Summary Slides

MILP Based Verification of Neural Networks

Presented by
Zhong Yuyi

Robustness Verification

- Recent research showing that even highly accurate NNs are vulnerable to adversarial examples.
- Wrongly label adversarial examples: typically obtained by slightly perturbing
- Prove robustness of neural networks:
  - Incomplete methods - abstract interpretation:
  - Overestimate the output of the network from a given input region
  - Draw the conclusion from the over-approximation
  - Pros: very efficient, easy to scale up
  - Cons: give false negatives, conclude that the network is not robust when it actually is

Complete Methods

- Can be divided into 3 main groups
  - MILP-based that represent the problem as Mixed Integer Linear Program
  - SMT-based that encode the problem as the satisfiability modulo theory problem
  - Techniques that use a combination of overestimation and refinement techniques to get a definite answer
- Complete verifiers reason over the exact result.
- Given sufficient time, a complete verifier can provide a definite answer
- Challenge: conquering scalability

MILP Based Verification

- Tjeng V, etc. Evaluating robustness of neural networks with mixed integer programming. ICLR 2019.
- Botoeva E, etc. Efficient Verification of ReLU-based Neural Networks via Dependency Analysis. AAAI 2020.

1. Formulating robustness evaluation as an MILP (mixed-integer linear programming)
2. Call existing MILP solvers
    - MILP program is feasible iff the answer to the verification problem is no.
    - The network is robust to perturbations on x iff MILP is infeasible

Contributions

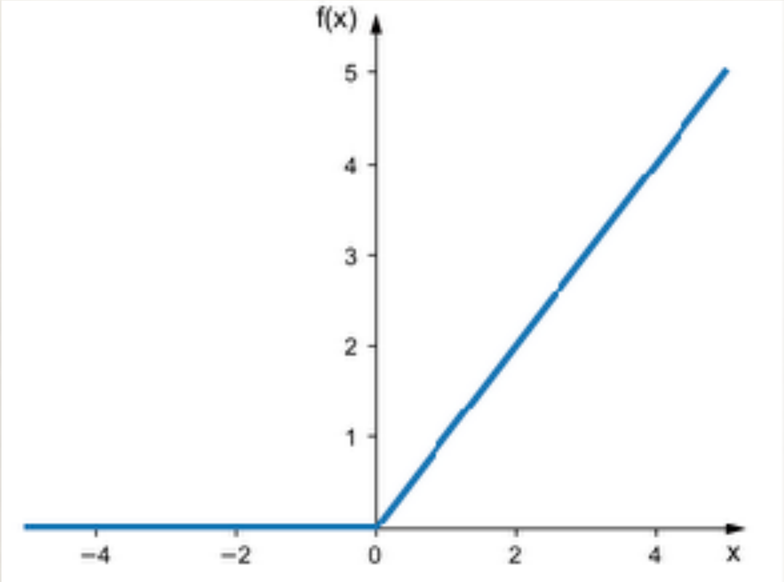
Tjeng V, etc. Evaluating robustness of neural networks with mixed integer programming. ICLR 2019.

The approach improves upon existing MILP-based approaches with

- A tighter formulation for non-linearities (ReLU)
- A novel presolve algorithm that makes use of all information available, leading to solve times several orders of magnitude faster than a naively implemented MILP-based approach.

Verification as Solving MILP

- A neural network $f(\theta) : R^m \rightarrow R^n$ parameterized by a vector of weights θ, composed of
  - Linear transformations (such as fully-connected, convolution layers)
  - Piecewise-linear function, a function composed of some number of linear segments defined over intervals (such as ReLU)



- General definition of verification: determine whether some property P on the output of a neural network holds for all input in a bounded input domain $C \subseteq R^m$.

- Expressible as solving an MILP, P must be expressible as the conjunction or disjunction of linear properties $P_{i,j}$ over some set of polyhedra C_i, where $C = \cup C_i$.

What is MILP?

