymbol{A}}_{ij} \leq \boldsymbol{U}_{ij}$. For the lower bound we have (i) $\boldsymbol{L}_{ii} = \mathcal{T}(\boldsymbol{A})_{ii}$, since the self loops in the preprocessing procedure cannot be removed and $\frac{1}{d_i - r_i} \geq \frac{1}{d_i}$ (ii) everywhere else we have $\boldsymbol{L}_{ij} = 0$, representing a potential deletion of the edge.

The upper bound is $\boldsymbol{U}_{ij} = \min\{\boldsymbol{A}_{ij}, \big((d_i - \boldsymbol{q}_i)(d_j - \boldsymbol{q}_j)\big)^{-\frac{1}{2}}\}$ for $i \neq j$. The first term within the min ensures that no edges are added in the certification procedure. This technically means that, in the following, we *only* need to consider entries (i, j) for which there exists an edge in $\boldsymbol{A}$, leading to a sparse optimization problem. For the second term, note that the degrees of all nodes can never increase and since they appear in the denominator (see Eq. (6)), $\big((d_i - \boldsymbol{q}_i)(d_j - \boldsymbol{q}_j)\big)^{-\frac{1}{2}}$ is an upper bound on the individual entries $\mathcal{T}(\boldsymbol{A})_{ij}$. For $\boldsymbol{U}_{ii}$ we use only the second term of the min.

Please note that from the above discussion we can also conclude that $\hat{\boldsymbol{A}}_{ij} = 0$ if the edge between i and j is deleted and $\hat{\boldsymbol{A}}_{ij} \geq \mathcal{T}(\boldsymbol{A})_{ij}$ else. While we cannot use this insight immediately (since it forms a non-convex set), we will show in Section 3.4.3 how to use it.

Row-wise bounds (I). The element-wise bounds do not take dependencies between elements into account. Thus, it would, e.g., be possible to set for a single node i all non-diagonal $\hat{\boldsymbol{A}}_{ij}$ to 0;

likely violating the local and global budget. To prevent this, we now introduce constraints on specific row sums of $\hat{\boldsymbol{A}}$.

For this we first rephrase the row sum as

$$\sum_j \hat{\boldsymbol{A}}_{ij} = \frac{1}{\sqrt{d_i - r_i}} \sum_{\substack{j \neq i \\ j \in \mathcal{N}_i \setminus \mathcal{D}_i}} \frac{1}{\sqrt{d_j - r_j}} + \frac{1}{d_i - r_i} \tag{8}$$

When removing an edge from node i (i.e. increasing r_i) we can observe two opposing effects: the term in front of the sum increases; while the sum itself contains fewer summands and, thus, decreases (assuming the other r_j fixed). Thus, our general idea is to upper/lower bound the value the sum can take for a fixed r_i. Plugging this bound on the sum in Eq. (8) leads to an upper/lower bound for a fixed r_i, denoted by $U_i^{\text{row}}(r_i)$ and $L_i^{\text{row}}(r_i)$, respectively. Now, by simply evaluating $U_i^{\text{row}}(r_i)$ and $L_i^{\text{row}}(r_i)$ for $0 \leq r_i \leq \boldsymbol{q}_i$ we can find the overall row-bound in an efficient way (e.g. for the lower bound $L_i^{\text{row}} = \min_{r_i} L_i^{\text{row}}(r_i)$).

(1) We start with the lower bound $L_i^{\text{row}} \leq \sum_j \hat{\boldsymbol{A}}_{ij}$ since the solution is easy to see. For a fixed r_i, the smallest possible sum is achieved by keeping those nodes which have the largest degree. Clearly, we set $r_j = 0$ since this leads to the smallest value. In short: To obtain the lower bound on the sum, we simply sort the nodes based on their degree in descending order and sum up the first $d_i - 1 - r_i$ terms $1/\sqrt{d_j}$ (-1 since the self-loops is counted in d_i).

(2) For the upper bound $\sum_j \hat{\boldsymbol{A}}_{ij} \leq U_i^{\text{row}}$, we need to find an upper bound for the sum. Clearly, in such a scenario $r_j > 0$ since then the fractions get larger. However, we cannot simply set every $r_j = q_j$ since this would violate the overall budget constraint. The question is now twofold: how to select the terms/nodes to remove from the sum, and how to spend the remaining budget to increase the remaining terms? Clearly, we select the smallest r_i summands (corresponding to the neighbors with largest degree) to drop from the sum. For the remaining nodes $\mathcal{M}$, we have to solve the following optimization problem:

$$\max_{r_j, j \in \mathcal{M}} \sum_{j \in \mathcal{M}} \frac{1}{\sqrt{d_j - r_j}} \qquad \text{s.t.} \sum_{j \in \mathcal{M}} r_j \leq B \tag{9}$$

where B is the remaining budget one can spend. Note that we cannot assume that the remaining budget is simply $Q - r_i$. The reason is that a single edge removal (i.e. one 'unit' from Q) can lead to *two* neighbors' degrees reducing by one, if these neighbors are connected by this edge. Thus, in the worst case, if all neighbors are connected with each other, we can effectively spend twice the remaining removal budget to increase the r_j. Therefore we denote the number of neighbors of i that are connected to each other as $\mathcal{N}_i^{\text{conn}}$, from which $r^{\text{nbs}} = \min\{Q - r_i, |\mathcal{N}_i^{\text{conn}}|\}$ can become disconnected using the remaining budget, leading to a total remaining budget to increase r_j of $B = 2 \cdot r^{\text{nbs}} + (Q - r_i - r^{\text{nbs}})$.

As it turns out, we can solve the optimization problem (9) exactly using a greedy approach: We maximize the term by sorting the neighbors $\mathcal{N}_i$ by their degree in ascending order, and, starting with the lowest degree, use all available budget $\boldsymbol{q}_j$ to reduce the neighbors' degrees, until the budget B is depleted. This is a valid upper bound on the sum because $h(x) = \frac{1}{\sqrt{x}}, x > 0,\ x \in \mathbb{N}$ is a monotonically decreasing function whose second derivative is positive everywhere, i.e. $\frac{1}{\sqrt{u}} - \frac{1}{\sqrt{u-1}} \geq \frac{1}{\sqrt{u+1}} - \frac{1}{\sqrt{u}}$ for $u \in \mathbb{N} > 1$. Put simply: the larger the term $\frac{1}{\sqrt{d_j - r_j}}$ already is, the more effect has increasing r_j on increasing it even further. Thus, the strategy outlined above is guaranteed to maximize (9).

