

489 Appendix

490 Contents

491	A Formulations	14
492	A.1 MIP Formulation and LP & Planet Relaxation	14
493	A.2 Formulating Bounding Hyperplanes in Linear Bound Propagation	15
494	A.3 Verification Properties Given as Boolean Logical Formulae	17
495	B Proofs	17
496	B.1 Statement and proof of Lemma B.1	17
497	B.2 Proof of Theorem 3.1	18
498	B.3 Proof of Theorem 3.2	19
499	B.4 Proof of Order Dependency	20
500	C Algorithms	21
501	C.1 Overview	21
502	C.2 Sequential Clipping for Multiple Constraints	21
503	C.3 Algorithms for Clip-and-Verify in Input Branch-and-bound Scheme	22
504	C.4 Algorithms for Clip-and-Verify in ReLU Branch-and-bound Scheme	22
505	D Experiments	25
506	D.1 Experiments Settings	25
507	D.2 Ablation Studies	25
508	D.3 Details of Heuristics	26
509	E Limitation and Broader Impacts	27

510 A Formulations

511 A.1 MIP Formulation and LP & Planet Relaxation

512 **The MIP Formulation** The mixed integer programming (MIP) formulation is the root of many
513 NN verification algorithms. Given the RELU activation function’s piecewise linearity, the model
514 requires binary encoding variables, or ReLU indicators δ only for unstable neurons. We formulate
515 the optimization problem aiming to minimize the function $f(\mathbf{x})$, subject to a set of constraints that

516 encapsulate the DNN’s architecture and the perturbation limits around a given input \mathbf{x} , as follows:

$$f^* = \min_{\mathbf{x}, \hat{\mathbf{z}}, \delta} f(\mathbf{x}) \quad \text{s.t. } f(\mathbf{x}) = z^{(L)}; \hat{\mathbf{z}}^{(0)} = \mathbf{x} \in \mathcal{X} \quad (9a)$$

$$\hat{\mathbf{z}}^{(i)} = \mathbf{W}^{(i)} \hat{\mathbf{z}}^{(i-1)} + \mathbf{b}^{(i)}; \quad i \in [L] \quad (9b)$$

$$\mathcal{I}^{+(i)} := \{j : l_j^{(i)} \geq 0\} \quad (9c)$$

$$\mathcal{I}^{-(i)} := \{j : u_j^{(i)} \leq 0\} \quad (9d)$$

$$\mathcal{I}^{(i)} := \{j : l_j^{(i)} < 0, u_j^{(i)} > 0\} \quad (9e)$$

$$\mathcal{I}^{+(i)} \cup \mathcal{I}^{-(i)} \cup \mathcal{I}^{(i)} = \mathcal{J}^i \quad (9f)$$

$$\hat{x}_j^{(i)} \geq 0; j \in \mathcal{I}^{(i)}, i \in [L-1] \quad (9g)$$

$$\hat{z}_j^{(i)} \geq z_j^{(i)}; j \in \mathcal{I}^{(i)}, i \in [L-1] \quad (9h)$$

$$\hat{z}_j^{(j)} \leq u_j^{(i)} \delta_j^{(i)}; j \in \mathcal{I}^{(i)}, i \in [L-1] \quad (9i)$$

$$\hat{z}_j^{(i)} \leq z_j^{(i)} - l_j^{(i)} (1 - \delta_j^{(i)}); j \in \mathcal{I}^{(i)}, i \in [L-1] \quad (9j)$$

$$\delta_j^{(i)} \in \{0, 1\}; j \in \mathcal{I}^{(i)}, i \in [L-1] \quad (9k)$$

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

$$\hat{z}_j^{(i)} = 0; j \in \mathcal{I}^{-(i)}, i \in [L-1] \quad (9m)$$

517 To initialize intermediate bounds for each neuron, we replace the original objective $f(\mathbf{x})$ with the
 518 neuron’s pre-activation value $z_j^{(i)}$. This lets us solve the following bounds for every neuron j in layer
 519 i , with $i \in [L-1]$ and $j \in \mathcal{J}^{(i)}$:

$$l_j^{(i)} = \min_{\mathbf{x} \in \mathcal{X}} f_j^{(i)}(\mathbf{x}), \quad u_j^{(i)} = \max_{\mathbf{x} \in \mathcal{X}} f_j^{(i)}(\mathbf{x}).$$

520 Here, the set $\mathcal{J}^{(i)}$ comprises all neurons in layer i , which can be categorized into three groups:
 521 ‘active’ ($\mathcal{I}^{+(i)}$), ‘inactive’ ($\mathcal{I}^{-(i)}$), and ‘unstable’ ($\mathcal{I}^{(i)}$).

522 Next, the MIP formulation is initialized with the constraints

$$l_j^{(i)} \leq z_j^{(i)} \leq u_j^{(i)},$$

523 across all neurons and layers i . These bounds can be computed recursively, propagating from the first
 524 layer up to the i -th layer. However, since MIP problems involve integer variables, they are generally
 525 NP-hard, reflecting the computational challenge of this approach.

526 **The LP and Planet relaxation.** By relaxing the binary variables in (9k) to $\delta_j^{(i)} \in [0, 1]$, $j \in \mathcal{I}^{(i)}$, $i \in$
 527 $[L-1]$, we can get the LP relaxation formulation. By replacing the constraints in (9i), (9j), (9k) with

$$\hat{z}_j^{(i)} \leq \frac{u_j^{(i)}}{u_j^{(i)} - l_j^{(i)}} (z_j^{(i)} - l_j^{(i)}); \quad j \in \mathcal{I}^{(i)}, i \in [L-1] \quad (10)$$

528 we can eliminate the δ variables and get the well-known Planet relaxation formulation. Both of these
 529 two relaxations are solvable in polynomial time to yield lower bounds.

530 A.2 Formulating Bounding Hyperplanes in Linear Bound Propagation

531 In a feedforward network, $\underline{\mathbf{A}}^{(i)}$, $\overline{\mathbf{A}}^{(i)}$, $\underline{\mathbf{c}}^{(i)}$ and $\overline{\mathbf{c}}^{(i)}$ must be derived for every linear layer preceding an
 532 activation layer, as well as the final layer of the network. In order to derive the hyperplane coefficients
 533 ($\underline{\mathbf{A}}^{(i)} / \overline{\mathbf{A}}^{(i)}$) and biases ($\underline{\mathbf{c}}^{(i)} / \overline{\mathbf{c}}^{(i)}$), at this i^{th} layer, all preceding activation layers must have already
 534 had their inputs bounded. The following lemma describes how a ReLU activation layer may be
 535 relaxed which will be useful for defining bounding hyperplanes, $\underline{\mathbf{A}}^{(i)}$, $\overline{\mathbf{A}}^{(i)}$, $\underline{\mathbf{c}}^{(i)}$ and $\overline{\mathbf{c}}^{(i)}$.

536 **Lemma A.1.** (*Relaxation of a ReLU layer in CROWN*). Given the lower and upper bounds of $\mathbf{z}_j^{(i-1)}$,
 537 denoted as $\mathbf{l}_j^{(i-1)}$ and $\mathbf{u}_j^{(i-1)}$, respectively, the linear layer proceeding the ReLU activation layer
 538 may be lower-bounded element-wise by the following inequality:

$$\mathbf{z}^{(i)} = \mathbf{W}^{(i)} \sigma(\mathbf{z}^{(i-1)}) \geq \mathbf{W}^{(i)} \mathbf{D}^{(i-1)} \mathbf{z}^{(i-1)} + \mathbf{W}^{(i)} \underline{\mathbf{b}}^{(i-1)} \quad (11)$$

539 where $\mathbf{D}^{(i-1)}$ is a diagonal matrix with shape $\mathbb{R}^{n_{i-1} \times n_{i-1}}$ whose off-diagonal entries are 0, and
 540 on-diagonal entries are defined as:

$$\mathbf{D}_{j,j}^{(i-1)} := \begin{cases} 1, & \mathbf{l}_j^{(i-1)} \geq 0 \\ 0, & \mathbf{u}_j^{(i-1)} \leq 0 \\ \alpha_j^{(i-1)}, & \mathbf{l}_j^{(i-1)} < 0 < \mathbf{u}_j^{(i-1)} \text{ and } \mathbf{W}_j^{(i)} \geq 0 \\ \frac{\mathbf{u}_j^{(i-1)}}{\mathbf{u}_j^{(i-1)} - \mathbf{l}_j^{(i-1)}}, & \mathbf{l}_j^{(i-1)} < 0 < \mathbf{u}_j^{(i-1)} \text{ and } \mathbf{W}_j^{(i)} < 0 \end{cases} \quad (12)$$

541 and $\underline{\mathbf{b}}_j^{(i-1)}$ is a vector with shape $\mathbb{R}^{n_{i-1}}$ whose elements are defined as:

$$\underline{\mathbf{b}}_j^{(i-1)} := \begin{cases} 0, & \mathbf{l}_j^{(i-1)} > 0 \text{ or } \mathbf{u}_j^{(i-1)} \leq 0 \\ 0, & \mathbf{l}_j^{(i-1)} < 0 < \mathbf{u}_j^{(i-1)} \text{ and } \mathbf{W}_j^{(i)} \geq 0 \\ -\frac{\mathbf{u}_j^{(i-1)} \mathbf{l}_j^{(i-1)}}{\mathbf{u}_j^{(i-1)} - \mathbf{l}_j^{(i-1)}}, & \mathbf{l}_j^{(i-1)} < 0 < \mathbf{u}_j^{(i-1)} \text{ and } \mathbf{W}_j^{(i)} < 0 \end{cases} \quad (13)$$

542 In the above definitions, $\alpha_j^{(i-1)}$ is a parameter in range $[0, 1]$ and may be fixed or optimized as in
 543 [59].

