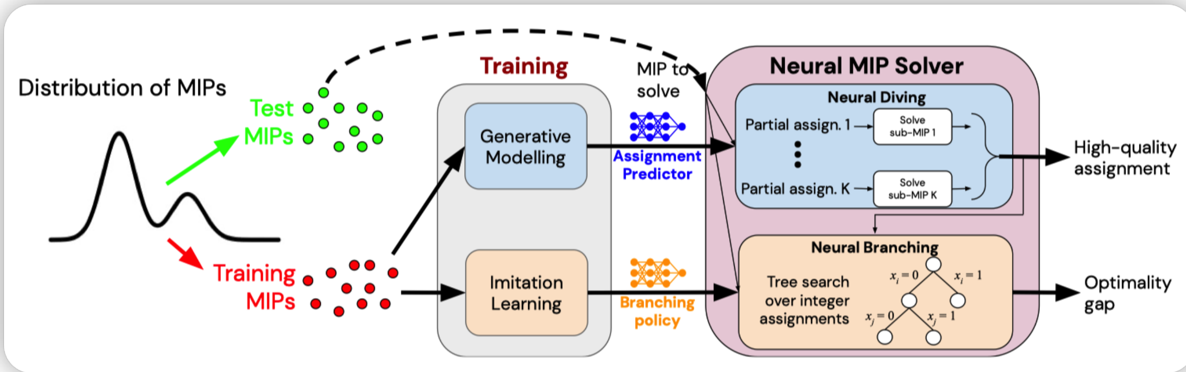
Solving MIPs Using NN

Summary of Solving Mixed Integer Programs Using Neural Networks

Nair, V., Bartunov, S., Gimeno, F., Von Glehn, I., Lichocki, P., Lobov, I., ... & Zwols, Y. (2020). Solving mixed integer programs using neural networks. *arXiv preprint arXiv:2012.13349*.

- Integer programming background
    - Preliminaries
- Neural diving
    - Neural network architecture
    - Evaluation
    - Energy function
    - Conditional generative modelling
    - Learning
    - Conditionally-independent model
- Neural branching
    - Imitating expert policy —— FSB
    - Expert score
    - Neural network architecture



Two neural network-based components to use in a MIP solver, Neural Diving and Neural Branching, and combine them to produce a Neural Solver customized to a given MIP dataset.

Integer programming background

A mixed integer linear program has the form

$$\begin{aligned} &minimize\ c^T x \\ &subject\ to\ Ax \leq b \\ &\qquad l \leq x \leq u \\ &\qquad x_i \in \mathbb{Z},\ i \in \mathcal{I} \end{aligned}$$

- A complete assignment is for any entry of $x \in \mathbb{R}^n$
- A partial assignment is an assignment that fixed some of the variable values
- A feasible assignment is an assignment that satisfies all the constraints in MIP
- An optimal assignment is a feasible assignment that also minimizes the objective

Preliminaries

- **LP relaxation:** Removing the integer constraints. The optimal value of the relaxed problem is guaranteed to be a lower bound for the original problem —— ***dual bound***.
- **Branch-and-Bound:** Recursively building a search tree with partial integer assignments at each node.

- **Primal heuristics:** A method that attempts to find a *feasible, but not necessarily optimal, assignment*. Any such feasible assignment provides an upper bound —— ***primal bound*** on the MIP
- **Primal-dual gap:**
  - Global primal bound: the minimum objective value of any feasible assignment
  - Global dual bound: the minimum dual bound across all leaves of the search tree
  - $gap = global\ primal\ bound - global\ dual\ bound$