Acknowledgement:

Mixed Integer Linear Programming from Javier Larrosa, Albert Oliveras and Enric Rodriguez-Carbonell https://www.cs.upc.edu/%7Eerodri/webpage/cps/theory/lp/milp/slides.pdf

- A mixed integer linear program (MILP, MIP) is of the form

$$\begin{aligned} &\min\ c^T x \\ &Ax = b \\ &x \geq 0 \\ &x_i \in \mathbb{Z}\ \ \forall i \in \mathcal{I} \end{aligned}$$

- If all variables need to be integer, it is called a (pure) integer linear program (ILP, IP)

- If all variables need to be 0 or 1 (binary, boolean), it is called a $0-1$ linear program

MILP: linear constraints, integrality constraints ($x_i \in Z$)

$$\begin{aligned} &\max\ x + y \\ &\quad -2x + 2y \geq 1 \\ &\quad -8x + 10y \leq 13 \\ &\quad x, y \geq 0 \\ &\quad x, y \in \mathbb{Z} \end{aligned}$$

What is MILP?

- Including integer variables increases enourmously the modeling power, at the expense of more complexity
- LP's can be solved in polynomial time with interior-point methods (ellipsoid method, Karmarkar's algorithm)
- Integer Programming is an NP-complete problem. So:
  - There is no known polynomial-time algorithm
  - There are little chances that one will ever be found
  - Even small problems may be hard to solve

How to Solve MILP?

- Given a MIP

$$(IP) \quad \begin{aligned} &\min\ c^T x \\ &Ax = b \\ &x \geq 0 \\ &x_i \in \mathbb{Z} \quad \forall i \in \mathcal{I} \end{aligned}$$

its linear relaxation is the LP obtained by dropping integrality constraints:

$$(LP) \quad \begin{aligned} &\min\ c^T x \\ &Ax = b \\ &x \geq 0 \end{aligned}$$

Formulate Robustness Evaluation as MILP

1. Evaluating Robustness
2. Evaluating Minimum Adversarial Distortion

Evaluating Robustness

- $\mathcal{G}(x)$ denote a region in the input domain corresponding to all allowable perturbations of a particular input x
- Perturbed inputs must also remain in the domain of valid inputs $\mathcal{X}_{valid}$.
- A neural network is robust, if the predicted probability of the true label $\lambda(x)$ exceeds that of every other label for all perturbations on x:

$$\forall x' \in (\mathcal{G}(x) \cap \mathcal{X}_{valid}) : \ \mathrm{argmax}_i(f_i(x')) = \lambda(x)$$

- Example $\mathcal{G}(x) = \{x' \mid \forall i : -\epsilon \le (x - x')_i \le \epsilon\}$
- Equivalently, the network is robust to perturbations on x iff Equation 2 (the MILP) is infeasible for x', where $f_i(\cdot)$ is the i^{th} output of the network

$$(x' \in (\mathcal{G}(x) \cap \mathcal{X}_{valid})) \wedge \left(f_{\lambda(x)}(x') < \max_{\mu \in [1,n] \setminus \{\lambda(x)\}} f_\mu(x') \right) \qquad (2)$$

Formulate Robustness Evaluation as MILP

Evaluating Minimum Adversarial Distortion

- Let $d(\cdot, \cdot)$ denote a distance metric that measures the similarity between two images
- The minimum adversarial distortion under d for input x with true label $\lambda(x)$ corresponds to the solution to the optimization:

$$
\begin{aligned}
& \min_{x'} d(x', x) && (3) \\
\text{subject to} \quad & \operatorname{argmax}_i (f_i(x')) \neq \lambda(x) && (4) \\
& x' \in \mathcal{X}_{valid} && (5)
\end{aligned}
$$

- Example: use distance metric l_1 where $d(x', x) = \|x' - x\|_1$
- Introduce the auxiliary variable , which bounds the element-wise absolute value from above $\delta_j \geq x'_j - x_j$, $\delta_j \geq x_j - x'_j$

$$
\begin{aligned}
& \min_{x'} \sum_j \delta_j && (17) \\
\text{subject to} \quad & \operatorname{argmax}_i (f_i(x')) \neq \lambda(x) && (18) \\
& x' \in \mathcal{X}_{valid} && (19) \\
& \delta_j \geq x'_j - x_j && (20) \\
& \delta_j \geq x_j - x'_j && (21)
\end{aligned}
$$

Formulate Network in MILP Framework

- A neural network is composed of
  - Linear transformations (such as fully-connected, convolution layers)
  - Piecewise-linear function, non-linear function (such as ReLU)

