Learning Cut Selection for DOGE-Train: Discrete Optimization on GPU with End-to-End Training

Published in AAAI 2024

Abbas, Ahmed and Swoboda, Paul

Lecture of Mathieu Lacroix

2024

Outline

Outline

A 3-papers work

- **Paper 1**: *Lange, Jan-Hendrik and Swoboda, Paul*, Efficient Message Passing for 0–1 ILPs with Binary Decision Diagrams, ICML 2021.
- **Paper 2**: *Abbas, Ahmed and Swoboda, Paul*, FastDOG: Fast Discrete Optimization on GPU, CVPR 2022.
- **Paper 3**: *Abbas, Ahmed and Swoboda, Paul*, Learning Cut Selection for DOGE-Train: Discrete Optimization on GPU with End-to-End Training, AAAI 2024.

Overview

Objective

Provide efficient and accurate heuristic for 0/1 general MIP

- Based on Lagrangian decomposition solved by block coordinate ascent and followed by a heuristic (paper 1)
- GPU Friendly (paper 2)
- ML used to update Lagrangian multipliers at each iteration (paper 3).

Lagrangian decomposition

Binary linear problem

$$\min c^\top x \quad (1)$$
$$a_j^\top x \leq b_j \qquad \forall j \in [m] \quad (2)$$
$$x_i \in \{0, 1\} \qquad \forall i \in [n] \quad (3)$$

- $\mathcal{I}_j$: Set of variables appearing in constraint number j
- $\mathcal{X}_j$: Set of binary assignment of variables of $\mathcal{I}_j$ satisfying constraint number j, that is, $\mathcal{X}_j = \{x \in \{0, 1\}^{\mathcal{I}_j} \mid \sum_{i \in \mathcal{I}_j} a_{ji} x_i \leq b_j\}$ (knapsack solutions)
- $\mathcal{J}_i$: Set of constraints (indexes) where variable x_i appears

Lagrangian decomposition

Idea: consider $\mathcal{J}_i$ copies of each variable x_i.

$$\min c^\top x^1 \tag{4}$$

$$\sum_{i \in \mathcal{I}_j} a_{ji} x_i^j \leq b_j \qquad \forall j \in [m] \tag{5}$$

$$x_i^j = x_i^1 \qquad \forall i \in [n], \forall j \in \mathcal{J}_i \setminus \{1\} \tag{6}$$

$$x_i^j \in \{0,1\} \qquad \forall i \in [n], \forall j \in \mathcal{J}_i \tag{7}$$

Lagrangian decomposition

Dualizing (6) and setting $\lambda_i^1 = c_i - \sum_{j \in \mathcal{J}_i}$ gives:

$LR(\lambda)$

$$\min \sum_{j \in [m]} \sum_{i \in \mathcal{I}_j} \lambda_i^j x_i^j \qquad (8)$$

$$\sum_{i \in \mathcal{I}_j} a_{ji} x_i^j \leq b_j \qquad \forall j \in [m] \qquad (9)$$

$$x_i^j \in \{0, 1\} \qquad \forall i \in [n], \forall j \in \mathcal{J}_i \qquad (10)$$

Remark : λ must satisfy $\sum_{j \in \mathcal{J}_i} \lambda_i^j = c_i$ for all $i \in [n]$.

$LR(\lambda)$ is decomposable into 1 subproblem per constraint:

$$LR(\lambda) = \sum_{j \in [n]} E(\lambda^j)$$

with $E(\lambda^j) = \max_{x \in \mathcal{X}_j} \sum_{i \in \mathcal{I}_j} \lambda_i^j x_i^j$

Lagrangian decomposition

LD

$$\max LR(\lambda) \tag{11}$$

$$\sum_{j \in \mathcal{J}_i} \lambda_i^j = c_i \qquad \forall i \in [n] \tag{12}$$

Block Coordinate Ascent Method

Idea: Update one Lagrangian vector $\lambda_i \in \mathbb{R}^{\mathcal{J}_i}$ at each time.

