Report on MINLP Paper

Siqi Yao

SDS

2025.1.2

香港中文大學（深圳）
The Chinese University of Hong Kong, Shenzhen

1. Problem Introduction: MINLP
2. L2O Formulation
3. Differentiating Through Discrete Operations
4. Algorithm Overview

Concept

- Mixed-Integer Non-Linear Programming

Concept

- Mixed-Integer Non-Linear Programming
- Optimization problem that contain both integer and non-integer variables, and non-linear (possibly non-convex, which makes the problem extremely hard!) objective functions or constraints.

Concept

- Mixed-Integer Non-Linear Programming
- Optimization problem that contain both integer and non-integer variables, and non-linear (possibly non-convex, which makes the problem extremely hard!) objective functions or constraints.
- Non-integer variables allow discrete decisions (e.g., number of items) and non-linearity models complex relationships.

Example: Gas Transmission (De Wolf and Smeers, 2000)

Problem Description: Supply gas to all demand points (all red points below) in Belgium while minimizing cost. Gas is pumped through the network from a series of supply points (not shown), and there are non-linear constraints on the pressure of the gas within the pipe.

Example: Gas Transmission (De Wolf and Smeers, 2000)

Problem Input:

- **Network** (N, A)
- $N_s \subseteq N$: Set of supply nodes.
- $c_i,\ i \in N_s$: Purchase cost of gas.
- $s_i,\ i \in N$: Supply at node i:
  - $s_i > 0 \implies$ node i is supply node.
  - $s_i < 0 \implies$ node i is demand node (consumes gas).
- $f_{ij},\ (i,j) \in A$: Flow along arc (i,j):
  - $f(i,j) > 0 \implies$ gas flows $i \to j$.
  - $f(i,j) < 0 \implies$ gas flows $j \to i$.
- $\underline{s}_i, \overline{s}_i$: Lower and upper bounds on gas "supply" at node i.
- $\underline{p}_i, \overline{p}_i$: Lower and upper bounds on gas pressure at node i.

Example: Gas Transmission (De Wolf and Smeers, 2000)

Objective:

$$\min \sum_{j \in N_s} c_j s_j$$

Subject to:

$$\sum_{j|(i,j)\in A} f_{ij} = s_i, \quad \forall i \in N$$

$$\mathrm{sign}(f_{ij}) f_{ij}^2 - \Psi_{ij}(p_i^2 - p_j^2) = 0, \quad \forall (i,j) \in A_p$$

$$\mathrm{sign}(f_{ij}) f_{ij}^2 - \Psi_{ij}(p_i^2 - p_j^2) \geq 0, \quad \forall (i,j) \in A_a$$

$$s_i \in [\underline{s}_i, \overline{s}_i], \quad \forall i \in N$$

$$p_i \in [\underline{p}_i, \overline{p}_i], \quad \forall i \in N$$

$$f_{ij} \geq 0, \quad \forall (i,j) \in A_a$$

Why is it Hard

- **Mixed Integer and Non-Integer Variables:**

Why is it Hard

- **Mixed Integer and Non-Integer Variables:**
  - Incompatible methods: Integer variables require discrete combinatorial techniques (e.g., branch and bound), while non-integer variables often rely on continuous gradient-based methods.

Why is it Hard

- **Mixed Integer and Non-Integer Variables:**
  - Incompatible methods: Integer variables require discrete combinatorial techniques (e.g., branch and bound), while non-integer variables often rely on continuous gradient-based methods.
  - Rounding issues: Approximating integer solutions from continuous relaxations can lead to suboptimal or infeasible results.

Why is it Hard

- **Mixed Integer and Non-Integer Variables:**
  - Incompatible methods: Integer variables require discrete combinatorial techniques (e.g., branch and bound), while non-integer variables often rely on continuous gradient-based methods.
  - Rounding issues: Approximating integer solutions from continuous relaxations can lead to suboptimal or infeasible results.
  - Large solution space due to combinatorial nature.

Why is it Hard

- **Mixed Integer and Non-Integer Variables:**
  - Incompatible methods: Integer variables require discrete combinatorial techniques (e.g., branch and bound), while non-integer variables often rely on continuous gradient-based methods.
  - Rounding issues: Approximating integer solutions from continuous relaxations can lead to suboptimal or infeasible results.
  - Large solution space due to combinatorial nature.
- **Non-Linear and Non-Convex Constraints/Objective:**

Why is it Hard

- **Mixed Integer and Non-Integer Variables:**
  - Incompatible methods: Integer variables require discrete combinatorial techniques (e.g., branch and bound), while non-integer variables often rely on continuous gradient-based methods.
  - Rounding issues: Approximating integer solutions from continuous relaxations can lead to suboptimal or infeasible results.
  - Large solution space due to combinatorial nature.
- **Non-Linear and Non-Convex Constraints/Objective:**
  - Local minima makes global optimization challenging, and applicability of efficient convex optimization algorithms are limited.

