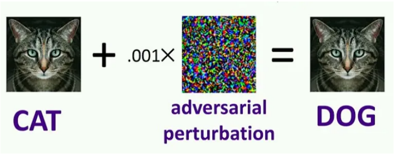
Efficient Neural Network Verification via Order Leading Exploration of Branch-and-Bound Trees

Guanqin Zhang[1,2], Kota Fukuda[3], Zhenya Zhang[3], Dilum Bandara[1,2], Shiping Chen[1,2], Jianjun Zhao[3], Yulei Sui[1]

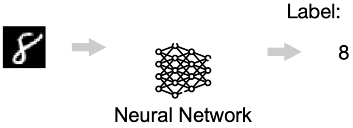
[1]University of New South Wales, Sydney, Australia
[2]CSIRO's Data61, Australia
[3]Kyushu University, Fukuoka, Japan

Are Neural Networks Robustness?



[1]

[2]

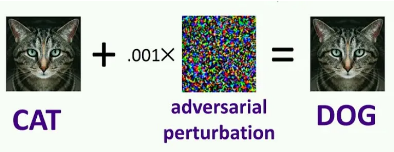


Small perturbations on the input can cause neural networks to yield **incorrect output**.

[1] Goodfellow et al., Explaining and Harnessing Adversarial Examples, ICLR'15

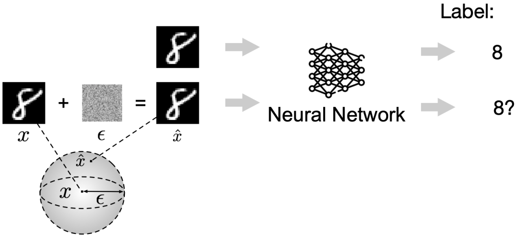
[2] Gnanasambandam et al., Optical Adversarial Attack, CVPR'21

Are Neural Networks Robustness?



[1]

[2]



Perturbed input $\hat{x} \in \Phi : \{\hat{x} \mid \|\hat{x} - x\|_\infty \leq \epsilon\}$,

To check $f(\hat{x}) \models \Psi$.

Small perturbations on the input can cause neural networks to yield **incorrect output**.

[1] Goodfellow et al., Explaining and Harnessing Adversarial Examples, ICLR'15

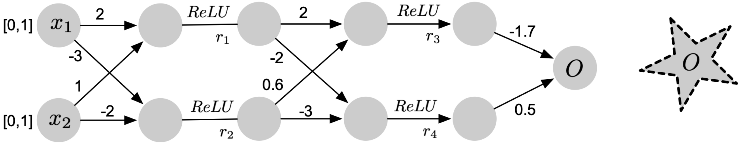
[2] Gnanasambandam et al., Optical Adversarial Attack,CVPR'21

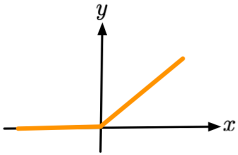
Over-Approximation for Neural Network Verification

Specification $\Phi \wedge \Psi$

Input: $\Phi := x_1 \in [0,1] \wedge x_2 \in [0,1]$ Output: $\Psi = (O \geq 0)$

Network f



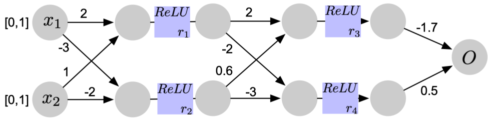


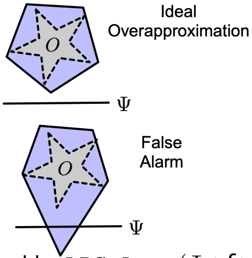
ReLU Activation function.

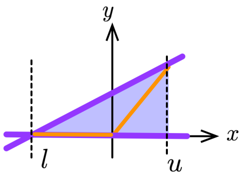
Over-Approximation for Neural Network Verification

Specification $\Phi \wedge \Psi$ Input: $\Phi := x_1 \in [0,1] \wedge x_2 \in [0,1]$ Output: $\Psi = (O \geq 0)$

Network f







DeepPoly[3]

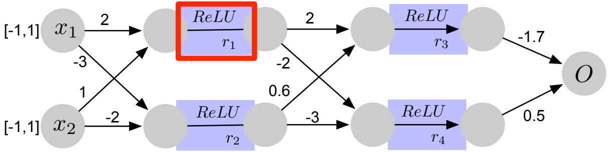
Lowerbound $\hat{p} = \min O$, computed by $\texttt{LPSolver}(\Phi \wedge f \wedge \Psi)$ $\hat{p} = -\mathbf{2.7}$ obtained by conservative over-approximation of active functions (i.e., ReLU) via linear solver and can be **imprecise (incomplete)** and may produce a **false alarm**, i.e., $\hat{p}! =! - 2.7$ is a **spurious value** that never occurs during runtime.