Complexity analysis: Both bounds require sorting the neighbors based on their degree, which can be done once. We have to compute $\boldsymbol{q}_i$ many terms $U_i^{\text{row}}(r_i)$ and $L_i^{\text{row}}(r_i)$. Each term, however, can be computed incrementally based on the previous r_i (since one only iterates over more elements in the sorted list). Thus, to compute all terms, we have to iterate through all neighbors at most once. Thus the overall complexity is $O(|\mathcal{N}_i| \log |\mathcal{N}_i| + |\mathcal{N}_i|) = O(d_i \log d_i)$

Row-wise bounds (II). (1) So far we have defined lower and upper bounds on the row sum $\sum_j \hat{\boldsymbol{A}}_{ij}$ of node i. We additionally define constraints on the L_1 norm of the difference between the perturbed message-passing matrix $\hat{\boldsymbol{A}}$ and the original message-passing matrix $\mathcal{T}(\boldsymbol{A})$: $\sum_j |\hat{\boldsymbol{A}}_{ij} - \mathcal{T}(\boldsymbol{A})_{ij}| \leq \bar{U}_i^{\text{row}}$. This covers the case where large positive and large negative changes mostly cancel each other out, such that the lower and upper bounds on the row sum are not violated even though the perturbed message-passing matrix is substantially different.

Again, we denote the set of nodes whose edges to i are removed by $\mathcal{D}_i$. For any term $j \in \mathcal{D}_i$ the absolute change is $|\hat{\boldsymbol{A}}_{ij} - \mathcal{T}(\boldsymbol{A})_{ij}| = \mathcal{T}(\boldsymbol{A})_{ij}$ since it corresponds to a removed edge. For any node $j \in \mathcal{N}_i \setminus \mathcal{D}_i$ (which also includes the node i itself) the absolute change is at most $\delta_{ij} = \boldsymbol{U}_{ij} - \mathcal{T}(\boldsymbol{A})_{ij}$. Assuming that all terms not removed take their maximum value, we aim to solve the following optimization problem:

$$\begin{aligned} \bar{U}_i^{\text{row}} &= \max_{\mathcal{D}_i} \sum_{j \in \mathcal{D}_i} \mathcal{T}(\boldsymbol{A})_{ij} + \sum_{j \in \mathcal{N}_i \setminus \mathcal{D}_i} \delta_{ij} \qquad (10) \\ &= \sum_{j \in \mathcal{N}_i} \delta_{ij} + \max_{\mathcal{D}_i} \sum_{j \in \mathcal{D}_i} \mathcal{T}(\boldsymbol{A})_{ij} - \sum_{j \in \mathcal{D}_i} \delta_{ij} \\ &= \sum_{j \in \mathcal{N}_i} \delta_{ij} + \max_{\mathcal{D}_i} \sum_{j \in \mathcal{D}_i} 2 \cdot \mathcal{T}(\boldsymbol{A})_{ij} - \boldsymbol{U}_{ij} \\ &\qquad s.t.\ |\mathcal{D}_i| \leq \boldsymbol{q}_i \end{aligned}$$

Hence we sort all neighbors of node i by their value $2 \cdot \mathcal{T}(\boldsymbol{A})_{ij} - \boldsymbol{U}_{ij}$ and choose the largest $\boldsymbol{q}_i$ values to obtain $\bar{U}^{\text{row}}$.

(2) Another measure to avoid cancelling positive and negative changes to $\mathcal{T}(\boldsymbol{A})$ is to constrain the total change in the negative direction. That is, we consider only the change regarding edge removal – effectively discarding from Eq. (10) the change in positive direction (i.e. the δ_{ij} terms):

$$\sum_{j \neq i} \left[\hat{\boldsymbol{A}}_{ij} - \mathcal{T}(\boldsymbol{A})_{ij}\right]_- \leq \Delta_i^- = \max_{\mathcal{D}_i} \sum_{j \in \mathcal{D}_i} \mathcal{T}(\boldsymbol{A})_{ij} \quad s.t.\ |\mathcal{D}_i| \leq \boldsymbol{q}_i$$

The solution $\mathcal{D}_i$ is simply the set of the $\boldsymbol{q}_i$ neighbors with lowest degree, which are allowed to be disconnected from i (i.e., doing so does not lead to singleton nodes).

Complexity analysis: Like before, the dominating complexity is sorting the neighbors, leading to $O(d_i \log d_i)$.

Global bounds. Besides entry-wise and row-wise constraints we can also make use of global constraints on $\hat{\boldsymbol{A}}$.

(1) An upper bound $\bar{U}_{\text{glob}}$ on the L_1 difference of $\hat{\boldsymbol{A}}$ and $\mathcal{T}(\boldsymbol{A})$ can be obtained in a similar fashion as the row-wise bound (Eq. 10).

Formally we want to solve

$$\begin{aligned}\bar{U}_{\text{glob}} &= \max_{\mathcal{D}} \sum_{\substack{i<j \\ (i,j)\in\mathcal{D}}} \mathcal{T}(\boldsymbol{A})_{ij} + \sum_{\substack{i\leq j \\ (i,j)\notin\mathcal{D}}} \delta_{ij} \\ &= \sum_{i\leq j} \delta_{ij} + \max_{\mathcal{D}} \sum_{\substack{i<j \\ (i,j)\in\mathcal{D}}} 2\cdot\mathcal{T}(\boldsymbol{A})_{ij} - \boldsymbol{U}_{ij} \\ &\text{s.t. } |\mathcal{D}| \leq Q\end{aligned}$$

where $\mathcal{D}$ represents the set of removed edges (this set does not include the self-loops since they cannot be removed). As before, the optimal solution is obtained by selecting the Q entries (i,j) with largest values $2\cdot\mathcal{T}(\boldsymbol{A})_{ij} - \boldsymbol{U}_{ij}$ (excl. diagonal terms).

(2) We can also compute an upper bound Δ^-_{glob} on the negative change analogously to Δ^-_i. That is,

$$\sum_{i<j} \left[\hat{\boldsymbol{A}}_{ij} - \mathcal{T}(\boldsymbol{A})_{ij}\right]_- \leq \Delta^-_{\text{glob}} = \max_{\mathcal{D}} \sum_{\substack{i<j \\ (i,j)\in\mathcal{D}}} \mathcal{T}(\boldsymbol{A})_{ij}, \quad s.t.\ |\mathcal{D}| \leq Q$$

which is maximized by setting $\mathcal{D}$ to the Q largest values $\mathcal{T}(\boldsymbol{A})_{ij}$, $i<j$.

Complexity analysis: The bounds can again be computed based on sorting; now based on the full edge set. This leads to $O(E\log E)$, where E is the number of edges in $\boldsymbol{A}$.