Min marginal averaging

For $i \in [n]$ and $j \in \mathcal{J}_i$ and $\beta \in \{0, 1\}$, let

$$m_{ij}^{\beta} = E(\lambda^j) \text{ with } x_i = \beta$$

m_{ij}^{β} is the value of the best solution of subproblem j when x_i is equal to β.

Lagrangian update

$$\lambda_i^j = \lambda_i^j + (m_{ij}^1 - m_{ij}^0) - \frac{1}{\mathcal{J}_i} \sum_{k \in J_i} (m_{ij}^1 - m_{ij}^0) \quad \forall j \in \mathcal{J}_i$$

New λ_i satisfies (12) and gives a non worse Lagrangian bound.
Remark: Needs to compute m_{ij}^{β}!

Block Coordinate Ascent Method

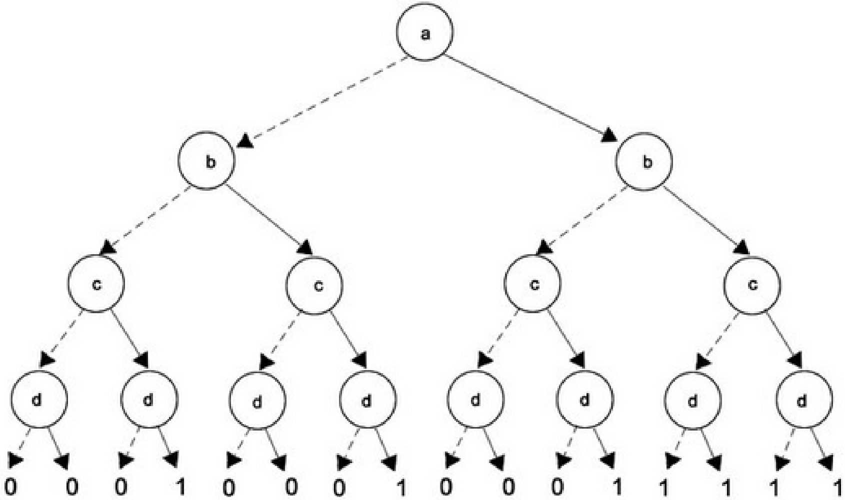
Algorithm 1: Min-marginal averaging

```
input objective vector c ∈ ℝ^n, constraint sets
  𝒳_j ⊂ {0,1}^{ℐ_j} for j ∈ [m]
Find variable ordering {i_1, ..., i_n} = [n].
Initialize dual variables λ_i^j = c_i / |𝒥_i| for all i ∈ [n]
  and j ∈ 𝒥_i.
while (stopping criterion not met) do
    Perform forward pass:
    for i = i_1, ..., i_n do
        for j ∈ 𝒥_i do
            Compute min-marginals for β ∈ {0,1}:
            m_ij^β = min_{x ∈ 𝒳_j} x^⊤ λ^j  s. t.  x_i = β
        for j ∈ 𝒥_i do
            Update dual variable:
              λ_i^j ← λ_i^j − (m_ij^1 − m_ij^0) +
              1/|𝒥_i| Σ_{k ∈ 𝒥_i} m_ik^1 − m_ik^0 .
    Perform backward pass analogously (set
      variable order to {i_n, ..., i_1})
```

Remark: It is a heuristic (may get stuck in suboptimal points)!

