Exact Verification of Graph Neural Networks with Incremental Constraint Solving

Minghao Liu, Chia-Hsuan Lu, Marta Kwiatkowska
University of Oxford, United Kingdom
minghao.liu@cs.ox.ac.uk, chia-hsuan.lu@cs.ox.ac.uk, marta.kwiatkowska@cs.ox.ac.uk

Abstract

Graph neural networks (GNNs) are increasingly employed in high-stakes applications, such as fraud detection or healthcare, but are susceptible to adversarial attacks. A number of techniques have been proposed to provide adversarial robustness guarantees, but support for commonly used aggregation functions in message-passing GNNs is still lacking. In this paper, we develop an exact (sound and complete) verification method for GNNs to compute guarantees against attribute and structural perturbations that involve edge addition or deletion, subject to budget constraints. Focusing on node classification tasks, our method employs constraint solving with bound tightening, and iteratively solves a sequence of relaxed constraint satisfaction problems while relying on incremental solving capabilities of solvers to improve efficiency. We implement GNNEV, a versatile solver for message-passing neural networks, which supports three aggregation functions, sum, max and mean, with the latter two considered here for the first time. Extensive experimental evaluation of GNNEV on two standard benchmarks (Cora and CiteSeer) and two real-world fraud datasets (Amazon and Yelp) demonstrates its usability and effectiveness, as well as superior performance compared to existing exact verification tools on sum-aggregated node classification tasks.

1 Introduction

Graph neural networks (GNNs) have been widely deployed in real-world applications, such as financial fraud detection (Wang et al. 2019; Motie and Raahemi 2024), autonomous driving (Casas et al. 2020; Cai et al. 2021), healthcare treatment (Li, Huang, and Zitnik 2022; Gao et al. 2024), and scientific discovery (Jha, Saha, and Singh 2022; Reiser et al. 2022). However, like all other neural network architectures, GNNs are vulnerable to adversarial attacks, where slight perturbations can cause a prediction change (Zügner, Akbarnejad, and Günnemann 2018; Dai et al. 2018; Tao et al. 2021). Therefore, it is desirable to ensure adversarial robustness of GNNs for high-stakes applications.

Formal verification is a rigorous methodology to mathematically prove that a system meets its specifications under all possible circumstances (Clarke and Wing 1996). To evaluate the reliability and trustworthiness of neural networks, a number of approaches have been developed to verify the adversarial robustness of fully connected neural networks (Katz et al. 2017; Weng et al. 2018; Wang et al. 2018; Zhang et al. 2018; Botoeva et al. 2020), convolutional neural networks (CNNs) (Huang et al. 2017; Boopathy et al. 2019; Tran et al. 2020), recurrent neural networks (RNNs) (Ko et al. 2019; Akintunde et al. 2019; Du et al. 2021), and transformers (Shi et al. 2020; Bonaert et al. 2021). Technically, these approaches can be classified into two categories. The first category is *exact* verification, also known as *complete* verification, where the verification task is formulated as a constraint satisfaction problem (CSP) (Rossi, Van Beek, and Walsh 2006) to deterministically prove whether the model is robust or not. The second category is *approximate* verification, also known as (sound but) *incomplete* verification, where the non-convex constraints are typically relaxed to convex ones, which admits efficient computation of a lower bound on the robust region. However, when the lower bound cannot be accurately computed, the outcome of the verification may be inconclusive, which does not offer strong robustness guarantees.

Recently, the concept of adversarial robustness has been extended to GNNs. Graph inputs present additional challenges, in that the perturbations can pertain to both node attributes (Zügner and Günnemann 2019b; An et al. 2024) and graph structure, i.e., the addition and deletion of edges (Bojchevski and Günnemann 2019; Zügner and Günnemann 2020; Jin et al. 2020; Ladner, Eichelbeck, and Althoff 2025), or even node injection (Lai et al. 2024). Since graph inputs are encoded using a combination of real-valued and discrete data, typical approaches to provide *deterministic* robustness guarantees are based on Mixed-Integer Programming (MIP) (Zhang et al. 2023; Hojny et al. 2024). We also mention methods based on randomised smoothing (Bojchevski, Klicpera, and Günnemann 2020; Osselin, Kenlay, and Dong 2023; Scholten et al. 2022), noting that their guarantees are *probabilistic* and thus not directly comparable.

Despite recent advances, existing exact verification approaches are still limited to GNNs with *sum* aggregation. However, other common aggregations, including *max* and *mean*, have been shown to be necessary and effective in both theory (Corso et al. 2020; Rosenbluth, Tönshoff, and Grohe 2023) and practice (Ying et al. 2018; Dehmamy, Barabási, and Yu 2019), and are supported as standard by the GraphSAGE library (Hamilton, Ying, and Leskovec 2017). Existing methods also typically focus on graph classification problems in the context of structural perturbations, implemented only for edge deletion for tractability reasons. Node classification tasks have been under-explored, and yet are relied upon in high-stakes applications, for example, fraud detection in financial networks (Wang et al. 2019) and cyber attack localization in smart grids (Haghshenas, Hasnat, and Naeini 2023). For such applications, in order to provide certified robustness guarantees, methods that yield conclusive outcomes are preferable.

We develop an exact verification method for GNNs to provide guarantees against attribute and structural perturbations subject to local and global budget constraints. Focusing on node classification tasks[1], we support three commonly used aggregation functions, sum, max and mean, with the latter two being non-linear and not tackled previously. We employ constraint solving with bound tightening and iteratively solve a sequence of relaxed CSPs while relying on incremental solving capabilities of solvers to improve efficiency. We implement GNNEV, a versatile and efficient exact verifier for GNNs, and conduct extensive experiments on two standard node classification benchmarks, Cora and CiteSeer, and two real-world fraud datasets, Amazon and Yelp. The results indicate that GNNEV outperforms the verifier by Hojny et al. (2024), the only exact method known to us, on sum aggregation and edge deletion under different model sizes and perturbation budgets. On a range of aggregation functions, we show that GNNEV can provide useful insight into the susceptibility of GNNs to adversarial attacks, which is important for deployment in high-stakes applications.

Our novel contributions can be summarised as follows.

- We introduce the first exact verification method for GNNs with two common aggregation functions, max and mean, and design specialised tightened bound propagation strategies to reduce computational cost.
- We propose an algorithm that iteratively encodes GNN layers backwards and utilises incremental constraint solving to speed up performance while maintaining exact verification capability.
- We implement GNNEV, a versatile and efficient exact verifier for GNNs that supports new aggregation functions and edge addition, and show its usability and superior performance through extensive experiments.

2 Related Work

Verification of neural networks Adversarial robustness verification methods can be classified into complete (exact) methods, such as constraint solving (Katz et al. 2017; Huang et al. 2017; Tjeng, Xiao, and Tedrake 2019), or sound though incomplete methods, e.g., convex relaxation (Salman et al. 2019; Xu et al. 2020), which can be strengthened to completeness by employing branch-and-bound procedures (Bunel et al. 2018; Lu and Kumar 2020). Katz et al. (2017) prove that exact NN verification is generally NP-hard. More recent works move beyond fully-connected and convolutional architectures, and include MIP-based exact verification of recurrent networks (Akintunde et al. 2019) and approximate verification of transformers utilising the zonotope abstract domain (Bonaert et al. 2021). Regarding GNN verification, existing approaches concern graph convolutional networks with respect to attribute (Zügner and Günnemann 2019b; An et al. 2024) or structural (Bojchevski and Günnemann 2019; Zügner and Günnemann 2020; Jin et al. 2020; Ladner, Eichelbeck, and Althoff 2025) perturbations. Most are approximate, relying on optimisation (Bojchevski and Günnemann 2019; Zügner and Günnemann 2020; Jin et al. 2020; Ladner, Eichelbeck, and Althoff 2025) or zonotope-based relaxation (Ladner, Eichelbeck, and Althoff 2025). Our approach is most similar to the exact MIP-based approach SCIP-MPNN (Hojny et al. 2024) for sum-aggregated message-passing GNNs and edge deletion only. Their method supports static and dynamic (aggressive) bound tightening over the complete network encoding. In contrast, our method also supports max and mean, as well as edge addition, and proceeds by building the encoding incrementally layer by layer while statically tightening the bounds. For node classification tasks involving large numbers of neighbouring nodes, our exact method can greatly reduce the costs of encoding, thus enhancing performance.

Adversarial attacks against GNNs Zügner, Akbarnejad, and Günnemann (2018); Dai et al. (2018); Zügner and Günnemann (2019a); Xu et al. (2019) demonstrate that adversarial attacks can induce incorrect predictions by GNNs on node classification. Chang et al. (2020); Mu et al. (2021) propose black-box attack approaches without accessing any knowledge of the GNN classifiers. Zou et al. (2021); Li et al. (2023b) further improve the efficiency and scalability of adversarial attack algorithms. Another direction is attacks on graph classification (Zhang et al. 2021) and link prediction (Zhou et al. 2019; Chen et al. 2023). We emphasise that a failure to find an attack does not imply robustness. In contrast, we aim to verify that GNNs are not vulnerable to any admissible attack, which provides strong guarantees and can thus be used for robustness certification.

We further remark that, while most studies focus on undirected graphs and claim potential extensions to directed graphs, some works (Geisler et al. 2021; Chen et al. 2023), along with our paper, consider both directed and undirected graphs.

Robustness enhancement of GNNs Adversarial training is a common technique applied to enhance the robustness of GNNs (Dai et al. 2019; Xu et al. 2019; Feng et al. 2021; Li et al. 2023a; Gosch et al. 2023). Wu et al. (2019); Zhang and Zitnik (2020) aim to enhance the robustness of GNNs by estimating and recovering potential adversarial perturbations. Moreover, generalised randomised smoothing (Bojchevski, Klicpera, and Günnemann 2020; Wang et al. 2021; Zhang et al. 2021) and partition-ensemble (Xia et al. 2024) techniques have been developed for GNNs to provide defences against attacks with probabilistic guarantees. Instead, we develop a formal verification algorithm, which is complementary to robustness enhancement and offers deterministic guarantees.

[1] An extension of our method to graph classification is discussed in Appendix A.

Theoretical analysis of GNN verification Expressiveness of sum-aggregated GNNs is studied in Barceló et al. (2020); Benedikt et al. (2024); Nunn et al. (2024) and mean-aggregated in Schönherr and Lutz (2025). The adversarial robustness problem for node classification is shown decidable in Sälzer and Lange (2023) under the assumption of bounded degree of input graphs. Benedikt et al. (2024) (and Nunn et al. (2024)) study verification of certain properties that quantify over all graph inputs proving PSPACE-completeness for GNNs with rational (resp. integer) coefficients, Boolean-attributed input graphs and truncated ReLU activation functions. We focus instead on a practical verification approach applicable to commonly used GNNs, with real-valued coefficients and attributes and ReLU activation functions. Our problem setting is clearly decidable because it only considers a finite and bounded input graph set.

3 Preliminaries

3.1 Graph Neural Networks

Let $G = \langle V, E, X \rangle$ be an *attributed directed graph*, where $V = \{v_1, v_2, \ldots, v_n\}$ is a set of nodes, $E \subseteq V \times V$ is a set of edges with no self-loops, and $X = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\}$ is a set of real *attribute vectors* for the nodes with the same dimension. For a node $v \in V$ and a natural number k, we denote the set of *k-hop incoming neighbours of* v by $\mathcal{N}_k(v)$. For simplicity, we write $\mathcal{N}(v)$ when $k = 1$. *Graph neural networks (GNNs)* are a class of neural network architectures designed to operate on graph data. In this paper, we only consider *node classification* task, where the goal is to classify a node $v \in V$ into one class in $C = \{c_1, c_2, \ldots, c_m\}$. More precisely, we view a GNN as a function f such that, given an attributed directed graph $G = \langle V, E, X \rangle$ and a node $v \in V$, the output of f is a class, i.e., $f(G, v) \in C$.

