General Cutting Planes for Bound-Propagation-Based Neural Network Verification

Anonymous Author(s)
Affiliation
Address
email

Abstract

Bound propagation methods, when combined with branch and bound, are among the most effective methods to formally verify properties of deep neural networks such as correctness, robustness, and safety. However, existing works cannot handle the general form of cutting plane constraints widely accepted in traditional solvers, which are crucial for strengthening verifiers with tightened convex relaxations. In this paper, we generalize the bound propagation procedure to allow the addition of arbitrary cutting plane constraints, including those involving relaxed integer variables that do not appear in existing bound propagation formulations. Our generalized bound propagation method, GCP-CROWN, opens up the opportunity to apply ***g**eneral **c**utting **p**lane methods* for neural network verification while benefiting from the efficiency and GPU acceleration of bound propagation methods. As a case study, we investigate the use of cutting planes generated by off-the-shelf mixed integer programming (MIP) solver. We find that MIP solvers can generate high-quality cutting planes for strengthening bound-propagation-based verifiers using our new formulation. Since the branching-focused bound propagation procedure and the cutting-plane-focused MIP solver can run in parallel utilizing different types of hardware (GPUs and CPUs), their combination can quickly explore a large number of branches with strong cutting planes, leading to strong verification performance. Experiments demonstrate that our method is the first verifier that can *completely* solve the `oval20` benchmark, and can verify *twice as many* instances on the `oval21` benchmark compared to the best tool in VNN-COMP 2021, and also noticeably outperforms state-of-the-art verifiers on a wide range of benchmarks.

1 Introduction

Neural network (NN) verification aims to formally prove or disprove certain properties (e.g., correctness and safety properties) of a NN under a certain set of inputs. These methods can provide worst-case performance guarantees of a NN, and have been applied to mission-critical applications that involve neural networks, such as automatic aircraft control [31, 4], learning-enabled cyber-physical systems [54], and NN based algorithms in an operating system [51].

The NN verification problem is generally NP-complete [30]. For piece-wise linear networks, it can be encoded as a mixed integer programming (MIP) [53] problem with the non-linear ReLU neurons described by binary variables. Thus, fundamentally, the NN verification problem can be solved using the branch and bound (BaB) [10] method similar to generic MIP solvers, by branching some binary variables and relaxing the rest into a convex problem such as linear programming (LP) to obtain bounds on the objective. Although early neural network verifiers relied on off-the-shelf CPU-based LP solvers [36, 9] for bounding in BaB, LP solvers do not scale well to large NNs. Thus, many recent verifiers are instead based on efficient and GPU-accelerated algorithms customized to NN verification, such as bound propagation methods [60, 57], Lagrangian decomposition methods [8, 17]

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

and others [16, 11]. Bound propagation methods, presented in a few different formulations [58, 18, 56, 61, 50, 25], empower state-of-the-art NN verifiers such as α,β-CROWN [61, 60, 57] and VeriNet [3], and can achieve two to three orders of magnitudes speedup compared to solving the NN verification problem using an off-the-shelf solver directly [57], especially on large networks.

Despite the success of existing NN verifiers, we experimentally find that state-of-the-art NN verifiers may timeout on certain hard instances which a generic MIP solver can solve relatively quickly, sometimes even without branching. Compared to an MIP solver, a crucial factor missing in most scalable NN verifiers is the ability to efficiently generate and solve general cutting planes (or "cuts"). In generic MIP solvers, cutting planes are essential to strengthen the convex relaxation, so that much less branching is required. Advanced cutting planes are among the most important factors in modern MIP solvers [6]; they can strengthen the convex relaxation without removing any valid integer solution from the MIP formulation. In the setting of NN verification, cutting planes reflects complex intra-layer and inter-layer dependencies between multiple neurons, which cannot be easily captured by existing bound propagation methods with single neuron relaxations [45]. This motivates us to seek the combination of efficient bound propagation method with effective cutting planes to further increase the power of NN verifiers.

A few key factors make the inclusion of *general* cutting planes in NN verifiers quite challenging. First, existing efficient bound propagation frameworks such as CROWN [61] and β-CROWN [57] cannot solve general cutting plane constraints that may involve variables across *different layers* in the MIP formulation. Particularly, these frameworks do not explicitly include the *integer variables* in the MIP formulation that are crucial when encoding many classical strong cutting planes, such as Gomory cuts and mixed integer rounding (MIR) cuts. Furthermore, although some existing works [47, 52, 40] enhanced the basic convex relaxation used in NN verification (such as the Planet relaxation [19]), these enhanced relaxations involve only one or a few neurons in a single layer or two adjacent layers, and are not general enough. In addition, an LP solver is often required to handle these additional cutting plane constraints [40], for which the efficient and GPU-accelerated bound propagation cannot be used, so the use of these tighter relaxations may not always bring improvements.

In this paper, we achieve major progress in using general cutting planes in bound propagation based NN verifiers. To mitigate the challenge of efficiently solving general cuts, **our first contribution** is to generalize existing bound propagation methods to their most general form, enabling constraints involving variables from neurons of any layer as well as integer variables that encode the status of a ReLU neuron. This allows us to consider any cuts during bound propagation without relying on a slow LP solver, and opens up the opportunity for using advanced cutting plane techniques efficiently for the NN verification problem. **Our second contribution** involves combining a cutting-plane-focused, off-the-shelf MIP solver with our GPU-accelerated, branching-focused bound propagation method capable of handling general cuts. We entirely disable branching in the MIP solver and use it only for generating high quality cutting planes not restricting to neurons within adjacent layers. Although an MIP solver often cannot verify large neural networks, we find that they can generate high quality cutting planes within a short time, significantly helping bound propagation to achieve better bounds.

Our experiments show that general cutting planes can bring significant improvements to NN verifiers: we are the first verifier that *completely* solves *all* instances in the `oval20` benchmark, with an average time of only 13 seconds per instance; on the even harder `oval21` benchmark in VNN-COMP 2021 [3], we can verify *twice as many instances* compared to the competition winner. We also outperform existing state-of-the-art bound propagation based methods including those using multi-neuron relaxations (a limited form of cutting planes).

2 Background

The NN verification problem We consider the verification problem for an L-layer ReLU-based Neural Network (NN) with inputs $\hat{\boldsymbol{x}}^{(0)} := \boldsymbol{x} \in \mathbb{R}^{d_0}$, weights $\mathbf{W}^{(i)} \in \mathbb{R}^{d_i \times d_{i-1}}$, and biases $\mathbf{b}^{(i)} \in \mathbb{R}^{d_i}$ ($i \in \{1, \cdots, L\}$). We can get the NN outputs $f(\boldsymbol{x}) = \boldsymbol{x}^{(L)} \in \mathbb{R}^{d_L}$ by sequentially propagating the input $\boldsymbol{x}$ through affine layers with $\boldsymbol{x}^{(i)} = \mathbf{W}^{(i)}\hat{\boldsymbol{x}}^{(i-1)} + \mathbf{b}^{(i)}$ and ReLU layer with $\hat{\boldsymbol{x}}^{(i)} = \mathrm{ReLU}(\boldsymbol{x}^{(i)})$. We also let scalars $\hat{x}_j^{(i)}$ and $x_j^{(i)}$ denote the post-activation and pre-activation, respectively, of j-th ReLU neuron in i-th layer. Throughout the paper, we use bold symbols to denote vectors (e.g., $\boldsymbol{x}^{(i)}$) and regular symbols to denote scalars (e.g., $x_j^{(i)}$ is the j-th element of $\boldsymbol{x}^{(i)}$). We use the shorthand $[N]$ to denote $\{1, \cdots, N\}$, and $\mathbf{W}_{:,j}^{(i)}$ is the j-th column of $\mathbf{W}^{(i)}$.

Commonly, the input $\boldsymbol{x}$ is bounded within a perturbation set $\mathcal{C}$ (such as an ℓ_p norm ball) and the verification specification defines a property of the output $f(\boldsymbol{x})$ that should hold for any $\boldsymbol{x} \in \mathcal{C}$, e.g., whether the true label's logit $f_y(\boldsymbol{x})$ will be always larger than another label's logit $f_j(\boldsymbol{x})$, (i.e., checking if $f_y(\boldsymbol{x}) - f_j(\boldsymbol{x})$ is always positive). Since we can append the verification specification (such as a linear function on neural network output) as an additional layer of the network, canonically, the NN verification problem requires one to solve the following one-dimensional ($d_L = 1$) optimization objective on $f(\boldsymbol{x})$:

$$f^* = \min_{\boldsymbol{x}} f(\boldsymbol{x}), \forall \boldsymbol{x} \in \mathcal{C} \tag{1}$$

with the relevant property defined to be proven if the optimal solution $f^* \geq 0$. Throughout this work, we consider the ℓ_∞ norm ball $\mathcal{C} := \{\boldsymbol{x} : \|\boldsymbol{x} - \boldsymbol{x}_0\|_\infty \leq \epsilon\}$ where $\boldsymbol{x}_0$ is a predefined constant (e.g., a clean input image), although it is possible to extend to other norms or specifications [43, 59].

The MIP and LP formulation for NN verification The mixed integer programming (MIP) formulation is the root of many NN verification algorithms. This formulation uses binary variables $\mathbf{z}$ to encode the non-linear ReLU neurons to make the non-convex optimization problem (1) tractable. Additionally, we assume that we know sound pre-activation bounds $\boldsymbol{l}^{(i)} \leq \boldsymbol{x}^{(i)} \leq \boldsymbol{u}^{(i)}$ for $\boldsymbol{x} \in \mathcal{C}$ which can be obtained via cheap bound propagation methods such as IBP [23] or CROWN [61]. Then ReLU neurons for each layer i can be classified into three classes [58], namely "active" ($\mathcal{I}^{+(i)}$), "inactive" ($\mathcal{I}^{-(i)}$) and "unstable" ($\mathcal{I}^{(i)}$) neurons, respectively:

$$\mathcal{I}^{+(i)} := \{j : l_j^{(i)} \geq 0\}; \quad \mathcal{I}^{-(i)} := \{j : u_j^{(i)} \leq 0\}; \quad \mathcal{I}^{(i)} := \{j : l_j^{(i)} \leq 0,\ u_j^{(i)} \geq 0\}$$

Based on the definition of ReLU, activate and inactive neurons are linear functions, so only unstable neurons require binary encoding. The *MIP formulation* of (1) is:

$$f^* = \min_{\boldsymbol{x}, \hat{\boldsymbol{x}}, \mathbf{z}} f(\boldsymbol{x}) \qquad \text{s.t. } f(\boldsymbol{x}) = \boldsymbol{x}^{(L)}; \quad \hat{\boldsymbol{x}}^{(0)} = \boldsymbol{x}; \quad \boldsymbol{x} \in \mathcal{C}; \tag{2}$$

$$\boldsymbol{x}^{(i)} = \mathbf{W}^{(i)} \hat{\boldsymbol{x}}^{(i-1)} + \mathbf{b}^{(i)}; \quad i \in [L], \tag{3}$$

$$\hat{x}_j^{(i)} \geq 0; \quad j \in \mathcal{I}^{(i)}, i \in [L-1] \tag{4}$$

$$\hat{x}_j^{(i)} \geq x_j^{(i)}; \quad j \in \mathcal{I}^{(i)}, i \in [L-1] \tag{5}$$

$$\hat{x}_j^{(i)} \leq u_j^{(i)} z_j^{(i)}; \quad j \in \mathcal{I}^{(i)}, i \in [L-1] \tag{6}$$

$$\hat{x}_j^{(i)} \leq x_j^{(i)} - l_j^{(i)}(1 - z_j^{(i)}); \quad j \in \mathcal{I}^{(i)}, i \in [L-1] \tag{7}$$

$$z_j^{(i)} \in \{0, 1\}; \quad j \in \mathcal{I}^{-(i)}, i \in [L-1] \tag{8}$$

$$\hat{x}_j^{(i)} = x_j^{(i)}; \quad j \in \mathcal{I}^{+(i)}, i \in [L-1] \tag{9}$$

$$\hat{x}_j^{(i)} = 0; \quad j \in \mathcal{I}^{-(i)}, i \in [L-1] \tag{10}$$

Since a MIP problem is slow or intractable to solve, it is commonly relaxed as a When the integer variables are relaxed to continuous ones, we obtain the *LP relaxation* of NN verification problem:

$$f_{\text{LP}}^* = \min_{\boldsymbol{x}, \hat{\boldsymbol{x}}, \mathbf{z}} f(\boldsymbol{x})$$

$$\text{s.t. } (3), (2), (4), (5), (6), (7), (9), (10), \quad 0 \leq z_j^{(i)} \leq 1; \quad j \in \mathcal{I}^{(i)},\ i \in [L-1] \tag{11}$$

The ReLU constraints involving z is often projected out, leading to the well-known *Planet relaxation* used in many NN verifiers, replacing (6), (7) and (8) with a single constraint to get an equivalent LP:

$$f_{\text{LP}}^* = \min_{\boldsymbol{x}, \hat{\boldsymbol{x}}, \mathbf{z}} f(\boldsymbol{x})$$

$$\text{s.t. } (3), (2), (4), (5), (9), (10), \quad \hat{x}_j^{(i)} \leq \frac{u_j^{(i)}}{u_j^{(i)} - l_j^{(i)}} (x_j^{(i)} - l_j^{(i)});\ j \in \mathcal{I}^{(i)},\ i \in [L-1] \tag{12}$$

Due to the relaxations, the objective of the LP formulation is always a lower bound of the MIP formulation: $f_{\text{LP}}^* \leq f^*$. A verifier using this formulation is incomplete: if $f_{\text{LP}}^* \geq 0$, then $f^* \geq 0$ and the property is verified; otherwise, we cannot conclude the sign of f^* so the verifier must return "unknown". Branch and bound can be used to improve the lower bound and achieve completeness [10, 57]. However, in this paper, we work on an orthogonal direction of strengthening the LP formulation by adding cutting planes to obtain larger bounds.