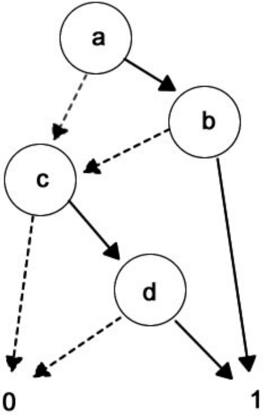
BDD

Start from binary tree



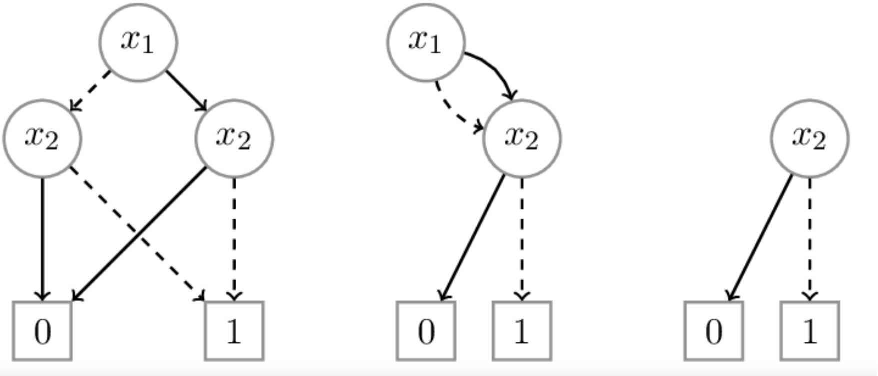
BDD

"Shrink" some parts



BDD

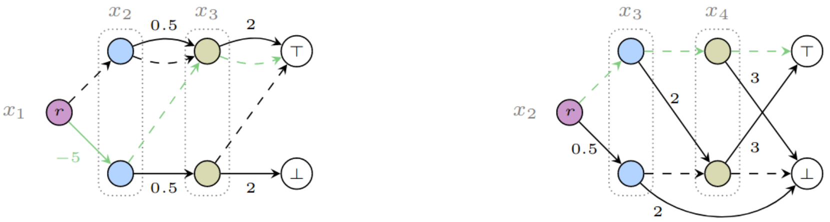
"Shrink" some parts



Remark: there may exist more than 2 nodes per variable.

BDD to compute m_{ij}^{β}

$$\min_{x \in \{0,1\}} -5x_1 + x_2 + 4x_3 + 3x_4$$
$$\mathcal{X}_1 : x_1 + x_2 + x_3 \leq 2, \qquad \mathcal{X}_2 : x_2 + x_3 - x_4 = 0$$



$\lambda^1 = (-5, 0.5, 2), (x_1, x_2, x_3) \in \mathcal{X}_1$ $\qquad$ $\lambda^2 = (0.5, 2, 3), (x_2, x_3, x_4) \in \mathcal{X}_2$

Figure 2. Example decomposition of a binary program into two subproblems, one for each constraint. Each subproblem is represented by a weighted BDD where solid arcs model the cost λ of assigning a 1 to the variable and dashed arcs have 0 cost which model assigning a 0. All $r - \top$ paths in BDDs encode feasible variable assignments of corresponding subproblems (and $r - \bot$ infeasible). Optimal assignments w.r.t current (non-optimal) λ are highlighted in green i.e. $x_1 = 1, x_2 = x_3 = 0$ for $\mathcal{X}_1$ and $x_2 = x_3 = x_4 = 0$ for $\mathcal{X}_2$. Our dual update scheme processes multiple variables in parallel which are indicated in same color (e.g. x_1, x_2 in $\mathcal{X}_1, \mathcal{X}_2$ resp.).

- Compute shortest paths from r and $\top$
- Select minimum path leaving node i with arc β.

Primal heuristic

For $i \in [n]$, let $M_i = \sum_{j \in \mathcal{J}_i} (m^1_{ij} - m^0_{ij})$. If $M_i > 0$, then it is preferable to fix x_i to 0, and to 0 otherwise.
Depth-first search considering variables following decreasing order $|M_i|$ (considering best fixation variable) until a solution is found.

Experiments

- Run Block Coordinate Ascent Method and then primal heuristic.
- Compare with Gurobi (TL 1h for both)

Result comments

- Provide weaker lower bounds than Gurobi (root LP relaxation (cutting?)) but faster
- Solutions of the primal heuristic are only slightly worse than those provided by Gurobi (sometimes better) and faster.
- There is a parallel version with a speed up to 6 when having 16 threads.