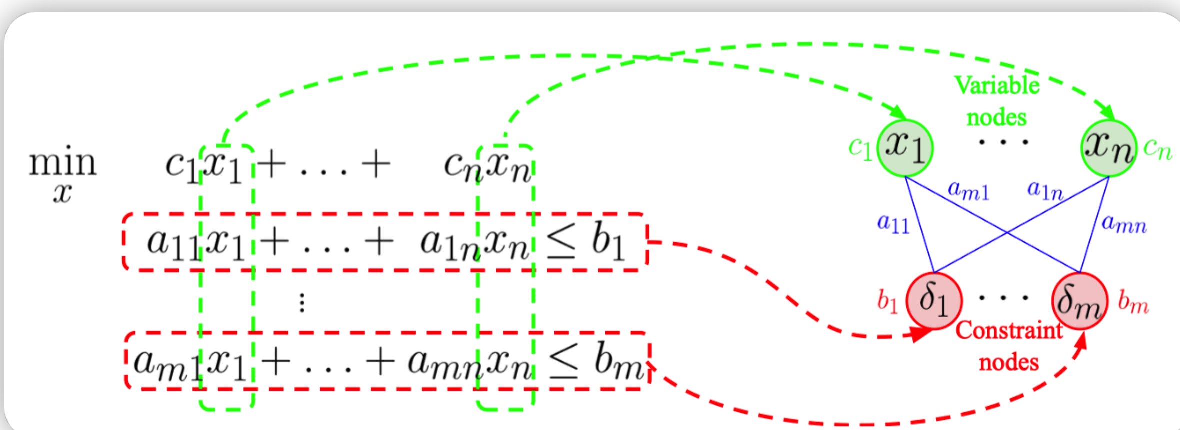
Neural diving

> ***This component finds high quality joint variable assignments.***

An instance of a primal heuristic training a deep neural network (***generative model***) to produce multiple partial assignments of integer variables and generate a series of sub-MIPs (solved by SCIP). Neural Diving gives higher probability to feasible assignments that have better objective values.

Neural network architecture

Using the form of graph convolutional network



Bipartite graph representation

Encoding a bipartite graph $G = (\mathcal{V}, \mathcal{E}, \mathcal{A})$ defined by the set of nodes $\mathcal{V}$, the set of edges $\mathcal{E}$, and the graph adjacency matrix $\mathcal{A}$.

- $\mathcal{V}$ is the union of n variable nodes and m constraint nodes, of size $N := |\mathcal{V}| = n + m$
- $\mathcal{A}$ is an $N \times N$ binary matrix (diagonal entries are 1) that $\mathcal{A}_{ij} = a_j i$ if constraint j involves variable i with coefficient a_{ij}
- The objective coefficients $\{c_1, \cdots c_n\}$ and the constraint bounds $\{b_1, \cdots, b_n\}$ are encoded in a $N \times D$ matrix U

A single-layer GCN learns to compute an *H-dimensional continuous vector* for each node:

$$
\begin{aligned}
&Z = \mathcal{A} f_\theta(U) \\
&Z^0 = U \quad \tilde{Z}^{(l)} = Z^0 \\
&Z^{(l+1)} = \mathcal{A} f_{\theta(l)}(Z^{(l)}), \quad l = 0, \cdots, L-1 \\
&\tilde{Z}^{(l+1)} = (Z^{(l+1)}, \tilde{Z}^{(l)})
\end{aligned}
$$

1. The network output is invariant to permutations of variables and constraints
2. The network can be applied to MIPs of different sizes using the same set of parameters

Evaluation

- **Primal gap:** $\gamma_p(t) = \begin{cases} 1 & p(t) \cdot p^\star < 0 \\ \frac{p(t) - p^\star}{max\{|p(t)|, |p^\star|, \epsilon\}} & otherwise \end{cases}$
- **Dual gap:** $\gamma_d(t) = \begin{cases} 1 & d(t) \cdot p^\star < 0 \\ \frac{p^\star - d(t)}{max\{|d(t)|, |p^\star|, \epsilon\}} & otherwise \end{cases}$
- **Primal-dual gap:** $\gamma_p d(t) = \begin{cases} 1 & d(t) \cdot p(t) < 0 \\ \frac{p(t) - d(t)}{max\{|d(t)|, |p(t)|, \epsilon\}} & otherwise \end{cases}$

where $\epsilon = 10^{-12}$, $p(t)$ is the primal bound at time t and $p^\star$ is the best known primal bound, $d(t)$ is the global dual bound at time t.

Energy function

$$M = (A, b, c) \quad E(x; M) = \begin{cases} c^T x & if\ x\ is\ feasible \\ \infty & otherwise \end{cases}$$

Conditional generative modelling

$$p(x|M) = \frac{exp(-E(x; M))}{Z(M)}$$
$$Z(M) = \sum_{x'} exp(-E(x'; M))$$

Feasible assignments with lower objective values have higher probability. Infeasible one has zero probability.

The inverse temperature parameter β multiplying c changes the distribution.