Current Solutions

- Various Branch and Bound methods.
- Outer Approximation.
- Various decomposition methods such as Benders Decomposition.
- Have trouble dealing large scale problems!

Problem Formulation

Generic Learning-to-Optimize Formulation for MINLP

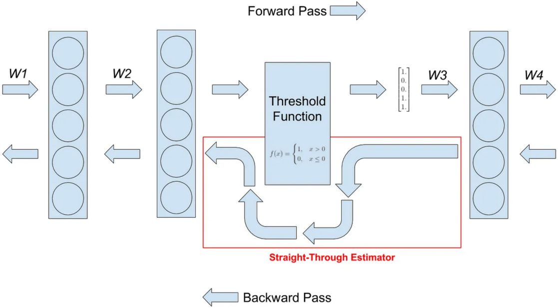
$$
\begin{aligned}
\min_{\Theta} \quad & \mathbb{E}_{\xi \sim \mathcal{P}_\xi}\, f(\mathbf{x}^\xi, \xi) \\
\text{s.t.} \quad & g(\mathbf{x}^\xi, \xi) \le 0, \quad \forall \xi \in \mathcal{P}_\xi \\
& \mathbf{x}^\xi \in \mathbb{R}^{n_r} \times \mathbb{Z}^{n_z}, \quad \forall \xi \in \mathcal{P}_\xi \\
& \mathbf{x}^\xi = \psi_\Theta(\xi), \quad \forall \xi \in \mathcal{P}_\xi
\end{aligned} \tag{1}
$$

Problem Formulation

Regularized Loss Function

$$\mathcal{L}(\Theta) = \frac{1}{m}\sum_{i=1}^{m}\left(f(\mathbf{x}^i, \xi^i) + \lambda \cdot \left\|\left(g(\mathbf{x}^i, \xi^i)\right)_+\right\|_2^2\right) \text{ with } \mathbf{x}^i = \psi_\Theta(\xi^i)\ \forall i \tag{2}$$

Straight-through Estimator



Straight-through Estimator

- Enables back propagation through discrete operations.
- Forward pass: Simply apply the non-differentiable discrete operation such as rounding, binarizing, or using indicator function $\mathbb{I}(\cdot)$.
- Backward pass: Replaces the non-existent gradient of these discrete functions with soft approximations.
    - For rounding operations: Use the gradient of the identity function.
    - For binarization or indicator functions: Use the gradient of the Sigmoid function.
- Lacks stochasticity.

Gumbel-Sigmoid Noise

A Gumbel-Sigmoid Trick

Works by adding noise g to the logits h, where g is sampled from Gumbel(0, 1) distribution using inverse transform sampling:

$$g = -\log(-\log(U)), \quad U \sim \text{Uniform}(0, 1).$$

The Gumbel-Sigmoid function produces a soft approximation with random perturbation:

$$\text{Gumbel-Sigmoid}(h) = \frac{1}{1 + \exp\left(-\frac{h + g_1 - g_2}{\tau}\right)}$$

where $g_1, g_2 \overset{\text{iid}}{\sim} \text{Gumbel}(0, 1)$, and the temperature parameter τ controls the smoothness. Set $\tau = 1$ for simplicity.

Gumbel-Sigmoid Further Explained

- Gumbel-Max:

$$z = \text{one\_hot}\left(\arg\max_i \left[g_i + \log \pi_i\right]\right)$$

- Gumbel-Softmax:

$$y_i = \frac{\exp\left((\log(\pi_i) + g_i)/\tau\right)}{\sum_{j=1}^{k} \exp\left((\log(\pi_j) + g_j)/\tau\right)} \quad \text{for } i = 1, \ldots, k.$$

- Gumbel-Sigmoid:

$$\sigma(h) = \frac{\exp\left((h + g_1)/\tau\right)}{\exp\left((h + g_1)/\tau\right) + \exp\left(g_2/\tau\right)}$$

$$= \frac{1}{1 + \exp\left(-(h + g_1 - g_2)/\tau\right)}$$

Gumbel-Sigmoid Further Explained

- Why noise at all?
  - **Introducing randomness:** Noise adds randomness during training, acting similarly to simulated annealing. If the Gumbel noise is removed, naively rounding up/down is very likely to lead to a local optimum since the problem is highly non-convex.
  - **Improving robustness of the network:** The noise prevents the network from overfitting or collapsing into deterministic behaviors too early, leading to a more robust training process.
- After computing Gumbel-Sigmoid, apply hard binarization (with threshold=0.5, similar to regular logistic regression). Use gradient of Sigmoid during backward pass.
- Only used during training!