There are a variety of GNN architectures. Existing formal verification works mostly focus on graph convolutional networks (GCNs) (Kipf and Welling 2017), which utilise a specific and fixed aggregation function and lack universality. In this work, we consider *GraphSAGE* (Hamilton, Ying, and Leskovec 2017), also known as *Message-Passing Neural Networks (MPNNs)* (Gilmer et al. 2017), which are a more general GNN architecture because (i) they support the choice of multiple neighbourhood aggregation functions and (ii) the expressive power of MPNNs has been shown to encompass other popular variations such as GCNs and graph attention networks (GATs) (Bronstein et al. 2021).

A GNN consists of K layers. The dimension of a GNN is a $(K + 1)$ tuple of positive integers $d_0, d_1, \ldots, d_K$. For an input attributed graph $G = \langle V, E, X \rangle$, for each node $v \in V$, its real-valued embedding vector $\mathbf{h}_v^{(0)}$ is a d_0-dimension vector initialised to its corresponding attribute vector $\mathbf{x}_v$. For the k-th layer, GNNs compute the d_k-dimension embedding $\mathbf{h}_v^{(k)}$ by

$$\mathbf{h}_v^{(k)} = \sigma\left(\begin{matrix}\mathbf{W}_1^{(k)} \cdot \mathbf{h}_v^{(k-1)} + \\ \mathbf{W}_2^{(k)} \cdot \mathbf{aggr}\left(\left\{\!\!\left\{\mathbf{h}_u^{(k-1)} \mid u \in \mathcal{N}(v)\right\}\!\!\right\}\right)\end{matrix}\right), \quad (1)$$

where $\mathbf{W}_1^{(k)}, \mathbf{W}_2^{(k)}$ are learnable parameter matrices with dimension $d_k \times d_{k-1}$ for the k-th layer, $\sigma(x) = \max(0, x)$ is the ReLU activation function, $\mathbf{aggr} \in \{\text{sum}, \text{max}, \text{mean}\}$ is the aggregation function, and $\{\!\{\}\!\}$ denotes a multiset. Finally, the last dimension d_K is exactly m, and the predicted class $\hat{c}_v$ of a node $v \in V$ is obtained by the Softmax function:

$$\hat{c}_v = \operatorname*{arg\,max}_{c \in C} \left(\text{Softmax}\left(\mathbf{h}_v^{(K)}\right)[c]\right). \quad (2)$$

3.2 Adversarial Robustness of GNNs

Node embeddings are directly affected by attribute perturbations and indirectly affected by structural perturbations through message passing, which may result in prediction instability. Changes in predictions caused by admissible perturbations indicate a lack of *adversarial robustness*. We slightly adapt the definition of adversarial robustness for GNNs in Bojchevski and Günnemann (2019) by (i) allowing perturbations in both graph structure and node attributes, and (ii) removing the redundant fixed edge set. Given an attributed directed graph $G = \langle V, E, X \rangle$, the attacker typically has limited capabilities to perturb it. Let $F \subseteq V \times V$ be a set of *fragile edges*. Intuitively, an attacker can either remove edges in $F \cap E$ or insert edges in $F \backslash E$. For structural perturbations, we assume a global budget Δ and a local budget δ_v for each $v \in V$, both of which are non-negative integers. For attribute perturbations, we work with a budget $\epsilon_{v,i}$ for each $v \in V$ and $i \in \{1, 2, \ldots, d_0\}$, which is a non-negative real value representing the maximum perturbation allowed for the i-th dimension of $\mathbf{x}_v \in X$. The admissible perturbation space is defined as follows.

Definition 1 (Admissible perturbation space of graph)**.** Given an attributed directed graph $G = \langle V, E, X \rangle$, a set of fragile edges $F \subseteq V \times V$, and perturbation budgets Δ, δ, ϵ, the *admissible perturbation space* $\mathcal{Q}(G)$ of G with respect to F and budgets is the set of attributed directed graphs $\tilde{G} = \langle V, \tilde{E}, \tilde{X} \rangle$ that satisfy

1. $E \backslash F \subseteq \tilde{E} \subseteq E \cup F$,
2. $|E \backslash \tilde{E}| + |\tilde{E} \backslash E| \leq \Delta$,
3. For every $v \in V$, $|\mathcal{N}(v) \backslash \tilde{\mathcal{N}}(v)| + |\tilde{\mathcal{N}}(v) \backslash \mathcal{N}(v)| \leq \delta_v$,
4. For every $v \in V$ and $1 \leq i \leq d_0$, $|\mathbf{x}_v[i] - \tilde{\mathbf{x}}_v[i]| \leq \epsilon_{v,i}$,

where we denote the incoming neighbours of v in the graph $\tilde{G}$ by $\tilde{\mathcal{N}}(v)$. Note that this is a general formulation as F can be defined arbitrarily for different purposes, and allows us to consider edge addition when the size of F is moderate[2]. For example, if $F = E$, only edge deletion is allowed, and any existing edge can be deleted. If $F = V \times V$, edges can be added or deleted between any pair of nodes.

Finally, we introduce the formal definition of adversarial robustness of GNNs.

Definition 2 (Adversarial robustness of GNNs)**.** Given a node-classification GNN f, an attributed directed graph G with admissible perturbation space $\mathcal{Q}(G)$, a target node $t \in V$, and its predicted class $\hat{c}_t \in C$, we say that f is *adversarially robust* for t with class $\hat{c}_t$ if and only if, for every perturbed graph $\tilde{G} \in \mathcal{Q}(G)$, it holds that $f(\tilde{G}, t) = \hat{c}_t$.

[2] Edge addition was viewed as impractical and not implemented under the definition of Hojny et al. (2024), Eq. (14), which is a special case of ours when $F = E$.

Adversarial robustness verification aims to guarantee that the prediction for a given node will not change under any admissible perturbations, subject to budget constraints, and can be reduced to computing the worst-case margin between the correct class and other classes.

4 Exact Verification of GNNs

In this section, we introduce our method, which encodes the verification task as a constraint satisfaction problem (CSP). Next, the bound tightening strategies for max and mean-aggregated GNNs are proposed. Finally, we illustrate an efficient exact verification algorithm based on incremental solving. Note that, while the emphasis of this paper is on node classification, our method can also be generalised to graph classification, as discussed in Appendix A.

4.1 Encoding

A CSP aims to determine whether there exists an assignment of variables that satisfies a given set of constraints. We encode the exact verification tasks as CSPs, which consist of three parts: input perturbation, GNN architecture, and verification objective. Following existing MIP encodings for GNNs (Zhang et al. 2023; Hojny et al. 2024), we extend them to accommodate max and mean aggregations, except that, rather than directly encoding max constraints using the big-M method, we rely on the solver.

Input perturbation Consider an input graph $G = \langle V, E, X \rangle$, where our goal is to verify the adversarial robustness for the target node $t \in V$. Note that, for K-layer GNNs, only perturbations to nodes in $\mathcal{N}_K(t)$ can affect the prediction of t. Two kinds of perturbations are allowed. The first is attribute perturbation, where the node attributes can be modified within a specified range. For each node $v \in \mathcal{N}_K(t)$, we set d_0 real variables $attr_{v,1}, attr_{v,2}, \ldots, attr_{v,d_0}$ as the attribute values after perturbations. Given the perturbation budget $\epsilon_{v,i}$ for the i-th dimension of $\mathbf{x}_v$, the following constraint is applied:

$$\mathbf{x}_v[i] - \epsilon_{v,i} \ \leq \ attr_{v,i} \ \leq \ \mathbf{x}_v[i] + \epsilon_{v,i}. \tag{3}$$

The second type is structural perturbation, which can impact the message-passing mechanism. For each edge $(u, v) \in F$, we set a Boolean variable $pe_{u,v}$ to represent if (u, v) is perturbed. Given the global budget Δ, we have the constraint

$$\sum_{(u,v)\in F} pe_{u,v} \leq \Delta. \tag{4}$$

For each $v \in \mathcal{N}_{K-1}(t)$, given the local budget δ_v, we have

$$\sum_{(u,v)\in F} pe_{u,v} \leq \delta_v. \tag{5}$$

GNN architecture We encode the architecture of GNNs as follows. For the k-th layer as shown in Eq. (1), the output embedding of node $v \in \mathcal{N}_{K-k}(t)$ is represented by d_k real variables $h^{(k)}_{v,1}, h^{(k)}_{v,2}, \ldots, h^{(k)}_{v,d_k}$. Let $h^{(0)}_{v,i} = attr_{v,i}$. The encoding for each layer consists of three parts.

First, the neighbours of node v are aggregated to produce the message. We set d_{k-1} real variables $msg^{(k)}_{v,1}, msg^{(k)}_{v,2}, \ldots, msg^{(k)}_{v,d_{k-1}}$ as the message vector. We support three common aggregation functions: sum, max, and mean. Note that the last two functions are nonlinear, which accounts for higher computational complexity for verification. Before we give the encoding for the aggregation functions, we set the auxiliary variables $a^{(k)}_{v,i,u}$, which indicate the contribution of node u to $msg^{(k)}_{v,i}$ through the edge (u, v). The following constraints restrict the values of $a^{(k)}_{v,i,u}$: for every $(u, v) \in E \backslash F$,

$$a^{(k)}_{v,i,u} = h^{(k-1)}_{v,i}; \tag{6}$$

for every $(u, v) \in E \cap F$,

$$pe_{u,v} \to a^{(k)}_{v,i,u} = 0 \quad \text{and} \quad \neg pe_{u,v} \to a^{(k)}_{v,i,u} = h^{(k-1)}_{v,i}; \tag{7}$$

for every $(u, v) \in F \backslash E$,

$$pe_{u,v} \to a^{(k)}_{v,i,u} = h^{(k-1)}_{v,i} \quad \text{and} \quad \neg pe_{u,v} \to a^{(k)}_{v,i,u} = 0. \tag{8}$$

For sum and max aggregation, the constraints are shown in Eq. (9) and (10), respectively.

$$msg^{(k)}_{v,i} = \sum_{(u,v)\in F\cup E} a^{(k)}_{v,i,u}. \tag{9}$$

$$msg^{(k)}_{v,i} = \max_{(u,v)\in F\cup E} a^{(k)}_{v,i,u}. \tag{10}$$

For mean aggregation, the constraints are set as follows:

$$\begin{aligned} deg_v = &\sum_{(u,v)\in F\cap E} (1 - pe_{u,v}) + \sum_{(u,v)\in F\backslash E} pe_{u,v} \\ &+ |(u,v) \in E\backslash F|, \\ deg_v \cdot msg^{(k)}_{v,i} = &\sum_{(u,v)\in F\cup E} a^{(k)}_{v,i,u}, \\ deg_v = 0 \to &\ msg^{(k)}_{v,i} = 0. \end{aligned} \tag{11}$$

Next, the new embedding of node v before ReLU is computed through a linear transformation as follows:

$$y^{(k)}_{v,i} = \sum_{j\in[1,d_{k-1}]} \mathbf{W}^{(k)}_1[i,j] \cdot h^{(k-1)}_{v,j} + \mathbf{W}^{(k)}_2[i,j] \cdot msg^{(k)}_{v,j}. \tag{12}$$

Finally, the node embedding is obtained via ReLU activation, except for the last layer:

$$h^{(k)}_{v,i} = \max\left(y^{(k)}_{v,i}, 0\right). \tag{13}$$

Verification objective According to Definition 2, our objective is to verify that no admissible attribute or structural perturbation can make the prediction of the GNN inconsistent with the input graph for a target node. Let the predicted class of target node t be $\hat{c}_t$, where the set of classes is denoted by C. Then we set the following constraint:

$$\left(\max_{c\in C\backslash\{\hat{c}_t\}} y^{(K)}_{t,c}\right) \geq y^{(K)}_{t,\hat{c}_t}, \tag{14}$$

which expresses the worst-case margin at the logit level. Let $\mathbf{y}_t^{(K)} = \left[y_{t,1}^{(K)}, y_{t,2}^{(K)}, \dots, y_{t,|C|}^{(K)}\right]$. Since the Softmax function is monotonically increasing, Eq. (14) implies that there exists $c \in C$ with $c \neq \hat{c}_t$ such that

$$\text{Softmax}\left(\mathbf{y}_t^{(K)}\right)[c] \geq \text{Softmax}\left(\mathbf{y}_t^{(K)}\right)[\hat{c}_t].$$

If Eq. (14) is feasible, this means that a perturbed graph has been found that makes the GNN misclassify node t. Therefore, the robustness verification task is translated to deciding whether the above CSP is unsatisfiable.