Summary. The intersection of the (linear) *induced constraints* defined above describes a convex set $\hat{\mathcal{A}}(\boldsymbol{A})$ of admissible perturbed message-passing matrices $\hat{\boldsymbol{A}}$.

$$\begin{aligned}\hat{\mathcal{A}}(\boldsymbol{A}) = \Big\{ &\hat{\boldsymbol{A}} \in [0,1]^{N\times N} \mid \hat{\boldsymbol{A}} = \hat{\boldsymbol{A}}^T \wedge\ \forall i,j : \boldsymbol{L}_{ij} \leq \hat{\boldsymbol{A}}_{ij} \leq \boldsymbol{U}_{ij} \\ &\wedge \forall i : L_i^{\text{row}} \leq \sum_j \hat{\boldsymbol{A}}_{ij} \leq U_i^{\text{row}} \ \wedge \forall i : \sum_j |\hat{\boldsymbol{A}}_{ij} - \mathcal{T}(\boldsymbol{A})_{ij}| \leq \bar{U}_i^{\text{row}} \\ &\wedge \forall i : \sum_{j\neq i} \left[\hat{\boldsymbol{A}}_{ij} - \mathcal{T}(\boldsymbol{A})_{ij}\right]_- \leq \Delta^-_i \\ &\wedge \sum_{i\leq j} |\hat{\boldsymbol{A}}_{ij} - \mathcal{T}(\boldsymbol{A})_{ij}| \leq \bar{U}_{\text{glob}} \ \sum_{i<j} \left[\hat{\boldsymbol{A}}_{ij} - \mathcal{T}(\boldsymbol{A})_{ij}\right]_- \leq \Delta^-_{\text{glob}} \Big\}\end{aligned} \tag{11}$$

By construction of the set $\hat{\mathcal{A}}(\boldsymbol{A})$ it holds:

Theorem 3.1. *$\mathcal{T}(\mathcal{A}(\boldsymbol{A})) \subseteq \hat{\mathcal{A}}(\boldsymbol{A})$, i.e. for $\tilde{\boldsymbol{A}} \in \mathcal{A}(\boldsymbol{A})$ we have $\mathcal{T}(\tilde{\boldsymbol{A}}) \in \hat{\mathcal{A}}(\boldsymbol{A})$.*

From Theorem 3.1 it follows that the optimal solution of Eq. (7) leads to a valid lower bound on the original problem in Eq. (4).

3.3 Relaxation of the neural network

The above constraints can be computed efficiently and form a convex set, thus the constraint in Equation (7) can be efficiently handled. Still, however, the objective function, which is based on the original neural network f_θ, is challenging due to the nonlinearities (see **Challenge (1)** mentioned in the introduction). Here we follow the relaxation approach described in [13, 19]. Under this relaxation, the output $\boldsymbol{H}$ of the ReLU activation function in Eq. (1) is no longer deterministic but instead treated as a variable (like $\hat{\boldsymbol{A}}$) with the following constraints:

$$\begin{aligned}&\boldsymbol{H}^{(l)}_{nj} \geq 0, \qquad \boldsymbol{H}^{(l)}_{nj} \geq \hat{\boldsymbol{H}}^{(l)}_{nj} \\ &\boldsymbol{H}^{(l)}_{nj}\left(\boldsymbol{S}^{(l)}_{nj} - \boldsymbol{R}^{(l)}_{nj}\right) \leq \boldsymbol{S}^{(l)}_{nj}\left(\hat{\boldsymbol{H}}^{(l)}_{nj} - \boldsymbol{R}^{(l)}_{nj}\right) \quad \text{if } (n,j)\in\mathcal{I}^{(l)}\end{aligned} \tag{12}$$

Here, $\boldsymbol{S}^{(l)}_{nj}$ and $\boldsymbol{R}^{(l)}_{nj}$ are upper and lower bounds on the pre-ReLU activation $\hat{\boldsymbol{H}}^{(l)}_{nj}$ (which we describe shortly). $\mathcal{I}^{(l)}$ is the set of tuples (n,j) for which $\boldsymbol{S}^{(l)}_{nj}$ and $\boldsymbol{R}^{(l)}_{nj}$ have different signs. Analogously, $\mathcal{I}^{(l)}_+$ and $\mathcal{I}^{(l)}_-$ respectively consist of tuples (n,j) for which the lower and upper bounds are both positive / negative. For $\mathcal{I}^{(l)}_+$ and $\mathcal{I}^{(l)}_-$ we have the following constraints on $\boldsymbol{H}^{(l)}_{nj}$:

$$\boldsymbol{H}^{(l)}_{nj} = \hat{\boldsymbol{H}}^{(l)}_{nj} \ \text{ if } (n,j)\in\mathcal{I}^{(l)}_+ \qquad \boldsymbol{H}^{(l)}_{nj} = 0 \ \text{ if } (n,j)\in\mathcal{I}^{(l)}_- \tag{13}$$

We denote the set of hidden activations compliant with the above constraints as $\mathcal{Z}(\boldsymbol{A})$.

Computation of $\boldsymbol{S}^{(l)}_{nj}$ and $\boldsymbol{R}^{(l)}_{nj}$. For the convex relaxation described above we need valid lower and upper bounds on the pre-ReLU activations in the graph neural network. In contrast to [19], who make heavy use of the given (static) graph structure, in our case the graph structure itself changes. Instead we use the following upper and lower bounds (inspired by [11] for standard neural networks):

$$\begin{aligned}\boldsymbol{S}^{(l)} &= \boldsymbol{U}\left[\boldsymbol{S}^{(l-1)}\boldsymbol{W}^{(l-1)}\right]_+ - \boldsymbol{L}\left[\boldsymbol{R}^{(l-1)}\boldsymbol{W}^{(l-1)}\right]_- + \boldsymbol{b}^{(l-1)} \\ \boldsymbol{R}^{(l)} &= \boldsymbol{L}\left[\boldsymbol{R}^{(l-1)}\boldsymbol{W}^{(l-1)}\right]_+ - \boldsymbol{U}\left[\boldsymbol{S}^{(l-1)}\boldsymbol{W}^{(l-1)}\right]_- + \boldsymbol{b}^{(l-1)} \\ &\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \text{for } 2\leq l\leq L\end{aligned}$$

where $\boldsymbol{S}^{(1)} = \boldsymbol{R}^{(1)} = \boldsymbol{X}$ and $\boldsymbol{U}$, $\boldsymbol{L}$ are the element-wise lower and upper bounds on the message passing matrix (see Sec. 3.2.1).