Tight formulations of ReLU is critical to good performance of the MILP solver

- Assume that we have element-wise bounds on the inputs to ReLU
- Let $y = max(x,0)$, and $l \leq x \leq u$, it includes three possibilities:
- Stably inactive: $u \leq 0$, then $y \equiv 0$
- Stably active: $l \geq 0$, then $y \equiv x$
- Unstable: introduce an indicator decision variable $a = 1_{x \geq 0}$
- Is equivalent to the set of linear and integer constraints in Equation 6.

$$(y \leq x - l(1-a)) \wedge (y \geq x) \wedge (y \leq u \cdot a) \wedge (y \geq 0) \wedge (a \in \{0,1\}) \tag{6}$$

Linear Constraints for ReLU

We reproduce our formulation for the ReLU below.

$$y \leq x - l(1 - a) \tag{8}$$
$$y \geq x \tag{9}$$
$$y \leq u \cdot a \tag{10}$$
$$y \geq 0 \tag{11}$$
$$a \in \{0, 1\} \tag{12}$$

- When $a = 0$, the constraints in Equation 10 and 11 are binding, and together imply that y = 0
- When $a = 1$, the constraints in Equation 8 and 9 are binding, and together imply that y = x.
- The idea of “Stably inactive” and “Stably active” helps to reduce the number of auxiliary binary variables.

Progressive Bound Tightening

- We previously assumed that we have element-wise bounds on the inputs to ReLU. In practice, we have to carry out a presolve step to determine these bounds.
- Determining tight bounds is critical for problem tractability: tight bounds strengthen the problem formulation and thus improve solve times - if we can prove that the phase of a ReLU is stable, we can avoid introducing a binary variable
- They use two procedures to determine bounds: interval arithmetic (IA); the slower but tighter linear programming (LP) approach.
- A tradeoff between higher build times (to determine tighter bounds on inputs to non-linearities), and higher solve times (to solve the main MILP problem in Equation 2 or Equation 3-5).

$$(x' \in (\mathcal{G}(x) \cap \mathcal{X}_{valid})) \wedge \left(f_{\lambda(x)}(x') < \max_{\mu \in [1,n] \setminus \{\lambda(x)\}} f_{\mu}(x') \right) \tag{2}$$

$$\begin{aligned} & \min_{x'} d(x', x) && (3)\\ \text{subject to} \quad & \operatorname{argmax}_i(f_i(x')) \neq \lambda(x) && (4)\\ & x' \in \mathcal{X}_{valid} && (5) \end{aligned}$$

Progressive Bound Tightening

- The knowledge of the non-linearities allows us to reduce average build times without affecting the strength of the problem formulation
- There are thresholds beyond which further refining a bound will not improve the problem formulation.
- Eg: no need to further refine $0 \leq x \leq u$ to $1 \leq x \leq u$ or $0 \leq x \leq u - 1$
- **Progressive bounds tightening approach**
  - begin by determining coarse bounds using fast procedures
  - only spend time refining bounds using procedures with higher computational complexity if doing so could provide additional information to improve the problem formulation

Progressive Bound Tightening

GETBOUNDSFORRELU(x, fs)