Note that the encoding involves at most $5N^2D$ real variables, N^2 Boolean variables, and $8ND + 2$ constraints, where N is the number of nodes in $\mathcal{N}_K(t)$ and $D := \sum_{0 \leq i \leq K} d_i$. Furthermore, the encoding can be computed in time polynomial in the size of the GNN.

4.2 Bound Tightening for Aggregations

Tightening of variable bounds has been found to be critical to enhancing the efficiency of MIP solvers. Simultaneously, high-quality bounds form the basis of our incremental solving algorithm introduced in Section 4.3. Starting from the input variables (i.e., *attr*), whose bounds have been determined by Eq. (3), we propagate these bounds layer by layer to obtain tight bounds for other variables.

Since tightened bound propagation is straightforward for linear transformations and ReLU functions, our main focus is on tightening the bounds for aggregation functions. We formalise the problem as follows: given three finite sets of variables X_1, X_2, and X_3, the upper and lower bounds for each variable, and a non-negative integer s, compute the upper and lower bounds for the variable

$$z := \mathbf{aggr}(X_1 \cup X_2' \cup X_3'),$$

where $\mathbf{aggr} \in \{\text{sum}, \text{max}, \text{mean}\}$, $X_2' \subseteq X_2$, $X_3' \subseteq X_3$, and $|X_2 \backslash X_2'| + |X_3'| \leq s$. Let N be the total size of the sets X_1, X_2, and X_3. The intuition behind the formalisation is as follows: X_1, X_2, and X_3 represent the embedding vectors of nodes connected to the target node via non-fragile edges (E/F), fragile edges ($E \cap F$), and fragile non-edges (F/E), respectively. The attacker can either delete fragile edges, that is, remove elements from X_2, or insert fragile non-edges, that is, select elements from X_3. The number of deletion and insertions is constrained by the budget s.

For a set of variables X and an integer k, let $hi(X, k)$ denote the k-th *largest upper bound* among the variables in X. If $k > |X|$, we define $hi(X, k) = -\infty$. Similarly, let $lo(X, k)$ denote the k-th *smallest lower bound* among the variables in X, with $lo(X, k) = \infty$, if $k \leq 0$. Note that $hi(X, k)$ and $lo(X, k)$ can be computed in $O(|X| \log k)$ time by maintaining a max- or min-heap. For multiple values $k_1, k_2, \dots, k_\ell$, we can reuse the heap, resulting in an overall complexity of $O(|X| \log k)$, where k is the maximum value among $k_1, k_2, \dots, k_\ell$.

For a variable x, its upper and lower bounds are denoted by $\overline{x}$ and $\underline{x}$ respectively. The case for sum aggregation has been shown by Hojny et al. (2024). We restate their bounds in the context of our formalisation for completeness:

$$\overline{z} = \sum_{x \in X_1 \cup X_2} \overline{x} + \sum_{1 \leq i \leq s} \max(hi(Y, i), 0),$$

$$\underline{z} = \sum_{x \in X_1 \cup X_2} \underline{x} + \sum_{1 \leq i \leq s} \min(lo(Y, i), 0),$$

where $Y = \{-x \mid x \in X_2\} \cup X_3$. The upper and lower bounds of z can be computed in time $O(N \log s)$.

Max aggregation The upper and lower bounds of z for the max aggregation can be obtained by case analysis. For the corner case $s = 0$, $z = \max(X_1 \cup X_2)$. Note that if $X_1 \cup X_2 = \emptyset$, then $\overline{z} = \underline{z} = 0$. We now focus on the case $s > 0$. For the upper bound:

$$\overline{z} = \begin{cases} \max(0, hi(X_2, 1), hi(X_3, 1)), & \text{if } X_1 = \emptyset, s \geq |X_2|, \\ \max(hi(X_1, 1), hi(X_2, 1), hi(X_3, 1)), & \text{otherwise.} \end{cases}$$

On the other hand, for the lower bound:

$$\underline{z} = \begin{cases} \min(0, lo(X_2, 1), lo(X_3, 1)), & \text{if } X_1 = \emptyset, s \geq |X_2|, \\ lo(X_2, |X_2| - s), & \text{if } X_1 = \emptyset, s < |X_2|, \\ \max(lo(X_1, |X_1|), \min(lo(X_2, 1), lo(X_3, 1))), & \\ & \text{if } X_1 \neq \emptyset, s \geq |X_2|, \\ \max(lo(X_1, |X_1|), lo(X_2, |X_2| - s)), & \\ & \text{if } X_1 \neq \emptyset, s < |X_2|. \end{cases}$$

Note that the upper and lower bounds for z can be computed in time $O(N \log s)$.

Mean aggregation In this case, we reduce the original problem to a combination of subproblems. For non-negative integers s_2 and s_3, define the variable

$$z_{s_2,s_3} := \mathbf{mean}(X_1 \cup X_2' \cup X_3'),$$

where $X_2' \subseteq X_2$, $X_3' \subseteq X_3$, $|X_2 \backslash X_2'| = s_2$, and $|X_3'| = s_3$. The main difference is that we now explicitly fix separate budgets for deletions and insertions. The upper and lower bounds of the original problem z are given by

$$\overline{z} = \max_{\substack{s_2+s_3 \leq s \\ 0 \leq s_2 \leq |X_2| \\ 0 \leq s_3 \leq |X_3|}} \overline{z}_{s_2,s_3} \quad \text{and} \quad \underline{z} = \min_{\substack{s_2+s_3 \leq s \\ 0 \leq s_2 \leq |X_2| \\ 0 \leq s_3 \leq |X_3|}} \underline{z}_{s_2,s_3}.$$

For the subproblem z_{s_2,s_3}, since the budgets for deletion and insertion are separate, the upper and lower bounds can be obtained through greedy choices. In the corner case where X_1 is empty, $s_2 = |X_2|$, and $s_3 = 0$, $\overline{z}_{s_2,s_3} = \underline{z}_{s_2,s_3} = 0$. In all other cases, we have $\overline{z}_{s_2,s_3} = \overline{w}_{s_2,s_3}/n_{s_2,s_3}$ and $\underline{z}_{s_2,s_3} = \underline{w}_{s_2,s_3}/n_{s_2,s_3}$, where $n_{s_2,s_3} = |X_1| + |X_2| - s_2 + s_3$ and

$$\overline{w}_{s_2,s_3} = \sum_{x \in X_1} \overline{x} + \sum_{1 \leq i \leq |X_2| - s_2} hi(X_2, i) + \sum_{1 \leq i \leq s_3} hi(X_3, i),$$

$$\underline{w}_{s_2,s_3} = \sum_{x \in X_1} \underline{x} + \sum_{1 \leq i \leq |X_2| - s_2} lo(X_2, i) + \sum_{1 \leq i \leq s_3} lo(X_3, i).$$

Since the summations over lo and hi can be computed incrementally among subproblems and there are at most $(s+1)^2$ subproblems, we can compute the upper and lower bounds for z in time $O(s^2 + N \log s)$.

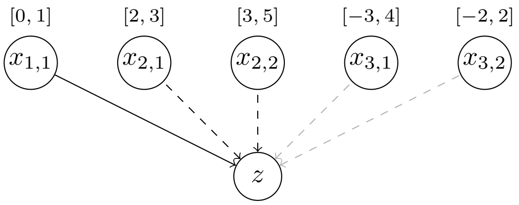
Remark 1. The upper and lower bounds provided in this section for max and mean aggregations are tight. That is, there exist sets $X_2' \subseteq X_2$ and $X_3' \subseteq X_3$ with $|X_2 \backslash X_2'| + |X_3'| \leq s$ that achieve these bounds exactly.

The proof is included in Appendix B.

Example 1. Consider sets of variables $X_1 = \{x_{1,1}\}$, $X_2 = \{x_{2,1}, x_{2,2}\}$, and $X_3 = \{x_{3,1}, x_{3,2}\}$. Each variable has upper and lower bounds as shown in Figure 1. Let the budget $s = 1$. We now consider tightened bound propagation:

- For max aggregation, the upper bound is 5 by no operation, the same as the plain method[3]; the lower bound is 2 by deleting $x_{2,2}$, improved by 5 over the plain method.
- For mean aggregation, the upper bound is 3.25 by inserting $x_{3,1}$, improved by 0.08; the lower bound is 0.5 by inserting $x_{3,1}$, improved by 2.17.

For both aggregations, tightened bound propagation reduces the domain of the variable z.



	Plain	Tightened
max (*ub*)	$\max(1,3,5,4,2) = \mathbf{5}$	$\max(1,5,4) = \mathbf{5}$
max (*lb*)	$\min(0,2,3,-3,-2) = -3$	$\max(0,2) = \mathbf{2}$
mean (*ub*)	$\frac{1+5+4}{3} = 3.33$	$\frac{1+5+3+4}{4} = \mathbf{3.25}$
mean (*lb*)	$\frac{0-3-2}{3} = -1.67$	$\frac{0+2+3-3}{4} = \mathbf{0.5}$

Figure 1: An illustration of tightened bound propagation. Solid lines denote the non-fragile edges, black dashed lines denote fragile edges, and gray dashed lines denote fragile non-edges. The tighter bound is shown in **bold**.

4.3 Verification with Incremental Solving

With tightened bound propagation, we can verify adversarial robustness with enhanced efficiency. However, graphs in node classification tasks are typically large in size, and for a node t we typically have that $|\mathcal{N}_{k+1}(t)| \gg |\mathcal{N}_k(t)|$ for $1 \leq k < K$, leading to rapid expansion of the encoding and a decrease in efficiency. To improve performance while maintaining exact verification capability, we utilise *incremental solving*, an effective mechanism in existing CSP solvers, to iteratively solve a series of simplified relaxation problems compared to the original problem.

Algorithm 1 describes our incremental solving-based verification method for GNNs. First, the CSP is formulated and the bounds of variables are derived based on the rules of bound tightening of Section 4.2 and layer-by-layer propagation (lines 1–6). Next, we solve the formula Φ iteratively using an incremental solving mechanism (lines 7–15). Initially, $\Phi = \varphi_{obj}$, which means Φ only contains the variables representing the final embeddings of the target node t, which define the verification objective. In each iteration, new variables and the corresponding constraints are added to Φ, and a MIP solver S_Θ is then called (line 11), where Θ represents the generated cuts, which are redundant constraints produced by the solver to prune the search space. If Φ is found to be unsatisfiable, then the GNN has been verified to be `robust`. Otherwise, the encoding of the previous layer is added to Φ. Finally, if the result of the last iteration remains satisfiable, then the GNN is verified as `nonrobust`. Note that, when each additional layer is added in the graph, the number of neighbouring nodes can increase significantly. Therefore, our algorithm effectively reduces the size of the encoding in early iterations, which leads to a more efficient exact verification process.