3.4 Jointly Constrained Bilinear Program

Using the relaxation from Section 3.3 and the constraints on the message passing matrix from Section 3.2 we can rephrase the objective function in Eq. 7 as

$$\hat{\boldsymbol{H}}^{(L)}_{ty^*} - \hat{\boldsymbol{H}}^{(L)}_{ty} = \hat{\boldsymbol{H}}^{(L)}_{t}\boldsymbol{c} = \hat{\boldsymbol{A}}_t\boldsymbol{H}^{(L-1)}\boldsymbol{W}^{(L-1)}\boldsymbol{c} + \boldsymbol{b}^{(L-1)^\top}\boldsymbol{c}$$

where $\boldsymbol{c} = \boldsymbol{e}_{y^*} - \boldsymbol{e}_y$ is a K-dimensional constant vector with value 1 at y^*, -1 at y, and 0 else. Since $\boldsymbol{b}^{(L-1)^\top}\boldsymbol{c}$ is constant this leads to the overall problem:

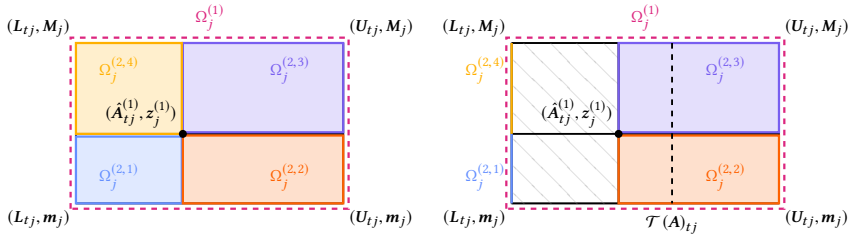
$$\begin{aligned}\tilde{m}^t(y^*, y) := \underset{\hat{\boldsymbol{A}},\boldsymbol{H}^{(\cdot)}}{\text{minimize}}\ & \hat{\boldsymbol{A}}_t\boldsymbol{H}^{(L-1)}\boldsymbol{W}^{(L-1)}\boldsymbol{c} \\ \text{subject to } & \boldsymbol{H}^{(\cdot)} \in \mathcal{Z}(\boldsymbol{A}),\ \hat{\boldsymbol{A}} \in \hat{\mathcal{A}}(\boldsymbol{A})\end{aligned}$$

Note that in the objective function we have the term $\hat{\boldsymbol{A}}_t\boldsymbol{H}^{(L-1)}$ and thus we have a bilinear objective function.

3.4.1 Canonical form. We will bring the above problem in a canonical form to simplify further analysis. From now on we consider the special case of a GCN with two message passing steps (i.e. $L = 3$), and subsequently show how to generalize to $L > 3$. First, as a shorthand, we define the vector $\boldsymbol{w} = \boldsymbol{W}^{(2)}\boldsymbol{c} \in \mathbb{R}^{h^{(2)}}$ and the matrix

$$\boldsymbol{P} \in \mathbb{R}^{|\mathcal{N}_t|\times|\mathcal{N}_t|\cdot h^{(2)}} = \begin{bmatrix} \boldsymbol{w}^T & \boldsymbol{0}^T & \cdots & \boldsymbol{0}^T \\ \boldsymbol{0}^T & \boldsymbol{w}^T & \cdots & \boldsymbol{0}^T \\ \vdots & \vdots & \ddots & \vdots \\ \boldsymbol{0}^T & \cdots & \cdots & \boldsymbol{w}^T \end{bmatrix},$$

where $\boldsymbol{0}$ is a $h^{(2)}$-dimensional vector of zeros. Next, we denote with $\boldsymbol{H}^{(2)}_{\text{vec}} \in \mathbb{R}^{|\mathcal{N}_t|\cdot h^{(2)}}$ the result of flattening the matrix $\boldsymbol{H}^{(2)}$ into a



(a) Standard branching as in [1]. (b) Improved branching for graph certificates.

Figure 1: Overview of the branching step in the first iteration on dimension j

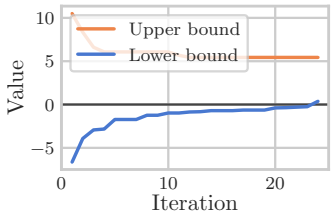


Figure 2: Converging lower and upper bounds; node from the CORA-ML dataset.

vector.[1] Finally, we introduce another variable $z \in \mathbb{R}^{|\mathcal{N}_t|} = \boldsymbol{P}\boldsymbol{H}_{\text{vec}}^{(2)}$. With this, the above problem can equivalently be written as

$$\begin{aligned} &\underset{\hat{\boldsymbol{A}}, \boldsymbol{z}, \boldsymbol{H}^{(2)}}{\text{minimize}}\ \hat{\boldsymbol{A}}_t \boldsymbol{z} \\ &\text{subject to } \boldsymbol{H}^{(2)} \in \mathcal{Z}(\boldsymbol{A}),\ \hat{\boldsymbol{A}} \in \hat{\mathcal{A}}(\boldsymbol{A}),\ \boldsymbol{z} = \boldsymbol{P}\boldsymbol{H}_{\text{vec}}^{(2)} \end{aligned} \tag{14}$$

Note that all constraints are linear since the constraints on $\boldsymbol{H}^{(2)}$ (see Eqs. (12, 13)) relate to the value of $\hat{\boldsymbol{H}}^{(2)} = \hat{\boldsymbol{A}}\boldsymbol{X}\boldsymbol{W}^{(1)} + \boldsymbol{b}^{(1)}$ (see Eq. (2), recall that $\boldsymbol{H}^{(1)} = \boldsymbol{X}$), which is linear in $\hat{\boldsymbol{A}}$. Furthermore, we can easily add the following constraint to the problem without changing its solution: $(\hat{\boldsymbol{A}}_t, \boldsymbol{z}) \in \Omega$ where

$$\Omega = \left\{(\hat{\boldsymbol{A}}_t, \boldsymbol{z}) : \boldsymbol{L}_{tj} \le \hat{\boldsymbol{A}}_{tj} \le \boldsymbol{U}_{tj}, \boldsymbol{m}_j \le \boldsymbol{z}_j \le \boldsymbol{M}_j; 1 \le j \le |\mathcal{N}_t|\right\} \tag{15}$$

forms a simple rectangular constraint on the variables (see Figure 1a). Here, $\boldsymbol{L}$ and $\boldsymbol{U}$ are the familiar entry-wise box constraints on $\hat{\boldsymbol{A}}$, and $\boldsymbol{m}$ and $\boldsymbol{M}$ are bounds on $\boldsymbol{z}$ which, e.g., can be obtained via standard interval arithmetic from the bounds on $\boldsymbol{H}^{(2)}$:

$$\boldsymbol{m} = \left[\boldsymbol{R}^{(2)}\right]_+ [\boldsymbol{w}]_+ - \left[\boldsymbol{S}^{(2)}\right]_+ [\boldsymbol{w}]_-, \quad \boldsymbol{M} = \left[\boldsymbol{S}^{(2)}\right]_+ [\boldsymbol{w}]_+ - \left[\boldsymbol{R}^{(2)}\right]_+ [\boldsymbol{w}]_-$$

So far, it seems that this new constraint adds no benefit – but it indeed now opens the door for solving the problem: Using the above reformulation, we can apply the principle proposed in [1] for solving so-called jointly constrained bilinear optimization problems.