Bound propagation methods Instead of solving the LP formulation directly using a LP solver, bound propagation methods aims to quickly give a lower bound for f^*_{LP}. For example, CROWN [61] and β-CROWN [57] propagate a sound linear lower bound backwards for $f_L(\boldsymbol{x})$ with respect to each intermediate layer. For example, suppose we know

$$\min_{\boldsymbol{x}\in\mathcal{C}} f_L(\boldsymbol{x}) \geq \min_{\boldsymbol{x}\in\mathcal{C}} \mathbf{a}^{(i)\top}\hat{\boldsymbol{x}}^{(i)} + c^{(i)} \tag{13}$$

With $i = L-1$ the above is trivially hold with $\mathbf{a}^{(L-1)} = \mathbf{W}^{(L)}$, $c^{(L-1)} = \mathbf{b}^{(L)}$. In bound propagation methods, a propagation rule propagates an inequality (13) through a previous layer $\hat{\boldsymbol{x}}^{(i)} := \text{ReLU}(\mathbf{W}^{(i)}\hat{\boldsymbol{x}}^{(i-1)} + \mathbf{b}^{(i)})$ to obtain a *sound* inequality with respect to $\hat{\boldsymbol{x}}^{(i-1)}$:

$$\min_{\boldsymbol{x}\in\mathcal{C}} f_L(\boldsymbol{x}) \geq \min_{\boldsymbol{x}\in\mathcal{C}} \mathbf{a}^{(i-1)\top}\hat{\boldsymbol{x}}^{(i-1)} + c^{(i-1)}$$

Here $\mathbf{a}^{(i-1)}$, $c^{(i-1)}$ can be calculated in close-form via $\mathbf{a}^{(i)}$, $c^{(i)}$, $\mathbf{W}^{(i)}$, $\mathbf{b}^{(i)}$, $\boldsymbol{l}^{(i)}$ and $\boldsymbol{u}^{(i)}$ such that the bound still holds (see Lemma 1 in [57]). Applying the procedure repeatedly will eventually reach the input layer:

$$\min_{\boldsymbol{x}\in\mathcal{C}} f_L(\boldsymbol{x}) \geq \min_{\boldsymbol{x}\in\mathcal{C}} \mathbf{a}^{(0)\top}\boldsymbol{x} + c^{(0)} \tag{14}$$

The minimization on linear layer can be solved easily when $\mathcal{C}$ is a ℓ_p norm ball to obtain a valid lower bound of f^*. Since the bounds propagate layer-by-layer, this process can be implemented efficiently on GPUs [59] without relying on a slow LP solver, which greatly improves the scalability and solving time. Additionally, it is often used to obtain intermediate layer bounds $\boldsymbol{l}^{(i)}$ and and $\boldsymbol{u}^{(i)}$ required for the MIP formulation (6)(7), by treating each $x_j^{(i)}$ as the output neuron. The bound propagation rule can either be derived in primal space [61], dual space [58] or abstract interpretations [50]. In Sec.3.1, we will discuss the our bound propagation procedure with general cutting plane constraints.

3 Neural Network Verification with General Cutting Planes

3.1 GCP-CROWN: General Cutting Planes in Bound Propagation

In this section, we generalize existing bound propagation method to handle general cutting plane constraints. Our goal is to derive a bound propagation rule similar to CROWN and β-CROWN discussed in Section 2, however considering additional constraints among any variables within the LP relaxation. To achieve this, we first derive the dual problem of the LP; inspired by the dual formulation, we derive the bound prorogation rule in a layer by layer manner that takes all cutting plane constraints into consideration. The derivation process is inspired by [58, 45] and [57].

LP relaxation with cutting planes. In this section, we derive the bound propagation procedure under the presence of general cutting plane constraints. A cutting plane is a constraint involving any variables $\boldsymbol{x}^{(i)}$ (pre-activation), $\hat{\boldsymbol{x}}^{(i)}$ (post-activation), $\mathbf{z}^{(i)}$ (ReLU indicators) from any layer i:

$$\sum_{i=1}^{L-1}\left(\boldsymbol{h}^{(i)\top}\boldsymbol{x}^{(i)} + \boldsymbol{g}^{(i)\top}\hat{\boldsymbol{x}}^{(i)} + \boldsymbol{q}^{(i)\top}\mathbf{z}^{(i)}\right) \leq d$$

Here $\boldsymbol{h}^{(i)}$, $\boldsymbol{g}^{(i)}$ and $\boldsymbol{q}^{(i)}$ are coefficients for this cut constraint. The difference between a valid cutting plane and an arbitrary constraint is that a valid cutting plane should not remove any valid integer solution from the MIP formulation. Our new bound propagation procedure can work for any constraints, although in this work we focus on studying the impacts of cutting planes. When there are N cutting planes, we write them in a matrix form:

$$\sum_{i=1}^{L-1}\left(\boldsymbol{H}^{(i)}\boldsymbol{x}^{(i)} + \boldsymbol{G}^{(i)}\hat{\boldsymbol{x}}^{(i)} + \boldsymbol{Q}^{(i)}\mathbf{z}^{(i)}\right) \leq \boldsymbol{d} \tag{15}$$

where $\boldsymbol{H}^{(i)}, \boldsymbol{G}^{(i)}, \boldsymbol{Q}^{(i)} \in \mathbb{R}^{N\times d_i}$. The LP relaxation with all cutting planes is:

$$f^*_{\text{LP-cut}} = \min_{\boldsymbol{x},\hat{\boldsymbol{x}},\mathbf{z}} f(x)$$

$$\text{s.t. } (3),(2),(4),(5),(6),(7),(9),(10), \quad 0 \leq z_j^{(i)} \leq 1; \quad j \in \mathcal{I}^{(i)},\ i \in [L-1]$$

$$\sum_{i=1}^{L-1}\left(\boldsymbol{H}^{(i)}\boldsymbol{x}^{(i)} + \boldsymbol{G}^{(i)}\hat{\boldsymbol{x}}^{(i)} + \boldsymbol{Q}^{(i)}\mathbf{z}^{(i)}\right) \leq \boldsymbol{d} \tag{16}$$

Since an additional constraint is added, $f^*_{\text{LP-cut}} \geq f^*_{\text{LP}}$ and we get closer to f^*. Unlike the original LP where each constraint only contains variables from two consecutive layers, our general cutting plane constraint may involve any variable from any layer in a single constraint.

The dual problem with cutting planes We first show the dual problem for the above LP. The dual problem we consider here is different from existing works in two ways: first, we have constraints with integer variables to support potential cutting planes on $\mathbf{z}$. Additionally, we have cutting planes constraints that may involve variables in *any* layer, so the dual in previous works such as [58] cannot be directly reused. Our dual problem is given below (derivation details in Appendix A):

$$
\begin{aligned}
f^*_{\text{LP-cut}} = \max_{\substack{\boldsymbol{\nu},\boldsymbol{\mu}\geq 0,\boldsymbol{\tau}\geq 0\\ \boldsymbol{\gamma}\geq 0,\boldsymbol{\pi}\geq 0,\boldsymbol{\beta}\geq 0}} & -\epsilon\|\boldsymbol{\nu}^{(1)\top}\mathbf{W}^{(1)}\boldsymbol{x}_0\|_1 - \boldsymbol{\beta}^\top \boldsymbol{d} - \sum_{i=1}^{L}\boldsymbol{\nu}^{(i)\top}\mathbf{b}^{(i)} \\
& + \sum_{i=1}^{L-1}\sum_{j\in\mathcal{I}^{(i)}}\left[\pi_j^{(i)} l_j^{(i)} - \text{ReLU}(u_j^{(i)}\gamma_j^{(i)} + l_j^{(i)}\pi_j^{(i)} - \boldsymbol{\beta}^\top \boldsymbol{Q}_{:,j}^{(i)})\right] \\
\text{s.t. } \boldsymbol{\nu}^{(L)} = -1;& \text{ and for each } i \in [L-1]: \\
\nu_j^{(i)} &= \boldsymbol{\nu}^{(i+1)\top}\mathbf{W}_{:,j}^{(i+1)} - \boldsymbol{\beta}^\top(\boldsymbol{H}_{:,j}^{(i)} + \boldsymbol{G}_{:,j}^{(i)});\ j \in \mathcal{I}^{+(i)} \\
\nu_j^{(i)} &= -\boldsymbol{\beta}^\top \boldsymbol{H}_{:,j}^{(i)};\ j \in \mathcal{I}^{-(i)} \\
& \text{and for each } j \in \mathcal{I}^{(i)} \text{ the two equalities below hold:} \\
\nu_j^{(i)} &= \pi_j^{(i)} - \tau_j^{(i)} - \boldsymbol{\beta}^\top \boldsymbol{H}_{:,j}^{(i)}; \quad \left(\pi_j^{(i)} + \gamma_j^{(i)}\right) - \left(\mu_j^{(i)} + \tau_j^{(i)}\right) = \boldsymbol{\nu}^{(i+1)\top}\mathbf{W}_{:,j}^{(i+1)} - \boldsymbol{\beta}^\top \boldsymbol{G}_{:,j}^{(i)}
\end{aligned}
$$

Instead of solving the dual problem exactly, we use it to obtain a lower bound of $f^*_{\text{LP-cut}}$. Intuitively, due to the definition of due problem, any valid setting of dual variables leads to a lower bound of $f^*_{\text{LP-cut}}$. Informally, starting from $\boldsymbol{\nu}^{(L)} = -1$, by applying the constraints in this dual formulation, we can compute $\boldsymbol{\nu}^{(L-1)}, \boldsymbol{\nu}^{(L-2)}, \cdots$ until $\boldsymbol{\nu}^{(1)}$. The final objective is a function of $\boldsymbol{\nu}^{(i)}$, $i \in [N]$ and other dual variables. Precisely, our GCP-CROWN bound propagation procedure with general cutting plane constraint is presented in the theorem below (proof in Appendix A):

Theorem 3.1 (Bound propagation with general cutting planes)**.** *Given any optimizable parameters $0 \leq \alpha_j^{(i)} \leq 1$ and $\boldsymbol{\beta} \geq 0$, $f^*_{\text{LP-cut}}$ is lower bounded by the following objective function:*

$$
g(\boldsymbol{\alpha}, \boldsymbol{\beta}) = -\epsilon\|\boldsymbol{\nu}^{(1)\top}\mathbf{W}^{(1)}\boldsymbol{x}_0\|_1 - \sum_{i=1}^{L}\boldsymbol{\nu}^{(i)\top}\mathbf{b}^{(i)} - \boldsymbol{\beta}^\top \boldsymbol{d} + \sum_{i=1}^{L-1}\sum_{j\in\mathcal{I}^{(i)}} h_j^{(i)}(\boldsymbol{\beta})
$$

where variables $\boldsymbol{\nu}^{(i)}$ are obtained by propagating $\boldsymbol{\nu}^{(L)} = -1$ throughout all $i \in [L-1]$:

$$
\begin{aligned}
\nu_j^{(i)} &= \boldsymbol{\nu}^{(i+1)\top}\mathbf{W}_{:,j}^{(i+1)} - \boldsymbol{\beta}^\top(\boldsymbol{H}_{:,j}^{(i)} + \boldsymbol{G}_{:,j}^{(i)}),\ \ j \in \mathcal{I}^{+(i)} \\
\nu_j^{(i)} &= -\boldsymbol{\beta}^\top \boldsymbol{H}_{:,j}^{(i)},\ \ j \in \mathcal{I}^{-(i)} \\
\nu_j^{(i)} &= \pi_j^{(i)^*} - \alpha_j^{(i)}[\hat{\nu}_j^{(i)}]_- - \boldsymbol{\beta}^\top \boldsymbol{H}_{:,j}^{(i)},\ \ j \in \mathcal{I}^{(i)}
\end{aligned}
$$