Algorithm 1: Incremental Solving-based Exact Robustness Verification on GNNs

Input: Trained GNN f with K layers, attributed graph $G = \langle V, E, X \rangle$, target node t, fragile edge set F, perturbation budgets Δ, δ, ϵ.
Output: Verification result.

```
 1: $Bounds \leftarrow \{\underline{attr}, \overline{attr}\}$;
 2: for $k$ from 1 to $K$ do
 3:     $\varphi_k \leftarrow$ EncodeGNNLayer($f, k, G, F, \Delta, \delta, \epsilon$);
 4:     $Bounds \leftarrow$ BoundPropagation($\varphi_k, Bounds$);
 5: end for
 6: $\varphi_{obj} \leftarrow$ EncodeObjective($\varphi_K$);
 7: $S_\Theta \leftarrow$ InitIncSolver();
 8: $\Phi \leftarrow \varphi_{obj}$;
 9: for $k$ from $K$ to 1 do
10:     $\Phi \leftarrow \Phi \wedge \varphi_k$;
11:     $result, \Theta \leftarrow$ IncSolve($S_\Theta, \Phi, Bounds$);
12:     if $result$ = unsat then
13:         return robust;
14:     end if
15: end for
16: return nonrobust;
```

Finally, we show the correctness of Algorithm 1.

Theorem 1. Given a GNN f, an attributed directed graph G, and a perturbation space $\mathcal{Q}(G)$, f is adversarially robust for a node t if and only if Algorithm 1 returns `robust`.

The proof is included in Appendix C.

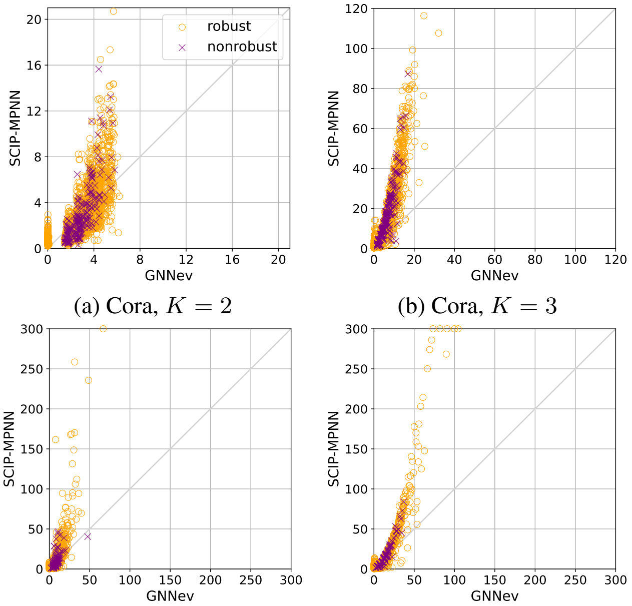
5 Experimental Evaluation

5.1 Implementation and Setup

We implement our method as GNNEV, a versatile and efficient exact verifier specifically designed for message-passing GNNs[4]. As a Python library, GNNEV inputs trained

[3] The plain method computes the upper (lower) bounds for max aggregation as the maximum (minimum) bound of all the variables, and for mean aggregation by sorting variable bounds from $X_2 \cup X_3$ in ascending order and finding the maximum (minimum) mean value among all postfix (prefix) sequences.

[4] The source code and data will be made publicly available in the final version.



(a) Cora, $K = 2$

(b) Cora, $K = 3$

(c) CiteSeer, $K = 2$

(d) CiteSeer, $K = 3$

Figure 2: Comparison of runtime (in seconds) for sum-aggregated node-classification GNNs. Each data point represents a robustness verification task for one node. A point above the diagonal line indicates that GNNEv outperforms SCIP-MPNN on that task.

models built by the SAGEConv module in PyTorch Geometric (Fey and Lenssen 2019) and calls the underlying Gurobi 11.0.3 solver[5] for MIP solving. Compared to available exact GNN verifiers, GNNEv introduces two new features: (i) it accepts three commonly used aggregation functions: sum, max, and mean, which broadens usability, and (ii) it allows for both edge addition and deletion as valid perturbations, making it applicable to a wider range of attack scenarios.

To evaluate performance, we trained a batch of GraphSAGE models on two standard node classification datasets, Cora and CiteSeer (Sen et al. 2008), and two real-world fraud datasets, Amazon (McAuley and Leskovec 2013) and Yelp (Rayana and Akoglu 2015), varying the number of layers $K \in \{2, 3\}$ and aggregation functions $\mathbf{aggr} \in \{\text{mean}, \text{max}, \text{sum}\}$. We regard the robustness verification for each node as a task, which checks prediction stability for any set of admissible perturbations up to a given global structural budget Δ. All experiments were conducted on a server with Intel Xeon Gold 6252 @ 2.10GHz CPU, 252GB RAM, and Ubuntu 18.04 OS. A verifier, including both GNNEv and the baselines, ran each task on a single CPU core for fair comparison. The time limit for each task was set to 300s. More details of the datasets and experimental setups can be found in Appendix D.

5.2 Comparison with Baselines

We compare GNNEv to SCIP-MPNN (Hojny et al. 2024), the only exact verifier immediately applicable to adversarial robustness of GNNs. It includes five versions with different underlying solvers and bound tightening strategies. To ensure fair comparison, we used the two versions that also call Gurobi 11.0.3: GRBbasic and GRBsbt[6]. For each task, we ran these two versions separately and reported the shorter runtime as the result of SCIP-MPNN. To align with the restrictions of SCIP-MPNN, we utilised GNNs with sum aggregation, only allowed edge deletions (i.e., $F = E$), and adapted our verification objective[7].

Results Figure 2 compares the verification runtime for the global budget $\Delta = 2$. For the 2-layer GNN on Cora (Figure 2a), while all tasks are quickly solvable, the maximum runtime for GNNEv is 6.2s, in contrast to 20.7s by SCIP-MPNN. Note that GNNEv solved 780 tasks within 0.1s, which can be attributed to determining `robust` in the first iteration and avoiding the overhead to process the entire encoding. We observe a similar trend for the 3-layer GNN on Cora (Figure 2b), and likewise CiteSeer (Figure 2c,2d), where GNNEv solved 5 challenging tasks on which SCIP-MPNN failed within the time limit (Figure 2d).

5.3 Performance on Broader Tasks

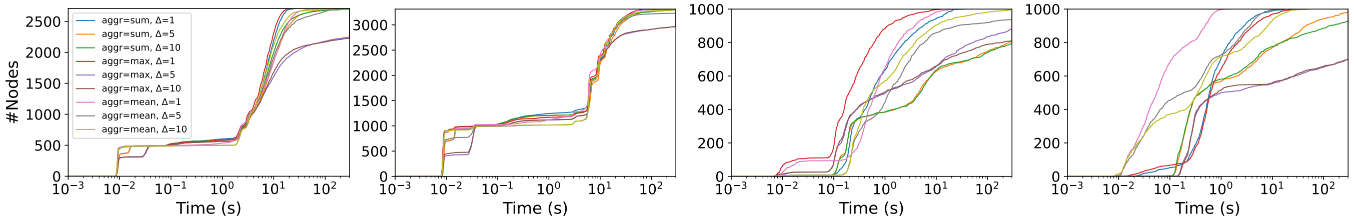
Next, we evaluated the performance of GNNEv on sum-, max- and mean-aggregated 3-layer GNNs. For this experiment we set the verification objective as Eq. (14), which is more challenging to compute than Hojny et al. (2024), Eq. (11). Due to the large number of nodes in Amazon and Yelp, we randomly selected 1,000 nodes for verification. Two fragile edge sets F were used: (i) only *edge deletions* are allowed for all edges, i.e., $F = E$, and (ii) only *edge additions* are allowed for selected edges, i.e., for each node v, a sampled non-edge $(u, v) \notin E$ is added to F. See Appendix E for more details.

Results Figure 3 illustrates the runtime of GNNEv under different aggregations and budgets for edge deletion, and Figure 4 in the Appendix for edge addition. First, GNNEv effectively verified most tasks within the time limit, even on Amazon and Yelp, which demonstrated its suitability to a broad range of tasks. Second, across different aggregation functions, there was a performance drop for max aggregation when $\Delta \geq 5$, likely due to the MIP solver struggling to efficiently process big-M encoding. Moreover, for sum aggregation, the performance on Amazon and Yelp dropped compared to Cora and CiteSeer. This can be attributed to the much larger gap between upper and lower bounds of variables for Amazon and Yelp, which can be observed from the statistics in Table 6 in the Appendix. This is most pronounced for sum aggregation, which increases the gap significantly after each layer. Third, we performed a systematic

[5] https://www.gurobi.com

[6] We excluded the three variants based on SCIP, noting that the experimental results in Hojny et al. (2024) showed that GRBbasic and GRBsbt have notably shorter runtime.

[7] According to Hojny et al. (2024), Eq. (11) and Section 4.2, given the predicted class $\hat{c}_t$ for target node t, SCIP-MPNN aims to verify that $\mathbf{h}_t^{(K)}[\hat{c}_t] > \mathbf{h}_t^{(K)}[(\hat{c}_t + 1)\%|C|]$ holds for all perturbed graphs. Their objective is simpler and more computationally tractable than ours stated in Eq. (14).



(a) Cora (b) CiteSeer (c) Amazon (d) Yelp

Figure 3: The number of tasks solved by GNNEV plotted against runtime under different aggregations and budgets. Only edge deletions are allowed, and all edges are set as fragile. The verification objective is as defined in Section 4.1, Eq. (14).

evaluation of the number of nodes verified plotted against an increasing global budget, shown in Figure 5,6 in the Appendix due to space constraints. An interesting finding is that mean-aggregated GNNs were more vulnerable to adversarial perturbations than other GNNs. This demonstrates that GNNEV is able to gain useful insight into the susceptibility of GNN models to adversarial attacks, which is of importance for pre-deployment analysis for high-stakes applications.

Finally, we conducted ablation experiments to show the effectiveness of bound tightening and incremental solving, with results detailed in Appendix E.

6 Conclusion and Future Work

We have developed an exact robustness verification method for message-passing GNNs supporting a range of aggregation functions, which is implemented in GNNEV, a versatile and efficient verifier. Our approach is based on a reduction to a constraint satisfaction problem with bound tightening, solved incrementally. The support for two common aggregation functions, max and mean, is formulated here for the first time. Experiments on node-classification GNNs show that GNNEV outperforms state-of-the-art baselines in both efficiency and functionality, and delivers strong performance on real-world fraud datasets. One practical limitation of our approach is that the computational complexity grows quickly with the size of the fragile edge set. Future work will include mitigating the impact of the fragile edge sets on performance by combining heuristic search and distributed frameworks, as well as providing support for graph classification tasks.

Acknowledgments

The authors would like to thank Michael Benedikt for the insightful comments and suggestions. This work was supported by the EPSRC Prosperity Partnership FAIR (grant number EP/V056883/1) and ELSA: European Lighthouse on Secure and Safe AI project (grant agreement No. 101070617 under UK guarantee). MK receives funding from the ERC under the European Union's Horizon 2020 research and innovation programme (FUN2MODEL, grant agreement No. 834115).

References

Akintunde, M. E.; Kevorchian, A.; Lomuscio, A.; and Pirovano, E. 2019. Verification of RNN-Based Neural Agent-Environment Systems. In *AAAI 2019*, 6006–6013.

An, D.; Zhang, H.; Zhao, Q.; Liu, J.; Shi, J.; Huang, Y.; Yang, Y.; Liu, X.; and Qin, S. 2024. Graph Convolutional Network Robustness Verification Algorithm Based on Dual Approximation. In *ICFEM 2024*, 146–161.