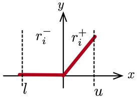
3 Singh et al., Abstract Domain and Analysis for ReLU Neural Networks, POPL'19

Branch-and-Bound-Based Approach (Bunel+, JMLR'20)

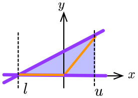
Specification: $\Phi \wedge \Psi$ Input: $\Phi := x_1 \in [-1,1] \wedge x_2 \in [-1,1]$ Output: $\Psi = (O > 0)$

Network: f





Precise spliting ReLU r_1

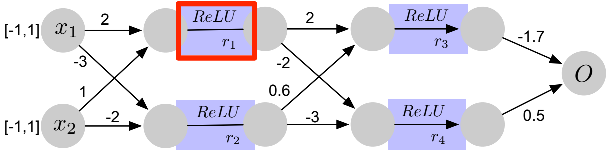


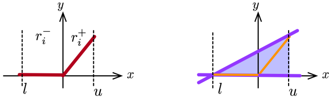
Over-approximations for ReLUs r_2 r_3 r_4

Branch-and-Bound-Based Approach (Bunel+, JMLR'20)

Specification: $\Phi \wedge \Psi$ Input: $\Phi := x_1 \in [-1,1] \wedge x_2 \in [-1,1]$ Output: $\Psi = (O > 0)$

Network: f





Precise spliting ReLU r_1 Over-approximations for ReLUs r_2 r_3 r_4

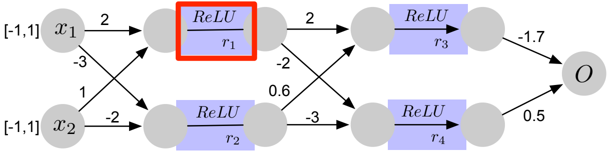
- The branch-and-bound[a] **aims to achieve ideal precise verification** by dividing a problem into subproblems (**branch**) and eliminating those that cannot lead to an optimal solution (**bound**)

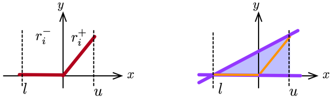
[a] https://en.wikipedia.org/wiki/Branch_and_bound

Branch-and-Bound-Based Approach (Bunel+, JMLR'20)

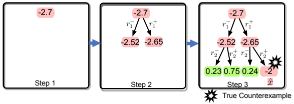
Specification: $\Phi \wedge \Psi$ Input: $\Phi := x_1 \in [-1,1] \wedge x_2 \in [-1,1]$ Output: $\Psi = (O > 0)$

Network: f





Precise spliting ReLU r_1 **Over-approximations for ReLUs** r_2 r_3 r_4



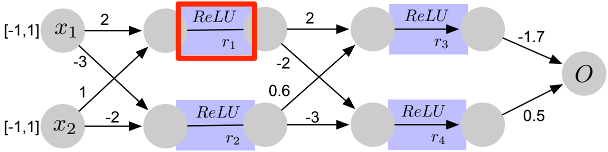
- The branch-and-bound[a] **aims to achieve ideal precise verification** by dividing a problem into subproblems (**branch**) and eliminating those that cannot lead to an optimal solution (**bound**)

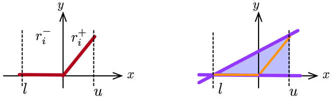
[a] https://en.wikipedia.org/wiki/Branch_and_bound

Branch-and-Bound-Based Approach (Bunel+, JMLR'20)

Specification: $\Phi \wedge \Psi$ Input: $\Phi := x_1 \in [-1, 1] \wedge x_2 \in [-1, 1]$ Output: $\Psi = (O > 0)$

Network: f

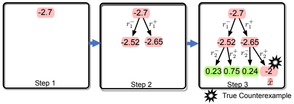




Precise spliting ReLU r_1 **Over-approximations for ReLUs** r_2 r_3 r_4

- The branch-and-bound[a] **aims to achieve ideal precise verification** by dividing a problem into subproblems (**branch**) and eliminating those that cannot lead to an optimal solution (**bound**)

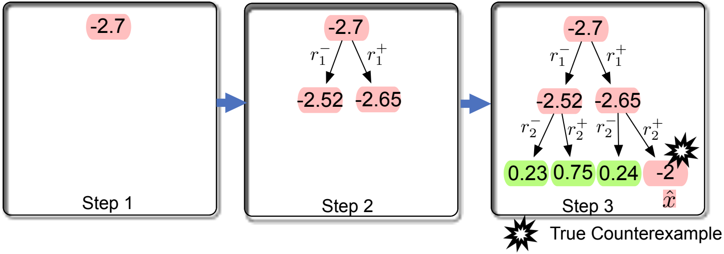
[a] https://en.wikipedia.org/wiki/Branch_and_bound



Branch-and-Bound (BaB) Tree

- Split activation *ReLU* r_1 **input** into r_1^+ $(x_1 \geq 0)$ and r_1^- $(x_1 < 0)$.
- Split activation *ReLU* r_2 **input** into r_2^+ $(x_2 \geq 0)$ and r_2^- $(x_2 < 0)$.