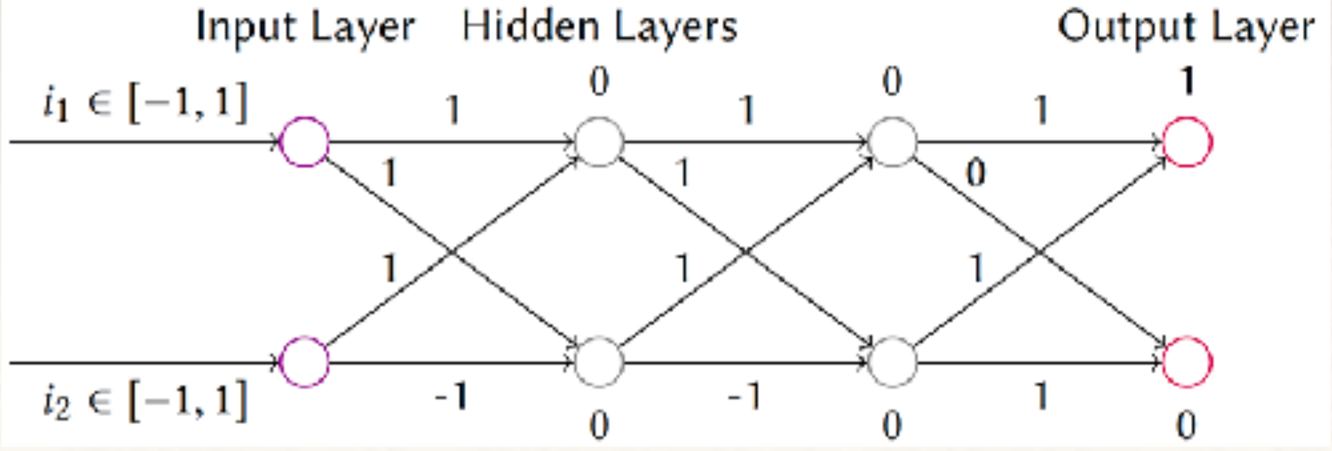
$\triangleright$ fs are the procedures to determine bounds, sorted in increasing computational complexity.
$l_{best} = -\infty$; $u_{best} = \infty$ $\quad\triangleright$ initialize best known upper and lower bounds on x
for f in fs: $\quad\triangleright$ carrying out progressive bounds tightening
    do $u = f(x, boundType = upper)$; $u_{best} = \min(u_{best}, u)$
        if $u_{best} \leq 0$ **return** (l_{best}, u_{best}) $\quad\triangleright$ Early return: $x \leq u_{best} \leq 0$; thus $\max(x, 0) \equiv 0$.
        $l = f(x, boundType = lower)$; $l_{best} = \max(l_{best}, l)$
        if $l_{best} \geq 0$ **return** (l_{best}, u_{best}) $\quad\triangleright$ Early return: $x \geq l_{best} \geq 0$; thus $\max(x, 0) \equiv x$
return (l_{best}, u_{best}) $\quad\triangleright$ x could be either positive or negative.

Progressive Bound Tightening

- The framework for determining bounds is to view the neural network as a computation graph G.
- Directed edges point from function input to output, and vertices represent variables.
- Source vertices in G correspond to the input of the network, and sink vertices correspond to the output
- The computation graph begins with defined bounds on the input variables (based on the input domain $\mathcal{G}(x) \cap \mathcal{X}_{valid}$)
- Any subgraph of G can be expressed as an MILP, with constraints derived from
  - input-output relationships along edges (additional integer constraints when edges describe a non-linear relationship)
  - bounds on the values of the source nodes in the subgraph.

Upper Bound Computation

- All the information required to determine the best possible bounds on variable v is contained in the subtree of G rooted at v, G_v
- Maximizing the value of v in the MILP M_v corresponding to G_v gives the optimal upper bound on v.
  - FULL: considers the full subtree G_v and does not relax any integer constraints in M_v. Get optimal bounds. But can be relatively inefficient, since solve times in the worst case are exponential in the number of binary variables in M_v
  - LINEAR PROGRAMMING (LP): considers the full subtree G_v but relaxes all integer constraints in M_v. A good middle ground between the optimality of FULL and the performance of IA.
  - INTERVAL ARITHMETIC (IA): considers the bounds on the variables in the previous layer, which is simply interval arithmetic. Efficient but can lead to overly coarse bounds for deep layers.



Experiments

- Dataset: MINIST & CIFAR-10
- Comparison to other MILP based verifier: expect several order of magnitude faster
- Comparison to other SMT based verifier Reluplex: improve on speed by 2-3 orders of magnitude
- Comparison to incomplete verifiers w.r.t. minimum adversarial distortion (a distance metric that measures the similarity between the input image and its adversarial image, greater the better): incomplete verifiers provide lower bounds while this tool can obtain the exact value.

Contributions

Botoeva E, etc. Efficient Verification of ReLU-based Neural Networks via Dependency Analysis. AAAI 2020.