544 *Proof.* For the j^{th} ReLU at the $(i-1)^{\text{th}}$ layer, it's result may be bounded as follows:

$$\alpha_j^{(i-1)} \mathbf{z}_j^{(i-1)} \leq \sigma(\mathbf{z}_j^{(i-1)}) \leq \frac{\mathbf{u}_j^{(i-1)}}{\mathbf{u}_j^{(i-1)} - \mathbf{l}_j^{(i-1)}} (\mathbf{z}_j^{(i-1)} - \mathbf{l}_j^{(i-1)}). \quad (14)$$

545 The right-hand side holds as this is the Planet-relaxation defined by equation (10). For the left-hand
 546 side, we first consider when $\mathbf{z}_j^{(i-1)} \leq 0$. For every input in this range, the result of the ReLU
 547 is $\sigma(\mathbf{z}_j^{(i-1)}) = 0$. $\alpha_j^{(i-1)} \mathbf{z}_j^{(i-1)}$ forms a line for which inputs in this range will always produce a
 548 non-positive result when $\alpha_j^{(i-1)} \in [0, 1]$. For inputs in the range $\mathbf{z}_j^{(i-1)} \geq 0$, the result of the ReLU
 549 is $\sigma(\mathbf{z}_j^{(i-1)}) = \mathbf{z}_j^{(i-1)}$. This result is never exceeded by $\alpha_j^{(i-1)} \mathbf{z}_j^{(i-1)}$ when $\alpha_j^{(i-1)} \in [0, 1]$.

550 When the result, $\sigma(\mathbf{z}_j^{(i-1)})$, is multiplied by a scalar such as $\mathbf{W}_j^{(i)}$, a valid lower-bound of
 551 $\mathbf{W}_j^{(i)} \sigma(\mathbf{z}_j^{(i-1)})$ requires a lower bound on $\sigma(\mathbf{z}_j^{(i-1)})$ when $\mathbf{W}_j^{(i)} \geq 0$, and an upper bound on
 552 $\sigma(\mathbf{z}_j^{(i-1)})$ when $\mathbf{W}_j^{(i)} < 0$. Such lower and upper bounds are indeed produced by $\mathbf{D}_{j,j}^{(i-1)}$ and $\underline{\mathbf{b}}_j^{(i-1)}$,
 553 whose definitions are derived from the inequality displayed in equation (14). This concludes the
 554 proof. \square

555 Lemma A.1 suggests a recursive approach to bounding a neural network as the bounds at the i^{th} layer
 556 depends on the bounds of the layer preceding it due to the dependence on $\mathbf{l}_j^{(i-1)}$ and $\mathbf{u}_j^{(i-1)}$. This is
 557 indeed the case, and we may define our hyperplane coefficients as $\underline{\mathbf{A}}^{(i)} = \mathbf{\Omega}^{(i,1)} \mathbf{W}^{(1)}$ where

$$\mathbf{\Omega}^{(i,k)} := \begin{cases} \mathbf{W}^{(i)} \mathbf{D}^{(i-1)} \mathbf{\Omega}^{(i-1)}, & i > k \\ \mathbf{I}, & i = k \end{cases} \quad (15)$$

558 To collect the remaining terms, we set $\underline{\mathbf{c}}^{(i)} = \sum_{k=2}^i (\mathbf{\Omega}^{(i,k)} \mathbf{W}^{(k)} \underline{\mathbf{b}}^{(k-1)}) + \sum_{k=1}^i (\mathbf{\Omega}^{(i,k)} \mathbf{b}^{(k)})$. To
 559 obtain an upper bound, Lemma A.1 and its proof may be adjusted accordingly where appearances
 560 of the inequalities $\mathbf{W}^{(i)} \geq 0$ and $\mathbf{W}^{(i)} < 0$ are flipped. In doing so, we may repeat this recursive
 561 process in order to obtain $\overline{\mathbf{A}}^{(i)}$ and $\overline{\mathbf{c}}^{(i)}$.

Though we have described how a ReLU feedforward network may be bounded, appropriately updating the definitions of $\mathbf{D}^{(i)}$ and $\mathbf{b}^{(i)}$ allows feedforward networks with general activation functions (that act element-wise) to be bounded. Such a general formulation is described in [64] that is similar to the template described above, and goes into further detail on how this formulation may be extended to quadratic bound propagation.

A.3 Verification Properties Given as Boolean Logical Formulae

Equation (1) is referred to as the *canonical* formulation [11, 9] as it is general enough to encompass any property involving boolean logical formulas over linear inequalities. To make this idea clear, consider a neural network whose output dimension is $n_L = 3$. We may wish to verify that the output corresponding to the true label, \mathbf{y}_0 , is always greater than all other outputs, \mathbf{y}_1 and \mathbf{y}_2 , for every input in \mathcal{X} . Explicitly, we want to verify:

$$(\mathbf{y}_0 > \mathbf{y}_1) \wedge (\mathbf{y}_0 > \mathbf{y}_2), \quad \forall \mathbf{x} \in \mathcal{X}. \quad (16)$$

To represent this formula, we may first introduce a new linear layer whose weight matrix is defined as \mathbf{C} , sometimes referred to as a *specification matrix*:

$$\mathbf{C} := \begin{bmatrix} 1 & -1 & 0 \\ 1 & 0 & -1 \end{bmatrix}. \quad (17)$$

This specification matrix allows the output neurons to be compared against one another. Applying this matrix to the output vector of the network, denoted as $\mathbf{y} := [\mathbf{y}_0, \mathbf{y}_1, \mathbf{y}_2]^\top$, results in the following new output:

$$\begin{bmatrix} \mathbf{y}_0 - \mathbf{y}_1 \\ \mathbf{y}_0 - \mathbf{y}_2 \end{bmatrix} = \mathbf{C}\mathbf{y}. \quad (18)$$

Next, we append a *MinPool* layer to the network which returns the minimum of these two results. Now, we may define a new neural network, $f'(\mathbf{x})$, which appends the aforementioned *MinPool* and linear layer to $f(\mathbf{x})$, resulting in the final output, $\min\{\mathbf{y}_0 - \mathbf{y}_1, \mathbf{y}_0 - \mathbf{y}_2\}$. It should now be apparent that if the property described by equation (16) can be verified as true, then the following must hold:

$$\min_{\mathbf{x} \in \mathcal{X}} f'(\mathbf{x}) > 0. \quad (19)$$

We may now proceed to produce a lower bound, $\underline{f}'(\mathbf{x})$, and only until $\underline{f}'(\mathbf{x}) > 0$, can we formally state that the network will correctly classify all inputs in \mathcal{X} . In the scenario that we instead want to verify the property $(\mathbf{y}_0 > \mathbf{y}_1 \vee \mathbf{y}_0 > \mathbf{y}_2)$ for all inputs in \mathcal{X} , we may simply redefine $f'(\mathbf{x})$ by replacing the *MinPool* layer with a *MaxPool* layer which would return $\max\{\mathbf{y}_0 - \mathbf{y}_1, \mathbf{y}_0 - \mathbf{y}_2\}$.

Finally, suppose we want to verify that the output corresponding to the true label is greater than all other outputs with some additional margin m such that $(\mathbf{y}_0 > \mathbf{y}_1 + m \wedge \mathbf{y}_0 > \mathbf{y}_2 + m), \forall \mathbf{x} \in \mathcal{X}$ where $m > 0$ (i.e. the network should always classify the first label with additional relative confidence defined by m). Then the only modification is to also incorporate a bias vector, $\mathbf{t} := [-m, -m]^\top$, at the final linear layer where \mathbf{C} was defined. This bias vector is sometimes referred to as the *threshold*. This results in the final output, $f'(\mathbf{x}) = \min\{\mathbf{y}_0 - \mathbf{y}_1 - m, \mathbf{y}_0 - \mathbf{y}_2 - m\}$. Hence, equation (1) is general enough to encompass more complex queries.

B Proofs

B.1 Statement and proof of Lemma B.1

We first introduce a preparatory result that establishes a closed-form solution for a linear optimization problem over a hyper-rectangular feasible set.

Lemma B.1 (Dual Norm Characterization for Hyper-Rectangular Domains). *Let $\epsilon \in \mathbb{R}^n$ with $\epsilon_i > 0$ for all $i \in [n]$, and define the hyper-rectangular domain:*

$$\mathcal{X} = \{\mathbf{x} \in \mathbb{R}^n : \hat{\mathbf{x}} - \epsilon \leq \mathbf{x} \leq \hat{\mathbf{x}} + \epsilon\}.$$

For any vector $\mathbf{v} \in \mathbb{R}^n$, the following hold:

$$\begin{aligned} \max_{\mathbf{x} \in \mathcal{X}} \mathbf{v}^\top \mathbf{x} &= \mathbf{v}^\top \hat{\mathbf{x}} + |\mathbf{v}|^\top \epsilon, \\ \min_{\mathbf{x} \in \mathcal{X}} \mathbf{v}^\top \mathbf{x} &= \mathbf{v}^\top \hat{\mathbf{x}} - |\mathbf{v}|^\top \epsilon, \end{aligned}$$

597 where $|\mathbf{v}| \in \mathbb{R}_+^n$ denotes the component-wise absolute value of \mathbf{v} .