Barceló, P.; Kostylev, E. V.; Monet, M.; Pérez, J.; Reutter, J. L.; and Silva, J. P. 2020. The Logical Expressiveness of Graph Neural Networks. In *ICLR*.

Benedikt, M.; Lu, C.; Motik, B.; and Tan, T. 2024. Decidability of Graph Neural Networks via Logical Characterizations. In *ICALP 2024*, 127:1–127:20.

Bojchevski, A.; and Günnemann, S. 2019. Certifiable Robustness to Graph Perturbations. In *NeurIPS 2019*, 8317–8328.

Bojchevski, A.; Klicpera, J.; and Günnemann, S. 2020. Efficient Robustness Certificates for Discrete Data: Sparsity-Aware Randomized Smoothing for Graphs, Images and More. In *ICML 2020*, 1003–1013.

Bonaert, G.; Dimitrov, D. I.; Baader, M.; and Vechev, M. T. 2021. Fast and precise certification of transformers. In *PLDI 2021*, 466–481.

Boopathy, A.; Weng, T.; Chen, P.; Liu, S.; and Daniel, L. 2019. CNN-Cert: An Efficient Framework for Certifying Robustness of Convolutional Neural Networks. In *AAAI 2019*, 3240–3247.

Botoeva, E.; Kouvaros, P.; Kronqvist, J.; Lomuscio, A.; and Misener, R. 2020. Efficient Verification of ReLU-Based Neural Networks via Dependency Analysis. In *AAAI 2020*, 3291–3299.

Bronstein, M. M.; Bruna, J.; Cohen, T.; and Velickovic, P. 2021. Geometric Deep Learning: Grids, Groups, Graphs, Geodesics, and Gauges. *CoRR*, abs/2104.13478.

Bunel, R.; Turkaslan, I.; Torr, P. H. S.; Kohli, P.; and Mudigonda, P. K. 2018. A Unified View of Piecewise Linear Neural Network Verification. In *NeurIPS 2018*, 4795–4804.

Cai, P.; Wang, H.; Sun, Y.; and Liu, M. 2021. DiGNet: Learning Scalable Self-Driving Policies for Generic Traffic Scenarios with Graph Neural Networks. In *IROS 2021*, 8979–8984.

Casas, S.; Gulino, C.; Liao, R.; and Urtasun, R. 2020. SpAGNN: Spatially-Aware Graph Neural Networks for Relational Behavior Forecasting from Sensor Data. In *ICRA 2020*, 9491–9497.

Chang, H.; Rong, Y.; Xu, T.; Huang, W.; Zhang, H.; Cui, P.; Zhu, W.; and Huang, J. 2020. A Restricted Black-Box Adversarial Framework Towards Attacking Graph Embedding Models. In *AAAI 2020*, 3389–3396.

Chen, J.; Zhang, J.; Chen, Z.; Du, M.; and Xuan, Q. 2023. Time-Aware Gradient Attack on Dynamic Network Link Prediction. *IEEE Trans. Knowl. Data Eng.*, 35(2): 2091–2102.

Clarke, E. M.; and Wing, J. M. 1996. Formal Methods: State of the Art and Future Directions. *ACM Comput. Surv.*, 28(4): 626–643.

Corso, G.; Cavalleri, L.; Beaini, D.; Liò, P.; and Velickovic, P. 2020. Principal Neighbourhood Aggregation for Graph Nets. In *NeurIPS 2020*.

Dai, H.; Li, H.; Tian, T.; Huang, X.; Wang, L.; Zhu, J.; and Song, L. 2018. Adversarial Attack on Graph Structured Data. In *ICML 2018*, 1123–1132.

Dai, Q.; Shen, X.; Zhang, L.; Li, Q.; and Wang, D. 2019. Adversarial Training Methods for Network Embedding. In *WWW 2019*, 329–339.

Dehmamy, N.; Barabási, A.; and Yu, R. 2019. Understanding the Representation Power of Graph Neural Networks in Learning Graph Topology. In *NeurIPS 2019*, 15387–15397.

Du, T.; Ji, S.; Shen, L.; Zhang, Y.; Li, J.; Shi, J.; Fang, C.; Yin, J.; Beyah, R.; and Wang, T. 2021. Cert-RNN: Towards Certifying the Robustness of Recurrent Neural Networks. In *CCS 2021*, 516–534.

Feng, F.; He, X.; Tang, J.; and Chua, T. 2021. Graph Adversarial Training: Dynamically Regularizing Based on Graph Structure. *IEEE Trans. Knowl. Data Eng.*, 33(6): 2493–2504.

Fey, M.; and Lenssen, J. E. 2019. Fast Graph Representation Learning with PyTorch Geometric. *CoRR*, abs/1903.02428.

Gao, C.; Yin, S.; Wang, H.; Wang, Z.; Du, Z.; and Li, X. 2024. Medical-Knowledge-Based Graph Neural Network for Medication Combination Prediction. *IEEE Trans. Neural Networks Learn. Syst.*, 35(10): 13246–13257.

Geisler, S.; Schmidt, T.; Şirin, H.; Zügner, D.; Bojchevski, A.; and Günnemann, S. 2021. Robustness of Graph Neural Networks at Scale. In *NeurIPS 2021*.

Gilmer, J.; Schoenholz, S. S.; Riley, P. F.; Vinyals, O.; and Dahl, G. E. 2017. Neural Message Passing for Quantum Chemistry. In *ICML 2017*, 1263–1272.

Gosch, L.; Geisler, S.; Sturm, D.; Charpentier, B.; Zügner, D.; and Günnemann, S. 2023. Adversarial Training for Graph Neural Networks: Pitfalls, Solutions, and New Directions. In *NeurIPS 2023*.

Haghshenas, S. H.; Hasnat, A.; and Naeini, M. 2023. A Temporal Graph Neural Network for Cyber Attack Detection and Localization in Smart Grids. In *ISGT 2023*, 1–5.

Hamilton, W. L.; Ying, Z.; and Leskovec, J. 2017. Inductive Representation Learning on Large Graphs. In *NeurIPS 2017*, 1024–1034.

Hojny, C.; Zhang, S.; Campos, J. S.; and Misener, R. 2024. Verifying message-passing neural networks via topology-based bounds tightening. In *ICML 2024*.

Huang, X.; Kwiatkowska, M.; Wang, S.; and Wu, M. 2017. Safety Verification of Deep Neural Networks. In *CAV 2017*, 3–29.

Jha, K.; Saha, S.; and Singh, H. 2022. Prediction of protein–protein interaction using graph neural networks. *Scientific Reports*, 12(1): 8360.

Jin, H.; Shi, Z.; Peruri, V. J. S. A.; and Zhang, X. 2020. Certified Robustness of Graph Convolution Networks for Graph Classification under Topological Attacks. In *NeurIPS 2020*.

Katz, G.; Barrett, C. W.; Dill, D. L.; Julian, K.; and Kochenderfer, M. J. 2017. Reluplex: An Efficient SMT Solver for Verifying Deep Neural Networks. In *CAV 2017*, 97–117.

Kipf, T. N.; and Welling, M. 2017. Semi-Supervised Classification with Graph Convolutional Networks. In *ICLR 2017*.

Ko, C.; Lyu, Z.; Weng, L.; Daniel, L.; Wong, N.; and Lin, D. 2019. POPQORN: Quantifying Robustness of Recurrent Neural Networks. In *ICML 2019*, 3468–3477.

Ladner, T.; Eichelbeck, M.; and Althoff, M. 2025. Formal Verification of Graph Convolutional Networks with Uncertain Node Features and Uncertain Graph Structure. *Trans. Mach. Learn. Res.*, 2025.

Lai, Y.; Zhu, Y.; Pan, B.; and Zhou, K. 2024. Node-aware Bi-smoothing: Certified Robustness against Graph Injection Attacks. In *IEEE SP 2024*, 2958–2976.

Li, J.; Peng, J.; Chen, L.; Zheng, Z.; Liang, T.; and Ling, Q. 2023a. Spectral Adversarial Training for Robust Graph Neural Network. *IEEE Trans. Knowl. Data Eng.*, 35(9): 9240–9253.

Li, J.; Xie, T.; Chen, L.; Xie, F.; He, X.; and Zheng, Z. 2023b. Adversarial Attack on Large Scale Graph. *IEEE Trans. Knowl. Data Eng.*, 35(1): 82–95.

Li, M. M.; Huang, K.; and Zitnik, M. 2022. Graph representation learning in biomedicine and healthcare. *Nature Biomedical Engineering*, 6(12): 1353–1369.

Lu, J.; and Kumar, M. P. 2020. Neural Network Branching for Neural Network Verification. In *ICLR 2020*.

McAuley, J. J.; and Leskovec, J. 2013. From amateurs to connoisseurs: modeling the evolution of user expertise through online reviews. In *WWW 2013*, 897–908.

Motie, S.; and Raahemi, B. 2024. Financial fraud detection using graph neural networks: A systematic review. *Expert Syst. Appl.*, 240: 122156.

Mu, J.; Wang, B.; Li, Q.; Sun, K.; Xu, M.; and Liu, Z. 2021. A Hard Label Black-box Adversarial Attack Against Graph Neural Networks. In *CCS 2021*, 108–125.

Nunn, P.; Sälzer, M.; Schwarzentruber, F.; and Troquard, N. 2024. A Logic for Reasoning about Aggregate-Combine Graph Neural Networks. In *IJCAI 2024*, 3532–3540.

Osselin, P.; Kenlay, H.; and Dong, X. 2023. Structure-aware robustness certificates for graph classification. In *UAI 2023*, 1596–1605.

Rayana, S.; and Akoglu, L. 2015. Collective Opinion Spam Detection: Bridging Review Networks and Metadata. In *KDD 2015*, 985–994.

Reiser, P.; Neubert, M.; Eberhard, A.; Torresi, L.; Zhou, C.; Shao, C.; Metni, H.; van Hoesel, C.; Schopmans, H.; Sommer, T.; and Friederich, P. 2022. Graph neural networks for materials science and chemistry. *Communications Materials*, 3(1): 93.

Rosenbluth, E.; Tönshoff, J.; and Grohe, M. 2023. Some Might Say All You Need Is Sum. In *IJCAI 2023*, 4172–4179.

Rossi, F.; Van Beek, P.; and Walsh, T. 2006. *Handbook of Constraint Programming*. Elsevier.

Salman, H.; Yang, G.; Zhang, H.; Hsieh, C.; and Zhang, P. 2019. A Convex Relaxation Barrier to Tight Robustness Verification of Neural Networks. In *NeurIPS 2019*, 9832–9842.

Sälzer, M.; and Lange, M. 2023. Fundamental Limits in Formal Verification of Message-Passing Neural Networks. In *ICLR 2023*.

Scholten, Y.; Schuchardt, J.; Geisler, S.; Bojchevski, A.; and Günnemann, S. 2022. Randomized Message-Interception Smoothing: Gray-box Certificates for Graph Neural Networks. In *NeurIPS 2022*.

Schönherr, M.; and Lutz, C. 2025. Logical Characterizations of GNNs with Mean Aggregation. arXiv:2507.18145.

Sen, P.; Namata, G.; Bilgic, M.; Getoor, L.; Gallagher, B.; and Eliassi-Rad, T. 2008. Collective Classification in Network Data. *AI Mag.*, 29(3): 93–106.

Shi, Z.; Zhang, H.; Chang, K.; Huang, M.; and Hsieh, C. 2020. Robustness Verification for Transformers. In *ICLR 2020*.

Tao, S.; Cao, Q.; Shen, H.; Huang, J.; Wu, Y.; and Cheng, X. 2021. Single Node Injection Attack against Graph Neural Networks. In *CIKM 2021*, 1794–1803.