Here $\hat{\nu}_j^{(i)}$, $\pi_j^{(i)^*}$ and $h_j^{(i)}(\boldsymbol{\beta})$ are defined for each unstable neuron $j \in \mathcal{I}^{(i)}$ as:

$$
\begin{aligned}
\hat{\nu}_j^{(i)} &:= \boldsymbol{\nu}^{(i+1)\top}\mathbf{W}_{:,j}^{(i+1)} - \boldsymbol{\beta}^\top \boldsymbol{G}_{:,j}^{(i)} \\
\pi_j^{(i)^*} &= \max\left(\min\left(\frac{u_j^{(i)}[\hat{\nu}_j^{(i)}]_+ + \boldsymbol{\beta}^\top \boldsymbol{Q}_{:,j}^{(i)}}{u_j^{(i)} - l_j^{(i)}}, [\hat{\nu}_j^{(i)}]_+\right), 0\right)
\end{aligned}
$$

$$
h_j^{(i)}(\boldsymbol{\beta}) = \begin{cases} l_j^{(i)}\pi_j^{(i)^*} & \text{if } \ l_j^{(i)}[\hat{\nu}_j^{(i)}]_+ \leq \boldsymbol{\beta}^\top \boldsymbol{Q}_{:,j}^{(i)} \leq u_j^{(i)}[\hat{\nu}_j^{(i)}]_+ \\ 0 & \text{if } \ \boldsymbol{\beta}^\top \boldsymbol{Q}_{:,j}^{(i)} \geq u_j^{(i)}[\hat{\nu}_j^{(i)}]_+ \\ \boldsymbol{\beta}^\top \boldsymbol{Q}_{:,j}^{(i)} & \text{if } \ \boldsymbol{\beta}^\top \boldsymbol{Q}_{:,j}^{(i)} \leq l_j^{(i)}[\hat{\nu}_j^{(i)}]_+ \end{cases}
$$

Based on Theorem 3.1, to obtain an lower bound of $f^*_{\text{LP-cut}}$, we start with any valid setting of $0 \leq \boldsymbol{\alpha} \leq 1$ and $\boldsymbol{\beta} \geq 0$ and $\boldsymbol{\nu}^{(L)} = -1$. According to the bound propagation rule, we can compute

each $\boldsymbol{\nu}^{(i)}$, $i \in [L-1]$, in a layer by layer manner. Then objective $g(\boldsymbol{\alpha}, \boldsymbol{\beta})$ can be evaluated based on all $\boldsymbol{\nu}^{(i)}$ to give an lower bound of $f^*_{\text{LP-cut}}$. Since any valid setting of $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ lead to a valid lower bound, we can optimize $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ using gradient ascent in a similar manner as in [60, 57] to tighten this lower bound. The entire procedure can also run on GPU for great acceleration.

Connection to Convex Outer Adversarial Polytope In convex outer adversarial polytope [58], a bound propagation rule was developed in a similar manner in the dual space without considering cutting plane constraints, and is a special case of ours. We denote their bound propagation objective function as g_{WK} which also contains optimizable parameters $\boldsymbol{\alpha}_{\text{WK}}$.

Proposition 3.2. *Given the same input $\boldsymbol{x}$, perturbation set $\mathcal{C}$, network weights, and N cutting plane constraints,*

$$\max_{\boldsymbol{\alpha},\boldsymbol{\beta}} g(\boldsymbol{\alpha}, \boldsymbol{\beta}) \geq \max_{\boldsymbol{\alpha}_{WK}} g_{WK}(\boldsymbol{\alpha}_{WK})$$

Proof. In Theorem 3.1, when all $\boldsymbol{\beta}$ are set to 0, then $\pi_j^{(i)*} = \frac{u_j^{(i)}[\hat{\nu}_j^{(i)}]_+}{u_j^{(i)} - l_j^{(i)}}$ and $h_j^{(i)}(\boldsymbol{\beta}) = \pi_j^{(i)*} l_j^{(i)}$, we recover exactly the same bound propagation equations as in [58]. However, since we allow the addition of cutting plane methods and we can maximize over the additional parameter $\boldsymbol{\beta}$, the objective given by our bound propagation is always at least as good as g_{KW}. □

Connection to CROWN-like bound propagation methods CROWN [61] and α-CROWN [60] use the same bound propagation rule as [58] so Proposition 3.2 also applies, although they were derived from primal space without explicitly formulating the problem as a LP. Salman et al. [45] showed that many other bound propagation methods [50, 55] are equivalent to or weaker than [58]. Recently, Wang et al. [57] extends CROWN to β-CROWN to handle split constraints (e.g., $x_j^{(i)} \geq 0$). It can be seen a special case as GCP-CROWN where all $\boldsymbol{H}$, $\boldsymbol{G}$ and $\boldsymbol{Q}$ matrices are zeros except:

$$\boldsymbol{H}^{(i)}_{j,j} = 1, j \in \mathcal{I}^{-(i)} \qquad \text{for } x_j^{(i)} \leq 0 \text{ split;} \qquad \boldsymbol{H}^{(i)}_{j,j} = -1, j \in \mathcal{I}^{+(i)} \qquad \text{for } x_j^{(i)} \geq 0 \text{ split}$$

In addition, [20] encodes multi-neuron relaxations using sparse $\boldsymbol{H}^{(i)}$ and $\boldsymbol{G}^{(i)}$ and each cut contains a small number of neurons involving $\boldsymbol{x}^{(i)}$ and $\hat{\boldsymbol{x}}^{(i)}$ for the same layer i. Wang et al. [57] derived bound propagation rules from both the dual LP and the primal space with a Lagrangian without LP. However, in our case, it is not intuitive to derive bound propagation without LP due to the potential cutting planes on relaxed integer variables $\mathbf{z}$, which do not appear without the explicit LP formulation. Furthermore, although we derived cutting planes for bound propagation methods, it is technically also possible to derive them using other bounding frameworks such as Lagrangian decomposition [8].

3.2 Branch-and-bound with GCP-CROWN and MIP Solver Generated Cuts

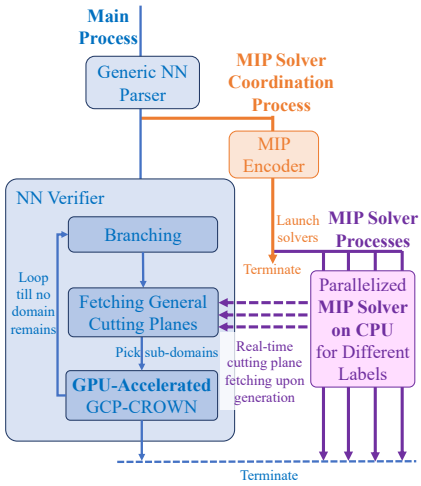

Figure 1: Overview of our cutting-plane-enhanced, fully-parallelized NN verifier.

To build a complete NN verifier, we follow the popular branch-and-bound (BaB) procedure [10, 9] in state-of-the-art NN verifiers with GPU accelerated bound propagation method [8, 60, 16, 57], and our GCP-CROWN is used as the bounding procedure in BaB. We refer the readers to Appendix B for a more detailed background on branch-and-bound. Having the efficient bound propagation procedure with general cutting plane constraints, we now need to find a good set of general cutting planes $\boldsymbol{H}^{(i)}, \boldsymbol{G}^{(i)}, \boldsymbol{Q}^{(i)}$ to accelerate NN verification. Since GCP-CROWN can adopt any cutting planes, to fully exploit its power, we propose to use off-the-shelf MIP solvers to generate cutting planes and create an NN verifier combining GPU-accelerated bound propagation with strong cuts generated by a MIP solver. We make the bound propagation on GPU and the MIP solver on CPU run in parallel with cuts added on-the-fly, so the original strong performance of bound-propagation-based NN verifier will never be affected by a potentially slow MIP solver. This allows us to make full use of available computing resource (GPU + CPU). The architecture of our verifier is shown in Fig 1.

MIP solvers for cutting plane generation Generic MIP solvers such as `cplex`[28] and `gurobi`[24] also apply a branch-and-bound strategy, conceptually similar to state-of-the-art NN verifiers. They often tend to be slower than specialized NN verifiers because MIP solvers rely on slower bounding procedures (e.g., Simplex or barrier method) and cannot apply an GPU-accelerated method such as bound propagation or Lagrangian decomposition. However, we still find that MIP solvers are a strong baseline when *combined with tight intermediate layer bounds*. For example, in the `oval21` benchmark in Table 2, α-CROWN+MIP (MIP solver combined with tight intermediate layer bound computed by α-CROWN) is able to solve 4 more instances compared to all other tools in the competition. Our investigation found that α-CROWN+MIP explores much less branches than other state-of-the-art branch-and-bound based NN verifiers, however before branching starts, the MIP solver produces very effective cutting planes that can often verify an instance with little branching. MIP solvers is able to discover sophisticated cutting planes involving several NN layers reflecting complex correlations among neurons, while existing NN verifiers only exploit specific forms of constraints within same layer or between adjacent layers [52, 47, 40]. This motivates us to combine the strong cutting planes generated by MIP solvers with GPU-accelerated bound-propagation-based BaB.

In this work, we use `cplex` as the MIP solver for cutting plane generation since `gurobi` cannot export cuts. We entirely disable all branching features in `cplex` and make it focus on finding cutting planes only. These cutting planes are generated only for the root node of the BaB search tree so they are sound for any subdomains with neuron splits in BaB. We conduct branching using our generalized bound propagation procedure in Section (3.1) with the cutting planes generated by `cplex`.

Fully-parallelized NN verifier design We design our verifier as shown in Figure 1. After parsing the NN under verification, we launch a separate process to encode the NN verification problem as MIP problem and start multiple MIP solvers, one for each target label to be verified. At the same time, the main NN verifier process executes branch-and-bound without waiting for the MIP solver process. In each iteration of branch and bound, we query the MIP solving processes and fetch any newly generated cutting planes. If any cutting planes are produced, they are added as $\boldsymbol{H}^{(i)}, \boldsymbol{G}^{(i)}, \boldsymbol{Q}^{(i)}$ in GCP-CROWN and tighten the bounds for subsequent branching and bounding. If no cutting planes are produces, GCP-CROWN reduces to β-CROWN [57]. Since our verifier is based on strengthening the bounds in β-CROWN with sound cutting planes, we are also sound and complete.

Adjustments to existing branch and bound method. We implement GCP-CROWN into the α,β-CROWN verifier [61, 59, 57], the winning verifier in VNN-COMP 2021 [3], as the backbone for BaB with bound propagation. To better exploit the power of cutting planes under a fully parallel and asynchronous design, we made a key changes to the BaB procedure. When the number of BaB subdomains are greater than batch size, we rank the subdomains by their lower bounds and choose the *easiest domains* with largest lower bounds first to verify with GCP-CROWN, unlike most existing verifiers which solve the worst domains first. We use such a order because the MIP solver generates cutting planes incrementally. Solving these easier subdomains tend to require no or fewer cutting planes, so we solve them at earlier stages where cutting planes have not been generated or are relatively weak. On the other hand, if we split worst subdomains first, the number of subdomains will grow quickly, and it can take a long time to verify these domains when stronger cuts become available later. Under a similar rationale, when verifying a multi-class model and BaB needs to verify each target class label one by one, we start BaB from the easiest label first (α-CROWN bound closest to 0), allowing the MIP solver to run longer for harder labels, generating stronger cuts for harder labels.

4 Experiments

We now evaluate our verifier, GCP-CROWN with MIP cuts, on a few popular verification benchmarks. Since our verifier uses a MIP solver in our pipeline, we also include a new baseline, α-CROWN+MIP, which uses `gurobi` as the MIP solver with the tightest possible intermediate layer bounds from α-CROWN [59]. We use the same branch and bound algorithm as in β-CROWN and we use filtered smart branching (FSB) [16] as the branching heuristic in all experiments. Without cutting planes, GCP-CROWN becomes vanilla β-CROWN as we share the same code base as β-CROWN. We include detailed setup for our experiments in Appendix C.

Results on the `oval20` benchmark in VNN-COMP 2020. `oval20` is a popular benchmark consistently used in huge amount of NN verifiers and it perfectly reflects the progress of NN verifiers.

Table 1: Average runtime and average number of branches on `oval20` benchmarks with 100 properties per model. Timeout is set to 3,600 seconds (consistent with other literature results).