3.4.2 Branch-and-bound algorithm. In the following we give an overview of the branch-and-bound algorithm we employ to solve the bilinear program in Eq. (14). We refer the interested reader to [1] for more details on the procedure and convergence proofs.

The idea is twofold: (1) We use the convex envelope[2] of the objective function to compute *lower bounds* on a rectangular domain over $\hat{\boldsymbol{A}}_i$ and $\boldsymbol{z}$. That is, instead of minimizing the objective in Eq. (14) we are minimizing

$$\sum_{j=1}^{|\mathcal{N}_t|} \max\{\boldsymbol{m}_j \hat{\boldsymbol{A}}_{tj} + \boldsymbol{L}_{tj}\boldsymbol{z}_j - \boldsymbol{L}_{tj}\boldsymbol{m}_j, \boldsymbol{M}_j \hat{\boldsymbol{A}}_{tj} + \boldsymbol{U}_{tj}\boldsymbol{z}_j - \boldsymbol{U}_{tj}\boldsymbol{M}_j\},$$

When using this objective function in Eq. (14), we (i) obtain a convex problem, more precisely even an LP, which is extremely efficient to solve; and (ii) we obtain a *lower bound* v on the global optimum v^* of the original problem on the hyperrectangle Ω.

(2) This brings us to the second idea: We use a branch-and-bound procedure to recursively subdivide the rectangular domain. (a) We initialize the branch-and-bound procedure (i.e. $k = 1$) with the initial hyperrectangle $\Omega = \Omega^{(1)}$ and solve the problem using an off-the-shelf LP solver to obtain a lower bound $v^{(1)}$ (along with the optimal solution $(\hat{\boldsymbol{A}}_t^{(1)}, \boldsymbol{z}^{(1)})$). The tuple $(\Omega^{(1)}, v^{(1)})$ is added to the list $\mathcal{L}$ of (yet to be branched) rectangles.

(b) In the branching step, i.e. in iteration k, we select the tuple from $\mathcal{L}$ with the smallest lower bound for branching (recall that our goal is to solve a minimization problem). The branching is illustrated in Figure 1. We choose a dimension j to split (see Appendix A.1 on how we choose j) and divide 'around' the optimal solution that was obtained on the overall rectangle. Figure 1 illustrates how we divide $\Omega_j^{(1)}$ into the subregions $\Omega_j^{(2,1)}, \Omega_j^{(2,2)}, \Omega_j^{(2,3)}$, and $\Omega_j^{(2,4)}$ where the optimal solution $(\hat{\boldsymbol{A}}_{tj}^{(1)}, \boldsymbol{z}_j^{(1)})$ is indicated by the black dot. We solve these 4 problems, obtain 4 new tuples, and add them to $\mathcal{L}$. Since dividing the domain Ω into smaller sub-rectangles leads to tighter convex envelopes, we have $v^{(k+1)} \ge v^{(k)}$ (see Fig. 2), so that for $k \to \infty$ we recover the global solution. While (theoretically) this might require infinitely many iterations, in practice this is not a concern as we will discuss next along with further improvement.

Moreover note that with the above procedure we also easily get upper bounds as a by-product. Every time we solve an LP we use its optimal solution $(\hat{\boldsymbol{A}}_t^{(k)}, \boldsymbol{z}^{(k)})$ and plug it into the original Eq. (14). This leads to an upper bound. The upper bound $V^{(k)}$ in iteration k is then the minimum of all upper bounds obtained so far.

3.4.3 Improvements for Graph Certificates. It is crucial to observe that we are actually not interested in finding the global solution of Eq. (14), but we are only interested in its sign (positive or negative). Since in each iteration of the above approach we obtain increasingly accurate upper and lower bounds on the global solution, we can perform an early stopping. We can stop (i) if the lower bound is positive, $v^{(k)} > 0$. In this case we have successfully certified robustness. We can also stop (ii) if the upper bound is negative, $V^{(k)} < 0$. In this case we are certain that we cannot certify robustness for this instance. Note that (ii) *does not* imply that the classifier is not robust; it only means that no robustness guarantee can be made. In Fig. 2 we show a real-world example of this early stopping.

[1] We slightly abuse the notation here. $\boldsymbol{H}$ actually has a shape of $\mathbb{R}^{N \times h^{(2)}}$. For the objective function, however, we only need the elements from N_t, i.e. we can slice $\boldsymbol{H}$ to a smaller matrix. Equivalently we can slice $\hat{\boldsymbol{A}}_t$ to have shape $\mathbb{R}^{|\mathcal{N}_t|}$. This step also leads to more efficient computation. Indeed, we can start our overall procedure by slicing $\boldsymbol{A}$ into shape $\mathcal{N}_t^{(2)} \times \mathcal{N}_t^{(2)}$, where $\mathcal{N}_t^{(2)}$ are the two-hop neighbors of t, since t's prediction does not depend on any other nodes for $L = 3$.

[2] The convex envelope of a function f on a domain Ω is the pointwise supremum of all convex functions that underestimate f over Ω [1]. It can be efficiently computed for the scalar product of two d-dimensional vectors as in our objective function.

Dataset	Q	q	% Cert. Robust	% Cert. Nonrobust	% Certifiable
Citeseer	1	1	88.4	7.2	95.6
	5	3	78.8	11.8	90.6
	10	5	73.0	12.8	85.8
Cora-ML	1	1	80.8	11.6	92.4
	5	3	56.2	21.2	77.4
	10	5	43.8	29.0	72.8
PubMed	1	1	78.8	9.2	88.0
	5	3	58.2	14.8	73.0
	10	5	49.0	18.4	67.4

(a) Certification results for standard GCN.