Counterexample-Guided Verification

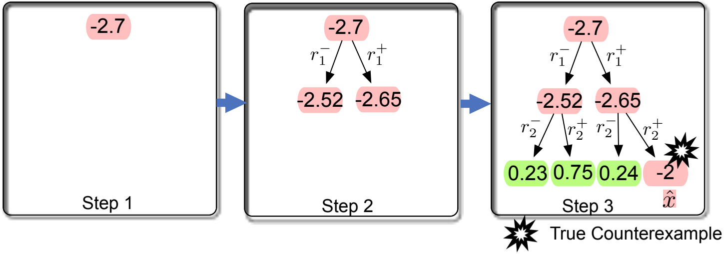


- **Counterexamples**, i.e., inputs violating specifications, can be found in partitioned problem spaces via BaB trees, enabling early termination.

[5] Bunel et al., Branch and bound for piecewise linear neural network verification, JMLR'20

[4] Fukuda et al., Adaptive Branch-and-Bound Tree Exploration for Neural Network Verification, DATE'25

Counterexample-Guided Verification

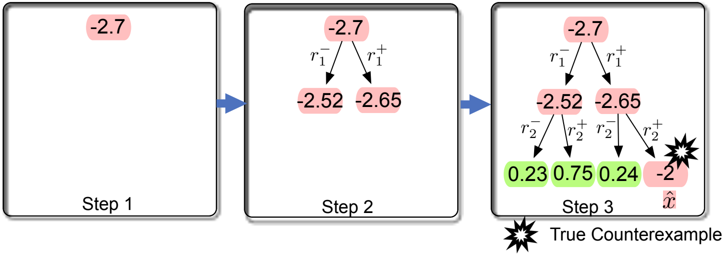


- **Counterexamples**, i.e., inputs violating specifications, can be found in partitioned problem spaces via BaB trees, enabling early termination.
- **Efficiently Finding** counterexamples is the **key** to scalable NN verification!

[5] Bunel et al., Branch and bound for piecewise linear neural network verification, JMLR'20

[4] Fukuda et al., Adaptive Branch-and-Bound Tree Exploration for Neural Network Verification, DATE'25

Counterexample-Guided Verification

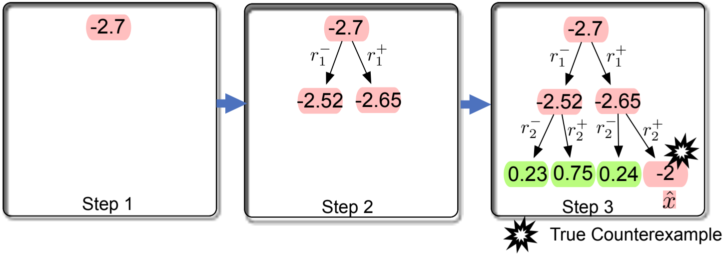


- **Counterexamples**, i.e., inputs violating specifications, can be found in partitioned problem spaces via BaB trees, enabling early termination.
- **Efficiently Finding** counterexamples is the **key** to scalable NN verification!
- Conventional BaB algorithm[5] (**breadth-first search**) can be **inefficient**

[5] Bunel et al., Branch and bound for piecewise linear neural network verification, JMLR'20

[4] Fukuda et al., Adaptive Branch-and-Bound Tree Exploration for Neural Network Verification, DATE'25

Counterexample-Guided Verification

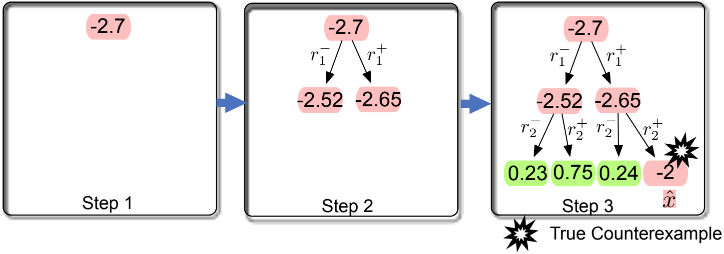


- **Counterexamples**, i.e., inputs violating specifications, can be found in partitioned problem spaces via BaB trees, enabling early termination.
- **Efficiently Finding** counterexamples is the **key** to scalable NN verification!
- Conventional BaB algorithm[5] (**breadth-first search**) can be **inefficient**
- **Potentiality** of **counterexample existence** can be inferred by two attributes[4]:
  1. Tree nodes (**output lower bound** $\hat{p}$) with smaller values.
  2. Tree node's **depth** with deeper level.