	CIFAR-10 Base			CIFAR-10 Wide			CIFAR-10 Deep		
Method	time(s)	branches	%timeout	time(s)	branches	%timeout	time(s)	branches	%timeout
MIPplanet [19]	2849.69	-	68.00	2417.53	-	46.00	2302.25	-	40.00
BaBSR [9]	2367.78	1020.55	36.00	2871.14	812.65	49.00	2750.75	401.28	39.00
GNN-online [36]	1794.85	565.13	33.00	1367.38	372.74	15.00	1055.33	131.85	4.00
BDD+ BaBSR [8]	807.91	195480.14	20.00	505.65	74203.11	10.00	266.28	12722.74	4.00
Fast-and-Complete [60]	695.01	119522.65	17.00	495.88	80519.85	9.00	105.64	2455.11	1.00
OVAL (BDD+ GNN)* [8, 36]	662.17	67938.38	16.00	280.38	17895.94	6.00	94.69	1990.34	1.00
A.set BaBSR [16]	381.78	12004.60	7.00	165.91	2233.10	3.00	190.28	2491.55	2.00
BigM+A.set BaBSR [16]	390.44	11938.75	7.00	172.65	4050.59	3.00	177.22	3275.25	2.00
BaDNB (BDD+ FSB)[17]	309.29	38239.04	7.00	165.53	11214.44	4.00	10.50	368.16	**0.00**
ERAN* [47, 48, 50, 49]	805.94	-	5.00	632.20	-	9.00	545.72	-	**0.00**
β-CROWN [57]	118.23	208018.21	3.00	78.32	116912.57	2.00	5.69	41.12	**0.00**
α-CROWN+MIP†	335.50	8523.37	3.00	203.87	2029.60	**0.00**	76.90	1364.24	**0.00**
GCP-CROWN with MIP cuts	**13.40**	25436.28	**0.00**	**3.02**	2095.18	**0.00**	**3.70**	110.92	**0.00**

* Results from VNN-COMP 2020 report [34]. † A new baseline proposed and evaluated in this work, not presented in previous papers.

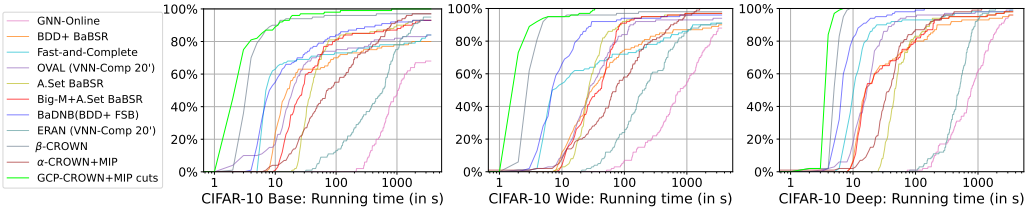

Figure 2: Percentage of solved properties on the `oval20` benchmark vs. running time (timeout 1hr).

We include literature results for many baselines in Table 1. We are the only verifier that can *completely solve* all three models without any timeout. Our average runtime is significantly lower compared to the time of baselines because we have no timeout (counted as 3600s), and our slowest instance only takes about a few minutes while easy ones only take a few seconds, as shown in Figure 2. Additionally, we often use less number of branches compared to the state-of-the-art verifier, β-CROWN, since our strong cutting planes help us to eliminate many hard to solve subdomains in BaB. Furthermore, we highlight that α-CROWN+MIP also achieves a low timeout rate, although it is much slower than our bound propagation based approach combined with cuts from a MIP solver.

Results on VNN-COMP 2021 benchmarks. Among the eight scoring benchmarks in VNN-COMP 2021 [3], only two (`oval21` and `cifar10-resnet`) are most suitable for the evaluation of this work. Among other benchmarks, `acasxu` and `nn4sys` have low input dimensionality and require input space branching rather than ReLU branching; `verivital`, `mnistfc`, and `eran` benchmarks consist of small MLP networks that can be solved directly by MIP solvers; `marabou` contains mostly adversarial examples, making it a good benchmark for falsifiers rather than verifiers. We present our results in Table 2. Besides results from 6 VNN-COMP participants, we also include two additional baselines, α-CROWN+MIP (same as in Table 1), and MN-BaB [20], a recently proposed branch and bound framework with multi-neuron relaxations [39], which can be viewed as a restricted form of cutting planes. On the `oval21` benchmark, OVAL, VeriNet and α,β-CROWN are the best performing tools, verified 11 out of 27 instances, while we can verify *twice more* instances (22 out of 27) on this benchmark. On the `cifar10-resnet` benchmark, our verifier also solves the most number of instances and achieves the lowest average time. In fact, α-CROWN+MIP is also a strong baseline, solving 4 more instances than all competition participants, showing the importance of strong cutting planes. Our GCP-CROWN with MIP cuts combines the benefits of fast bound propagation on GPU with the strong cutting planes generated by a MIP solver and achieves the best performance.

Results on SDP-FO benchmarks. We further evaluate our method on the SDP-FO benchmarks in [15, 57]. This benchmark contains 7 mostly adversarially trained MNIST and CIFAR models with 200 instances each, which are hard for many existing verifiers. Beyond the baselines reported in [57], we also include two additional baselines, α-CROWN+MIP (same as in Table 1) and MN-BaB [20] (same as in Table 2). Table 3 shows that our method improves the percentage of verified images ("verified accuracy") on all models compared to state-of-the-art verifiers, further closing the gap between verified accuracy and empirical robust accuracy obtained by PGD attack (reported as "upper bound" in Table 3).

Table 2: VNN-COMP 2021 benchmarks: `oval21` and `cifar10-resnet`. Results marked with VNN-COMP are from publicly available benchmark data on VNN-COMP 2021 Github. "-" indicates unsupported model.

	`oval21` (30 properties; PGD upper bound 27)			`cifar10-resnet` (72 properties)		
Method	time(s)	# verified	%timeout	time(s)	# verified	%timeout
nnenum (VNN-COMP) [2, 4]	630.06	2	86.66	-	-	-
Marabou (VNN-COMP) [31]	429.13	5	73.33	157.70	39	45.83
ERAN (VNN-COMP)[40, 38]	233.84	6	70.00	129.48	43	40.28
OVAL (VNN-COMP) [17, 16]	393.14	11	53.33	-	-	-
VeriNet (VNN-COMP) [25, 26]	414.61	11	53.33	105.91	48	33.33
MN-BaB [20]	435.46	10	56.66	-	-	-
Venus2 [33]	386.71	17	33.33	-	-	-
α,β-CROWN (VNN-COMP) [61, 60, 57]	395.32	11	53.33	99.87	58	19.44
α-CROWN+MIP	301.23	15	40.00	125.48	46	36.11
GCP-CROWN with MIP cuts	**187.91**	**22**	**16.66**	**77.72**	**61**	**15.28**

Table 3: **Verified accuracy (%)** and avg. per-example verification time (s) on 7 models from SDP-FO [15].

Dataset	Model	SDP-FO [15]*		PRIMA [40]		β-CROWN [57]		MN-BaB [20]		Venus2 [33]		α-CROWN+MIP		GCP-CROWN		Upper
$\epsilon = 0.3$ and $\epsilon = 2/255$		Verified%	Time (s)	Ver.%	Time(s)	Ver.%	Time(s)	Ver.%	Time(s)	Ver.%	Time(s)	Ver.%	Time(s)	bound		
MNIST	CNN-A-Adv	43.4	>20h	44.5	135.9	70.5	21.1	-	-	35.5	148.4	56.5	224.3	**71.0**	20.75	76.5
CIFAR	CNN-B-Adv	32.8	>25h	38.0	343.6	46.5	32.2	-	-	-	-	27.0	360.6	**48.5**	59.4	65.0
	CNN-B-Adv-4	46.0	>25h	53.5	43.8	54.0	11.6	-	-	-	-	52.5	129.5	**58.0**	21.9	63.5
	CNN-A-Adv	39.6	>25h	41.5	4.8	44.0	5.8	42.5	68.3	47.5	26.0	46.0	63.1	**48.0**	14.4	50.0
	CNN-A-Adv-4	40.0	>25h	45.0	4.9	46.0	5.6	46.0	37.7	47.5	13.1	**48.5**	16.4	**48.5**	5.7	49.5
	CNN-A-Mix	39.6	>25h	37.5	34.3	41.5	49.6	35.0	140.3	33.5	72.4	32.5	231.3	**46.0**	39.1	53.0
	CNN-A-Mix-4	47.8	>25h	48.5	7.0	50.5	5.9	49.0	70.9	49.0	37.3	52.5	77.7	**54.0**	13.4	57.5

* We run α-CROWN+MIP and MN-BaB with 600s timeout threshold for all models. "-" indicates that we could not run a model due to verifier errors. We run our GCP-CROWN with MIP cuts with a shorter 200s timeout for all models and it achieves better verified accuracy than all other baselines. Other results are reported from [57].

5 Related Work

Cutting plane method is a classic technique to strengthen the convex relaxation of an integer programming problem. Generic cutting planes such as Gomory's cut [21, 22], Chvátal–Gomory cut [12], implied bound cut [27], lift-and-project [35], reformulation-linearization techniques [46] and mixed integer rounding cuts [41, 37] can be applied to almost any LP relaxed problems, and problem specific cutting planes such as Knapsack cut [14], Flow-cover cut [42] and Clique cut [29] require specific problem structures. Modern MIP solvers typically uses a branch-and-cut strategy, which tends to generate a large number of cuts before starting the next iteration of branching, and solve the LP relaxation of the MIP problem with cutting planes with an exact method such as the Simplex method. Our GCP-CROWN is a specialized solver for the NN verification problem, which can quickly obtain a lower bound of the LP relaxation with cutting planes specially for the NN verification problem.

The verification of piece-wise linear NNs can be formulated as a MIP problem, so early works [30, 31, 53] solve an integer or combinatorial formulation directly. For efficiency reasons, most recent works use a convex relaxation such as linear relaxation [19, 58] or semidefinite relaxation [44, 15]. Salman et al. [45] discussed the limitation of many convex relaxation based NN verification algorithms and coined the term "convex relaxation barrier", specifically for the popular single-neuron "Planet" relaxation [19]. Several works developed novel techniques to break this barrier. [47] added constraints that depends on the aggregation of multiple neurons, and these constraints were passed to a LP solver. [40] enhanced the multi-neuron formulation of [47] to obtain tighter relaxations. [1] studied stronger convex relaxations for a ReLU neuron after an affine layer, and [52] constructed efficient algorithms based on this relaxation for incomplete verification. [16] extended the formulation in [1] to a dual form and combined it with branch and bound to achieve completeness. [7] proposed specialized cuts considering neuron dependencies and solve them using a MIP solver. [20] combined the multi-neuron relaxation [40] with branch and bound. Although these works can be seen as a special form of cutting planes, they mostly focused on enhancing the relaxation for several neurons within a single layer or two adjacent layers. GCP-CROWN can efficiently handle general cutting plane constraints with neurons from any layers in a bound propagation manner, and the cutting planes we find from a MIP solver can be seen as tighter convex relaxations encoding multi-neuron and multi-layer correlations.

6 Conclusion

In this paper, we propose GCP-CROWN, an efficient and GPU-accelerated bound propagation method for NN verification capable of handling any cutting plane constraints. We combine GCP-CROWN with branch and bound and high quality cutting planes generated by a MIP solver to tighten the convex relaxation for NN verification. The combination of fast bound propagation and strong cutting planes lead to state-of-the-art verification performance on multiple benchmarks. Our work opens up a great opportunity for studying more efficient and powerful cutting planes for NN verification.

References

[1] Ross Anderson, Joey Huchette, Christian Tjandraatmadja, and Juan Pablo Vielma. Strong convex relaxations and mixed-integer programming formulations for trained neural networks. In *Mathematical Programming*, 2020.

[2] Stanley Bak. nnenum: Verification of relu neural networks with optimized abstraction refinement. In *NASA Formal Methods Symposium*, pages 19–36. Springer, 2021.

[3] Stanley Bak, Changliu Liu, and Taylor Johnson. The second international verification of neural networks competition (vnn-comp 2021): Summary and results. *arXiv preprint arXiv:2109.00498*, 2021.

[4] Stanley Bak, Hoang-Dung Tran, Kerianne Hobbs, and Taylor T. Johnson. Improved geometric path enumeration for verifying relu neural networks. In *Proceedings of the 32nd International Conference on Computer Aided Verification*. Springer, 2020.

[5] Dimitris Bertsimas and Robert Weismantel. *Optimization over integers.* Athena Scientific, 2005.

[6] Robert Bixby and Edward Rothberg. Progress in computational mixed integer programming—a look back from the other side of the tipping point. *Annals of Operations Research*, 149(1):37–41, 2007.

[7] Elena Botoeva, Panagiotis Kouvaros, Jan Kronqvist, Alessio Lomuscio, and Ruth Misener. Efficient verification of relu-based neural networks via dependency analysis. In *AAAI Conference on Artificial Intelligence (AAAI)*, 2020.