Learning

Learn to approximate $p(x|M)$ using a generative model $p_\theta(x|M)$

Loss function: $L(\theta) = -\sum_{i=1}^{N}\sum_{j=1}^{N} w_{ij}\, log p_\theta(x^{i,j}|M_i) \quad w_{ij} = \frac{exp(-x_i^T x^{i,j})}{\sum_{k=1}^{N_i} exp(-c_i^T x^{i,k})}$

Training sample: $X_i = \{x^{i,j}\}_{j=1}^{N_i},\ M_i \sim p(M)$

Conditionally-independent model

Generative model:

$$p_\theta(x|M) = \prod_{d \in \mathcal{I}} p_\theta(x_d|M)$$

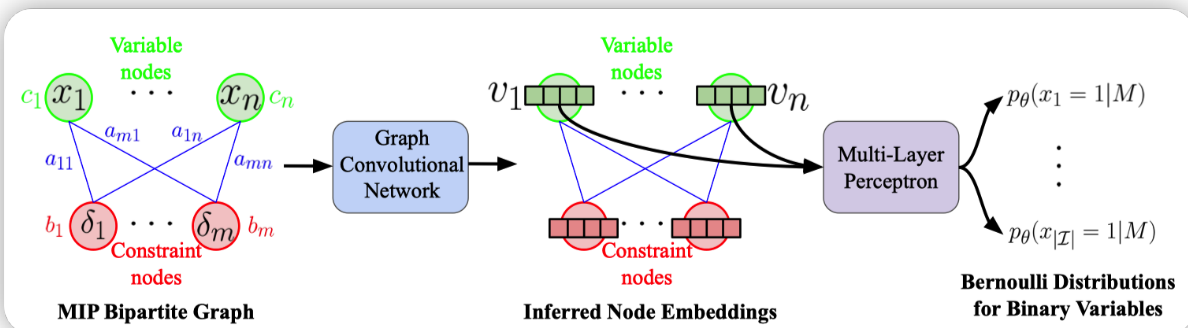
$\mathcal{I}$ is the set of dimensions of x corresponding to the integer variables. Let x_d denotes the d^{th} dimension of x.

Predicting a distribution for x^d is independent of the others conditioned on M

For simplicity, assuming $x^d \in \{0, 1\}$ is binary, the success probability μ_d for **Bernoulli distribution** $p_\theta(x^d|M)$ is

$$t_d = MLP(v_d; \theta)$$
$$\mu_d = p_\theta(x^d = 1|M) = \frac{1}{1 + exp(-t_d)}$$

where v_d is the embedding from GCN corresponding to x_d, **MLP** is the multi-layer perceptron for all variables



Handle general integers: change to a sequence of binary prediction tasks in a sequence of $\lceil log_2(card(z)) \rceil$ bits for a integer variable z (**binary tree search with maximum depth** n_b).

Most significant bit: the first bit in the sequence.

Additional binary classifier: decide which variables to predict a value for and optimize the ratio of predicted variables.

Loss function:

$$l_{selectiive}(\theta, x, M) = \frac{-\sum_{d\in\mathcal{I}} logp_\theta(x_d|M)\cdot y_d}{\sum_{d\in\mathcal{I}} y_d} + \lambda\Psi(C - \frac{1}{|\mathcal{I}|}\sum_{d\in\mathcal{I}} y_d),$$
$$L_{selective}(\theta) = \sum_{i,j} w_{ij}\cdot l_{selectiive}(\theta, x^{i,j}, M_i).$$

C is threshold of coverage ratio, Ψ is a quadratic penalty term with hyper-parameter λ.

Advantage: parallel computing, generate series of sub-MIPs, solving by solver.

Algorithm 1: Generating variable assignments and tightenings

Input: Learned distributions $p_\theta(x|M)$, $p_\theta(y|M)$, MIP instance M

Output: Variable assignment and bound tightenings

```
assignment := {}
tightenings := {}
for x_i ∈ Variables(M) do
    if x_i is binary variable then
        Sample p_x from p_θ(x_i|M)
        Sample y from p_θ(y_i|M)
        if y = 1 then
            Add (x_i := round(p_x)) to assignment
    if x_i is non-binary integer variable then
        lb := lower bound of x_i
        ub := upper bound of x_i
        b_0, ..., b_k := binary representation of (ub − lb), b_0 being most significant bit
        for j ∈ {0, ..., k} do
            Sample p_x from p_θ(x_{i,j}|M)
            Sample y from p_θ(y_{i,j}|M)
            if y = 1 then
                if round(p_x) = 1 then
                    lb := lb + ⌈(ub − lb)/2⌉
                else
                    ub := lb + ⌊(ub − lb)/2⌋
            else
                Add (lb ≤ x_i ≤ ub) to tightenings
                break
return assignment, tightenings
```