[5] Bunel et al., Branch and bound for piecewise linear neural network verification, JMLR'20

[4] Fukuda et al., Adaptive Branch-and-Bound Tree Exploration for Neural Network Verification, DATE'25

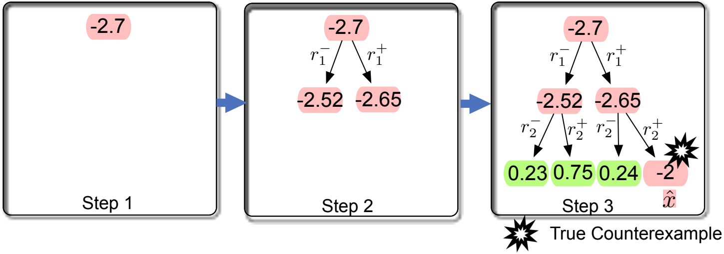
MCTS-Based Method (Fukuda+, DATE'25)



- (Fukuda+, DATE'25) adopts Monte Carlo Tree Search (MCTS)

[4] Fukuda et al., Adaptive Branch-and-Bound Tree Exploration for Neural Network Verification, DATE'25

MCTS-Based Method (Fukuda+, DATE'25)

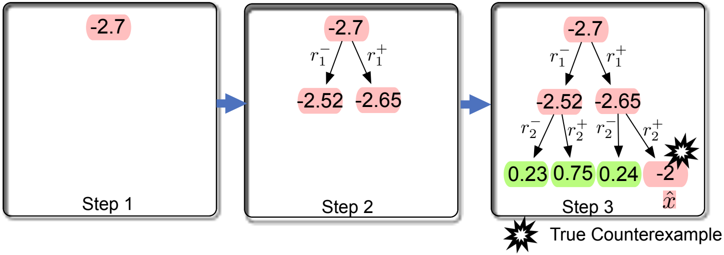


- (Fukuda+, DATE'25) adopts Monte Carlo Tree Search (MCTS)
- **Compute Rewards**[4] (counterexample potentiality (CePO)) of subproblems:

$$[\![\Gamma]\!] = \begin{cases} -\infty & \textit{if } \hat{p} > 0 \\ +\infty & \textit{true CE} \\ \lambda \frac{|\Gamma|}{K} + (1-\lambda)\frac{\hat{p}}{\hat{p}_{min}} & \textit{otherwise} \end{cases}$$

[4] Fukuda et al., Adaptive Branch-and-Bound Tree Exploration for Neural Network Verification, DATE'25

MCTS-Based Method (Fukuda+, DATE'25)



- (Fukuda+, DATE'25) adopts Monte Carlo Tree Search (MCTS)
- **Compute Rewards**[4] (counterexample potentiality (CePO)) of subproblems:

$$[\![\Gamma]\!] = \begin{cases} -\infty & \textit{if } \hat{p} > 0 \\ +\infty & \textit{true CE} \\ \lambda \frac{|\Gamma|}{K} + (1-\lambda)\frac{\hat{p}}{\hat{p}_{min}} & \textit{otherwise} \end{cases}$$

- (Fukuda+, DATE'25) outperforms naive BaB (breadth-first search) approach

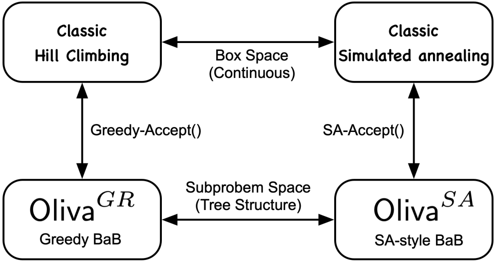
[4] Fukuda et al., Adaptive Branch-and-Bound Tree Exploration for Neural Network Verification, DATE'25

Limitations and Motivations

- MCTS-based approach (Fukuda+, DATE'25) is **deterministic**:
  - If this MCTS **failed** to find a counterexample, repeating the same run is meaningless and it does not give a different answer.
  - Inferring counterexamples (MCTS rewards) is a **heuristic** method, and it may **fail** to provide accurate guidance frequently.