[8] Rudy Bunel, Alessandro De Palma, Alban Desmaison, Krishnamurthy Dvijotham, Pushmeet Kohli, Philip H. S. Torr, and M. Pawan Kumar. Lagrangian decomposition for neural network verification. *Conference on Uncertainty in Artificial Intelligence (UAI)*, 2020.

[9] Rudy Bunel, Jingyue Lu, Ilker Turkaslan, P Kohli, P Torr, and P Mudigonda. Branch and bound for piecewise linear neural network verification. *Journal of Machine Learning Research (JMLR)*, 2020.

[10] Rudy R Bunel, Ilker Turkaslan, Philip Torr, Pushmeet Kohli, and Pawan K Mudigonda. A unified view of piecewise linear neural network verification. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2018.

[11] Shaoru Chen, Eric Wong, J Zico Kolter, and Mahyar Fazlyab. Deepsplit: Scalable verification of deep neural networks via operator splitting. *arXiv preprint arXiv:2106.09117*, 2021.

[12] Vasek Chvátal. Edmonds polytopes and a hierarchy of combinatorial problems. *Discrete mathematics*, 4(4):305–337, 1973.

[13] Michele Conforti, Gérard Cornuéjols, and Giacomo Zambelli. *Integer programming / Michele Conforti, Gérard Cornuéjols, Giacomo Zambelli.* Graduate texts in mathematics, . 271. Springer, Cham, 2014.

[14] Harlan Crowder, Ellis L Johnson, and Manfred Padberg. Solving large-scale zero-one linear programming problems. *Operations Research*, 31(5):803–834, 1983.

[15] Sumanth Dathathri, Krishnamurthy Dvijotham, Alexey Kurakin, Aditi Raghunathan, Jonathan Uesato, Rudy R Bunel, Shreya Shankar, Jacob Steinhardt, Ian Goodfellow, Percy S Liang, et al. Enabling certification of verification-agnostic networks via memory-efficient semidefinite programming. *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

[16] Alessandro De Palma, Harkirat Singh Behl, Rudy Bunel, Philip H. S. Torr, and M. Pawan Kumar. Scaling the convex barrier with active sets. *International Conference on Learning Representations (ICLR)*, 2021.

[17] Alessandro De Palma, Rudy Bunel, Alban Desmaison, Krishnamurthy Dvijotham, Pushmeet Kohli, Philip HS Torr, and M Pawan Kumar. Improved branch and bound for neural network verification via lagrangian decomposition. *arXiv preprint arXiv:2104.06718*, 2021.

[18] Krishnamurthy Dvijotham, Robert Stanforth, Sven Gowal, Timothy Mann, and Pushmeet Kohli. A dual approach to scalable verification of deep networks. *Conference on Uncertainty in Artificial Intelligence (UAI)*, 2018.

[19] Ruediger Ehlers. Formal verification of piece-wise linear feed-forward neural networks. In *International Symposium on Automated Technology for Verification and Analysis (ATVA)*, 2017.

[20] Claudio Ferrari, Mark Niklas Muller, Nikola Jovanovic, and Martin Vechev. Complete verification via multi-neuron relaxation guided branch-and-bound. *arXiv preprint arXiv:2205.00263*, 2022.

[21] Paul C Gilmore and Ralph E Gomory. A linear programming approach to the cutting-stock problem. *Operations research*, 9(6):849–859, 1961.

[22] Paul C Gilmore and Ralph E Gomory. A linear programming approach to the cutting stock problem—part ii. *Operations research*, 11(6):863–888, 1963.

[23] Sven Gowal, Krishnamurthy Dvijotham, Robert Stanforth, Rudy Bunel, Chongli Qin, Jonathan Uesato, Timothy Mann, and Pushmeet Kohli. On the effectiveness of interval bound propagation for training verifiably robust models. *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, 2019.

[24] LLC Gurobi Optimization. Gurobi optimizer - gurobi, 2022.

[25] Patrick Henriksen and Alessio Lomuscio. Efficient neural network verification via adaptive refinement and adversarial search. In *ECAI 2020*, pages 2513–2520. IOS Press, 2020.

[26] Patrick Henriksen and Alessio Lomuscio. Deepsplit: An efficient splitting method for neural network verification via indirect effect analysis. In *Proceedings of the 30th international joint conference on artificial intelligence (IJCAI21)*, pages 2549–2555, 2021.

[27] Karla L Hoffman and Manfred Padberg. Solving airline crew scheduling problems by branch-and-cut. *Management science*, 39(6):657–682, 1993.

[28] IBM. Ibm ilog cplex optimizer, 2022.

[29] Ellis L Johnson and Manfred W Padberg. Degree-two inequalities, clique facets, and biperfect graphs. In *North-Holland Mathematics Studies*, volume 66, pages 169–187. Elsevier, 1982.

[30] Guy Katz, Clark Barrett, David L Dill, Kyle Julian, and Mykel J Kochenderfer. Reluplex: An efficient smt solver for verifying deep neural networks. In *International Conference on Computer Aided Verification (CAV)*, 2017.

[31] Guy Katz, Derek A Huang, Duligur Ibeling, Kyle Julian, Christopher Lazarus, Rachel Lim, Parth Shah, Shantanu Thakoor, Haoze Wu, Aleksandar Zeljić, et al. The marabou framework for verification and analysis of deep neural networks. In *International Conference on Computer Aided Verification (CAV)*, 2019.

[32] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *International Conference on Learning Representations (ICLR)*, 2015.

[33] Panagiotis Kouvaros and Alessio Lomuscio. Towards scalable complete verification of relu neural networks via dependency-based branching. In *IJCAI*, pages 2643–2650, 2021.

[34] Changliu Liu and Taylor Johnson. Vnn comp 2020.

[35] László Lovász and Alexander Schrijver. Cones of matrices and set-functions and 0–1 optimization. *SIAM journal on optimization*, 1(2):166–190, 1991.

[36] Jingyue Lu and M Pawan Kumar. Neural network branching for neural network verification. *International Conference on Learning Representation (ICLR)*, 2020.

[37] Hugues Marchand and Laurence A Wolsey. Aggregation and mixed integer rounding to solve mips. *Operations research*, 49(3):363–371, 2001.

[38] Christoph Müller, Francois Serre, Gagandeep Singh, Markus Püschel, and Martin Vechev. Scaling polyhedral neural network verification on gpus. *Proceedings of Machine Learning and Systems*, 3:733–746, 2021.

[39] Christoph Müller, Gagandeep Singh, Markus Püschel, and Martin Vechev. Neural network robustness verification on gpus. *arXiv preprint arXiv:2007.10868*, 2020.

[40] Mark Niklas Müller, Gleb Makarchuk, Gagandeep Singh, Markus Püschel, and Martin Vechev. Precise multi-neuron abstractions for neural network certification. *arXiv preprint arXiv:2103.03638*, 2021.

[41] George L Nemhauser and Laurence A Wolsey. A recursive procedure to generate all cuts for 0–1 mixed integer programs. *Mathematical Programming*, 46(1):379–390, 1990.

[42] Manfred W Padberg, Tony J Van Roy, and Laurence A Wolsey. Valid linear inequalities for fixed charge problems. *Operations Research*, 33(4):842–861, 1985.

[43] Chongli Qin, Brendan O'Donoghue, Rudy Bunel, Robert Stanforth, Sven Gowal, Jonathan Uesato, Grzegorz Swirszcz, Pushmeet Kohli, et al. Verification of non-linear specifications for neural networks. *arXiv preprint arXiv:1902.09592*, 2019.

[44] Aditi Raghunathan, Jacob Steinhardt, and Percy S Liang. Semidefinite relaxations for certifying robustness to adversarial examples. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2018.

[45] Hadi Salman, Greg Yang, Huan Zhang, Cho-Jui Hsieh, and Pengchuan Zhang. A convex relaxation barrier to tight robustness verification of neural networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

[46] Hanif D Sherali and Warren P Adams. *A reformulation-linearization technique for solving discrete and continuous nonconvex problems*, volume 31. Springer Science & Business Media, 2013.

[47] Gagandeep Singh, Rupanshu Ganvir, Markus Püschel, and Martin Vechev. Beyond the single neuron convex barrier for neural network certification. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

[48] Gagandeep Singh, Timon Gehr, Matthew Mirman, Markus Püschel, and Martin Vechev. Fast and effective robustness certification. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2018.

[49] Gagandeep Singh, Timon Gehr, Markus Püschel, and Martin Vechev. Boosting robustness certification of neural networks. In *International Conference on Learning Representations*, 2018.

[50] Gagandeep Singh, Timon Gehr, Markus Püschel, and Martin Vechev. An abstract domain for certifying neural networks. *Proceedings of the ACM on Programming Languages (POPL)*, 2019.

[51] Cheng Tan, Yibo Zhu, and Chuanxiong Guo. Building verified neural networks with specifications for systems. In *Proceedings of the 12th ACM SIGOPS Asia-Pacific Workshop on Systems*, pages 42–47, 2021.

[52] Christian Tjandraatmadja, Ross Anderson, Joey Huchette, Will Ma, Krunal Patel, and Juan Pablo Vielma. The convex relaxation barrier, revisited: Tightened single-neuron relaxations for neural network verification. *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

[53] Vincent Tjeng, Kai Xiao, and Russ Tedrake. Evaluating robustness of neural networks with mixed integer programming. *International Conference on Learning Representations (ICLR)*, 2019.

[54] Hoang-Dung Tran, Xiaodong Yang, Diego Manzanas Lopez, Patrick Musau, Luan Viet Nguyen, Weiming Xiang, Stanley Bak, and Taylor T Johnson. Nnv: The neural network verification tool for deep neural networks and learning-enabled cyber-physical systems. In *International Conference on Computer Aided Verification*, pages 3–17. Springer, 2020.

[55] Shiqi Wang, Kexin Pei, Justin Whitehouse, Junfeng Yang, and Suman Jana. Efficient formal safety analysis of neural networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2018.

[56] Shiqi Wang, Kexin Pei, Justin Whitehouse, Junfeng Yang, and Suman Jana. Formal security analysis of neural networks using symbolic intervals. In *USENIX Security Symposium*, 2018.

[57] Shiqi Wang, Huan Zhang, Kaidi Xu, Xue Lin, Suman Jana, Cho-Jui Hsieh, and J Zico Kolter. Beta-crown: Efficient bound propagation with per-neuron split constraints for neural network robustness verification. *Advances in Neural Information Processing Systems*, 34, 2021.

[58] Eric Wong and Zico Kolter. Provable defenses against adversarial examples via the convex outer adversarial polytope. In *International Conference on Machine Learning (ICML)*, 2018.

[59] Kaidi Xu, Zhouxing Shi, Huan Zhang, Yihan Wang, Kai-Wei Chang, Minlie Huang, Bhavya Kailkhura, Xue Lin, and Cho-Jui Hsieh. Automatic perturbation analysis for scalable certified robustness and beyond. *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

[60] Kaidi Xu, Huan Zhang, Shiqi Wang, Yihan Wang, Suman Jana, Xue Lin, and Cho-Jui Hsieh. Fast and complete: Enabling complete neural network verification with rapid and massively parallel incomplete verifiers. *International Conference on Learning Representations (ICLR)*, 2021.

[61] Huan Zhang, Tsui-Wei Weng, Pin-Yu Chen, Cho-Jui Hsieh, and Luca Daniel. Efficient neural network robustness certification with general activation functions. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2018.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes]
   (c) Did you discuss any potential negative societal impacts of your work? [Yes]
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [Yes]
   (b) Did you include complete proofs of all theoretical results? [Yes]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [No] Code will be available when the final version of the this paper is released.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes]
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [No] Not applicable
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes]
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes]
   (b) Did you mention the license of the assets? [Yes]
   (c) Did you include any new assets either in the supplemental material or as a URL? [Yes]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [No] We only use public data.

(e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [No] We only use public data.

5. If you used crowdsourcing or conducted research with human subjects...

(a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]

(b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]

(c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

In Section A we derive the bound propagation procedure of GCP-CROWN and give a proof for Theorem 3.1 (GCP-CROWN bound propagation). In Section B we give additional background on branch and bound. In section C we give more details of experiments, more results as well as a case study for cutting planes used in GCP-CROWN with MIP cuts.