💡 **For non-binary variable, implement the binary tree search through nodes**
p_x**: up or down**
y**: whether or not (once not, stop tightening)**

Neural branching

> ***This component is mainly used to bound the gap between the objective value of the best assignment and an optimal one.***

A form of *branch-and-bound* progressively tightening the bound and helps find feasible assignments. Training **a deep neural network policy** to imitate choices made by Full Strong Branching (FSB).

Large Neighborhood Search

Focusing: Which variable to branch on impacting the number of steps

Imitating expert policy —— FSB

- Maintain approximately the same decision quality but substantially reduce the time.
- It simulates one branching step for all candidate variables and picks the one that provides the largest improvement in the **dual bound.**
- Denote $x^\star$ as the solution to LP relaxation at current node.
- Require to solve $2 \times n_{cands}$ program, n_{cands} is the number of possible candidates

For each candidate i, FSB solves two LPs:

$$\begin{aligned} &minimize && c^T x^{up} \\ &subject\ to && Ax^{up} \leq b \\ & && l^{(i)} \leq x^{up} \leq u \end{aligned} \qquad\qquad \begin{aligned} &minimize && c^T x^{down} \\ &subject\ to && Ax^{down} \leq b \\ & && l \leq x^{down} \leq u^{(i)} \end{aligned}$$

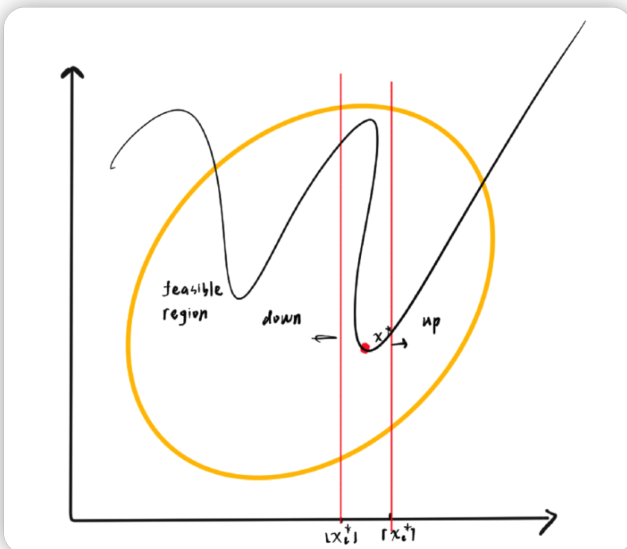
where $x^{up}, x^{down} \in \mathbb{R}^n, l^{(i)} = \lceil x_i^\star \rceil, u^{(i)} = \lfloor x_i^\star \rfloor$, other entries of l and u are unchanged.

Decide the branching variable by optimal values of above two LPs.

The Splitting Conic Solver based on ADMM can simultaneously handle many LPs in a batch.

Expert score

$$s_i = (OPT_i^{up} - OPT + \epsilon)(OPT_i^{down} - OPT + \epsilon)$$



OPT^{up} is the optimal value of $c^T x^{up}$, OPT^{down} is the optimal value of $c^T x^{down}$, OPT is the optimal objective value of the LP at current node.

Given a set $\mathcal{C}$ of candidates, converting scores into categorical distribution,

$$p_i^{expert} = \frac{s_i}{\sum_{c \in \mathcal{C}} s_c}$$

Neural network architecture

Same as the architecture in Neural diving,

Denote v_c be the embedding computed by a GCN for candidate x_c, and ϕ be the learnable parameter of the policy. The probability $p_\phi(x_c|M)$ of selecting x_c for branching is given by

$$t_c = MLP(v_c; \phi)$$
$$p_\phi(x_c|M) = \frac{exp(-t_c)}{\sum_{c' \in \mathcal{C}} exp(t_{c'})}$$

Loss function: $L(\phi) = \sum_{c \in \mathcal{C}} p_c^{expert} log p_\phi(x_c|M)$