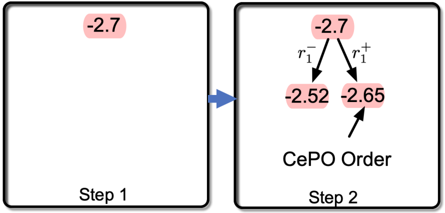
- This work is a **stochastic optimization-based** approach
  - Counterexample finding through an effective optimization-based sampling, e.g., **hill climbing** (HC), **simulated annealing** (SA)
  - **Repeated** runs with different seeds can explore the tree in different ways.

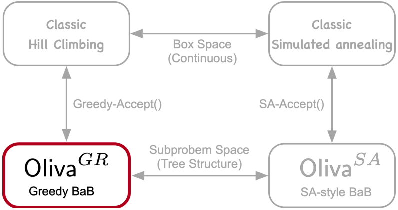
Contribution



- We propose Oliva that adopts stochastic optimization for neural network verification
  1. Oliva^{GR}: a greedy approach
  2. Oliva^{SA}: simulated annealing (SA)-style approach
- Oliva^{SA} is achieved by identifying and extending **two relations**:
  1. Relation between Oliva^{GR} and classic hill climbing
  2. Relation between classic simulated annealing and classic hill climbing

Greedy Approach (Oliva^{GR})

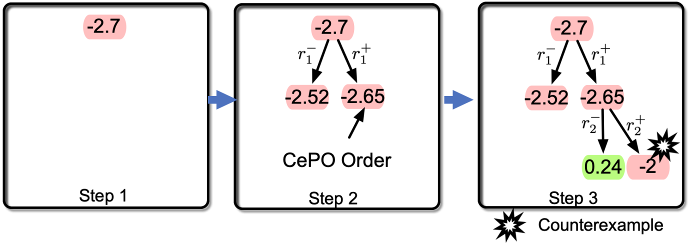


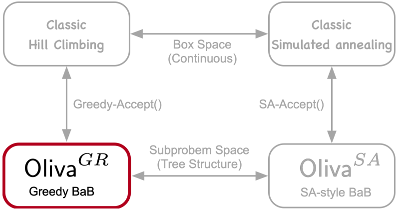


Oliva^{GR} is inspired by (Fukuda+,DATE'25)

- Driven by Greediness, we directly select **deeper** and **smaller** ones, until the subproblem is verified;

Greedy Approach (Oliva^{GR})

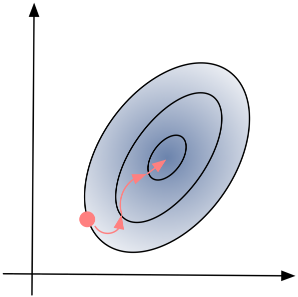


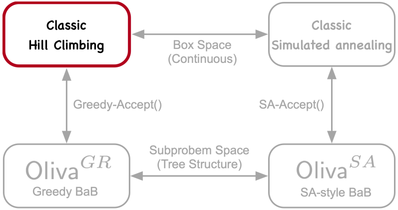


Oliva^{GR} is inspired by (Fukuda+, DATE'25)

- Driven by Greediness, we directly select **deeper** and **smaller** ones, until the subproblem is verified;
- Oliva^{GR} may fail to find a counterexample (even if it exists)
- "CePO" order is a **heuristic**, but not always promising.

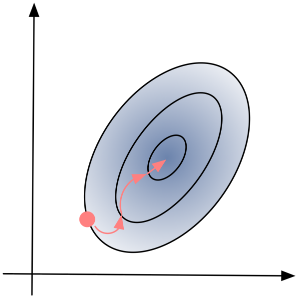
Connection Between Hill Climbing and Oliva^{GR}

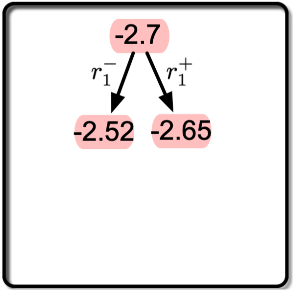


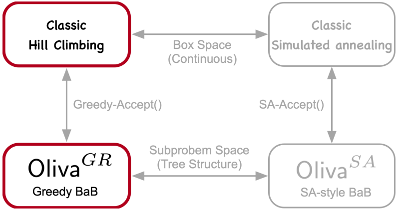


- **Hill Climbing** samples and optimizes within a continuous box domain, gradually converging to a local optimum.