A The dual problem with cutting planes

In this section we derive the dual formulation for neural network verification problem with general cutting planes (or arbitrarily added linear constraints across any layers). Our derivation is based on the linearly relaxed MIP formulation with original binary variables $\mathbf{z}$ intact, to allow us to add cuts also to the integer variable (the mostly commonly used triangle relaxation does not have $\mathbf{z}$). We first write the full LP relaxation with arbitrary cutting planes, as well as their corresponding dual variables:

$$
\begin{aligned}
& f^*_{\text{LP-cut}} = \min_{\boldsymbol{x},\hat{\boldsymbol{x}},\mathbf{z}} f(x) \\
\text{s.t.} \quad & f(\boldsymbol{x}) = \boldsymbol{x}^{(L)}; \quad \boldsymbol{x}_0 - \epsilon \leq \boldsymbol{x} \leq \boldsymbol{x}_0 + \epsilon; && (2)\\
& \boldsymbol{x}^{(i)} = \mathbf{W}^{(i)}\hat{\boldsymbol{x}}^{(i-1)} + \mathbf{b}^{(i)}; \quad i \in [L], && \Rightarrow \boldsymbol{\nu}^{(i)} \in \mathbb{R}^{d_i} \quad (3)\\
& \hat{x}^{(i)}_j \geq 0;\ j \in \mathcal{I}^{(i)} && \Rightarrow \mu^{(i)}_j \in \mathbb{R} \quad (4)\\
& \hat{x}^{(i)}_j \geq x^{(i)}_j;\ j \in \mathcal{I}^{(i)} && \Rightarrow \tau^{(i)}_j \in \mathbb{R} \quad (5)\\
& \hat{x}^{(i)}_j \leq u^{(i)}_j z^{(i)}_j;\ j \in \mathcal{I}^{(i)} && \Rightarrow \gamma^{(i)}_j \in \mathbb{R} \quad (6)\\
& \hat{x}^{(i)}_j \leq x^{(i)}_j - l^{(i)}_j(1 - z^{(i)}_j);\ j \in \mathcal{I}^{(i)} && \Rightarrow \pi^{(i)}_j \in \mathbb{R} \quad (7)\\
& \hat{x}^{(i)}_j = x^{(i)}_j;\ j \in \mathcal{I}^{+(i)} && (9)\\
& \hat{x}^{(i)}_j = 0;\ j \in \mathcal{I}^{-(i)} && (10)\\
& 0 \leq z^{(i)}_j \leq 1; \quad j \in \mathcal{I}^{(i)},\ i \in [L] && (11)\\
& \sum_{i=1}^{L-1} \left(\boldsymbol{H}^{(i)}\boldsymbol{x}^{(i)} + \boldsymbol{G}^{(i)}\hat{\boldsymbol{x}}^{(i)} + \boldsymbol{Q}^{(i)}\mathbf{z}^{(i)} \right) \leq \boldsymbol{d} && \Rightarrow \boldsymbol{\beta} \in \mathbb{R}^N \quad (15)
\end{aligned}
$$

The Lagrangian function can be constructed as:

$$
\begin{aligned}
f^*_{\text{LP-cut}} = & \min_{\boldsymbol{x},\hat{\boldsymbol{x}},\mathbf{z}} \max_{\boldsymbol{\nu},\boldsymbol{\mu},\boldsymbol{\tau},\boldsymbol{\gamma},\boldsymbol{\pi},\boldsymbol{\beta}} \boldsymbol{x}^{(L)} + \sum_{i=1}^{L} \boldsymbol{\nu}^{(i)\top} \left(\boldsymbol{x}^{(i)} - \mathbf{W}^{(i)}\hat{\boldsymbol{x}}^{(i-1)} + \mathbf{b}^{(i)} \right) \\
& + \sum_{i=1}^{L-1} \sum_{j \in \mathcal{I}^{(i)}} \left[\mu^{(i)}_j(-\hat{x}^{(i)}_j) + \tau^{(i)}_j(x^{(i)}_j - \hat{x}^{(i)}_j) + \gamma^{(i)}_j(\hat{x}^{(i)}_j - u^{(i)}_j z^{(i)}_j) + \pi^{(i)}_j(\hat{x}^{(i)}_j - x^{(i)}_j + l^{(i)}_j - l^{(i)}_j z^{(i)}_j) \right] \\
& + \boldsymbol{\beta}^\top \left[\sum_{i=1}^{L-1} \left(\boldsymbol{H}^{(i)}\boldsymbol{x}^{(i)} + \boldsymbol{G}^{(i)}\hat{\boldsymbol{x}}^{(i)} + \boldsymbol{Q}^{(i)}\mathbf{z}^{(i)} \right) - \boldsymbol{d} \right] \\
\text{s.t.} \ & \hat{x}^{(i)}_j = 0, j \in \mathcal{I}^{-(i)}; \quad \hat{x}^{(i)}_j = x^{(i)}_j, j \in \mathcal{I}^{+(i)}; \quad \boldsymbol{x}_0 - \epsilon \leq \boldsymbol{x} \leq \boldsymbol{x}_0 + \epsilon; \quad 0 \leq z^{(i)}_j \leq 1, j \in \mathcal{I}^{(i)} \\
& \boldsymbol{\mu} \geq 0; \quad \boldsymbol{\tau} \geq 0; \quad \boldsymbol{\gamma} \geq 0; \quad \boldsymbol{\pi} \geq 0; \quad \boldsymbol{\beta} \geq 0
\end{aligned}
$$

Here $\boldsymbol{\mu}, \boldsymbol{\tau}, \boldsymbol{\gamma}, \boldsymbol{\pi}$ are shorthands for all dual variables in each layer and each neuron. Note that for some constraints, their dual variables are not created because they are trivial to handle in the step steps. Rearrange the equation and swap the min and max (strong duality) gives us:

$$
\begin{aligned}
f^*_{\text{LP-cut}} = \max_{\boldsymbol{\nu},\boldsymbol{\mu},\boldsymbol{\tau},\boldsymbol{\gamma},\boldsymbol{\pi},\boldsymbol{\beta}} \min_{\boldsymbol{x},\hat{\boldsymbol{x}},\boldsymbol{z}} \; & (\boldsymbol{\nu}^{(L)}+1)\boldsymbol{x}^{(L)} - \boldsymbol{\nu}^{(1)\top}\mathbf{W}^{(1)}\hat{\boldsymbol{x}}^{(0)} \\
& + \sum_{i=1}^{L-1}\sum_{j\in\mathcal{I}^{+(i)}} \left(\nu_j^{(i)} + \boldsymbol{\beta}^\top \boldsymbol{H}_{:,j}^{(i)} - \boldsymbol{\nu}^{(i+1)\top}\mathbf{W}_{:,j}^{(i+1)} + \boldsymbol{\beta}^\top \boldsymbol{G}_{:,j}^{(i)}\right) x_j^{(i)} \\
& + \sum_{i=1}^{L-1}\sum_{j\in\mathcal{I}^{-(i)}} \left(\nu_j^{(i)} + \boldsymbol{\beta}^\top \boldsymbol{H}_{:,j}^{(i)}\right) x_j^{(i)} \\
& + \sum_{i=1}^{L-1}\sum_{j\in\mathcal{I}^{(i)}} \Big[\left(\nu_j^{(i)} + \boldsymbol{\beta}^\top \boldsymbol{H}_{:,j}^{(i)} + \tau_j^{(i)} - \pi_j^{(i)}\right) x_j^{(i)} \\
& + \left(-\boldsymbol{\nu}^{(i)\top}\mathbf{W}_{:,j}^{(i)} - \mu_j^{(i)} - \tau_j^{(i)} + \gamma_j^{(i)} + \pi_j^{(i)} + \boldsymbol{\beta}^\top \boldsymbol{G}_{:,j}^{(i)}\right) \hat{x}_j^{(i)} \\
& + \left(-u_j^{(i)}\gamma_j^{(i)} - l_j^{(i)}\pi_j^{(i)} + \boldsymbol{\beta}^\top \boldsymbol{Q}_{:,j}^{(i)}\right) z_j^{(i)} \Big] \\
& - \sum_{i=1}^{L} \boldsymbol{\nu}^{(i)\top}\mathbf{b}^{(i)} + \sum_{i=1}^{L-1}\sum_{j\in\mathcal{I}^{(i)}} \pi_j^{(i)} l_j^{(i)} - \boldsymbol{\beta}^\top \boldsymbol{d} \\
\text{s.t.} \quad & \boldsymbol{x}_0 - \epsilon \le \boldsymbol{x} \le \boldsymbol{x}_0 + \epsilon; \quad 0 \le z_j^{(i)} \le 1, j \in \mathcal{I}^{(i)} \\
& \boldsymbol{\mu} \ge 0; \quad \boldsymbol{\tau} \ge 0; \quad \boldsymbol{\gamma} \ge 0; \quad \boldsymbol{\pi} \ge 0; \quad \boldsymbol{\beta} \ge 0
\end{aligned}
$$

Here $\mathbf{W}_{:,j}^{(i+1)}$ denotes the j-th column of $\mathbf{W}^{(i+1)}$. Note that for the term involving $j \in \mathcal{I}^{+(i)}$ we have replaced $\hat{\boldsymbol{x}}^{(i)}$ with $\boldsymbol{x}^{(i)}$ to obtain the above equation. For the the term $x_j^{(i)}, j \in \mathcal{I}^{+(i)}$ it is always 0 so does not appear.

Solving the inner minimization gives us the dual formulation:

$$
\begin{aligned}
f^*_{\text{LP-cut}} = \max_{\boldsymbol{\nu},\boldsymbol{\mu},\boldsymbol{\tau},\boldsymbol{\gamma},\boldsymbol{\pi},\boldsymbol{\beta}} \; & -\epsilon\|\boldsymbol{\nu}^{(1)\top}\mathbf{W}^{(1)}\boldsymbol{x}_0\|_1 - \sum_{i=1}^{L}\boldsymbol{\nu}^{(i)\top}\mathbf{b}^{(i)} - \boldsymbol{\beta}^\top \boldsymbol{d} \\
& + \sum_{i=1}^{L-1}\sum_{j\in\mathcal{I}^{(i)}} \left[\pi_j^{(i)} l_j^{(i)} - \text{ReLU}(u_j^{(i)}\gamma_j^{(i)} + l_j^{(i)}\pi_j^{(i)} - \boldsymbol{\beta}^\top \boldsymbol{Q}_{:,j}^{(i)})\right] \qquad (17) \\
\text{s.t.} \; & \boldsymbol{\nu}^{(L)} = -1 \qquad (18) \\
& \nu_j^{(i)} = \boldsymbol{\nu}^{(i+1)\top}\mathbf{W}_{:,j}^{(i+1)} - \boldsymbol{\beta}^\top(\boldsymbol{H}_{:,j}^{(i)} + \boldsymbol{G}_{:,j}^{(i)}), \quad \text{for } j \in \mathcal{I}^{+(i)},\ i \in [L-1] \qquad (19) \\
& \nu_j^{(i)} = -\boldsymbol{\beta}^\top \boldsymbol{H}_{:,j}^{(i)}, \quad \text{for } j \in \mathcal{I}^{-(i)},\ i \in [L-1] \qquad (20) \\
& \nu_j^{(i)} = \pi_j^{(i)} - \tau_j^{(i)} - \boldsymbol{\beta}^\top \boldsymbol{H}_{:,j}^{(i)} \quad \text{for } j \in \mathcal{I}^{(i)},\ i \in [L-1] \qquad (21) \\
& \left(\pi_j^{(i)} + \gamma_j^{(i)}\right) - \left(\mu_j^{(i)} + \tau_j^{(i)}\right) = \boldsymbol{\nu}^{(i+1)\top}\mathbf{W}_{:,j}^{(i+1)} - \boldsymbol{\beta}^\top \boldsymbol{G}_{:,j}^{(i)}, \quad \text{for } j \in \mathcal{I}^{(i)},\ i \in [L-1] \qquad (22) \\
\text{s.t.} \; & \boldsymbol{\mu} \ge 0; \quad \boldsymbol{\tau} \ge 0; \quad \boldsymbol{\gamma} \ge 0; \quad \boldsymbol{\pi} \ge 0; \quad \boldsymbol{\beta} \ge 0
\end{aligned}
$$

The $\|\cdot\|_1$ comes from minimizing over $\boldsymbol{x}_0$ with the constraint $\boldsymbol{x}_0 - \epsilon \le \boldsymbol{x} \le \boldsymbol{x}_0 + \epsilon$, and the $\text{ReLU}(\cdot)$ term comes from minimizing over $z_j^{(i)}$ with the constraint $0 \le z_j^{(i)} \le 1$.