598 *Proof.* We begin by rewriting the feasible set as

$$\mathcal{X} = \{\mathbf{x} \in \mathbb{R}^n : \hat{\mathbf{x}} - \epsilon \leq \mathbf{x} \leq \hat{\mathbf{x}} + \epsilon\} = \{\hat{\mathbf{x}} + \epsilon \circ \mathbf{x} : \mathbf{x} \in \mathbb{R}^n, \|\mathbf{x}\|_\infty \leq 1\},$$

where \circ denotes the Hadamard (component-wise) product. Then, the maximization problem becomes

$$\begin{aligned} \max_{\mathbf{x} \in \mathcal{X}} \mathbf{v}^\top \mathbf{x} &= \max_{\|\mathbf{x}\|_\infty \leq 1} \mathbf{v}^\top \hat{\mathbf{x}} + (\mathbf{v} \circ \epsilon)^\top \mathbf{x} \\ &= \mathbf{v}^\top \hat{\mathbf{x}} + \sum_{i=1}^n |v_i| \epsilon_i = \mathbf{v}^\top \hat{\mathbf{x}} + |\mathbf{v}|^\top \epsilon, \end{aligned}$$

599 where the final equality follows from the definition of the dual norm of the ∞ -norm. The derivation
600 for the minimization problem follows analogously by replacing the maximization with a minimization,
601 which flips the sign of $|\mathbf{v}|^\top \epsilon$. Thus, the claim follows. \square

602 B.2 Proof of Theorem 3.1

603 *Proof.* The problem (3) is a linear program over a compact convex set \mathcal{X} with an additional linear
604 constraint. Let feasible set $\mathcal{F} = \{\mathbf{x} \in \mathcal{X} : \mathbf{g}^\top \mathbf{x} + h \leq 0\}$ be the feasible set for (3).

605 Case 1: $\mathcal{F} = \emptyset$, then by definition, $L^* = +\infty$. In this case, it implies that for all $\mathbf{x} \in \mathcal{X}$, $\mathbf{g}^\top \mathbf{x} + h > 0$.

Case 2: \mathcal{F} is non-empty. Since \mathcal{X} is compact and \mathcal{F} is a closed non-empty subset of \mathcal{X} (as it's the intersection of \mathcal{X} with a closed half-space), \mathcal{F} is also compact. The objective function $\mathbf{a}^\top \mathbf{x} + c$ is continuous. Therefore, L^* is finite and attained. We introduce the Lagrangian for problem (3) by partially dualizing the constraint $\mathbf{g}^\top \mathbf{x} + h \leq 0$:

$$\mathcal{L}(\mathbf{x}, \beta) = (\mathbf{a}^\top \mathbf{x} + c) + \beta(\mathbf{g}^\top \mathbf{x} + h) \quad \text{for } \mathbf{x} \in \mathcal{X}, \beta \geq 0.$$

The primal problem can be written as:

$$L^* = \min_{\mathbf{x} \in \mathcal{X}} \sup_{\beta \geq 0} \mathcal{L}(\mathbf{x}, \beta)$$

To swap the min and sup, we can apply Sion's Minimax Theorem. Let $K = \mathcal{X}$ (a compact convex set in \mathbb{R}^n) and $M = \{\beta \in \mathbb{R} : \beta \geq 0\}$ (a closed convex set in \mathbb{R}). The function $\mathcal{L}(\mathbf{x}, \beta)$ has the following properties: 1) For any fixed $\beta \in M$, $\mathcal{L}(\mathbf{x}, \beta) = (\mathbf{a} + \beta \mathbf{g})^\top \mathbf{x} + (c + \beta h)$ is linear in \mathbf{x} , and thus convex and continuous on K . 2) For any fixed $\mathbf{x} \in K$, $\mathcal{L}(\mathbf{x}, \beta) = (\mathbf{g}^\top \mathbf{x} + h)\beta + (\mathbf{a}^\top \mathbf{x} + c)$ is linear in β , and thus concave and continuous on M . Since these conditions are met, Sion's Minimax Theorem states:

$$\min_{\mathbf{x} \in K} \sup_{\beta \in M} \mathcal{L}(\mathbf{x}, \beta) = \sup_{\beta \in M} \min_{\mathbf{x} \in K} \mathcal{L}(\mathbf{x}, \beta)$$

Therefore,

$$L^* = \sup_{\beta \geq 0} \min_{\mathbf{x} \in \mathcal{X}} \mathcal{L}(\mathbf{x}, \beta)$$

Since \mathcal{F} is non-empty, L^* is finite, implying that the supremum is attained (or is the limit if approached at infinity, but $D(\beta)$ is continuous), so we can write max instead of sup. Let $d(\beta) = \min_{\mathbf{x} \in \mathcal{X}} \mathcal{L}(\mathbf{x}, \beta)$.

$$\begin{aligned} d(\beta) &= \min_{\mathbf{x} \in \mathcal{X}} (\mathbf{a}^\top \mathbf{x} + c + \beta \mathbf{g}^\top \mathbf{x} + \beta h) \\ &= \min_{\mathbf{x} \in \mathcal{X}} ((\mathbf{a} + \beta \mathbf{g})^\top \mathbf{x}) + c + \beta h \end{aligned}$$

The inner minimization is optimizing a linear function $(\mathbf{a} + \beta \mathbf{g})^\top \mathbf{x}$ over the hyper-rectangle \mathcal{X} . Using Lemma B.1 (with $\mathbf{v} = \mathbf{a} + \beta \mathbf{g}$):

$$\min_{\mathbf{x} \in \mathcal{X}} ((\mathbf{a} + \beta \mathbf{g})^\top \mathbf{x}) = (\mathbf{a} + \beta \mathbf{g})^\top \hat{\mathbf{x}} - |\mathbf{a} + \beta \mathbf{g}|^\top \boldsymbol{\epsilon}$$

where $|\mathbf{a} + \beta \mathbf{g}|^\top \boldsymbol{\epsilon} = \sum_{j=1}^n |a_j + \beta g_j| \epsilon_j$. Substituting this into the expression for $d(\beta)$, we get:

$$d(\beta) = (\mathbf{a} + \beta \mathbf{g})^\top \hat{\mathbf{x}} - |\mathbf{a} + \beta \mathbf{g}|^\top \boldsymbol{\epsilon} + c + \beta h$$

This is exactly the dual objective function $D(\beta)$ defined in the theorem statement (4). Thus, we have established that $L^* = \max_{\beta \geq 0} D(\beta)$. Then we analyze the properties of the dual objective $D(\beta)$:

1. Concavity: The dual function $d(\beta)$ is always concave, as it is the pointwise minimum of a family of functions that are affine in β (indexed by $\mathbf{x} \in \mathcal{X}$).
2. Piecewise-Linearity: The term $-|\mathbf{a} + \beta \mathbf{g}|^\top \boldsymbol{\epsilon} = -\sum_{j=1}^n |a_j + \beta g_j| \epsilon_j$ involves the absolute value function. Each term $-|a_j + \beta g_j| \epsilon_j$ is concave and piecewise-linear, with a breakpoint (a point where the slope changes) at $\beta = -a_j/g_j$ (if $g_j \neq 0$). The other terms in $D(\beta)$ are linear in β . Since $D(\beta)$ is a sum of concave piecewise-linear functions and linear functions, it is itself concave and piecewise-linear. The breakpoints of $D(\beta)$ are the collection of all values $\beta = -a_j/g_j \geq 0$ where $g_j \neq 0$.

Thus, we need to maximize the concave, piecewise-linear function $D(\beta)$ over the interval $[0, \infty)$. Since $D(\beta)$ is concave, its maximum over a convex set occurs either at a point where the super-gradient contains zero, or potentially at the boundary point $\beta = 0$. Because $D(\beta)$ is piecewise-linear, its super-gradient $\partial D(\beta)$ is constant within the linear segments between breakpoints. At a breakpoint β_k , the super-gradient is an interval $[\partial D(\beta_k^-), \partial D(\beta_k^+)]$ (the range between the left and right derivatives). The maximum occurs at a point β^* such that $0 \in \partial D(\beta^*)$. This β^* must be either $\beta = 0$ (if the derivative is non-positive for $\beta > 0$) or one of the breakpoints $\beta_k > 0$ where the derivative changes sign from positive to non-positive (i.e., $0 \in [\partial D(\beta_k^-), \partial D(\beta_k^+)]$), or the function increases indefinitely (which corresponds to an infeasible or unbounded primal, but we assumed feasibility and the primal is bounded over the compact \mathcal{X} , so L^* is finite, thus the dual maximum is finite).

Therefore, the maximum L^* can be found non-iteratively by:

1. Identifying all non-negative breakpoints $\beta_k = -a_j/g_j \geq 0$.
2. Sorting these unique breakpoints $0 = \beta_0 < \beta_1 < \dots < \beta_p$.
3. Evaluating the derivative (slope) of $D(\beta)$ within each segment (β_k, β_{k+1}) and potentially at $\beta = 0$.
4. Finding the point β^* (either 0 or some β_k) where the slope transitions from non-negative to non-positive. The value $D(\beta^*)$ is the maximum L^* .

This process involves a finite number of analytical calculations (evaluating slopes and function values at breakpoints) rather than iterative optimization, justifying the claim of efficiency. \square

B.3 Proof of Theorem 3.2

Proof. We consider the upper bound; the lower bound can be derived analogously. First, we rewrite the input region as

$$\mathcal{X} = \{\mathbf{x} \in \mathbb{R}^n : \underline{\mathbf{x}} \leq \mathbf{x} \leq \bar{\mathbf{x}}\} = \{\mathbf{x} \in \mathbb{R}^n : \hat{\mathbf{x}} - \boldsymbol{\epsilon} \leq \mathbf{x} \leq \hat{\mathbf{x}} + \boldsymbol{\epsilon}\},$$

where $\hat{\mathbf{x}} = \frac{\underline{\mathbf{x}} + \bar{\mathbf{x}}}{2}$ and $\boldsymbol{\epsilon} = \frac{\bar{\mathbf{x}} - \underline{\mathbf{x}}}{2}$. Suppose the linear inequality constraint is given by $\mathbf{a}^\top \mathbf{x} + b \leq 0$, where $\mathbf{a} \in \mathbb{R}^n$ and $b \in \mathbb{R}$. We note that the intersection $\mathcal{X} \cap \{\mathbf{x} \in \mathbb{R}^n : \mathbf{a}^\top \mathbf{x} + b \leq 0\}$ is nonempty if and only if

$$0 \geq \min_{\mathbf{x} \in \mathcal{X}} \mathbf{a}^\top \mathbf{x} + b = \mathbf{a}^\top \hat{\mathbf{x}} + b - \sum_{i=1}^n |a_i| \epsilon_i$$

639 by Lemma B.1. Henceforth, we will assume this inequality is satisfied.