Connection Between Hill Climbing and Oliva^{GR}

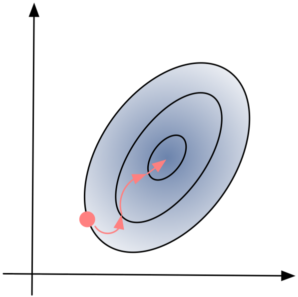


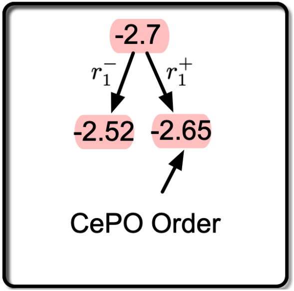


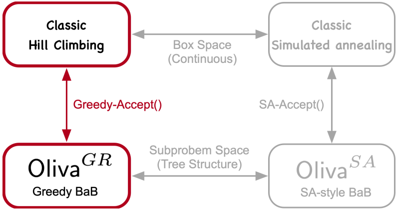


- **Hill Climbing** optimizes within a continuous box domain by sampling, gradually converging to a local optimum.
- Oliva^{GR} works on a **tree structure** to select the subproblems.

Connection Between Hill Climbing and Oliva^{GR}

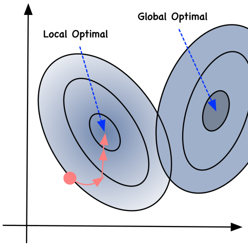


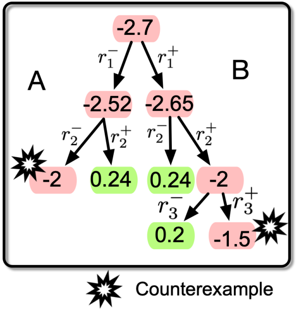


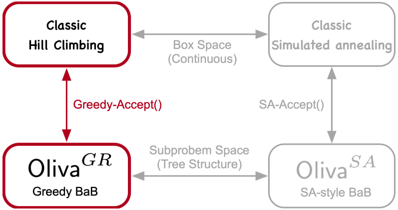


- Connection: action of **accepting** better candidates.
  - Oliva^{GR}: accept only good child nodes
  - Hill climbing: accept only good samples
- We build the edge between the two that shares the same "greedily accept" policy.

Issues of Hill Climbing and Oliva^{GR}

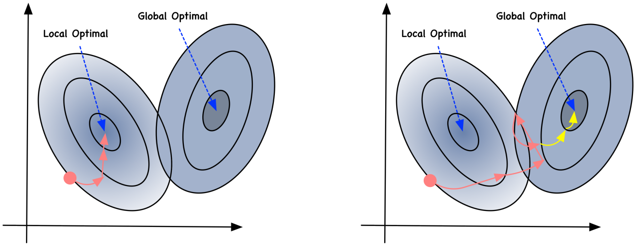


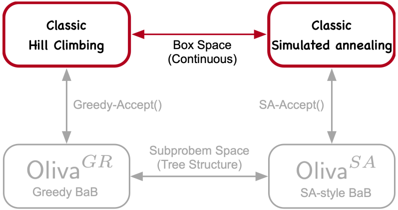




- The issues are also essentially the same:
  - Hill climbing can be trapped in local optima and lose the global optimum;
  - Oliva^{GR} can be misled by the heuristic order, resulting in suboptimal performance

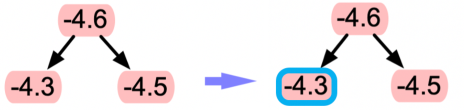
Classic Simulated Annealing (SA)

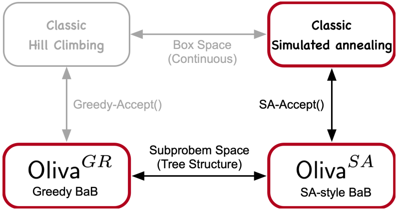




- In stochastic optimization, SA is a solution to "local optima" issue of hill climbing;
- Compared to hill climbing, it assigns a probability to **accept** a worse sample;
- The probability keeps evolving over the process, controlled by **temperature**
  - At initial stage, temperature is high, SA favors **exploration**;
  - Later, temperature becomes low, SA favors **exploitation**.

Our Proposed Oliva^{SA} Approach





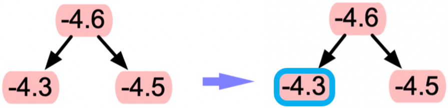
While $T \leftarrow \alpha \cdot T$

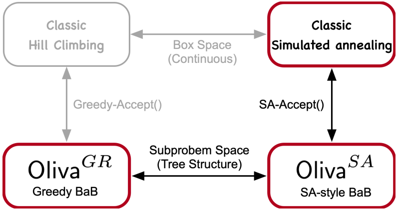
$$\Delta p \leftarrow \exp\left(\frac{\min \mathsf{R}(\Gamma\cdot a) - \max \mathsf{R}(\Gamma\cdot a)}{T}\right) \quad \text{s.t.} \;\; a \in \{r_k^+, r_k^-\}$$

$$\Gamma^* \leftarrow \Gamma \cdot a^* \quad \text{s.t.} \;\; a^* \leftarrow \begin{cases} \text{randomly choose } r_k^+ \text{ or } r_k^- & \text{if } \mathtt{rand}(0,1) < \Delta p \\ \underset{a \in \{r_k^+, r_k^-\}}{\arg\max}\ \mathsf{R}(\Gamma \cdot a) & \text{otherwise} \end{cases}$$