The approach uses dependency relation to improve the performance of MILP formulation

- Develops effective methods to exploit these dependencies
- Reducing the search space during a <u>branch-and-bound</u> approach

Branch and Bound

Acknowledgement:

Mixed Integer Linear Programming from Javier Larrosa, Albert Oliveras and Enric Rodriguez-Carbonell https://www.cs.upc.edu/%7Eerodri/webpage/cps/theory/lp/milp/slides.pdf

Page 4, 7, 8

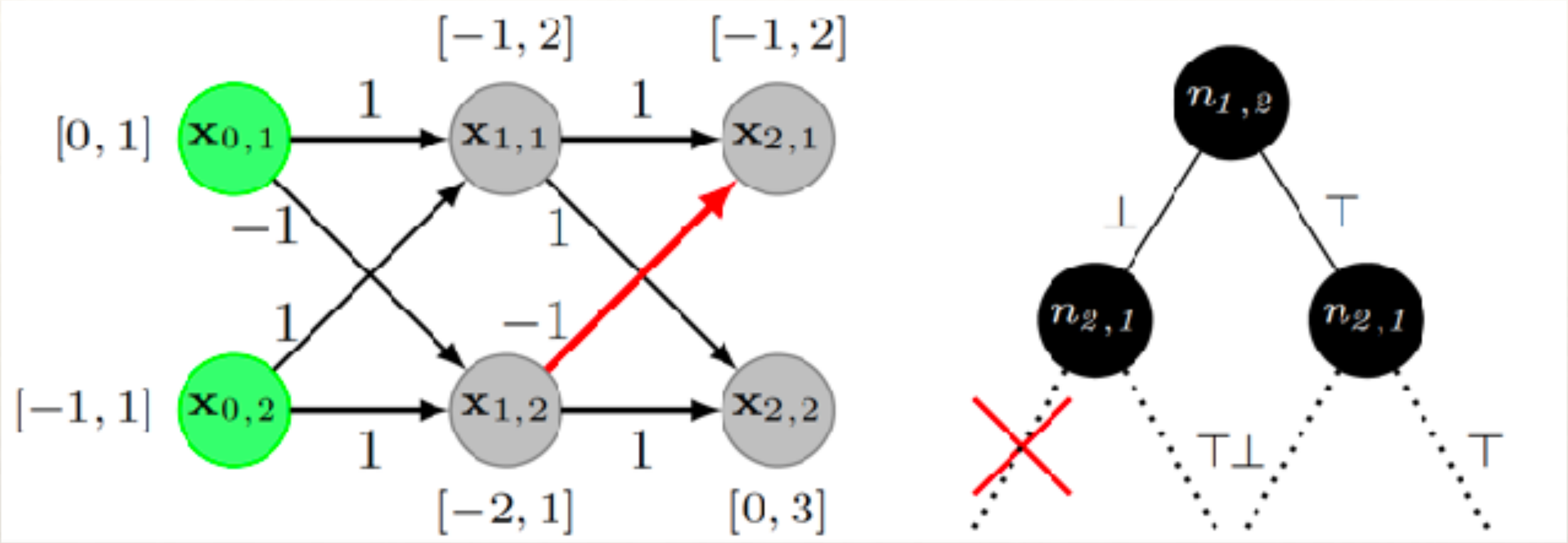
Dependency Relation

- $n_{i,q}$ to refer to the q-th node of layer i
- A node $n_{i,q}$'s output $x_{i,q}$ results from applying an activation function to the pre-activation $\hat{x}_{i,q}$ of the node, which is the weighted sum of the outputs of the nodes from the previous layer, $x_{i,q} = \mathrm{ReLU}(\hat{x}_{i,q})$
- Write $\hat{l}_{i,q}$ and $\hat{u}_{i,q}$ for the pre-activation's lower and upper bounds.
- Stably inactive: $\hat{u}_{i,q} \leq 0$, then $\mathrm{st}(n_{i,q}) = \bot$
- Stably active: $\hat{l}_{i,q} \geq 0$, then $\mathrm{st}(n_{i,q}) = \top$
- Else unstable then $\mathrm{st}(n_{i,q}) = ?$

Dependency relation: Given a neural network f that comprises a set of unstable nodes U, the dependency relation for $U, D_f \subseteq U \times U$ is the set of all pairs $(n_{i,q}, n_{j,r})$ such that $\mathrm{st}(n_{i,q}) \neq ? \implies \mathrm{st}(n_{j,r}) \neq ?$