640 We now compute

$$\begin{aligned}
\bar{x}_i^{(new)} &= \max_{\mathbf{x} \in \mathcal{X}} \{ \mathbf{e}_i^\top \mathbf{x} : \mathbf{a}^\top \mathbf{x} + b \leq 0 \} \\
&= \max_{\mathbf{x} \in \mathcal{X}} \min_{\lambda \in \mathbb{R}_+} \mathbf{e}_i^\top \mathbf{x} - \lambda(\mathbf{a}^\top \mathbf{x} + b) \\
&= \min_{\lambda \in \mathbb{R}_+} \max_{\mathbf{x} \in \mathcal{X}} \mathbf{e}_i^\top \mathbf{x} - \lambda(\mathbf{a}^\top \mathbf{x} + b) \\
&= \min_{\lambda \in \mathbb{R}_+} \mathbf{e}_i^\top \hat{\mathbf{x}} - \lambda(\mathbf{a}^\top \hat{\mathbf{x}} + b) + |\mathbf{e}_i - \lambda \mathbf{a}|^\top \boldsymbol{\epsilon},
\end{aligned}$$

641 where third line follows from Sion's minimax theorem since \mathcal{X} is a compact set, and the final line
642 follows from Lemma B.1

643 Rearranging the term yields

$$\bar{x}_i^{(new)} = \min_{\lambda \in \mathbb{R}_+} \mathbf{e}_i^\top \hat{\mathbf{x}} + \lambda \left(\sum_{j \neq i} \epsilon_j |a_j| - \mathbf{a}^\top \hat{\mathbf{x}} - b \right) + \epsilon_i |1 - \lambda a_i|. \quad (20)$$

644 We now analyze three cases for a_i and derive a closed-form expression for the scalar minimization.
645 Recall that we have assumed $0 \geq \mathbf{a}^\top \hat{\mathbf{x}} + c - \sum_{i=1}^n a_i \epsilon_i$ or equivalently $\sum_{j \neq i} \epsilon_j |a_j| - \mathbf{a}^\top \hat{\mathbf{x}} - b \geq$
646 $-\epsilon_i |a_i|$. We have:

647 1. $a_i > 0$: The function $\epsilon_i |1 - \lambda a_i|$ attains its minimum at $\lambda = \frac{1}{a_i} > 0$, with slope $\epsilon_i a_i$ on the
648 right and $-\epsilon_i a_i$ on the left. Thus:

- If $|\sum_{j \neq i} \epsilon_j |a_j| - \mathbf{a}^\top \hat{\mathbf{x}} - b| \leq \epsilon_i a_i$ then the minimum of (20) is attained at $\lambda^* = \frac{1}{a_i}$ and

$$\bar{x}_i^{(new)} = \hat{x}_i + \frac{\sum_{j \neq i} \epsilon_j |a_j| - \mathbf{a}^\top \hat{\mathbf{x}} - b}{a_i} = \frac{\sum_{j \neq i} \epsilon_j |a_j| - \sum_{j \neq i} a_j \hat{x}_j - b}{a_i}.$$

- If $\sum_{j \neq i} \epsilon_j |a_j| - \mathbf{a}^\top \hat{\mathbf{x}} - b > \epsilon_i a_i$ then $\lambda^* = 0$ and

$$\bar{x}_i^{(new)} = \hat{x}_i + \epsilon_i = \bar{x}_i.$$

649 2. $a_i < 0$: In this case, the minimizing $\lambda = \frac{1}{a_i} < 0$ is infeasible to (20). Hence, $\lambda^* = 0$ and
650 $\bar{x}_i^{(new)} = \bar{x}_i$.

651 3. $a_i = 0$: Here, the objective function in (20) becomes affine in λ . Since $\sum_{j \neq i} \epsilon_j |a_j| -$
652 $\mathbf{a}^\top \hat{\mathbf{x}} - b \geq 0$, we again have $\lambda^* = 0$ and $\bar{x}_i^{(new)} = \bar{x}_i$.

653 In summary, if $\boldsymbol{\epsilon}^\top |\mathbf{a}| - \mathbf{a}^\top \hat{\mathbf{x}} - b < 0$ then $\emptyset = \mathcal{X} \cap \{\mathbf{x} \in \mathbb{R}^n : \mathbf{a}^\top \mathbf{x} + c \leq 0\}$; otherwise,

$$\bar{x}_i^{(new)} = \begin{cases} \min \left\{ \frac{\sum_{j \neq i} \epsilon_j |a_j| - \sum_{j \neq i} a_j \hat{x}_j - b}{a_i}, \bar{x}_i \right\} & \text{if } a_i > 0 \\ \bar{x}_i & \text{if } a_i \leq 0. \end{cases}$$

654 This completes the proof. \square

655 B.4 Proof of Order Dependency

656 Here we briefly introduce Theorem B.2 to demonstrate why sequential processing the constraints for
657 clipping achieving better results.

658 **Theorem B.2** (Order Dependency of Constraint Intersection). *Let $\mathcal{X} = \{x \in \mathbb{R}^n : x_L \leq x \leq x_U\}$
659 be a box domain, and let $F_j = \{x \in \mathcal{X} : \mathbf{a}_j^\top x + c_j \leq 0\}$ denote the feasible set for the j -th constraint.
660 Define the full feasible set as $F = \bigcap_{j=1}^m F_j$.*

Let $\pi : \{1, \dots, m\} \rightarrow \{1, \dots, m\}$ be a permutation (i.e., a reordering) of the constraints. For sequential intersection, define:

$$F^{(k)} = \bigcap_{t=1}^k F_{\pi(t)}, \quad k = 1, \dots, m,$$

with $F^{(0)} = \mathcal{X}$.

(Order-Independent Bounds): The variable-wise bounds satisfy:

$$\max_{x \in F} x_i \leq \min_j \max_{x \in F_j} x_i \quad \text{and} \quad \min_{x \in F} x_i \geq \max_j \min_{x \in F_j} x_i.$$

These bounds are independent of π if computed via $F = \bigcap_{j=1}^m F_j$.

(Order-Dependent Refinement): If constraints are intersected sequentially (i.e., $F^{(k)}$ depends on π), then there exist permutations $\pi_1 \neq \pi_2$ such that:

$$F_{\pi_1}^{(m)} \neq F_{\pi_2}^{(m)}.$$

Proof. Order-Independent Bounds. Same as Corollary B.2. The inequalities follow from $F \subseteq F_j$ for all j . Simultaneous intersection satisfies:

$$\max_{x \in F} x_i \leq \min_j \max_{x \in F_j} x_i \quad \text{and} \quad \min_{x \in F} x_i \geq \max_j \min_{x \in F_j} x_i,$$

as the intersection F cannot exceed the tightest bound from any F_j .

Order-Dependent Refinement. Let $F^{(k)} = \bigcap_{t=1}^k F_{\pi(t)}$. For dependent constraints (e.g., F_1 bounds x_1 , F_2 depends on x_1), the sequence $F^{(k)}$ depends on π . A counterexample with $F_1 : x_1 + x_2 \leq 2$ and $F_2 : x_1 - x_2 \leq 0$ shows:

1. Intersecting F_1 first gives $x_1 \leq 2 - x_2$, then F_2 further tightens $x_1 \leq x_2$.
2. Intersecting F_2 first gives $x_1 \leq x_2$, then F_1 tightens $x_1 \leq 1$.

The final bounds differ: $x_1 \leq \min(2 - x_2, x_2)$ vs. $x_1 \leq 1$. □

C Algorithms

C.1 Overview

Fig. 3 shows the pipeline of our algorithm.

C.2 Sequential Clipping for Multiple Constraints

When several linear constraints are present, we apply Theorem 3.2 *sequentially* to each row of $Ax + c \leq 0$. Algorithm 3 outlines this procedure: First, for each constraint k , compute \hat{x} and ϵ from the *current* bounds, \underline{x} and \bar{x} . Then, use Theorem 3.2 to refine \underline{x}_i and \bar{x}_i for all i . Finally, proceed to the next constraint, using the newly clipped bounds as the domain. Note that if $\underline{x}_i > \bar{x}_i$ occurs in any dimension, the input region of this subproblem is infeasible, and we can directly verify this subproblem without performing further verification.

Because the bounds are updated after each constraint, the final domain is an over-approximation of the true feasible region under all constraints simultaneously. However, it is still far more efficient than solving a multi-constraint system in one shot, making it well suited for large-scale verification where we repeatedly clip domains across many subproblems.