Tjeng, V.; Xiao, K. Y.; and Tedrake, R. 2019. Evaluating Robustness of Neural Networks with Mixed Integer Programming. In *ICLR 2019*.

Tran, H.; Bak, S.; Xiang, W.; and Johnson, T. T. 2020. Verification of Deep Convolutional Neural Networks Using ImageStars. In *CAV 2020*, 18–42.

Wang, B.; Jia, J.; Cao, X.; and Gong, N. Z. 2021. Certified Robustness of Graph Neural Networks against Adversarial Structural Perturbation. In *KDD 2021*, 1645–1653.

Wang, D.; Qi, Y.; Lin, J.; Cui, P.; Jia, Q.; Wang, Z.; Fang, Y.; Yu, Q.; Zhou, J.; and Yang, S. 2019. A Semi-Supervised Graph Attentive Network for Financial Fraud Detection. In *ICDM 2019*, 598–607.

Wang, S.; Pei, K.; Whitehouse, J.; Yang, J.; and Jana, S. 2018. Efficient Formal Safety Analysis of Neural Networks. In *NeurIPS 2018*, 6369–6379.

Weng, T.; Zhang, H.; Chen, H.; Song, Z.; Hsieh, C.; Daniel, L.; Boning, D. S.; and Dhillon, I. S. 2018. Towards Fast Computation of Certified Robustness for ReLU Networks. In *ICML 2018*, 5273–5282.

Wu, H.; Wang, C.; Tyshetskiy, Y.; Docherty, A.; Lu, K.; and Zhu, L. 2019. Adversarial Examples for Graph Data: Deep Insights into Attack and Defense. In *IJCAI 2019*, 4816–4823.

Xia, Z.; Yang, H.; Wang, B.; and Jia, J. 2024. GNNCert: Deterministic Certification of Graph Neural Networks against Adversarial Perturbations. In *ICLR 2024*.

Xu, K.; Chen, H.; Liu, S.; Chen, P.; Weng, T.; Hong, M.; and Lin, X. 2019. Topology Attack and Defense for Graph Neural Networks: An Optimization Perspective. In *IJCAI 2019*, 3961–3967.

Xu, K.; Shi, Z.; Zhang, H.; Wang, Y.; Chang, K.; Huang, M.; Kailkhura, B.; Lin, X.; and Hsieh, C. 2020. Automatic Perturbation Analysis for Scalable Certified Robustness and Beyond. In *NeurIPS 2020*.

Ying, R.; He, R.; Chen, K.; Eksombatchai, P.; Hamilton, W. L.; and Leskovec, J. 2018. Graph Convolutional Neural Networks for Web-Scale Recommender Systems. In *KDD 2018*, 974–983.

Zhang, H.; Weng, T.; Chen, P.; Hsieh, C.; and Daniel, L. 2018. Efficient Neural Network Robustness Certification with General Activation Functions. In *NeurIPS 2018*, 4944–4953.

Zhang, S.; Campos, J. S.; Feldmann, C.; Walz, D.; Sandfort, F.; Mathea, M.; Tsay, C.; and Misener, R. 2023. Optimizing over trained GNNs via symmetry breaking. In *NeurIPS 2023*.

Zhang, X.; and Zitnik, M. 2020. GNNGuard: Defending Graph Neural Networks against Adversarial Attacks. In *NeurIPS 2020*.

Zhang, Z.; Jia, J.; Wang, B.; and Gong, N. Z. 2021. Backdoor Attacks to Graph Neural Networks. In *SACMAT 2021*, 15–26.

Zhou, K.; Michalak, T. P.; Waniek, M.; Rahwan, T.; and Vorobeychik, Y. 2019. Attacking Similarity-Based Link Prediction in Social Networks. In *AAMAS 2019*, 305–313.

Zou, X.; Zheng, Q.; Dong, Y.; Guan, X.; Kharlamov, E.; Lu, J.; and Tang, J. 2021. TDGIA: Effective Injection Attacks on Graph Neural Networks. In *KDD 2021*, 2461–2471.

Zügner, D.; Akbarnejad, A.; and Günnemann, S. 2018. Adversarial Attacks on Neural Networks for Graph Data. In *KDD 2018*, 2847–2856.

Zügner, D.; and Günnemann, S. 2019a. Adversarial Attacks on Graph Neural Networks via Meta Learning. In *ICLR 2019*.

Zügner, D.; and Günnemann, S. 2019b. Certifiable Robustness and Robust Training for Graph Convolutional Networks. In *KDD 2019*, 246–256.

Zügner, D.; and Günnemann, S. 2020. Certifiable Robustness of Graph Convolutional Networks under Structure Perturbations. In *KDD 2020*, 1656–1665.

A Extension to Graph Classification

While our paper focuses on node classification tasks, it is straightforward to extend our method to graph classification. The goal of graph classification is to classify an input attributed graph $G = \langle V, E, X \rangle$ into one class in $C = \{c_1, c_2, \ldots, c_m\}$. For graph classification GNNs, the process to compute the embedding vectors is the same as for node classification GNNs, as shown in Eq. (1) in the main paper. The difference arises from how the predicted class is obtained, namely for graph classification we have:

$$\hat{c} = \arg\max_{c \in C} \left(\text{Softmax} \left(\mathbf{W}_3 \cdot \sum_{v \in V} \mathbf{h}_v^{(K)} \right) [c] \right), \quad (15)$$

where $\mathbf{W}_3$ is a learnable parameter matrix with dimension $m \times d_K$. A graph-classification GNN f is called adversarially robust if and only if, for every perturbed $\tilde{G} \in \mathcal{Q}(G)$, it holds that $f(\tilde{G}) = \hat{c}$.

Our method can be extended to graph classification in the following manner. First, we simply need to make slight modifications to the last part of the CSP encoding to align with the operation in Eq. (15), which includes a summation of all the nodes and a linear transformation. Second, we remark that our bound tightening strategies are compatible with the modified encoding. Third, we note that our incremental solving-based verification algorithm can also be used directly. Although graph classification requires encoding of all the nodes in the graph, resulting in the same number of nodes in each GNN layer, our algorithm only encodes variables associated with the $(K-k)$-th to K-th GNN layers in k-th iteration, potentially reducing encoding costs and enhancing efficiency. We intend to provide support for graph classification as future work.

B Proof of the Tightness of Bounds

Recall the problem:

Given three finite sets of variables X_1, X_2, and X_3, the upper and lower bounds for each variable, and a non-negative integer s, compute the upper and lower bounds for the variable

$$z := \mathbf{aggr}(X_1 \cup X_2' \cup X_3'),$$

where $\mathbf{aggr} \in \{\text{sum}, \max, \text{mean}\}$, $X_2' \subseteq X_2$, $X_3' \subseteq X_3$, and $|X_2 \backslash X_2'| + |X_3'| \leq s$.

Max aggregation In this case, the problem is straightforward through case-by-case analysis. The only subtlety is when X_1 is empty and $s \geq |X_2|$, in which case we may have $X_2' = X_3' = \emptyset$ and thus $X_1 \cup X_2' \cup X_3' = \emptyset$, which implies that the corresponding upper and lower bounds are 0. Therefore, we must take 0 into consideration.

Mean aggregation In this case, recall that we reduce the original problem to a combination of subproblems. For non-negative integers s_2 and s_3, define the variable

$$z_{s_2,s_3} := \mathbf{mean}(X_1 \cup X_2' \cup X_3'),$$

where $X_2' \subseteq X_2$, $X_3' \subseteq X_3$, $|X_2 \backslash X_2'| = s_2$, and $|X_3'| = s_3$. The upper and lower bounds of the original problem z are given by

$$\overline{z} = \max_{\substack{s_2+s_3 \leq s \\ 0 \leq s_2 \leq |X_2| \\ 0 \leq s_3 \leq |X_3|}} \overline{z}_{s_2,s_3} \quad \text{and} \quad \underline{z} = \min_{\substack{s_2+s_3 \leq s \\ 0 \leq s_2 \leq |X_2| \\ 0 \leq s_3 \leq |X_3|}} \underline{z}_{s_2,s_3}. \quad (16)$$

Note that Eq. (16) covers all possible allocations of budgets (s_2, s_3) such that $s_2 + s_3 \leq s$. The correctness of this reduction is obvious.

Next, we show that the upper and lower bounds of the variable z_{s_2,s_3} can be computed efficiently. Here we focus on the upper bound; a similar argument applies to the lower bound.

Let (s_2, s_3) be a fixed pair of non-negative integers satisfying $s_2 + s_3 \leq s$, $0 \leq s_2 \leq |X_2|$, and $0 \leq s_3 \leq |X_3|$. Let

$$\begin{aligned} n_{s_2,s_3} &:= |X_1| + |X_2'| + |X_3'| \\ &= |X_1| + |X_2| - s_2 + s_3. \end{aligned}$$

Note that

$$\begin{aligned} z_{s_2,s_3} &= \mathbf{mean}(X_1 \cup X_2' \cup X_3') \\ &= \frac{1}{n_{s_2,s_3}} \cdot \mathbf{sum}(X_1 \cup X_2' \cup X_3') \\ &= \frac{1}{n_{s_2,s_3}} \cdot (\mathbf{sum}(X_1) + \mathbf{sum}(X_2') + \mathbf{sum}(X_3')), \end{aligned}$$

Because the values of s_2 and s_3 are fixed, the selections of $X_2' \subseteq X_2$ and $X_3' \subseteq X_3$ are independent. Thus, the upper bound of z_{s_2,s_3} is given by the sum of upper bounds of $\mathbf{sum}(X_1)$, $\mathbf{sum}(X_2')$, and $\mathbf{sum}(X_3')$, divided by n_{s_2,s_3} where $X_2' \subseteq X_2$, $X_3' \subseteq X_3$, $|X_2 \backslash X_2'| = s_2$, and $|X_3'| = s_3$.

Observe that, for a set of variables Y, because the sum function is monotonic w.r.t. each variable, the upper bound of the sum of the set Y is given by the sum of the upper bounds of the variables in Y. Therefore, we have the following conclusions:

- The upper bound of $\mathbf{sum}(X_1)$ is $\sum_{x \in X_1} \overline{x}$.
- For $\mathbf{sum}(X_2')$ where $X_2' \subseteq X_2$ and $|X_2 \backslash X_2'| = s_2$. Since $|X_2'| = |X_2| - s_2$, the upper bound of $\mathbf{sum}(X_2')$ is given by the sum of the $(|X_2| - s_2)$-largest upper bounds in X_2, that is, $\sum_{1 \leq i \leq |X_2| - s_2} hi(X_2, i)$.
- Similarly, since $X_3' \subseteq X_3$ and $|X_3'| = s_3$, the upper bound of $\mathbf{sum}(X_3')$ is $\sum_{1 \leq i \leq s_3} hi(X_3, i)$.

Thus $\overline{z}_{s_2,s_3}$ is upper bounded by

$$\frac{1}{n_{s_2,s_3}} \left(\sum_{x \in X_1} \overline{x} + \sum_{1 \leq i \leq |X_2| - s_2} hi(X_2, i) + \sum_{1 \leq i \leq s_3} hi(X_3, i) \right).$$

Note that the bound is tight. That is, there exist sets $X_2' \subseteq X_2$ and $X_3' \subseteq X_3$ with $|X_2 \backslash X_2'| = s_2$ and $|X_3'| = s_3$ that achieve this bound exactly.

C Proof of Theorem 1

Recall the theorem:

Given a GNN f, an attributed directed graph G, and a perturbation space $\mathcal{Q}(G)$, f is adversarially robust for a node t if and only if Algorithm 1 returns `robust`.