Dataset	Q	q	% Cert. Robust	% Cert. Nonrobust	% Certifiable
Citeseer	1	1	96.2	2.6	98.8
	5	3	83.4	7.4	90.8
	10	5	76.4	10.4	86.8
Cora-ML	1	1	87.8	5.6	93.4
	5	3	57.2	15.2	72.4
	10	5	47.0	22.6	69.6
PubMed	1	1	89.4	4.8	94.2
	5	3	72.8	10.6	83.4
	10	5	63.2	14.4	77.6

(b) Results for GCN with robust training proposed in [19].

Table 1: Robustness certification results. Our method can certify a large percentage of the nodes.

We further improve the procedure by exploiting knowledge about the graph domain, which we visualize in Fig. 1b. More precisely, when we branch on $\hat{\boldsymbol{A}}_{tj}$ and we find that the parent problem's optimal value for $\hat{\boldsymbol{A}}_{tj}$ is *smaller* than the original value $\mathcal{T}(\boldsymbol{A})_{tj}$, we set both the upper and lower bounds on $\hat{\boldsymbol{A}}_{tj}$ to zero for the sub-rectangles $\Omega_j^{(k+1,1)}$ and $\Omega_j^{(k+1,4)}$. This reflects the fact that we can only decrease values in $\mathcal{T}(\boldsymbol{A})$ by *removing* the corresponding edge (see element-wise bounds in Sec. 3.2.1). While we cannot encode this efficiently into the original bilinear problem, we can exploit this fact during the branch step. Note that the two sub-rectangles collapse into *lines* on the border of $\Omega_j^{(k)}$. Since the convex envelope is exact on the border of the rectangle, this not only leads to faster optimization but also tighter lower bounds on the original problem's global optimum.

3.4.4 Generalizing to $L > 3$. Our robustness certificates can be generalized to arbitrary number of message-passing steps. To this end we first observe that for $2 < l < L$ the constraints in the ReLU convex relaxation are no longer linear. For instance in the constraint

$$\boldsymbol{H}^{(l)} \geq \hat{\boldsymbol{H}}^{(l)} = \hat{\boldsymbol{A}}\boldsymbol{H}^{(l-1)}\boldsymbol{W}^{(l-1)} + \boldsymbol{b}^{(l-1)}$$

we have the bilinear term $\hat{\boldsymbol{A}}\boldsymbol{H}^{(l-1)}$, which is not convex. One solution is to use the reformulation-linearization technique Qualizza et al. [10] to define a new variable $\tilde{\boldsymbol{H}}^{(l)}$ and use the entry-wise lower and upper bounds on $\hat{\boldsymbol{A}}$ and $\boldsymbol{H}^{(l-1)}$ to derive upper and lower bounds on $\tilde{\boldsymbol{H}}^{(l)}$ using interval arithmetic. This leads to the constraints being linear again and we can apply our improved branch and bound procedure from above.

3.4.5 Alternative solution approaches. A bilinear program (BLP) is a special case of a quadratic program (QP). However, expressing a BLP as a QP leads to a matrix $\boldsymbol{Q}$ with zero diagonal, hence cannot be positive semidefinite and therefore the QP is not convex. Semidefinite relaxations such as in [11], where $\boldsymbol{Q}$ is replaced by a psd matrix, do not apply either due to the zero diagonal on $\boldsymbol{Q}$.

Another relaxation approach for nonconvex QPs, resp. BLPs, is the reformulation-linearization technique (RLT) [10]. Since RLT performs a relaxation on the original BLP, the result serves as a lower bound on the BLP's optimal value. We derived the RLT for our problem and compare with this alternative in our experiments.

Summary. We have rephrased our problem of certifying robustness of GCN under perturbation of the graph structure as a jointly constrained bilinear program (see Eq. (14)). While the BLP is not convex, we can use the branch-and-bound framework to get increasingly accurate lower and upper bounds on the global optimum. Our improved branch-and-bound algorithm, combined with early stopping when either the upper or lower bound crosses zero, leads to an efficient procedure for robustness certification.

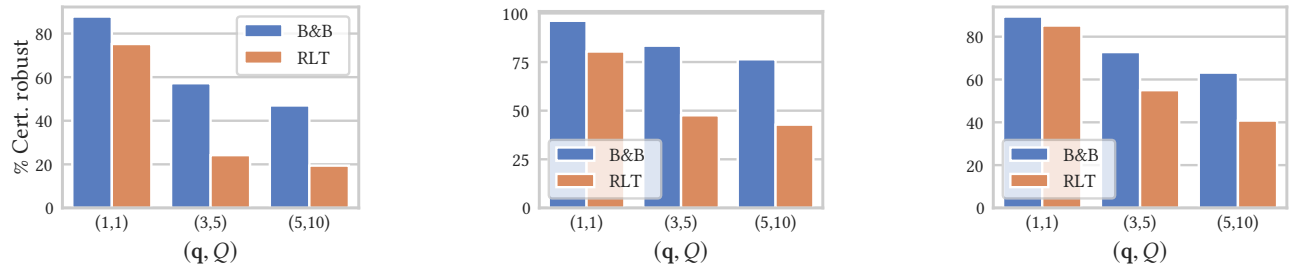
4 EXPERIMENTS

We evaluate our robustness certification method[3] on the publicly available and widely used datasets Citeseer (N=3,312, E=4,715, D=3,703, K=6) [12], Cora-ML (N=2,995, E=8,416, D=2,879, K=7) [9], and PubMed (N=19,717, E=44,324, D=500, K=3) [12]. On each dataset we train a GCN with two message-passing steps (i.e. $L = 3$) with hidden dimension 32, using 10% of the labels during training. We provide further details on our experimental setup and hyperparameters in the appendix.

Robustness certificates. We first present our results on our core contribution: the first method for certified robustness of GCN under perturbations of the graph structure. To evaluate our method on different severities of perturbations we use three different local ($\boldsymbol{q}$) and global (Q) perturbation budgets ($\boldsymbol{q}, Q$): $(1, 1)$, $(3, 5)$, and $(5, 10)$. Recall that our method can certify robustness, but not *non-robustness*. Obtaining the true number of non-robust nodes requires a brute-force search over the set of all possible perturbations and is therefore intractable for all above cases except $(1, 1)$. However, we obtain an estimate (lower bound) on the number of non-robust nodes by also performing gradient-based adversarial attacks with these budgets. We therefore report the share of nodes certified robust by our method, the share of nodes certified *non-robust* by the adversarial attack, and the total share of certified nodes, which is the sum of the former two. The gap to 100% are nodes for which no certificate can be given.