By clipping the domain repeatedly, Algorithm 3 retains enough precision to prune large regions yet remains computationally lightweight enough for repeated invocation on many subdomains during verification. It may even be the case that the constraints passed to our domain clipping algorithm reveal the region is entirely *infeasible*. Such a scenario may occur when two ReLU assignments admit an infeasible domain, or when the desired property admits multiple constraints that produce

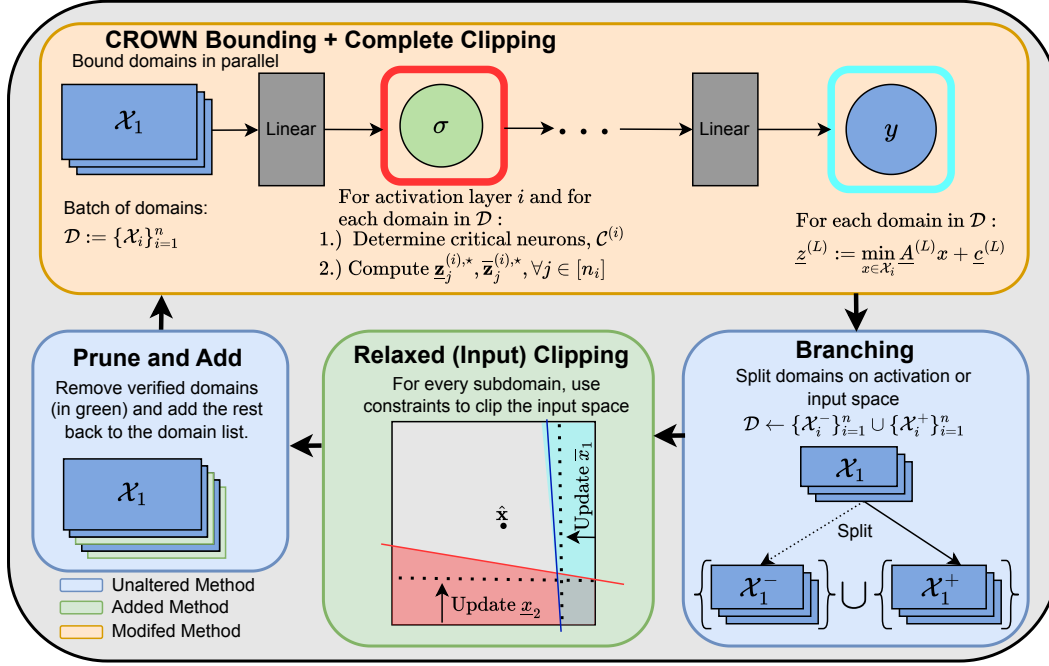


Figure 3: Full Clip-and-Verify pipeline for (Input and ReLU Activation) BaB integration.

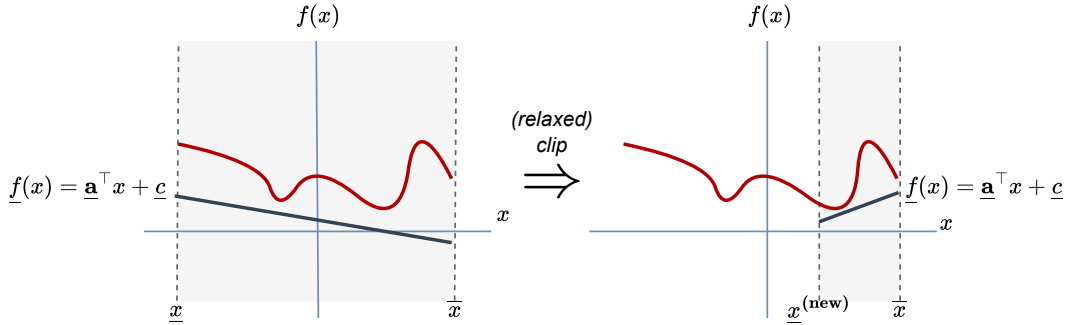


Figure 4: Simple 1D visualization of relaxed clipping reducing the input interval, potentially enabling a second pass of bound propagation to produce a tighter bound.

689 infeasibility. In such scenarios, Algorithm 3 will return clipped bounds where \underline{x} will be smaller than
 690 \bar{x} along some dimension(s). In the context of branch-and-bound, this subdomain may be effectively
 691 pruned, avoiding the process of running bound propagation once more on the domain, avoiding
 692 unnecessary computations

693 C.3 Algorithms for Clip-and-Verify in Input Branch-and-bound Scheme

694 Algorithm 4 describes our modifications to the standard BaB procedure. When all subproblems can
 695 be verified, then the verification problem is referred to as UNSAT (i.e., the complementary property,
 696 $\exists \mathbf{x} \in \mathcal{X}, f(\mathbf{x}) < 0$, is *unsatisfiable*), and the network is *safe* from counter-examples. Otherwise, it
 697 is insufficient to determine if the property is UNSAT without further refinement or falsification.

698 C.4 Algorithms for Clip-and-Verify in ReLU Branch-and-bound Scheme

699 Algorithm 5 begins by initializing the verification procedure. First, we call $\text{Init}(f, \emptyset)$ to obtain an
 700 empty set of partial ReLU assignments (or subdomains) along with an initial global lower bound \underline{f}
 701 for the property to be checked. This global lower bound can, for example, be the result of a quick

Algorithm 3 Linear Constraint-Driven Relaxed Clipping (*sequential*)

Require: \underline{x} : Lower bounds; \bar{x} : Upper bounds; \mathbf{A} : Constraint matrix; \mathbf{c} : Constraint vector.

```
1:  $m \leftarrow \text{rows}(\mathbf{A}), n \leftarrow \text{cols}(\mathbf{A})$ 
2: for for each constraint  $k \in \{1, \dots, m\}$  do
3:   for each input dimension  $i \in \{1, \dots, n\}$  do
4:      $\hat{x} \leftarrow \frac{\bar{x} + \underline{x}}{2}, \epsilon \leftarrow \frac{\bar{x} - \underline{x}}{2}$ 
5:     if  $\mathbf{A}_{k,i} \neq 0$  then
6:        $\mathbf{x}_i^{(\text{new})} \leftarrow \frac{-\sum_{j \neq i} \mathbf{A}_{k,j} \hat{x}_j + \sum_{j \neq i} |\mathbf{A}_{k,j}| \epsilon_j - b}{\mathbf{A}_{k,i}}$ 
7:       if  $\mathbf{A}_{k,i} > 0$  then
8:          $\bar{x}_i^{(\text{new})} \leftarrow \min(\bar{x}_i, \mathbf{x}_i^{(\text{new})})$ 
9:       else
10:         $\underline{x}_i^{(\text{new})} \leftarrow \max(\underline{x}_i, \mathbf{x}_i^{(\text{new})})$ 
11:       $\underline{x}_i \leftarrow \max(\underline{x}_i, \underline{x}_i^{(\text{new})})$ 
12:       $\bar{x}_i \leftarrow \min(\bar{x}_i, \bar{x}_i^{(\text{new})})$ 
Ensure: Clipped  $\underline{x}, \bar{x}$ 
```

Algorithm 4 Clip-and-Verify for Input Branch-and-bound

Require: f : model to verify; n : batch size; timeout : time-out threshold

```
1:  $\mathcal{D}_{\text{Unknown}}, \underline{f} \leftarrow \text{Init}(f, \emptyset)$  {Initialize the set of unknown subdomains  $\mathcal{D}_{\text{Unknown}}$  and global bound  $\underline{f}$ }
2: while  $|\mathcal{D}_{\text{Unknown}}| > 0$  and not timed out do
3:    $\{\mathcal{X}_i, \underline{\mathbf{A}}_i^{\text{prev}}, \underline{\mathbf{c}}_i^{\text{prev}}\}_{i=1}^n \leftarrow \text{Batch\_Pick\_Out}(\mathcal{D}_{\text{Unknown}}, n)$  {Pick up to  $n$  subdomains  $\mathcal{X}_i$  and their associated hyperplanes  $(\underline{\mathbf{A}}_i^{\text{prev}}, \underline{\mathbf{c}}_i^{\text{prev}})$  from the previous iteration.}
4:    $\{\mathcal{C}_i\}_{i=1}^n \leftarrow \text{Top-K\_Heuristic}(\{\mathcal{X}_i, \underline{\mathbf{A}}_i^{\text{prev}}, \underline{\mathbf{c}}_i^{\text{prev}}\}_{i=1}^n)$  {Determine critical neurons  $\mathcal{C}_i$  using a top-k heuristic over each domain and its previous hyperplanes.}
5:    $(\underline{f}_{\mathcal{X}_1}, \underline{\mathbf{A}}_{\mathcal{X}_1}, \underline{\mathbf{c}}_{\mathcal{X}_1}, \dots, \underline{f}_{\mathcal{X}_n}, \underline{\mathbf{A}}_{\mathcal{X}_n}, \underline{\mathbf{c}}_{\mathcal{X}_n}) \leftarrow \text{Solve\_Bound}(f, \{\mathcal{X}_i, \mathcal{C}_i\}_{i=1}^n)$  {Compute bounds and plane coefficients on each subdomain; Refine critical neurons using planes as constraints and Complete Clipping.}
6:    $\{\mathcal{X}_i^-, \mathcal{X}_i^+, \underline{\mathbf{A}}_i, \underline{\mathbf{c}}_i\}_{i=1}^n \leftarrow \text{Batch\_Split}(\{\mathcal{X}_i, \underline{\mathbf{A}}_i, \underline{\mathbf{c}}_i\}_{i=1}^n)$  {Split each subdomain; share hyperplanes with both children.}
7:    $\{\mathbf{x}_i^{(-)}, \mathbf{x}_i^{(+)}\}_{i=1}^n \leftarrow \text{Relaxed\_Clipper}(\{\mathcal{X}_i^-, \mathcal{X}_i^+, \underline{\mathbf{A}}_i, \underline{\mathbf{c}}_i\}_{i=1}^n)$  {Refine input box on each child subdomain using the plane coefficients.}
8:    $\{\mathcal{X}_i^-, \mathcal{X}_i^+\}_{i=1}^n \leftarrow \text{Domain\_Update}(\{\mathbf{x}_i^{(-)}, \mathbf{x}_i^{(+)}\}_{i=1}^n)$  {Update each child subdomain's input bounds.}
9:    $\mathcal{D}_{\text{Unknown}} \leftarrow \mathcal{D}_{\text{Unknown}} \cup \text{Domain\_Filter}([f_{\mathcal{X}_1^-}, \mathcal{X}_1^-, \underline{\mathbf{A}}_1^{(-)}, \underline{\mathbf{c}}_1^{(-)}], \dots)$  {Filter out verified/infeasible subdomains. Retain unknowns and their bounding hyper-planes.}
Ensure: UNSAT if  $|\mathcal{D}_{\text{Unknown}}| = 0$  else Unknown
```

702 bounding pass. The set $\mathcal{D}_{\text{Unknown}}$ is then populated with a single “root” subdomain representing the
703 entire input domain.