Proof. For $1 \leq k \leq K$, let $\Phi_k := \varphi_{obj} \wedge \bigwedge_{1 \leq i \leq k} \varphi_i$. Recall that φ_k is the encoding of the computation of the k-th layer of the GNN f. Following the intuition described in Section 4.1, it is straightforward to show that f is adversarially robust for t if and only if Φ_K is unsatisfiable.

If Algorithm 1 returns `nonrobust`, then Φ_K is satisfiable. Hence, f is not adversarially robust for t.

On the other hand, observe that for $1 \leq k \leq K$,

$$\Phi_K = \Phi_k \wedge \bigwedge_{k < i \leq K} \varphi_i.$$

If f is not adversarially robust for t, then Φ_K is satisfiable, which implies that Φ_k is also satisfiable for $1 \leq k \leq K$. Therefore, Algorithm 1 returns `nonrobust`. □

D Model Implementation and Training

Implementation To evaluate the performance of verifiers, we implemented a batch of GraphSAGE models built using PyTorch and the SAGEConv module of PyTorch Geometric. The aggregation function of SAGEConv was configured as sum, max, and mean, respectively, to build distinct models. The models consist of 2 or 3 SAGEConv layers (i.e., $K \in \{2, 3\}$), with input dimension being the dimension of attributes, all hidden layers having a dimension of 32, and output dimension being the number of classes. Except for the last layer, the output of each layer is passed through a ReLU activation function. Finally, a Softmax function is applied to the output of the last layer to obtain predictions for each class.

Datasets We trained models on 4 node classification datasets. Cora and CiteSeer are two standard graph benchmarks that have been employed for evaluating verification algorithms in a number of previous works (Bojchevski and Günnemann 2019; Wang et al. 2021; Scholten et al. 2022; Hojny et al. 2024). Amazon and Yelp are two Internet fraud datasets, and we employed them to evaluate the performance of our algorithm on real-world high-stakes tasks. Because Amazon and Yelp are heterogeneous graphs with three types of edges, we used only the U-P-U edges for Amazon and R-U-R edges for Yelp to ensure compatibility with our GNN architectures. Detailed statistical information of these datasets is shown in Table 1.

Dataset	#Nodes	#Edges	#Attributes	#Classes
Cora	2,708	5,429	1,433	7
CiteSeer	3,312	4,715	3,703	6
Amazon	11,944	351,216	25	2
Yelp	45,954	98,630	32	2

Table 1: Statistical information of datasets.

Training For Cora and Citeseer, we randomly selected 30% of the nodes as the training set, 20% as the validation set, and the remaining 50% as the test set. For Amazon and Yelp, given the highly imbalanced distribution of the two classes, we randomly selected 350 (from Amazon) and 2,000 (from Yelp) nodes per class as the training set, 70 (from Amazon) and 400 (from Yelp) nodes per class as the validation set, and the remaining nodes as the test set. All models were trained for 400 epochs with learning rate 0.001 and weight decay 5×10^{-5}, and early stopping is applied if the validation loss exceeded the average of the last 10 epochs after training for more than 200 epochs. The accuracy of the models on test sets is demonstrated in Table 2.

K	**aggr**	Cora	CiteSeer	Amazon	Yelp
2	sum	80.87%	73.19%	95.96%	79.04%
	max	80.65%	71.92%	96.31%	78.39%
	mean	82.05%	72.89%	96.41%	79.14%
3	sum	81.02%	71.68%	95.97%	79.12%
	max	80.35%	72.40%	96.19%	79.02%
	mean	81.39%	72.58%	97.55%	80.52%

Table 2: Accuracy of the trained models on test sets.

E Comprehensive Experimental Results

Comparison with SCIP-MPNN

Table 3 and Table 4 show detailed comparison results of GNNEV and SCIP-MPNN under different global structural perturbation budgets Δ. The average runtime, the number of solved tasks, and the number of winning tasks (i.e., when the runtime of the current verifier is shorter than other verifier(s) on that task) for each verifier are listed for the set of all instances and the set of robust instances, respectively. Across models with varying layer depth and under different budgets, GNNEV consistently outperformed SCIP-MPNN in terms of the number of winning tasks, especially on the robust instances.

Verification Results and Performance

We evaluated the performance of GNNEV on 3-layer GNNs. Due to the large number of nodes in Amazon and Yelp, we sampled 1,000 nodes from each dataset to form the instances for verification. Initially, we selected nodes, for which the size of K-hop incoming neighbours set was within the range [2, 200] and which were correctly predicted by the model. Next, if the number of nodes in a class was less than 500, all of them were selected, and nodes in another class were uniformly sampled to make the instances up to 1,000; otherwise, 500 nodes were uniformly sampled from each class.

Figure 4 illustrates the runtime of GNNEV under different aggregation functions and budgets for edge addition. GNNev successfully solved all tasks within the time limit for Cora, Amazon, and Yelp (Figure 4a,4c,4d). For CiteSeer, there were 2 ($\Delta = 1$), 78 ($\Delta = 5$) and 152 ($\Delta = 10$) tasks left unsolved with max aggregation, indicating that the difficulty of the problem increases with larger budgets (Figure 4b). Since the fragile edge set F only included one non-edge for each node, the number of edges that can be perturbed in each task is much lower for edge addition, which makes the corresponding CSP easier to solve compared with edge deletion. Note that setting F to a large size can lead to a notable decrease in verification efficiency.

Figure 5 and Figure 6 show the evolution of the number of robust and nonrobust nodes verified by GNNEV as the perturbation budget increases. We observed that the robustness of mean-aggregated GNNs is significantly lower compared to other GNNs, on Cora, CiteSeer and Amazon for edge deletion, and on Amazon and Yelp for edge addition. This indicates that GNNEV can reveal the robustness flaws of GNN models, which is beneficial for deploying these models in high-stakes applications. Furthermore, for most nodes, the transition from robust to nonrobust occurred when $\Delta \leq 5$, which also demonstrates the vulnerability of GNN models to adversarial attacks.

Ablation Studies

We evaluated the effectiveness of the proposed incremental solving algorithm. A variant of GNNEV was developed, where the incremental solving algorithm was replaced with a one-time solution of the complete encoding. We refer to this variant as GNNEV w/o IS. The experiments were conducted on 3-layer GNNs, with perturbation budget $\Delta = 2$ and fragile edge set $F = E$.

The results of the comparative experiments are shown in Table 5. The number of winning tasks and the average runtime on Cora and CiteSeer are reported. Compared to its variant without incremental solving, GNNEV demonstrated lower average runtime and was more competitive on more tasks. Specifically, on robust instances, the incremental solving algorithm lowers the runtime of up to 82% of tasks (on CiteSeer, mean aggregation). These results indicate that the proposed incremental solving algorithm effectively enhances the practical performance of our exact verifier, offering the greatest advantage on robust instances, which is consistent with its design.

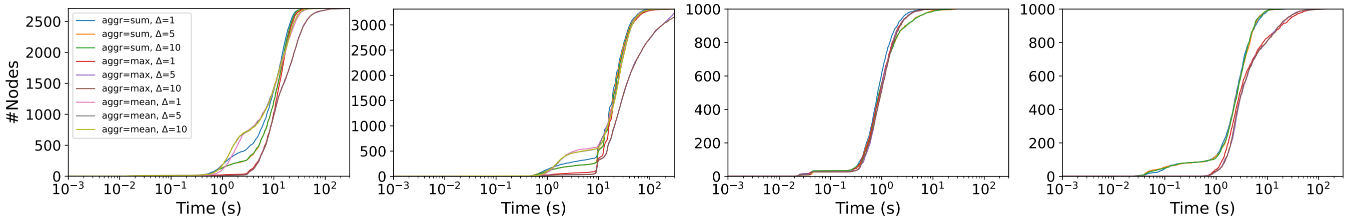
We have also shown the effectiveness of the proposed bound tightening strategies. Table 6 summarizes the mean and maximum gaps between the upper and lower bounds of the node embedding variables $y_{v,i}^{(k)}$ defined by Eq. (12) across the 4 datasets. The experiments were conducted on 3-layer GNNs, with perturbation budget $\Delta = 1$ and fragile edges set $F = E$. Compared with the plain method as described in Section 4.2, our tightened bounds achieved lower mean and maximum gaps for all aggregations and layers k. Noting that the sum aggregation resulted in significantly larger gaps than mean and max, leading to performance drop in the solver.

K	Δ	SCIP-MPNN						GNNEV					
		All instances			Robust instances			All instances			Robust instances		
		Time(s)	#Solved	#Win	Time(s)	#Solved	#Win	Time(s)	#Solved	#Win	Time(s)	#Solved	#Win
2	1	2.47	2,708	1,302	2.37	2,584	1,206	2.49	2,708	**1,406**	2.36	2,584	**1,378**
	2	2.48	2,708	1,203	2.22	2,473	1,046	2.29	2,708	**1,505**	2.04	2,473	**1,427**
	5	2.46	2,708	1,285	2.15	2,335	1,161	2.71	2,708	**1,423**	1.96	2,335	**1,174**
	10	2.48	2,708	1,278	2.15	2,328	**1,165**	2.72	2,708	**1,430**	1.95	2,328	1,163
	50	2.47	2,708	1,271	2.15	2,328	1,157	2.72	2,708	**1,437**	1.95	2,328	**1,171**
3	1	13.84	2,708	73	13.06	2,544	59	5.06	2,708	**2,635**	4.72	2,544	**2,485**
	2	13.86	2,708	58	12.15	2,404	45	5.12	2,708	**2,650**	4.45	2,404	**2,359**
	5	13.79	2,708	132	10.32	2,259	104	6.92	2,708	**2,576**	5.39	2,259	**2,155**
	10	13.87	2,706	162	10.25	2,249	136	8.10	2,708	**2,546**	6.53	2,251	**2,115**
	50	13.65	2,706	178	10.08	2,249	148	7.79	2,708	**2,530**	6.30	2,251	**2,103**

Table 3: The detailed comparison results of GNNEV and SCIP-MPNN on Cora for GNNs of varying depth and a range of perturbation budgets. The #Win column indicates the number of winning tasks (i.e. when the runtime of the current verifier is shorter than other verifier(s) on that task), and the larger number on the same set of instances is shown in **bold**.

K	Δ	SCIP-MPNN						GNNEV					
		All instances			Robust instances			All instances			Robust instances		
		Time(s)	#Solved	#Win	Time(s)	#Solved	#Win	Time(s)	#Solved	#Win	Time(s)	#Solved	#Win
2	1	5.92	3,312	1,358	5.66	3,174	1,244	5.14	3,312	**1,954**	4.81	3,174	**1,930**
	2	6.01	3,311	1,272	5.52	3,099	1,125	4.49	3,312	**2,040**	4.01	3,100	**1,975**
	5	6.02	3,311	1,430	5.14	3,035	1,237	5.31	3,312	**1,882**	4.44	3,036	**1,799**
	10	5.92	3,311	1,433	4.89	3,019	1,239	5.30	3,312	**1,879**	4.31	3,020	**1,781**
	50	5.91	3,311	1,439	4.72	3,016	1,241	5.31	3,312	**1,873**	4.27	3,017	**1,776**
3	1	9.59	3,308	1,126	9.12	3,156	1,009	6.14	3,312	**2,186**	5.70	3,160	**2,151**
	2	9.59	3,307	1,300	8.77	3,076	1,134	6.84	3,312	**2,012**	6.08	3,081	**1,947**
	5	9.63	3,307	1,550	8.04	3,010	1,327	9.07	3,308	**1,759**	7.56	3,010	**1,686**
	10	10.26	3,307	1,548	8.43	2,985	1,319	9.51	3,308	**1,761**	8.06	2,990	**1,672**
	50	10.37	3,305	1,561	7.52	2,978	1,328	9.41	3,295	**1,750**	7.14	2,980	**1,654**

Table 4: The detailed comparison results of GNNEV and SCIP-MPNN on CiteSeer for GNNs of varying depth and a range of perturbation budgets. The #Win column indicates the number of winning tasks (i.e. when the runtime of the current verifier is shorter than other verifier(s) on that task), and the larger number on the same set of instances is shown in **bold**.