Whenever $n_{i,q}$ is stable, $n_{j,r}$ has to be stable also

Dependency Relation



- Assume that a branch-and-bound method branches on node $n_{1,2}$, thereby splitting the optimisation problem into two sub-problems: one where $n_{1,2}$ is strictly active and one where $n_{1,2}$ is strictly inactive
- Consider the latter: We have $l_{1,2} = u_{1,2} = 0$, therefore $\hat{l}_{2,1} = 1 \cdot 0 + -1 \cdot 0 = 0$ and $\hat{u}_{2,1} = 1 \cdot 2 - 1 \cdot 0 = 2$
- Hence, $n_{2,1}$ is strictly active, and consequently $(n_{1,2}^{\perp}, n_{2,1}^{\top}) \in D_f$
- Each dependency provides a means to reduce the problem space by a factor of 1/4

How to Compute Dependency Relation

- Express dependency relations as unions of four disjoint sets $D_f = \cup_{z,z' \in \{\top,\bot\}} D_f^{z,z'}$
- Each $D_f^{z,z'} \doteq \{(n_{i,q}, n_{j,r}) \mid \mathrm{st}(n_{i,q}) = z \implies \mathrm{st}(n_{j,r}) = z'\}$, eg $(n_{1,2}^{\bot}, n_{2,1}^{\top})$
- Intra-layer dependencies: define $\hat{x}_{i,q,r=0}$ as the set of pre-activations of $n_{i,q}$ when the pre-activation of $n_{i,r}$ is zero

$$\hat{\mathbf{x}}_{i,q,r=0} \triangleq \{(W_i)_q \cdot \mathbf{x}_{i-1} + (b_i)_q \mid (W_i)_r \mathbf{x}_{i-1} + (b_i)_r = 0\}$$

Lemma 1. *For a neural network f and a pair of unstable nodes $(n_{i,q}, n_{i,r})$, the following hold:*

1. $(n_{i,q}, n_{i,r}) \in \mathcal{D}_f^{\top,\bot}$ *iff* $\hat{\mathbf{u}}_{i,q,r=0} < 0$ *and* $\hat{\mathbf{u}}_{i,r,q=0} < 0$.
2. $(n_{i,q}, n_{i,r}) \in \mathcal{D}_f^{\bot,\top}$ *iff* $\hat{\mathbf{l}}_{i,q,r=0} > 0$ *and* $\hat{\mathbf{l}}_{i,r,q=0} > 0$.
3. $(n_{i,q}, n_{i,r}) \in \mathcal{D}_f^{\top,\top}$ *iff* $\hat{\mathbf{u}}_{i,q,r=0} < 0$ *and* $\hat{\mathbf{l}}_{i,r,q=0} > 0$.
4. $(n_{i,q}, n_{i,r}) \in \mathcal{D}_f^{\bot,\bot}$ *iff* $\hat{\mathbf{l}}_{i,q,r=0} > 0$ *and* $\hat{\mathbf{u}}_{i,r,q=0} < 0$.

Lemma 1 gives a procedure for identifying intra-layer dependencies by computing the right hand side of each of the above clauses for every pair of unstable nodes in a layer.

How to Compute Dependency Relation

- Consecutive-layer dependencies: identifying consecutive layer dependencies by checking the right hand side of clauses (1) and (2) for every pair of unstable nodes in consecutive layers.

Lemma 3. *For a neural network f and a pair of unstable nodes $n_{i,q}, n_{j,r}$, for $j = i + 1$, the following hold:*

1. $(n_{i,q}, n_{j,r}) \in \mathcal{D}_f^{\perp,\perp} \Leftrightarrow \hat{\mathbf{u}}_{j,r} - (W_j)_{r,q} \cdot \mathbf{u}_{i,q} \leq 0.$
2. $(n_{i,q}, n_{j,r}) \in \mathcal{D}_f^{\perp,\top} \Leftrightarrow \hat{\mathbf{l}}_{j,r} - (W_j)_{r,q} \cdot \mathbf{u}_{i,q} \geq 0.$
3. $\mathcal{D}_f^{\top,\perp} = \emptyset$
4. $\mathcal{D}_f^{\top,\top} = \emptyset$

Takeaways

- Identifying dependency relations
- Leverage dependency relations to reduce the problem space during a branch-and-bound approach in MILP
- Compared to previous one, it leverages the "stable" idea to reduce the number of variables in the MILP formulation