Before we give the bound propagation procedure, we first give a technical lemma:

Lemma A.1. *Given $u \ge 0$, $l \le 0$, $\pi \ge 0$, $\gamma \ge 0$, and $\pi + \gamma = C$, and define the function:*

$$
g(\pi, \gamma) = -\text{ReLU}(u\gamma + l\pi + q) + l\pi
$$

Then

$$\max_{\pi\geq 0,\gamma\geq 0} g(\pi,\gamma) = \begin{cases} l\pi^*, & \textit{if } -uC \leq q \leq -lC \\ 0, & \textit{if } q < -uC \\ -q, & \textit{if } q > -lC \end{cases}$$

where the optimal values for π and γ are:

$$\pi^* = \max\left(\min\left(\frac{uC+q}{u-l}, C\right), 0\right), \quad \gamma^* = \max\left(\min\left(\frac{-lC-q}{u-l}, C\right), 0\right)$$

Proof. Case 1: when $u\gamma + l\pi + q \geq 0$, the objective becomes:

$$\begin{aligned} \max_{\pi\geq 0,\gamma\geq 0} g(\pi,\gamma) &= -u\gamma - q \\ \text{s.t. } \pi + \gamma &= C \\ u\gamma + l\pi + q &\geq 0 \end{aligned}$$

Since q is a constant and the object only involves a non-negative variable γ with non-positive coefficient $-u$, γ needs to be as smaller as possible as long as the constraint is satisfied. Substitute $\pi = C - \gamma$ into the constraint $u\gamma + l\pi + q \geq 0$,

$$u\gamma + l(C-\gamma) + q \geq 0 \Rightarrow \gamma \geq \frac{-lC-q}{u-l}$$

Considering γ is within $[0, C]$, the optimal γ and π are:

$$\gamma^* = \max\left(\min\left(\frac{-lC-q}{u-l}, C\right)\right), \quad \pi^* = \max\left(\min\left(\frac{uC+q}{u-l}, C\right)\right)$$

Case 2: when $u\gamma + l\pi + q \leq 0$, the objective becomes:

$$\begin{aligned} \max_{\pi\geq 0,\gamma\geq 0} g(\pi,\gamma) &= l\pi \\ \text{s.t. } \pi + \gamma &= C \\ u\gamma + l\pi + q &\leq 0 \end{aligned}$$

Since q is a constant and the object only involves a non-negative variable π with non-positive coefficient l, π needs to be as smaller as possible as long as the constraint is satisfied. Substitute $\gamma = C - \pi$ into the constraint $u\gamma + l\pi + q \leq 0$,

$$u(C-\pi) + l\pi + q \leq 0 \Rightarrow \pi \geq \frac{uC+q}{u-l}$$

The optimal π^* and γ^* are the same as in case 1. The optimal objective can be obtained by substitute π^* and γ^* into $g(\pi,\gamma)$. The objective depends on q because when q is too large ($q \geq -lC$) or too small ($q \leq -uC$), π^* and γ^* are fixed at either 0 or C. □

With this lemma, we are now ready to prove Theorem 3.1, the bound propagation rule with general cutting planes (GCP-CROWN):

Theorem 3.1 (Bound propagation with general cutting planes)**.** *Given the following bound propagation rule on $\boldsymbol{\nu}$ with optimizable parameter $0 \leq \alpha_j^{(i)} \leq 1$ and $\boldsymbol{\beta} \geq 0$:*

$$\begin{aligned}
&\boldsymbol{\nu}^{(L)} = -1 \\
&\nu_j^{(i)} = \boldsymbol{\nu}^{(i+1)\top}\mathbf{W}_{:,j}^{(i+1)} - \boldsymbol{\beta}^\top(\boldsymbol{H}_{:,j}^{(i)} + \boldsymbol{G}_{:,j}^{(i)}), && j \in \mathcal{I}^{+(i)},\ i \in [L-1] \\
&\nu_j^{(i)} = -\boldsymbol{\beta}^\top \boldsymbol{H}_{:,j}^{(i)}, && j \in \mathcal{I}^{-(i)},\ i \in [L-1] \\
&\hat{\nu}_j^{(i)} := \boldsymbol{\nu}^{(i+1)\top}\mathbf{W}_{:,j}^{(i+1)} - \boldsymbol{\beta}^\top \boldsymbol{G}_{:,j}^{(i)} && j \in \mathcal{I}^{(i)},\ i \in [L-1] \\
&\nu_j^{(i)} := \max\left(\min\left(\frac{u_j^{(i)}[\hat{\nu}_j^{(i)}]_+ + \boldsymbol{\beta}^\top \boldsymbol{Q}_{:,j}^{(i)}}{u_j^{(i)} - l_j^{(i)}}, [\hat{\nu}_j^{(i)}]_+\right), 0\right) - \alpha_j^{(i)}[\hat{\nu}_j^{(i)}]_- - \boldsymbol{\beta}^\top \boldsymbol{H}_{:,j}^{(i)} && j \in \mathcal{I}^{(i)},\ i \in [L-1]
\end{aligned}$$

Then $f^*_{LP\text{-}cut}$ is lower bounded by the following objective with any valid $0 \leq \boldsymbol{\alpha} \leq 1$ and $\boldsymbol{\beta} \geq 0$:

$$g(\boldsymbol{\alpha}, \boldsymbol{\beta}) = -\epsilon \|\boldsymbol{\nu}^{(1)\top} \mathbf{W}^{(1)} \boldsymbol{x}_0\|_1 - \sum_{i=1}^{L} \boldsymbol{\nu}^{(i)\top} \mathbf{b}^{(i)} - \boldsymbol{\beta}^\top \boldsymbol{d} + \sum_{i=1}^{L-1} \sum_{j \in \mathcal{I}^{(i)}} h_j^{(i)}(\boldsymbol{\beta})$$

where $h_j^{(i)}(\boldsymbol{\beta})$ is defined as:

$$h_j^{(i)}(\boldsymbol{\beta}) = \begin{cases} \frac{u_j^{(i)} l_j^{(i)} [\hat{\nu}_j^{(i)}]_+ + l_j^{(i)} \boldsymbol{\beta}^\top \boldsymbol{Q}_{:,j}^{(i)}}{u_j^{(i)} - l_j^{(i)}} & \textit{if} \;\; l_j^{(i)} [\hat{\nu}_j^{(i)}]_+ \leq \boldsymbol{\beta}^\top \boldsymbol{Q}_{:,j}^{(i)} \leq u_j^{(i)} [\hat{\nu}_j^{(i)}]_+ \\ 0 & \textit{if} \;\; \boldsymbol{\beta}^\top \boldsymbol{Q}_{:,j}^{(i)} \geq u_j^{(i)} [\hat{\nu}_j^{(i)}]_+ \\ \boldsymbol{\beta}^\top \boldsymbol{Q}_{:,j}^{(i)} & \textit{if} \;\; \boldsymbol{\beta}^\top \boldsymbol{Q}_{:,j}^{(i)} \leq l_j^{(i)} [\hat{\nu}_j^{(i)}]_+ \end{cases}$$

Proof. Given (22), for $j \in \mathcal{I}^{(i)}$, observing that the upper and lower bounds of ReLU relaxations for a single neuron cannot be tight simultaneously [58], $\pi_j^{(i)} + \gamma_j^{(i)}$ and $\mu_j^{(i)} + \tau_j^{(i)}$ cannot be both non-zero[1]. Thus, we have

$$\pi_j^{(i)} + \gamma_j^{(i)} = \left[\boldsymbol{\nu}^{(i+1)\top} \mathbf{W}_{:,j}^{(i+1)} - \boldsymbol{\beta}^\top \boldsymbol{G}_{:,j}^{(i)} \right]_+ := \left[\hat{\nu}_j^{(i)} \right]_+, \tag{23}$$

$$\mu_j^{(i)} + \tau_j^{(i)} = \left[\boldsymbol{\nu}^{(i+1)\top} \mathbf{W}_{:,j}^{(i+1)} - \boldsymbol{\beta}^\top \boldsymbol{G}_{:,j}^{(i)} \right]_- := \left[\hat{\nu}_j^{(i)} \right]_-. \tag{24}$$

To avoid clutter, we define $\hat{\nu}_j^{(i)} := \boldsymbol{\nu}^{(i+1)\top} \mathbf{W}_{:,j}^{(i+1)} - \boldsymbol{\beta}^\top \boldsymbol{G}_{:,j}^{(i)}$, and we define $[\,\cdot\,]_+ := \max(0, \cdot)$ and $[\,\cdot\,]_- := \min(0, \cdot)$.

To derived the proposed bound propagation rule, in (17), we must eliminate variable $\gamma_j^{(i)}$ and $\pi_j^{(i)}$. Ignoring terms related to $\boldsymbol{\nu}$, we observe that we can optimize the term $\pi_j^{(i)} l_j^{(i)} - \mathrm{ReLU}(u_j^{(i)} \gamma_j^{(i)} + l_j^{(i)} \pi_j^{(i)} - \boldsymbol{\beta}^\top \boldsymbol{Q}_{:,j}^{(i)})$ for each j for $\mathcal{I}^{(i)}$ individually. We seek the optimal solution for the follow optimization problem on function h:

$$\max_{\pi_j^{(i)} \geq 0, \gamma_j^{(i)} \geq 0} h_j^{(i)}(\pi_j^{(i)}, \gamma_j^{(i)}; \boldsymbol{\beta}) : = \pi_j^{(i)} l_j^{(i)} - \mathrm{ReLU}(u_j^{(i)} \gamma_j^{(i)} + l_j^{(i)} \pi_j^{(i)} - \boldsymbol{\beta}^\top \boldsymbol{Q}_{:,j}^{(i)}) \tag{25}$$

$$\text{s.t.} \;\; \pi_j^{(i)} + \gamma_j^{(i)} = \left[\hat{\nu}_j^{(i)} \right]_+$$

$$\pi_j^{(i)} \geq 0; \quad \gamma_j^{(i)} \geq 0$$

Note that we optimize over $\pi_j^{(i)}$ and $\gamma_j^{(i)}$ here and treat $\boldsymbol{\beta}$ as a constant. Applying Lemma A.1 with $\pi = \pi_j^{(i)}$, $\gamma = \gamma_j^{(i)}$, $u = u_j^{(i)}$, $l = l_j^{(i)}$, $q = -\boldsymbol{\beta}^\top \boldsymbol{Q}_{:,j}^{(i)}$, $C = [\hat{\nu}_j^{(i)}]_+$ we obtain the optimal objective for (25):

$$h_j^{(i)}(\boldsymbol{\beta}) = \begin{cases} l_j^{(i)} \pi_j^{(i)} & \text{if} \;\; l_j^{(i)} [\hat{\nu}_j^{(i)}]_+ \leq \boldsymbol{\beta}^\top \boldsymbol{Q}_{:,j}^{(i)} \leq u_j^{(i)} [\hat{\nu}_j^{(i)}]_+ \\ 0 & \text{if} \;\; \boldsymbol{\beta}^\top \boldsymbol{Q}_{:,j}^{(i)} \geq u_j^{(i)} [\hat{\nu}_j^{(i)}]_+ \\ \boldsymbol{\beta}^\top \boldsymbol{Q}_{:,j}^{(i)} & \text{if} \;\; \boldsymbol{\beta}^\top \boldsymbol{Q}_{:,j}^{(i)} \leq l_j^{(i)} [\hat{\nu}_j^{(i)}]_+ \end{cases}, \quad \pi_j^{(i)} = \max\left(\min\left(\frac{u_j^{(i)} [\hat{\nu}_j^{(i)}]_+ + \boldsymbol{\beta}^\top \boldsymbol{Q}_{:,j}^{(i)}}{u_j^{(i)} - l_j^{(i)}}, C \right), 0 \right)$$

Substitute this solution of $\pi_j^{(i)}$ into (21), and also observe that due to (24), $\tau_j^{(i)}$ is a variable between 0 and $\left[\hat{\nu}_j^{(i)} \right]_-$, so we can rewrite (21) as:

$$\nu_j^{(i)} := \max\left(\min\left(\frac{u_j^{(i)} [\hat{\nu}_j^{(i)}]_+ + \boldsymbol{\beta}^\top \boldsymbol{Q}_{:,j}^{(i)}}{u_j^{(i)} - l_j^{(i)}}, [\hat{\nu}_j^{(i)}]_+ \right), 0 \right) - \alpha_j^{(i)} [\hat{\nu}_j^{(i)}]_- - \boldsymbol{\beta}^\top \boldsymbol{H}_{:,j}^{(i)}$$

[1] In theorey, it is possible that $\tau_j^{(i)} \neq 0$, $\gamma_j^{(i)} \neq 0$, $\pi_j^{(i)} = \mu_j^{(i)} = 0$ or $\pi_j^{(i)} \neq 0$, $\mu_j^{(i)} \neq 0$, $\tau_j^{(i)} = \gamma_j^{(i)} = 0$. This situation may happen when $u_j^{(i)}$ or $l_j^{(i)}$ is exactly tight and the solution $x_j^{(i)}$ is at the intersection of one lower and upper linear equatlities. Practically, this can be avoided by adding a small δ to $u_j^{(i)}$ or $l_j^{(i)}$, and in most scenarios $u_j^{(i)}$ are $l_j^{(i)}$ are loose bounds so this situation will not occur. The same argument is also applicable to [58].