Algorithm Overview

Algorithm 1 Learning-to-optimize MINLPs with Correction Layers: Forward Pass.

Require: Training instance $\boldsymbol{\xi}^i$, neural networks $\pi_{\Theta_1}(\cdot)$ and $\delta_{\Theta_2}(\cdot)$

Predict a continuously relaxed solution $\bar{\mathbf{x}}^i \leftarrow \pi_{\Theta_1}(\boldsymbol{\xi}^i)$
Obtain an initial correction prediction $\mathbf{h}^i \leftarrow \delta_{\Theta_2}(\bar{\mathbf{x}}^i, \boldsymbol{\xi}^i)$
Update continuous variables: $\hat{\mathbf{x}}_r^i \leftarrow \bar{\mathbf{x}}_r^i + \mathbf{h}_r^i$
Round integer variables down: $\hat{\mathbf{x}}_z^i \leftarrow \lfloor \bar{\mathbf{x}}_z^i \rfloor$
if using *Rounding Classification* **then**
    Compute $\mathbf{b}^i$ as the rounding direction using Gumbel-Sigmoid$(\mathbf{h}_z^i)$
else if using *Learnable Threshold* **then**
    Compute $\mathbf{v}^i \in [0,1]^{n_z} \leftarrow$ Sigmoid$(\mathbf{h}_z^i)$
    Compute rounding direction: $\mathbf{b}^i \leftarrow \mathbb{I}\big((\bar{\mathbf{x}}_z^i - \hat{\mathbf{x}}_z^i) - \mathbf{v}^i > 0\big)$
end if
Update integer variables: $\hat{\mathbf{x}}_z^i \leftarrow \hat{\mathbf{x}}_z^i + \mathbf{b}^i$
return $\hat{\mathbf{x}}^i$

Differentiable Correction Layers

- Rounding Classification (Line 6):

$$b^i = \text{Gumbel-Sigmoid}(h_z^i)$$

  - Gumbel-Sigmoid works as a stochastic soft-rounding.
  - Entries of b^i decides the rounding directions.
  - In the backward pass, STE is used in line 4 for the rounding down operation. In line 6, the derivative of the Sigmoid function is used.

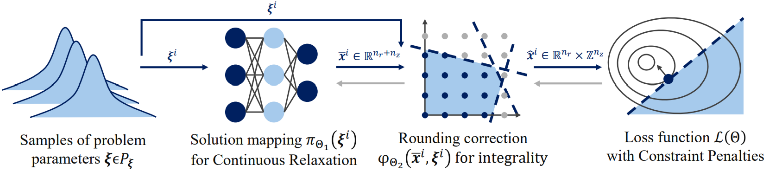
Differentiable Correction Layers

- Learnable Threshold (Line 8, 9):

$$\mathbf{v}^i \in [0,1]^{n_z} \leftarrow \text{Sigmoid}(h_z^i)$$
$$\mathbf{b}^i \leftarrow \mathbb{I}((\bar{\mathbf{x}}_z^i - \hat{\mathbf{x}}_z^i) - \mathbf{v}^i > 0)$$

  - Learn a vector of per-variable rounding thresholds $\mathbf{v}^i \in [0,1]^{n_z}$.
  - A variable is rounded up if the fractional part of its relaxed value, $(\bar{\mathbf{x}}_z^i - \hat{\mathbf{x}}_z^i)$, exceeds the threshold.
  - In the backward pass, the gradient of the indicator function $\mathbb{I}(\cdot)$ is approximated by that of the Sigmoid function.

Algorithm Overview



Thanks!

References

- Askary, H. (2021). Intuitive Explanation of Straight-Through Estimators with PyTorch Implementation. *Medium*. [Link]
- De Wolf, D., & Smeers, Y. (2000). The gas transmission problem solved by an extension of the simplex algorithm. *Management Science*, 46, 1454–1465.
- Jang, E., Gu, S., & Poole, B. (2017). Categorical reparameterization with gumbel-softmax. In *International Conference on Learning Representations (ICLR)*.
- Tsai, Y.-H., Liu, M.-Y., Sun, D., Yang, M.-H., & Kautz, J. (2018). Learning binary residual representations for domain-specific video streaming. *Proceedings of the AAAI Conference on Artificial Intelligence*, 32(1). `https://doi.org/10.1609/aaai.v32i1.12259`