704 Next, we retrieve the constraint information for all neurons (Line 2). Specifically,
705 $\text{Get_Constraints}(f)$ returns the linear coefficients and biases used to bound each neuron’s acti-
706 vation, distinguishing between the lower-bounding $(\underline{\mathbf{A}}^{(j)}, \underline{\mathbf{c}}^{(j)})$ and upper-bounding $(\bar{\mathbf{A}}^{(j)}, \bar{\mathbf{c}}^{(j)})$
707 linear functions. It also identifies indices of “unstable” ReLU neurons whose ranges straddle zero.
708 These constraints will later be used to restrict the feasible input region using both our Relaxed
709 Clipping and Complete Clipping algorithms.

710 We then initialize the Domain Clipper (Line 3) with the gathered linear constraints. This Clipper
711 component will be invoked whenever we branch on a ReLU neuron, so that the corresponding
712 partial assignment (e.g., $x_k^{(j)} \geq 0$) is “pushed back” onto the input domain. In doing so, we clip
713 the subdomain’s input box by applying Theorem 3.2 (or its extensions) to incorporate these newly

Algorithm 5 Clip-and-Verify for ReLU Branch-and-bound

Require: f : model to verify; n : batch size; timeout : time-out threshold

- 1: $\mathcal{D}_{\text{Unknown}}, \underline{f} \leftarrow \text{Init}(f, \emptyset)$ {Initialize the set of unknown subdomains $\mathcal{D}_{\text{Unknown}}$ and global bound \underline{f} }
- 2: $\{\underline{A}^{(j)}, \underline{c}^{(j)}, \overline{A}^{(j)}, \overline{c}^{(j)}, \text{unstable_neuron}(\text{idx})^{(j)}\}_{j=1}^J \leftarrow \text{Get_Constraints}(f)$ {Retrieve the set of coefficients and biases for each unstable neurons' upper and lower bound and get the indices of unstable neurons during Bound Propagation.}
- 3: $\text{Domain_Clipper} \leftarrow \text{Init_Clipper}(\{\underline{A}^{(j)}, \underline{c}^{(j)}, \overline{A}^{(j)}, \overline{c}^{(j)}, \text{idx}^{(j)}\}_{j=1}^J)$ {Initialize the Domain Clipper with the set of constraints information.}
- 4: **while** $|\mathcal{D}_{\text{Unknown}}| > 0$ **and** not timed out **do**
- 5: $\{\mathcal{Z}_i\}_{i=1}^n \leftarrow \text{Batch_Pick_Out}(\mathcal{D}_{\text{Unknown}}, n)$ {Pick at most n subdomains from the unknown set.}
- 6: $\{\mathcal{Z}_i^-, \mathcal{Z}_i^+\}_{i=1}^n \leftarrow \text{Batch_Split}(\{\mathcal{Z}_i\}_{i=1}^n)$ {Split each subdomain (e.g., ReLU split or input split) into two child subdomains.}
- 7: $\{\mathcal{C}_i^{(-)}, \mathcal{C}_i^{(+)}\}_{i=1}^n \leftarrow \text{Top-K_Heuristic}(\{\mathcal{Z}_i\}_{i=1}^n)$ {Determine critical neurons using a top-k heuristic (e.g. BaBSR) over each domain.}
- 8: $\{\mathbf{x}_i^{(-)}, \text{interm_bds}_i^{(-)}, \mathbf{x}_i^{(+)}, \text{interm_bds}_i^{(+)}\}_{i=1}^n \leftarrow \text{Domain_Clipper}(\{\mathcal{Z}_i^-, \mathcal{Z}_i^+, \mathcal{C}_i^{(-)}, \mathcal{C}_i^{(+)}\}_{i=1}^n)$ {Apply Relaxed Clipping to child subdomain's input and Complete Clipping on the critical neurons. Constraints are the ReLU splits.}
- 9: $\{\mathcal{Z}_i^-, \mathcal{Z}_i^+\}_{i=1}^n \leftarrow \text{Domain_Update}(\{\mathbf{x}_i^{(-)}, \text{interm_bds}_i^{(-)}, \mathbf{x}_i^{(+)}, \text{interm_bds}_i^{(+)}\}_{i=1}^n)$ {Update each child subdomain's input and intermediate bounds.}
- 10: $(\underline{f}_{\mathcal{Z}_1^-}, \dots, \underline{f}_{\mathcal{Z}_n^+}) \leftarrow \text{Solve_Bound}(f, \{\mathbf{x}_i, \text{interm_bounds}_i, \mathcal{Z}_i\})$ {Compute bounds on each newly clipped subdomain using a bound propagation solver.}
- 11: $\mathcal{D}_{\text{Unknown}} \leftarrow \mathcal{D}_{\text{Unknown}} \cup \text{Domain_Filter}(\underline{f}_{\mathcal{Z}_1^-}, \dots)$ {Filter out verified/infeasible subdomains. Keep remaining unknown subdomains in $\mathcal{D}_{\text{Unknown}}$.}

Ensure: UNSAT if $|\mathcal{D}_{\text{Unknown}}| = 0$ else Unknown

714 introduced constraints, thus discarding parts of the input space that violate them. In addition, when
 715 given a set of critical neurons, these ReLU split constraints will be used to directly refine the
 716 intermediate neurons using Theorem [3.1](#).

717 The main loop (Lines 5–11) iterates until either no subdomains remain unknown or a time-out is
 718 reached. In each iteration, we pick up to n unknown subdomains from $\mathcal{D}_{\text{Unknown}}$ (Line 5) for batched
 719 parallel processing. Each subdomain \mathcal{Z}_i is then split (Line 6) along one or more unstable ReLU
 720 neurons, creating child subdomains in which each split neuron is fixed to either the active (≥ 0) or
 721 inactive (≤ 0) regime.

722 At Line 7, we invoke a top-k heuristic (e.g. BaBSR) in order to determine the set of “critical neurons”
 723 that would contribute the most to providing a stronger convex relaxation if their bounds were to be
 724 refined.

725 At Lines 8–9, we invoke the Domain Clipper on these newly formed child subdomains. The Clipper
 726 translates each ReLU assignment into a linear constraint on the input, then refines (or “clips”) the
 727 child subdomain's input bounds, and the bounds of the “critical neurons” in each subdomain with
 728 respect to the heuristic choices from the step prior. This ensures that any portion of the parent domain
 729 that contradicts the new constraint is removed. Once clipped, the child subdomains' intermediate
 730 bounds are also updated (Line 10) so that subsequent bounding calculations reflect the tighter input
 731 ranges.

732 We then compute bounds on the newly clipped subdomains (11) using a chosen method—often a fast
 733 bound-propagation tool such as CROWN or a lightweight LP solver. This step yields lower bounds
 734 $\underline{f}_{\mathcal{Z}_i^\pm}$ on the network outputs for each subdomain. If these bounds confirm that the property holds
 735 (e.g., a robustness margin remains non-negative), the subdomain is verified and can be pruned. If
 736 the subdomain is infeasible (e.g., constraints are contradictory), it is also removed. All remaining
 737 subdomains (still “unknown”) return to $\mathcal{D}_{\text{Unknown}}$ for further splitting.

Finally, the loop terminates once there are no unknown subdomains left or the time-out is reached. If $\mathcal{D}_{\text{Unknown}}$ becomes empty, we conclude UNSAT, signifying that no violating input (counterexample) exists within any subdomain. Otherwise, we return “Unknown,” indicating that verification was not completed in time.

Overall, this procedure reflects a standard ReLU BaB flow, except that an additional “domain clipping” step (Lines 8–9) is inserted after each split, leveraging partial ReLU assignments to refine the input domain and select intermediate neurons before the next bounding pass. By applying our linear constraint-driven clipping algorithms whenever new constraints appear, we gain significantly tighter intermediate-layer bounds and thus reduce the branching burden throughout the verification process.

D Experiments

D.1 Experiments Settings

To allow for comparability of results, all tools for input BaB were evaluated on equal-cost hardware with a 32-vcore CPU, one NVIDIA RTX 4090 GPU with 24 GB memory, and 256 GB CPU memory. For ReLU based BaB experiment, we use a cluster with one AMD EPYC 9534 64-core CPU and the GPU is one NVIDIA RTX 5090 GPU with 32 GB memory and 512 GB CPU memory. Our implementation is based on the open-source α, β -CROWN verifier¹ with Clip-and-Verify related code added. For input bab, three different set-up of Clip-and-Verify are tested: Relaxed, Relaxed + Reorder, and Complete. Here Relaxed, Reorder and Complete refers to the methodology discussed in [3.2] and [3.3]. All experiments use 32 CPU cores and 1 GPU. The MIP cuts are acquired by the cplex [28] solver (version 22.1.0.0). We use the Adam optimizer [32] to solve both α, β, μ, τ . For the SDP-FO benchmarks, we optimize those parameters for 20 iterations with a learning rate of 0.1 for α and 0.02 for β, μ, τ . We decay the learning rates with a factor of 0.98 per iteration. The timeout is 200s per instance. For the VNN-COMP benchmarks, we use the same configuration as α, β -CROWN used in the respective competition and the same timeouts.