- Oliva^{SA} extends the **accept policy** of classic SA to **tree structures**

Our Proposed Oliva^{SA} Approach





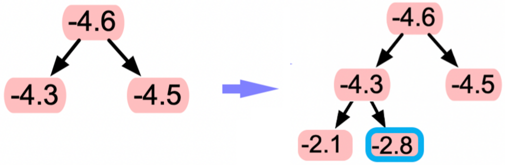
While $T \leftarrow \alpha \cdot T$

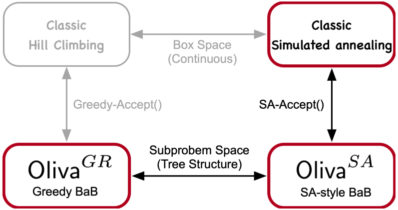
$$\Delta p \leftarrow \exp\left(\frac{\min \mathrm{R}(\Gamma\cdot a) - \max \mathrm{R}(\Gamma\cdot a)}{T \uparrow}\right) \quad \text{s.t.} \quad a \in \{r_k^+, r_k^-\}$$

$$\Gamma^* \leftarrow \Gamma \cdot a^* \quad \text{s.t.} \quad a^* \leftarrow \begin{cases} \text{randomly choose } r_k^+ \text{ or } r_k^- & \text{if } \mathtt{rand}(0,1) < \Delta p \uparrow \\ \arg\max_{a\in\{r_k^+, r_k^-\}} \mathrm{R}(\Gamma\cdot a) & \text{otherwise} \end{cases}$$

- Oliva^{SA} extends the **accept policy** of classic SA to **tree structures**
  - At initial stage, temperature is high, so it favors **exploration**

Our Proposed Oliva^{SA} Approach





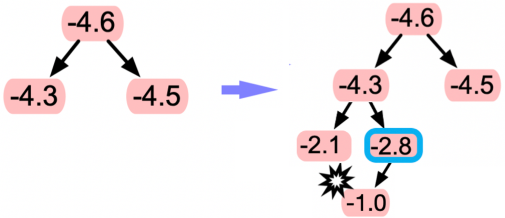
While $T \leftarrow \alpha \cdot T$

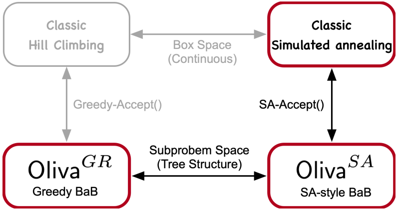
$$\Delta p \leftarrow \exp\left(\frac{\min \mathsf{R}(\Gamma \cdot a) - \max \mathsf{R}(\Gamma \cdot a)}{T \downarrow}\right) \quad \text{s.t. } a \in \{r_k^+, r_k^-\}$$

$$\Gamma^* \leftarrow \Gamma \cdot a^* \quad \text{s.t. } a^* \leftarrow \begin{cases} \text{randomly choose } r_k^+ \text{ or } r_k^- & \text{if } \mathtt{rand}(0,1) < \Delta p \downarrow \\ \arg\max\limits_{a \in \{r_k^+, r_k^-\}} \mathsf{R}(\Gamma \cdot a) & \text{otherwise} \end{cases}$$

- Oliva^{SA} extends the **accept policy** of classic SA to **tree structures**
  - At initial stage, temperature is high, so it favors **exploration**
  - Later, temperature becomes low, so it favors **exploitation**

Our Proposed Oliva^{SA} Approach





While $T \leftarrow \alpha \cdot T$

$$\Delta p \leftarrow \exp\left(\frac{\min \mathsf{R}(\Gamma \cdot a) - \max \mathsf{R}(\Gamma \cdot a)}{T\downarrow}\right) \quad \text{s.t.} \quad a \in \{r_k^+, r_k^-\}$$

$$\Gamma^* \leftarrow \Gamma \cdot a^* \quad \text{s.t.} \quad a^* \leftarrow \begin{cases} \text{randomly choose } r_k^+ \text{ or } r_k^- & \text{if } \texttt{rand}(0,1) < \Delta p \downarrow \\ \arg\max_{a \in \{r_k^+, r_k^-\}} \mathsf{R}(\Gamma \cdot a) & \text{otherwise} \end{cases}$$