In Table 1a we present our results on standard GCN. We see the general trend that increasing the perturbation budget leads to the fraction of certifiably robust nodes decreasing, while on the other hand the share of non-robust nodes increases. Somewhat

[3] Code available at https://www.daml.in.tum.de/robust-gcn/



(a) Cora-ML (b) Citeseer (c) Pubmed

Figure 3: Comparison of our branch-and-bound (B&B) algorithm with a reformulation-linearization (RLT) baseline.

Dataset	Cora-ML					Citeseer					PubMed				
(Rob.) Training	Standard	Robust GCN	ED	Atk.	B&B	Standard	Robust GCN	ED	Atk.	B&B	Standard	Robust GCN	ED	Atk.	B&B
% Cert. robust	56.2	**57.2**	55.4	55.8	56.6	78.8	**83.4**	78.0	79.2	79.8	58.2	**72.8**	60.4	61.2	60.4

Table 2: Results for employing various training schemes: standard training, robust training as in [19], ED (edge dropout with $p = 0.2$), ED with edges found by adversarial attacks, ED with edges found by B&B. $(q, Q) = (3, 5)$.

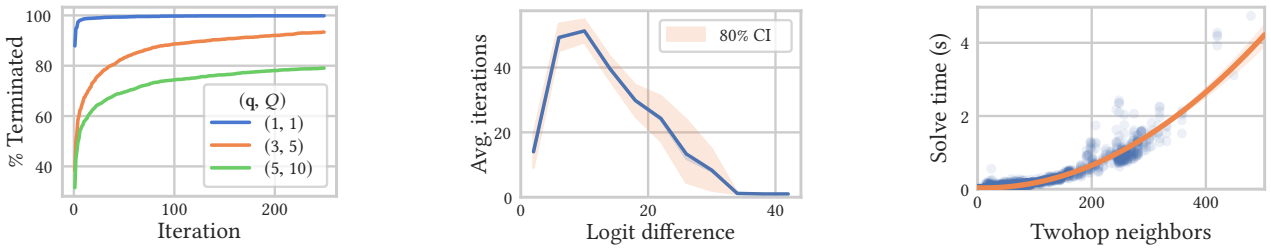
surprisingly, on all datasets, more than 5% of the nodes can change their predicted class label even when only a single edge removed, indicating that standard GCN is highly nonrobust. For the perturbation budget of (3, 5), we can certify as robust more than half of the nodes on each of the datasets, while the share of nonrobust nodes increases significantly.

In Table 1b we see the results of certifying GCN trained for robustness against *attribute* perturbations proposed in [19]. A striking difference is that on every dataset and every budget, we have more certifiably robust nodes and less nonrobust nodes compared to standard GCN. For PubMed and a budget (5, 10) we can even certify more than 25% (rel.) additional nodes as robust while having more than 20% (rel.) fewer nonrobust nodes. This result is remarkable since it suggests that robust training on a different objective (robustness to attribute perturbations) also has a beneficial effect on the robustness w.r.t. graph structure perturbations. Our method is the first that enables us to draw such conclusions based on robustness certification. Observe the high share of overall certifiable nodes; thus, for the vast majority of nodes we have a clear decision (robust/not robust). The cases in which no decision can be made can either be due to the relaxation in the certification method *or* the fact that we only have a heuristic adversarial attack (except for the budget (1, 1), where we compute the exact numbers by exhaustive search). On Cora-ML we also analyzed a budget of (10, 20): in such a setting still around 41% of the nodes can be certified as robust.

Comparison to linear relaxation. We further compare our branch-and-bound (B&B) algorithm to a relaxation via reformulation-linearization (RLT) of the bilinear program (see [10] for details on the relaxation). While the latter only requires solving a single linear program, its optimal solution is necessarily a lower bound on our solution obtained via B&B. In Fig. 3 we compare the share of successful robustness certifications for our B&B algorithm and the RLT relaxation. As expected, on every dataset and for every budget our B&B approach outperforms the baseline. Even more, the gap widens for increasing budgets, e.g. for Cora-ML and budgets (3, 5) and (5, 10) our method certifies more than twice as many nodes as the baseline. In conclusion, we find that our B&B algorithm has substantial advantages over the linear relaxation baseline.

Other types of robust training. Adversarial training, i.e. including adversarially perturbed examples into the training, is a standard defense against attacks [7]. In our setting, this corresponds to randomly dropping edges during training. To evaluate whether this leads to improved robustness we test a range of adversarial training techniques in Table 2. ED corresponds to edge dropout, i.e. randomly dropping edges (with $p = 0.2$) at each training iterations. Atk. is similar to ED but we drop out edges proportional to how often they were removed by adversarial attacks. In B&B we drop out edges proportional to how often they were set to zero in the optimal solutions obtained by B&B. We see that none of the robust training procedures consistently improve upon standard training, except the robust training for robustness w.r.t. feature perturbations proposed in [19]. Hence, we conclude that standard adversarial training does not lead to higher robustness. Note that this finding is in line with [5], who also observe only a minimal positive effect on robustness via edge dropout. Future work could explore training methods making direct use of the robustness goal (i.e. similar to [19]); however due to the branch-and-bound procedure for obtaining certificates, this approach is much more challenging.

Analysis of B&B. In Fig. 4 we present insights into the optimization procedure of our branch-and-bound algorithm. In Fig. 4a we can see that most instances terminate within the first few iterations of our branch-and-bound procedure (note that the y-axis starts at 40%; we force termination after 250 iterations). Recall that our algorithm terminates at iteration k if either the lower bound $v^{(k)}$ is positive (recall Fig. 2 for an example), leading to a robustness certificate, or that the upper bound $V^{(k)}$ is negative, meaning that we are



(a) Termination rate of B&B. **(b) Required iterations vs. logit difference.** **(c) Solve time vs. twohop neighbors.**

Figure 4: Analysis of the branch-and-bound procedure on the CORA-ML dataset. For (b) and (c) we have $(q, Q) = (3, 5)$.

certain that no certificate can be given. Furthermore, for increasing budgets, our procedure tends to require more iterations to converge. This can be explained by the fact that the domain Ω becomes larger for increasing budgets. Therefore we need more subdivision steps to achieve the accuracy required to make a decision.

Fig. 4b shows the relationship between the logit difference before the attack and the average number of iterations until convergence. Recall that the logit difference is the difference of the largest log-probability assigned by the classifier minus the second-largest log-probability. Therefore it indicates how confidently a node is classified in its current class. In the figure we can see that for nodes with a very large logit difference converge after very few iterations. This is because they are so confidently classified that our method terminates very quickly. Similarly, nodes that have a small logit difference (i.e., less confidently classified) require, on average, relatively few steps until termination. An explanation is that it is easy to change these nodes' classification. Nodes with a medium logit difference require the most iterations (on average), since for thes instances neither the lower nor upper bounds change their sign early in the procedure. Still, even for these cases, the average number of iterations to terminate is only around 50, meaning that our overall algorithm is very efficient.