D.2 Ablation Studies

We conduct a detailed ablation study on multiple adversarially-trained models spanning MNIST, CIFAR, and larger VNN-COMP benchmarks to evaluate Clip-and-Verify and its variants against three baselines: β -CROWN, GCP-CROWN, and BICCOS. Tables [3] and [4] highlight three key metrics: verified accuracy (Ver.%), average per-example verification time (Time), and average per-verified-example domain visited (D.V.). Domain visited (D.V.) is a metric specific to branch-and-bound (BaB) methods, indicating how many subproblems (domains) are explored to fully verify an instance. Crucially, a higher D.V. count may reflect verifying more difficult instances or a larger overall coverage, rather than inefficiency in the verification process. We also provide Figures [5] and [6] that visualize these metrics across all benchmarks.

Verified Accuracy and Time From Table [3], Clip-and-Verify variants nearly always achieve higher verified accuracy than the baselines. For instance, on CNN-B-Adv (CIFAR), Clip-and-Verify with BICCOS reaches 51.5% verified accuracy—surpassing the 47.0% (β -CROWN), 49.5% (GCP-CROWN with MIP cuts), and 51.0% (BICCOS alone) of the baselines. On cifar10-resnet, Clip-and-Verify (with MIP cuts or with BICCOS) achieves up to 88.89% verified accuracy (outperforming the 83.33% - 87.5% range from baselines). In terms of verification time on this benchmark, Clip-and-Verify with β -CROWN is the fastest overall (6.06s), and Clip-and-Verify with BICCOS (11.80s) remains competitive with standalone BICCOS (16.73s) and GCP-CROWN (17.99s). When scaling to deeper networks such as cifar100-2024 and tinyimagenet-2024, Clip-and-Verify with BICCOS maintains the leading verified accuracy (65.5% and 72.0%, respectively) while sustaining moderate average verification times (e.g., 8.17s for cifar100-2024 and 10.48s for tinyimagenet-2024), underscoring its suitability for larger-scale verification tasks.

Domain Visited (D.V.) vs. Difficulty In Table [4], we further examine the average domain visited (D.V.) across verified examples. While Clip-and-Verify variants may sometimes visit more domains (e.g., on oval22, Clip-and-Verify with MIP cuts visits 18891.25 domains compared to 16614.72 for

¹<https://github.com/huanzhang12/alpha-beta-CROWN>

Table 4: Ablation Studies on Verified accuracy (Var.%), avg. per-verified-example domain visited number (D.V.) analysis for all method verified instances on different Clip-and-Verify components.

Dataset	Model	β -CROWN		GCP-CROWN with MIP cuts		BICCOS		Clip-and-Verify with β -CROWN		Clip-and-Verify with MIP cuts		Clip-and-Verify with BICCOS		Upper bound
$\epsilon = 0.3$ and $\epsilon = 2/255$		Ver.%	D.V.	Ver.%	D.V.	Ver.%	D.V.	Ver.%	D.V.	Ver.%	D.V.	Ver.%	D.V.	
MNIST	CNN-A-Adv	71.0	2712.72	71.5	4447.54	76.0	3081.50	74.0	2395.96	73.5	1495.07	76.0	2636.99	76.5
	CNN-A-Adv	45.5	12621.38	48.5	8186.11	48.5	3622.71	45.5	1037.36	48.5	3704.86	48.5	2073.58	50.0
	CNN-A-Adv-4	46.5	2066.39	48.5	3907.30	48.5	1319.56	46.5	298.89	48.5	2396.36	48.5	843.47	49.5
	CNN-A-Mix	42.0	6108.57	47.5	17609.84	48.0	8015.12	43.0	4462.48	47.5	9836.06	48.0	4189.92	53.0
	CNN-A-Mix-4	51.0	482.43	55.0	8922.50	56.0	3319.90	51.0	150.46	55.0	4304.32	56.5	3501.06	57.5
	CNN-B-Adv	47.0	7255.68	49.5	9846.32	51.0	5758.00	49	4951.07	51.5	6979.27	51.5	2677.94	65.0
	CNN-B-Adv-4	55.0	1776.66	58.5	4688.22	59.5	2711.15	56.5	649.92	60.0	3565.74	60.5	1095.30	63.5
cifar10-resnet		83.33	2105.76	87.5	7091.30	87.5	5428.60	86.11	478.0	88.89	2545.28	88.89	2643.15	100.0
oval22		66.66	30949.95	83.33	16614.72	83.33	12730.08	73.33	20191.27	90.00	18891.25	90.00	14032.81	96.67
cifar100-2024		59.5	1535.60	-	-	60.5	769.87	63.0	152.96	-	-	65.5	122.00	84.0
tinyimagenet-2024		67.5	830.92	-	-	69.0	497.52	70.0	188.02	-	-	72.0	118.34	78.5

GCP-CROWN with MIP cuts), this often correlates with achieving higher verified accuracy (90.00% vs 83.33% in this case). This suggests that the method is effectively exploring the space to verify more challenging instances or a broader set of inputs, leading to a net increase in verified accuracy. For example, on CNN-A-Adv (CIFAR), Clip-and-Verify with MIP cuts visits 3704.86 domains on average and attains 48.5% verified accuracy. While its D.V. is slightly higher than standalone BICCOS (3622.71 D.V. for 48.5% accuracy), it's notably lower than β -CROWN (12621.38 D.V. for 45.5% accuracy) and GCP-CROWN with MIP cuts (8186.11 D.V. for 48.5% accuracy), while achieving comparable or better accuracy. In other scenarios (e.g., CNN-A-Adv-4 on CIFAR), Clip-and-Verify with BICCOS achieves 48.5% accuracy with a D.V. of only 843.47. This is the same or higher accuracy with a significantly smaller D.V. compared to standalone β -CROWN (46.5%, 2066.39 D.V.), GCP-CROWN with MIP cuts (48.5%, 3907.30 D.V.), and BICCOS (48.5%, 1319.56 D.V.). This illustrates that when Clip-and-Verify effectively prunes the search space, verification efficiency can improve even while tackling similarly challenging problems and achieving high accuracy.

Overall, these results show that Clip-and-Verify's framework of enhanced linear bounding and "clipping" robustly scales across varying network depths and adversarial training schemes. The additional integration of MIP cuts or BICCOS bounding routines consistently pushes verified accuracy closer to each benchmark's upper bound while balancing verification time and domain exploration. We plot and analyze these ablation studies across all benchmarks in Appendix D.2 confirming that our method not only raises coverage (Ver.%) but also leverages clipping and cutting plane methods to verify some of the hardest instances encountered.

D.3 Details of Heuristics

Neuron Selection Heuristic The neuron selection heuristic is to guide the search in the branch-and-bound algorithm more effectively. By selecting neurons that are deemed more critical or more likely to lead to significant tightening of bounds or faster pruning of the search space, the overall verification process can be accelerated by solving fewer optimization problems. It aims to make a more informed decision than random selection or simple ordering.

For unstable neurons in the neural network, an FSB intercept score [9] is calculated:

$$\text{score}_k = \left(\frac{\max(0, -l_k) \cdot \max(0, u_k)}{u_k - l_k} \right) \cdot \max(0, -\text{mean}_{A_k})$$

Where mean_{A_k} is a scalar value derived from $\mathbb{E}[A^{(k)}]$ (or $\mathbb{E}[\bar{A}^{(k)}]$), related to the linear lower bound of an associated neuron. Specifically, it's often related to the intercept or a critical coefficient of this bound. This score quantifies the importance or potential impact of branching on that particular neuron. In each layer, or globally among all unstable neurons, the heuristic selects the top-k neurons that have the highest fskb scores. These selected top-k neurons are then prioritized as the objective for the Algorithm 1 in the verification process.

Table 5 shows the ablation study on the top-k neuron selection heuristic on cifar_cnn_a_mix benchmark, reveals that strategically prioritizing neurons based on their FSB intercept scores significantly enhances verification efficiency. Employing a moderate k (specifically top-k 20 and 50) leads to the best outcomes, successfully verifying more properties (86) while substantially reducing both the number of domains visited (by up to 28% compared to no heuristic) and the overall time (by up to

Table 5: Ablation Study on Top-k Neuron Selection on `cifar_cnn_a_mix`

Top-k	# Verified	Avg. # Domain Visited	Avg. Time (s)
0 (reduce to β -CROWN)	84	6108.57	4.20
20	86	4462.47	4.24
50	86	4372.79	4.83
all	85 (1 timeout)	1881.77	8.23

19%). In contrast, not using the heuristic (top-k 0) results in a less efficient search, while applying it to all unstable neurons (top-k “all”) drastically cuts down visited domains but incurs a prohibitive time cost and a timeout, indicating that the overhead of processing too many prioritized neurons outweighs the benefits of more targeted branching. This demonstrates a crucial trade-off, with intermediate k values striking an optimal balance between guided search and computational overhead.

Constraint Importance Heuristic The constraint importance heuristic is to sort the constraints for better performance. The constraints corresponding to hyperplanes closer to the center of the current input domain might be more immediately relevant or impactful for tightening the bounds or for cutting off a significant portion of the current feasible region \mathcal{X} .