(a) Cora (b) CiteSeer (c) Amazon (d) Yelp

Figure 4: The number of tasks solved by GNNEV plotted against runtime under different aggregations and budgets. Only edge additions are allowed, and one non-edge to each node is selected as fragile.

Dataset	aggr	GNNEV		GNNEV w/o IS	
		#Win	Time(s)	#Win	Time(s)
All instances					
Cora	sum	**1,743**	**7.84**	965	9.21
	max	1,313	**19.81**	**1,383**	20.57
	mean	**1,532**	**32.06**	1,176	34.88
CiteSeer	sum	**2,491**	**10.09**	821	13.35
	max	**1,989**	**28.47**	1,222	29.82
	mean	**1,759**	**15.15**	1,553	17.41
Robust instances					
Cora	sum	**1,467**	**5.02**	476	6.49
	max	**1,084**	**11.87**	855	12.68
	mean	**931**	**20.33**	269	23.03
CiteSeer	sum	**2,217**	**7.34**	499	8.73
	max	**1,767**	**7.33**	778	8.72
	mean	**1,257**	**7.75**	273	10.09

Table 5: Comparative experiments on the incremental solving algorithm of GNNEV. The relatively better results are shown in **bold**.

aggr	k	Tightened Bounds		Plain Bounds	
		Mean gap	Max gap	Mean gap	Max gap
Cora					
sum	1	0.58	2.66	0.67	3.42
	2	7.46	38.26	9.29	50.58
	3	91.94	472.87	104.66	566.26
max	1	0.57	3.28	0.66	3.75
	2	5.25	17.12	6.42	22.10
	3	50.94	130.73	58.00	151.40
mean	1	0.42	2.35	0.73	3.68
	2	4.17	11.82	7.30	22.83
	3	41.60	93.48	65.96	162.08
CiteSeer					
sum	1	0.46	6.13	0.55	14.36
	2	4.99	97.94	6.40	189.29
	3	51.57	1,002.33	59.86	1,354.72
max	1	0.46	4.00	0.53	7.41
	2	3.70	23.50	4.55	37.25
	3	32.53	144.33	37.55	198.76
mean	1	0.34	2.00	0.51	7.66
	2	2.91	10.39	4.35	38.81
	3	25.83	77.49	35.81	203.99
Amazon					
sum	1	46.29	993.37	144.50	21,387.24
	2	391.64	14,691.80	562.75	25,208.20
	3	2,008.59	37,071.13	2,678.37	60,287.40
max	1	40.71	991.43	74.81	1,317.29
	2	101.59	1,626.05	134.98	2,378.28
	3	525.86	4,350.96	611.23	4,819.04
mean	1	19.53	1,025.55	68.98	1,126.02
	2	54.64	1,681.41	143.64	1,921.16
	3	226.63	3,832.60	466.67	4,021.53
Yelp					
sum	1	1.80	6.76	11.16	182.42
	2	28.76	387.72	391.38	17,145.60
	3	711.55	34,765.66	7,880.43	993,569.36
max	1	0.66	3.69	2.37	4.51
	2	5.42	27.59	17.85	38.48
	3	111.11	205.89	191.89	280.45
mean	1	0.09	2.11	0.43	2.49
	2	0.38	5.14	2.04	6.42
	3	12.45	48.56	34.62	54.53

Table 6: Comparison of the gaps between the upper and lower bounds of node embedding variables for the tightened and plain bound propagation methods.

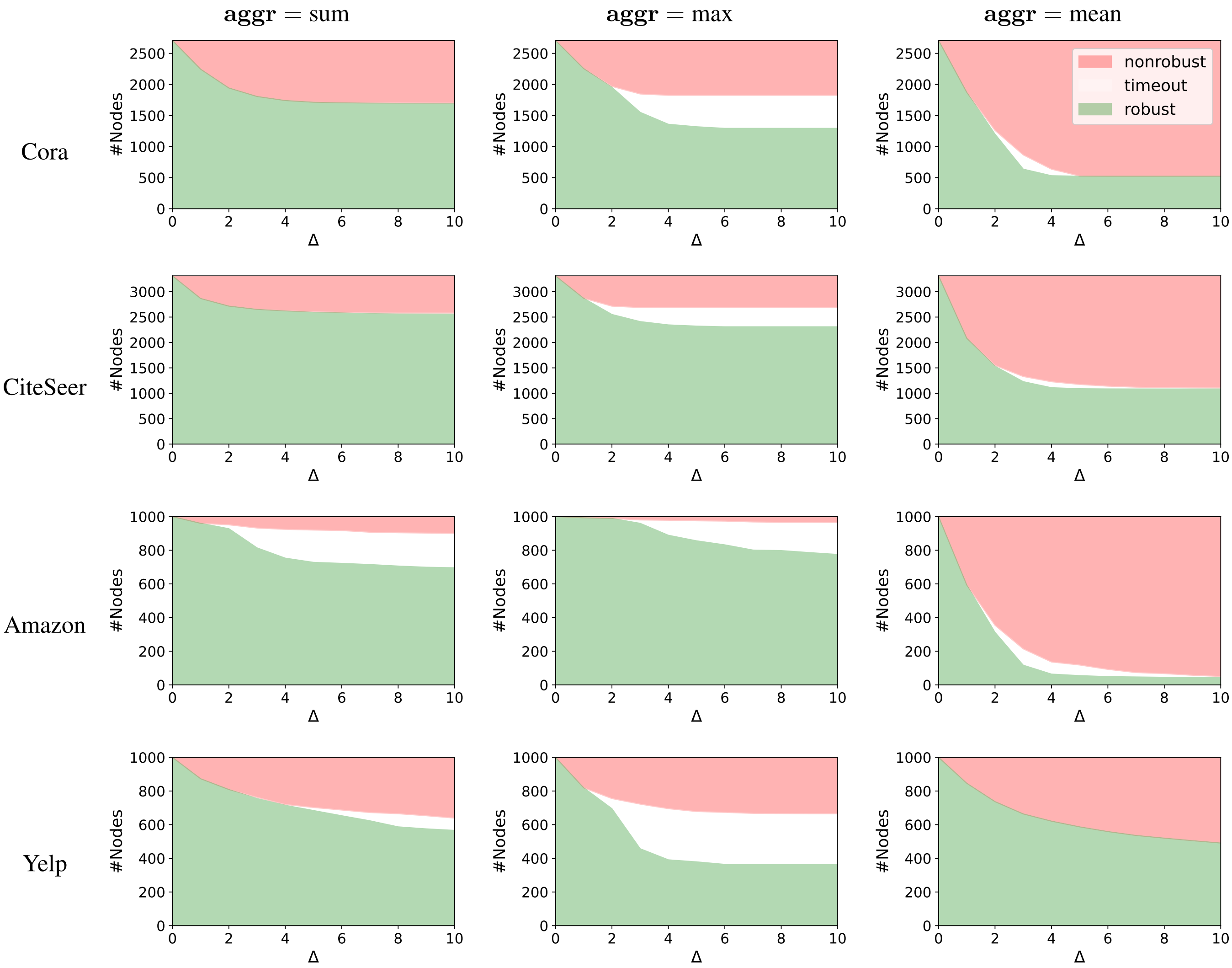


Figure 5: The evolution of the number of robust and nonrobust nodes verified by GNNEV as the global structural perturbation budget Δ increases. Only edge deletions are allowed, and all edges are set as fragile.

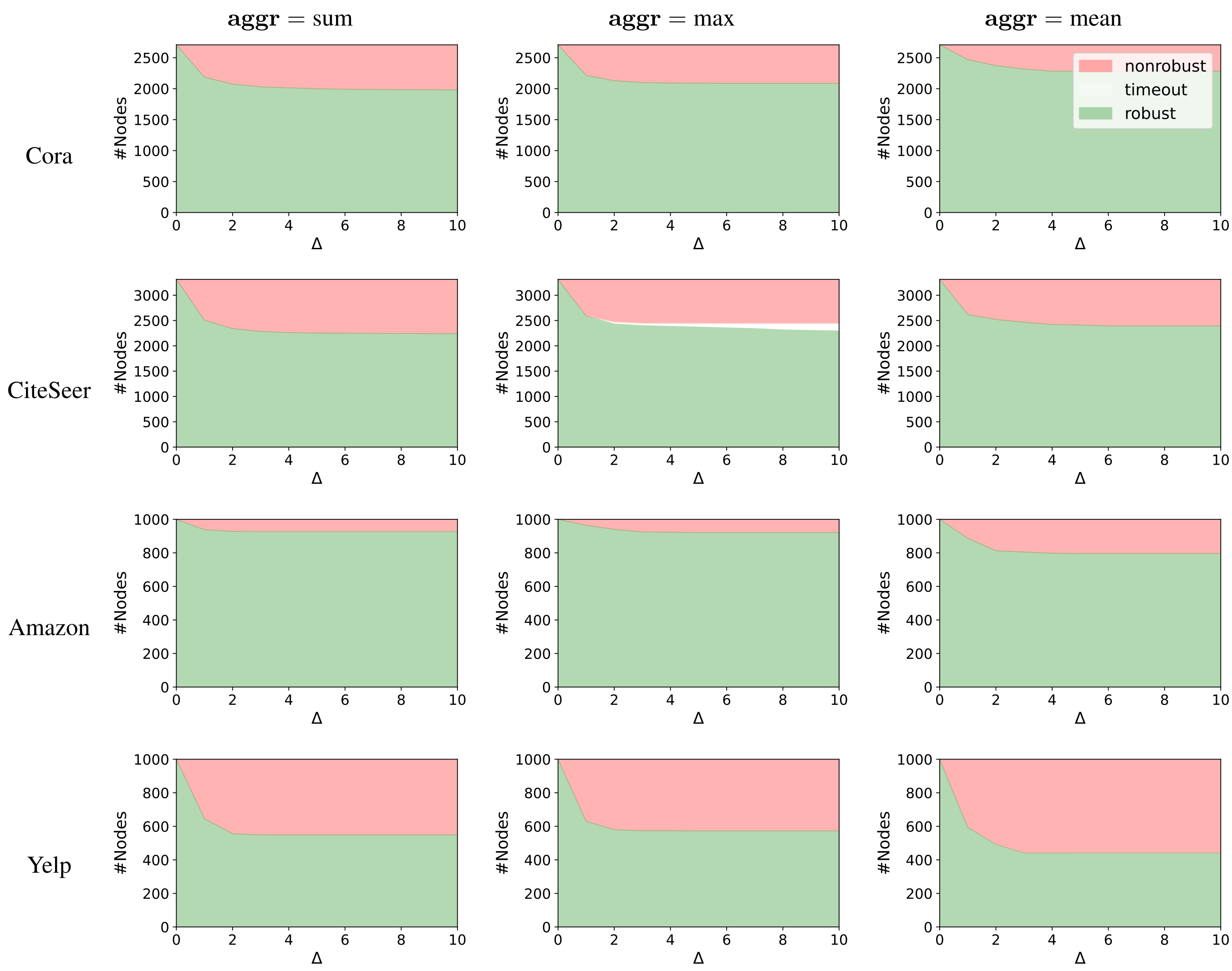


Figure 6: The evolution of the number of robust and nonrobust nodes verified by GNNEV as the global structural perturbation budget Δ increases. Only edge additions are allowed, and one non-edge to each node is selected as fragile.