In Fig. 4c we see how the time required to solve one problem instance (i.e., one iteration in our B&B procedure) relates to the number of twohop neighbors (i.e., nodes reachable within two or less hops) of the target node. Note that the median time required for solving is about 75 ms, and less than 5% of instances require more than one second to solve. The quadratic trend comes from the fact that the off-the-shelf LP solver we use does not fully support *sparse* variable matrices as we have in our problems – with full sparse support the solve time scales linearly in the number of edges in the twohop neighborhood. Even with this setup, the median total time until convergence is well below one second (ca. 820 ms), hence our procedure can be used to efficiently certify large numbers of nodes.

5 CONCLUSION

In this work we present the first method for certifiable robustness on GCN under structure perturbations. We show how to frame this problem as a jointly-constrained bilinear program, and propose a branch-and-bound procedure that makes use of knowledge about the graph domain. Our procedure decomposes the original problem into sub-problems, which are in turn linear programs and can be solved using highly optimized off-the-shelf solvers. Our certification method outperforms a reformulation-linearization baseline by a large margin and is able to certify a large fraction of the nodes.

Acknowledgements. This research was supported by the German Research Foundation, grant GU 1409/2-1, and the German Federal Ministry of Education and Research (BMBF), grant 01IS18036B. The authors of this work take full responsibility for its content.

REFERENCES

[1] Faiz A. Al-Khayyal and James E. Falk. 1983. Jointly Constrained Biconvex Programming. *Mathematics of Operations Research* 8, 2 (May 1983), 273–286.
[2] Aleksandar Bojchevski and Stephan Günnemann. 2019. Adversarial Attacks on Node Embeddings via Graph Poisoning. In *ICML*.
[3] Aleksandar Bojchevski and Stephan Günnemann. 2019. Certifiable Robustness to Graph Perturbations. In *NeurIPS*. 8317–8328.
[4] Aleksandar Bojchevski and Stephan Günnemann. 2020. Efficient Robustness Certificates for Discrete Data: Sparsity-Aware Randomized Smoothing for Graphs, Images and More. In *ICML*.
[5] Hanjun Dai, Hui Li, Tian Tian, Xin Huang, Lin Wang, Jun Zhu, and Le Song. 2018. Adversarial Attack on Graph Structured Data. In *ICML*.
[6] 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*. ACM, 169–177.
[7] Ian Goodfellow, Jonathon Shlens, and Christian Szegedy. 2014. Explaining and Harnessing Adversarial Examples. *arXiv 1412.6572* (12 2014).
[8] Thomas N Kipf and Max Welling. 2017. Semi-Supervised Classification with Graph Convolutional Networks. In *ICLR*.
[9] Andrew Kachites McCallum, Kamal Nigam, Jason Rennie, and Kristie Seymore. 2000. Automating the construction of internet portals with machine learning. *Information Retrieval* 3, 2 (2000), 127–163.
[10] Andrea Qualizza, Pietro Belotti, and François Margot. 2012. Linear Programming Relaxations of Quadratically Constrained Quadratic Programs. In *Mixed Integer Nonlinear Programming*, Jon Lee and Sven Leyffer (Eds.). Springer New York.
[11] Aditi Raghunathan, Jacob Steinhardt, and Percy S Liang. 2018. Semidefinite Relaxations for Certifying Robustness to Adversarial Examples. In *NIPS*.
[12] Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Galligher, and Tina Eliassi-Rad. 2008. Collective classification in network data. *AI magazine* 29, 3 (2008), 93.
[13] Eric Wong and Zico Kolter. 2018. Provable Defenses against Adversarial Examples via the Convex Outer Adversarial Polytope. In *ICML*. 5283–5292.
[14] Kaidi Xu, Hongge Chen, Sijia Liu, Pin-Yu Chen, Tsui-Wei Weng, Mingyi Hong, and Xue Lin. 2019. Topology Attack and Defense for Graph Neural Networks: An Optimization Perspective. In *IJCAI*. ijcai.org, 3961–3967.
[15] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L Hamilton, and Jure Leskovec. 2018. Graph convolutional neural networks for web-scale recommender systems. In *SIGKDD*. 974–983.
[16] Dingyuan Zhu, Ziwei Zhang, Peng Cui, and Wenwu Zhu. 2019. Robust graph convolutional networks against adversarial attacks. In *SIGKDD*. 1399–1407.
[17] Daniel Zügner, Amir Akbarnejad, and Stephan Günnemann. 2018. Adversarial Attacks on Neural Networks for Graph Data. In *SIGKDD*. 2847–2856.
[18] Daniel Zügner and Stephan Günnemann. 2019. Adversarial Attacks on Graph Neural Networks via Meta Learning. In *ICLR*.
[19] Daniel Zügner and Stephan Günnemann. 2019. Certifiable Robustness and Robust Training for Graph Convolutional Networks. In *SIGKDD*. 246–256.

A APPENDIX

A.1 Splitting and Branching Procedure

We follow Al-Khayyal and Falk [1]'s suggested procedures for splitting and branching. That is, for deciding on a problem to branch on, we choose the problem with the smallest lower bound among all open problems. To determine a dimension for the split, we choose the dimension for which the convex envelope has the largest difference to the corresponding value in the optimal solution, i.e. where the approximation is the coarsest. We refer the interested reader to [1] for more details.

A.2 Experimental setup

For all experiments we use Python 3.6.8. For the GCN training we use the hyperparameters shown in Table 3. For the robust training procedure proposed in [19] we use a local budget $q = 0.01D$ and global budget $Q = 12$, and a batch size of 8 (as suggested by the authors), and train for 1000 epochs. We use NVIDIA (R) GTX 1080 GPUs with 11 GB VRAM and use PyTorch 1.0.1. For all dataset we sample 500 nodes uniformly across the graph and certify robustness on this representative subset.

We use the IBM CPLEX solver[4] for all linear programs throughout this work, although we note that any off-the-shelf optimizer can be used. We further use CVXPY as the interface to Python. We solve each problem on a single core on an Intel(R) Xeon(R) CPU E5-2630 v4 @ 2.20GHz. We abort our branch and bound procedure after 250 iterations without termination and treat the instance as not certifiable.

Hyperparameter	Value
Learning rate	0.01
Hidden size	32
Hidden layers	1
Dropout	0.5
Training epochs	350
Training set size	10%
Weight decay	0.0001

Table 3: Hyperparameter configuration

[4] https://en.wikipedia.org/wiki/CPLEX