First, for the current input box domain \mathcal{X} , determine its centroid \hat{x} . We can calculate $\mathbf{g}^\top \hat{x} + h - \sum_{i=1}^n |g_i| \epsilon_i > 0$ (there is no $x \in \mathcal{X}$ satisfy the constraint in (3)) to check the infeasibility and $\mathbf{g}^\top \hat{x} + h + \sum_{i=1}^n |g_i| \epsilon_i \leq 0$ (for all $x \in \mathcal{X}$ satisfy the constraint in (3)) to remove redundancy. Second, for each available linear constraint $\mathbf{g}^\top \mathbf{x} + h = 0$, calculate the geometric distance from the centroid \hat{x} to this hyperplane:

$$d = \frac{|\mathbf{g}^\top \mathbf{x}_0 + h|}{\|\mathbf{g}\|_2}$$

Then, sort the constraints in ascending order based on these calculated distances to prioritize constraints that are closer to the centroid. In methods like Algorithm 1 (Complete Clipping), processing more impactful constraints earlier might lead to faster convergence or more significant bound improvements in the coordinate ascent. Such an important metric could also be relevant to Algorithm 3 (Relaxed Clipping).

We tested the bound tightness improvement from the heuristic on `acasxu` instance 65. Clip-and-Verify improves the intermediate bounds for 470,772,542 times during input BaB, and 3.962% of them can be further tightened with constraint importance heuristic. For all these further improved bounds, we computed the empirical quantiles of the relative improvements, shown in Table D.3.

Percentile of problems	Max	0.1th	1th	5th	10th	25th	50th	75th	90th
Bound Improvement	4242.6%	270.79%	3.07%	0.64%	0.29%	0.10%	0.03%	0.01%	0.002%

These results reveal that the head of the distribution contains substantial refinements, with the maximum observed improvement reaching up to 4000%. This indicates that the heuristic can produce tightened bounds, potentially leading to earlier branch pruning or faster convergence.

E Limitation and Broader Impacts

Limitation While Complete Clip utilizes GPU acceleration for parallel computing, its scalability is restricted, particularly for networks with large hidden layers that incorporate sparse constraints. Relaxed Clip offers high efficiency, however, its effectiveness can be substantially reduced in high-dimensional input domains due to the curse of dimensionality, potentially limiting its advantages.

Broader Impacts Neural network verification is crucial for ensuring the safety and reliability of AI systems in critical applications such as autonomous vehicles, medical diagnosis, and financial trading. By significantly accelerating the verification process through efficient domain reduction, our work makes formal verification more practical for larger and more complex neural networks. This advancement enables broader adoption of verification techniques in real-world applications,

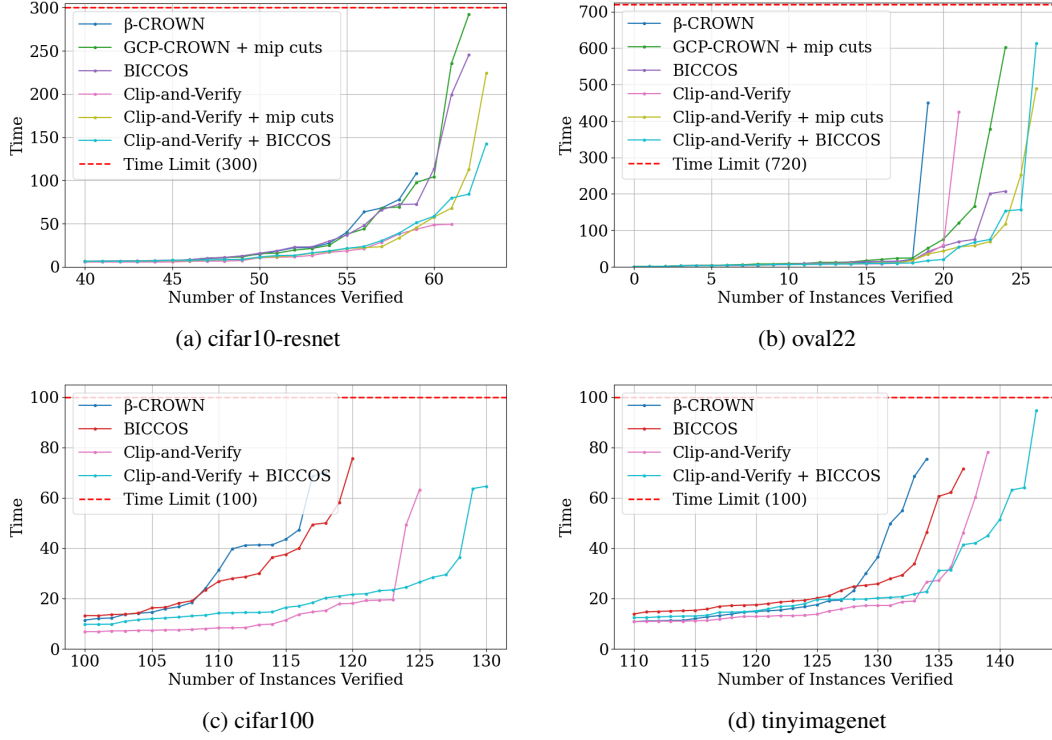
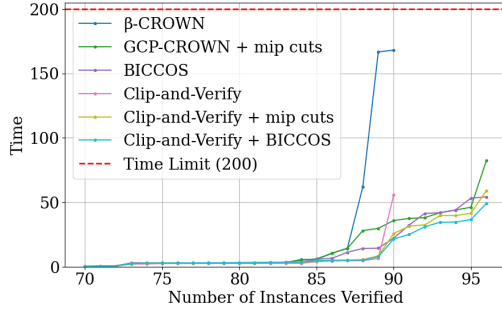
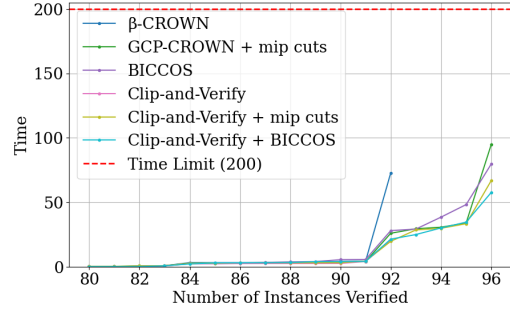


Figure 5: Plots for hard instances need to be solved by BaB in VNN-COMP benchmarks. It is important to note that in large-scale models with numerous properties to verify, the cutting plane method often incurs significant overhead. This is because our current approach processes all properties in batches, whereas the cutting plane method handles each property individually. Consequently, when addressing datasets like CIFAR-100 (99 properties) and TinyImageNet (199 properties), integrating with BICCOS introduces additional overhead.

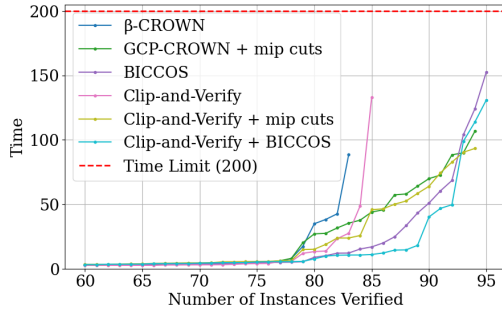
861 potentially preventing catastrophic failures and building trust in AI systems. Clip-and-Verify's
862 integration with existing frameworks ensures immediate applicability, allowing organizations to
863 implement stronger safety guarantees without substantial overhead. While this work strengthens the
864 safety of AI systems, it is important to note that verification tools should be part of an approach to AI
865 safety, including robust testing interpretability, and ethical guidelines.



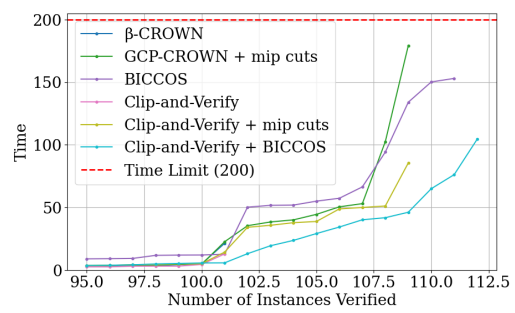
(a) cifar_cnn_a_adv



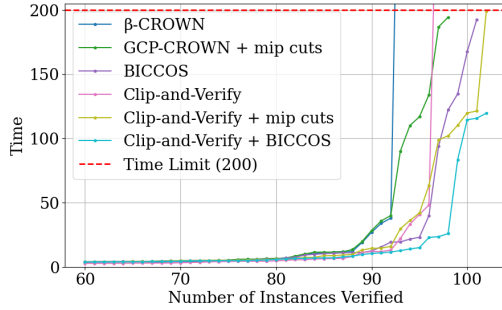
(b) cifar_cnn_a_adv4



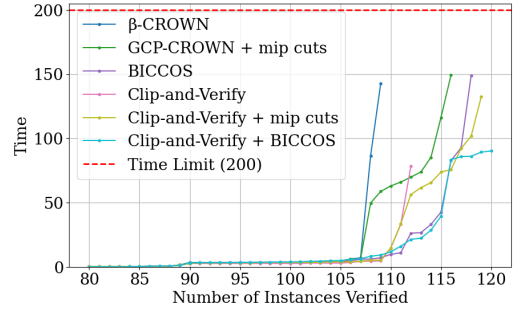
(c) cifar_cnn_a_mix



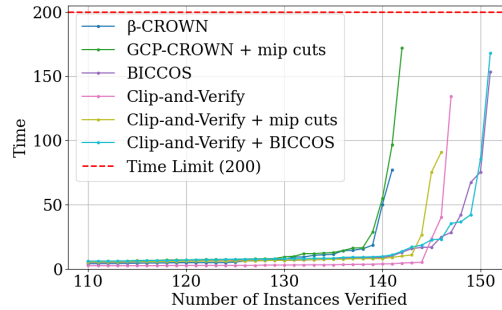
(d) cifar_cnn_a_mix4



(e) cifar_cnn_b_adv



(f) cifar_cnn_b_adv4



(g) mnist_cnn_a_adv

Figure 6: Plots for hard instances need to be solved by BaB in SDP benchmarks.