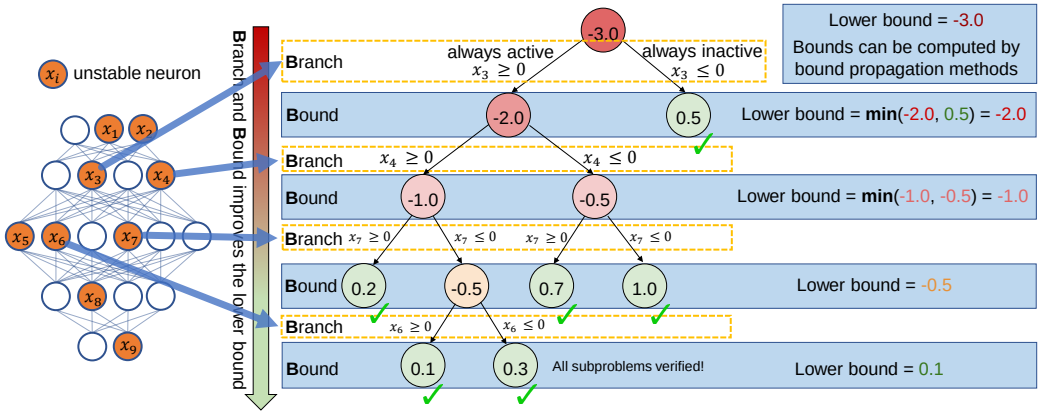

Figure 3: **B**ranch and **B**ound (BaB) for neural network verification. A *unstable* ReLU neuron (e.g., x_1 may receive either negative or positive inputs for some input $\boldsymbol{x}$ in perturbation set $\mathcal{C}$. On the other hand, a stable ReLU neuron is always positive or negative for all $\boldsymbol{x} \in \mathcal{C}$, so it is a linear operation and branching is not needed.

Here $0 \leq \alpha_j^{(i)} \leq 1$ is an optimizable parameter that can be updated via its gradient during bound propagation, and any $0 \leq \alpha_j^{(i)} \leq 1$ produces valid lower bounds. All above substitutions replace each dual variable $\boldsymbol{\pi}$, $\boldsymbol{\gamma}$ and $\boldsymbol{\mu}$ to a valid setting in the dual formulation, so the final objective $g(\boldsymbol{\alpha}, \boldsymbol{\beta})$ is always a sound lower bound of $f^*_{\text{LP-cut}}$. This theorem establishes the soundness of GCP-CROWN. □

Note that an optimal setting of $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ do not necessarily lead to the optimal primal value $f^*_{\text{LP-cut}}$. The main reason is that (25) gives a closed-form solution for $\boldsymbol{\pi}$ and $\boldsymbol{\gamma}$ to eliminate these variables without considering other dual variables like $\boldsymbol{\nu}$ to simplify the bound propagation process. To achieve theoretical optimality, $\boldsymbol{\pi}$ and $\boldsymbol{\gamma}$ can also be optimized.

B More technical details and background of GCP-CROWN with MIP cuts

Branch-and-bound Our work is based on branch-and-bound (BaB), a powerful framework for neural network verification [10] which many state-of-the-art verifiers are based on [57, 16, 26, 20]. Here we give a brief introduction for the concept of branch-and-bound for readers who are not familiar with this field.

We illustrate the branch-and-bound process in Figure 3. Neural network verification seeks to minimize the objective (1): $f^* = \min_{\boldsymbol{x}} f(\boldsymbol{x}), \forall \boldsymbol{x} \in \mathcal{C}$, where $f(\boldsymbol{x})$ is a ReLU network under our setting. A positive f^* indicates that the network can be verified. However, since ReLU neurons are non-linear, this optimization problem is non-convex, and it usually requires us to relax ReLU neurons with linear constraints (e.g., the LP relaxation in (11)) to obtain a lower bound for objective f^*. This lower bound (-3.0 at the root node in Figure 3) might become negative, even if f^* (the unknown ground-truth) is positive. Branch-and-bound is a systematic method to tighten this lower bound. First, we split an unstable ReLU neuron, such as x_3, into two cases: $x_3 \geq 0$ and $x_3 \leq 0$ (the *branching step*), producing two subproblems each with an additional constraint $x_3 \geq 0$ or $x_3 \leq 0$. In each subproblem, neuron x_3 does not need to be relaxed anymore, so the lower bound of f^* becomes tighter in each of the two subdomains in the subsequent *bounding step* (-3.0 becomes -2.0 and 0.5) in Figure 3. Any subproblem with a positive lower bound is successfully verified and no further split is needed. Then, we repeat the branching and bounding steps on subproblems that still have a negative lower bound. We terminate when all unstable neurons are split or all subproblems are verified. In Figure 3, all leaf subproblems have positive lower bounds so the network is verified. On the other hand, if we have split all unstable neurons and there still exist domains with negative bounds, a counter-example can be constructed.

The contribution of this work is to improve the bounding step using cutting planes. As one can observe in Figure 3, if we can make the bounds tighter, they approach to zero faster and less branching is needed. Since the number of the subproblems may be exponential to the number of unstable neurons

in the worst case, it is crucial to improve the bounds quickly. Informally, cutting planes are additional valid constraints for the relaxed subproblems, and adding these additional constraints as in (16) makes the lower bounds tighter. We refer the readers to integer programming literature [5, 13] for more details on cutting plane methods. Existing neural network verifiers mostly use bound propagation method [61, 50, 60, 57, 16] for the bounding step thanks to their efficiency and scalability, but none of them can handle general cutting planes. Our GCP-CROWN is the first bound propagation method that supports general cutting planes, and we demonstrated its effectiveness in Section 4 when combined with cutting planes generated by a MIP solver.

Soundness and Completeness GCP-CROWN is sound because Theorem 3.1 guarantees that we always obtain a sound lower bound of the verification problem as long as valid cutting planes (generated by a MIP solver in our case) are added. Our method, when combined with branch-and-bound, is also complete because we provide a strict improvement in the bounding step over existing methods such as [57], and the arguments for completeness in existing verifiers using branch-and-bound on splitting ReLU neurons such as [9, 57] are still valid for this work.

Limitations of this work Our work generalizes existing bound propagation methods that can handle only simple constraints (such as neuron split constraints in β-CROWN [57]) to general constraints, and we share a few common limitations as in previous works [10, 60, 16]: the branch-and-bound procces and bound propagation procedure are developed on ReLU networks, and it can be non-trivial to extend it to neural networks with non-piecewise-linear operations. In addition, we currently directly use cutting planes generated by a generic MIP solver, and there might exist stronger and faster cutting plane methods that can exploit the structure of the neural network verification problem. We hope these limitations can be addressed in future works.

Potential Negative Societal Impact Our work focuses on formally proving desired properties of a neural network under investigation such as safety and robustness, which is an important direction of trustworthy machine learning and has overall positive societal impact. Since our verifier is a complete verifier, it might be possible to use it to find weakness of a neural network and guide adversarial attacks. However, we believe that formally characterizing a model's behavior and potential weakness is important for building robust models and preventing real-world malicious attack.

C More details on experiments

C.1 Experimental Setup

Our experiments are conducted on a desktop with an AMD Ryzen 9 5950X CPU, one NVIDIA RTX 3090 GPU (24GB GPU memory), and 64GB CPU memory. Our implementation is based on the open-source α,β-CROWN verifier[2] with cutting plane related code added. Both α-CROWN+MIP and GCP-CROWN with MIP cuts use all 16 CPU cores and 1 GPU. `gurobi` is used as the MIP solver in α-CROWN+MIP. Although `gurobi` usually outperforms other MIP solvers for NN verification problems, it cannot export cutting planes, so our cutting planes are acquired by the `cplex` [28] solver (version 22.1.0.0). We use the Adam optimizer [32] to solve both $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ with 20 iterations. The learning rates are set as 0.1 and 0.02 (0.01 for `oval20`) for optimizing $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ respectively. We decay the learning rates with a factor of 0.9 (0.8 for `oval20`) per iteration. Timeout for properties in `oval20` are set as 1 hour follow the original benchmark. Timeout for `oval21` and `cifar10-resnet` are set as 720s and 300s respectively, the same as in VNN-COMP 2021. Timeout for SDP-FO models are set as 600s for α-CROWN+MIP and MN-BaB, and a shorter 200s timeout is used for GCP-CROWN with MIP cuts.

We summarize the model structures and batch size used in our experiments in Table 4. The CIFAR-10 Base, Wide and Deep models are used in `oval20` and `oval21` benchmarks.

C.2 A case study on cutting planes

The effectiveness of GCP-CROWN with MIP cuts motivates us to take a more careful look at the cutting planes generated by off-the-shelf MIP solvers, and understand how well it can contribute to

[2] `https://github.com/huanzhang12/alpha-beta-CROWN`

Table 4: Model structures used in our experiments. The notation Conv(a, b, c) stands for a conventional layer with a input channel, b output channels and a kernel size of $c \times c$. Linear(a, b) stands for a fully connected layer with a input features and b output features. ResBlock(a, b) stands for a residual block that has a input channels and b output channels. We have ReLU activation functions between two consecutive linear or convolutional layers.

Model name	Model structure	Batch size
Base (CIFAR-10)	Conv(3, 8, 4) - Conv(8, 16, 4) - Linear(1024, 100) - Linear(100, 10)	1024
Wide (CIFAR-10)	Conv(3, 16, 4) - Conv(16, 32, 4) - Linear(2048, 100) - Linear(100, 10)	1024
Deep (CIFAR-10)	Conv(3, 8, 4) - Conv(8, 8, 3) - Conv(8, 8, 3) - Conv(8, 8, 4) - Linear(412, 100) - Linear(100, 10)	1024
`cifar10-resnet2b`	Conv(3, 8, 3) - ResBlock(8, 16) - ResBlock(16, 16)) - Linear(1024, 100) - Linear(100, 10)	2048
`cifar10-resnet4b`	Conv(3, 16, 3) - ResBlock(16, 32) - ResBlock(32, 32)) - Linear(512, 100) - Linear(100, 10)	2048
CNN-A-Adv (MNIST)	Conv(1, 16, 4) - Conv(16, 32, 4) - Linear(1568, 100) - Linear(100, 10)	4096
CNN-A-Adv/-4 (CIFAR-10)	Conv(3, 16, 4) - Conv(16, 32, 4) - Linear(2048, 100) - Linear(100, 10)	4096
CNN-B-Adv/-4 (CIFAR-10)	Conv(3, 32, 5) - Conv(32, 128, 4) - Linear(8192, 250) - Linear(250, 10)	1024
CNN-A-Mix/-4 (CIFAR-10)	Conv(3, 16, 4) - Conv(16, 32, 4) - Linear(2048, 100) - Linear(100, 10)	4096

tighten the lower bounds and strengthen verification performance. We use the `oval21` as a sample case study because GCP-CROWN with MIP cuts is very effective on this benchmark.

The `oval21` benchmark has 30 instances, each with 9 target labels (properties) to verify. Among the total of 270 properties, we filter out the easy cases where fast incomplete verifiers like α-CROWN can verify directly, and 39 *hard properties* remain which must be solved using branch and bound and/or cutting planes. Cutting planes are generated on these hard properties and they greatly help GCP-CROWN.

Number of cuts used to verify each property. The maximal number of cuts applied per property is 4,162 and the minimal number is 318. On average, we have 1,683 cuts applied to our GCP-CROWN to solve these hard properties.

Improvements on lower bounds. In branch-and-bound, we lower bound the objective of Eq. 1 with ReLU split constraints [57]. A tighter lower bound can reduce the number of branches used and usually leads to stronger verification. We measure how well the cuts generated by off-the-shelf solvers in improving the tightness of the lower bound for verification. Without branching and cutting planes, the average α-CROWN lower bound is -2.54. With generated cutting planes, we can improve the lower bound by 0.51 on average and can directly verify 4 out of 39 hard properties without branching. The lower bound without branching can be maximally improved by 1.54 and minimally improved by 0.04.

Structure of Generated Cuts. Finally, we investigate the variables involved in each generated cutting plane. In total, the MIP solver generates 65,647 cutting planes in total for the 39 hard properties. 65,301 of the cuts involve variables across multiple layers. It indicates that single layer cutting planes (e.g., constraints involving pre- and post-activation variables of a single ReLU layer only) commonly used in previous works [47, 40] are not optimal in general. Additionally, all of 65,647 cuts have at least one ReLU *integer variable* $\mathbf{z}^{(i)}$ used, which was not supported in existing bound propagation methods. 65,197 of all cuts involve at least one variable of input neurons. 23,600 of all cuts have at least one pre-ReLU variables and 51,617 of them have at least one post-ReLU variables.