- Oliva^{SA} extends the **accept policy** of classic SA to **tree structures**
  - At initial stage, temperature is high, so it favors **exploration**
  - Later, temperature becomes low, so it favors **exploitation**

Comparison with (Fukuda+, DATE'25)

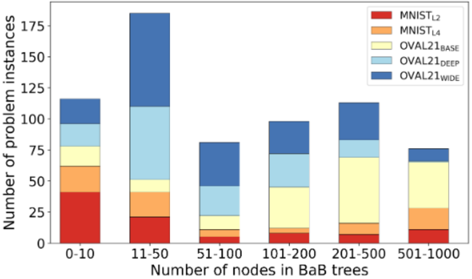
- Major technical differences
  - **Monte Carlo Tree Search** (i.e., MCTS in Fukuda+, DATE'25) originally deals with **tree structures**, so the application to BaB is **relatively straightforward**
  - Simulated Annealing (SA and **general stochastic optimization algorithms**) originally deals with **box domains**, so the application to BaB requires a **novel way** of adaptation in this work.

- Regarding verification effectiveness
  - MCTS involves a **fixed policy** of tree exploration; repeating the same run does not give a different answer.
  - Oliva^{SA} is stochastic, so **repeating experimental runs is useful** to find counterexamples

Experiment Settings

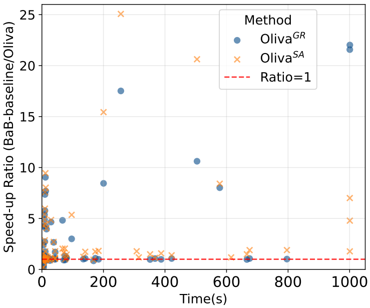
Model	Architecture	Dataset	#Activations	# Instances	#Images
$\texttt{MNIST}_{\texttt{L2}}$	2 × 256 linear	MNIST	512	100	70
$\texttt{MNIST}_{\texttt{L4}}$	4 × 256 linear	MNIST	1024	78	52
$\texttt{OVAL21}_{\texttt{BASE}}$	2 Conv, 2 linear	CIFAR-10	3172	173	53
$\texttt{OVAL21}_{\texttt{WIDE}}$	2 Conv, 2 linear	CIFAR-10	6244	196	53
$\texttt{OVAL21}_{\texttt{DEEP}}$	4 Conv, 2 linear	CIFAR-10	6756	143	40

- Following VNN-COMP[a]:
  - 690 instance across MNIST, CIFAR-10, with five different models.
  - Local robustness with $\epsilon \in [1/255, 16/255]$
  - Meaningful sub-problem selection.



[a]Müller et al., *arXiv preprint arXiv:2212.10376*.

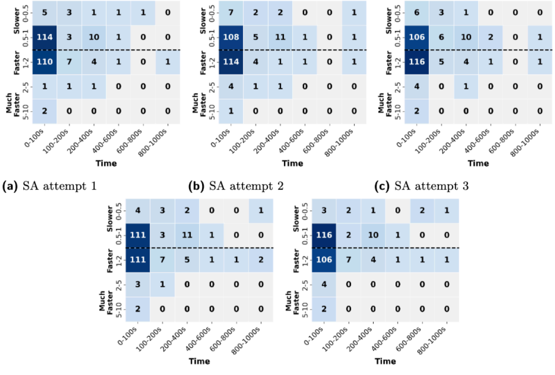
Experiment Results I



MNIST-L2 by 100 problem instances

- Each point is a verification problem
  - x-axis: time costs by BaB-baseline
  - y-axix: our speedup over BaB-baseline
- Points over the dashed red line are **faster** than BaB-baseline.

Experiments on Stochasticity of Oliva^{SA}



(a) SA attempt 1 **(b)** SA attempt 2 **(c)** SA attempt 3

(d) SA attempt 4 **(e)** SA attempt 5

Performances of 5 different runs of Oliva^{SA}

- Overall, the performance of Oliva^{SA} is stable;
- We observe such cases: while by one run we cannot find counterexamples, by repeating the same run with different seeds, we manage to find counterexamples.

Summary and Research Opportunities

- We propose Oliva, a metaheuristic optimization tool:
  1. Oliva^{GR}: Greedily driven by **Potentiality**, generalize "acceptance" in "hill-climbing" optimization.
  2. Oliva^{SA}: Simulated annealing **mitigates** the "greediness" of "hill-climbing" as a **stochastic** optimization.
- Other directions and our ongoing work for counterexample-guided NN verification
  1. Scalable incremental falsification of neural networks given a similar NN architecture and existing verification results.
  2. Efficient verification of the Transformer architecture and large language models.