Outline

Parallelizing Block Coordinate Ascent Mehtod

The problem

- Computing m_{ij}^{β} can be done in parallel
- Updating λ needs synchronisation since m_{ij}^{β} are needed!

The solution

Update λ with values m_{ij}^{β} (denoted $\overline{m}_{ij}^{\beta}$) computed at previous iteration!

$$\lambda_i^j = \lambda_i^j + \omega(m_{ij}^1 - m_{ij}^0) - \frac{\omega}{|\mathcal{J}_i|} \sum_{k \in J_i} (\overline{m}_{ik}^1 - \overline{m}_{ik}^0) \quad \forall j \in \mathcal{J}_i$$

Remark: ω dumping factor between 0 and 1 (fixed $\frac{1}{2}$) following (Werner, Průša, and Dlask, 2020).

Parallelizing primal heuristic

Algorithm 2: Perturbation Primal Rounding

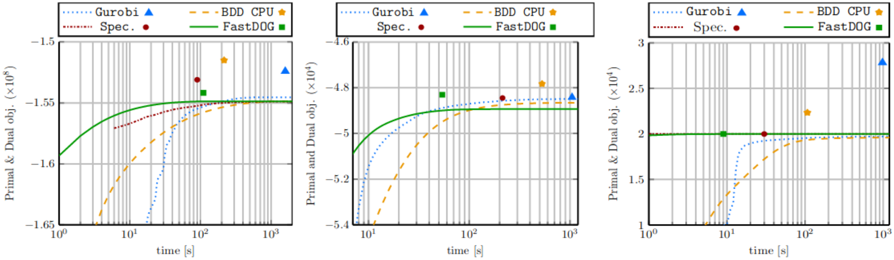
Input: Lagrange variables $\lambda_i^j \in \mathbb{R} \, \forall i \in [n], j \in \mathcal{J}_i$,
Constraint sets $\mathcal{X}_j \subset \{0,1\}^{\mathcal{I}_j} \, \forall j \in [m]$,
Initial perturbation strength $\delta \in \mathbb{R}_+$,
perturbation growth rate α

Output: Feasible labeling $x \in \{0,1\}^n$

Compute min-marginal differences $M_{ij} \, \forall i, j$ (MD)
while $\exists i \in [n]$ *and* $j \neq k \in \mathcal{J}_i$ *s.t.* $\text{sign}(M_{ij}) \neq \text{sign}(M_{ik})$ **do**
    for $i = 1, \dots, n$ *in parallel* **do**
        Sample r uniformly from $[-\delta, \delta]$
        if $M_{ij} > 0 \, \forall j \in \mathcal{J}_i$ **then**
            $\lambda_i^j \mathrel{+}= \delta \quad \forall j \in \mathcal{J}_i$
        else if $M_{ij} < 0 \, \forall j \in \mathcal{J}_i$ **then**
            $\lambda_i^j \mathrel{-}= \delta \quad \forall j \in \mathcal{J}_i$
        else if $M_{ij} = 0 \, \forall j \in \mathcal{J}_i$ **then**
            $\lambda_i^j \mathrel{+}= r \cdot \delta \quad \forall j \in \mathcal{J}_i$
        else
            Compute total min-marginal difference: $M_i = \sum_{j \in \mathcal{J}_i} M_{ij}$
            $\lambda_i^j \mathrel{+}= \text{sign}(M_i) \cdot |r| \cdot \delta \quad \forall j \in \mathcal{J}_i$
    Increase perturbation: $\delta \leftarrow \delta \cdot \alpha$
    Reoptimize perturbed λ via Algorithm 1
    Recompute $M_{ij} \, \forall i, j$ w.r.t optimized λ

while there exists a var with ≠ values in subproblems

Experiments



(a) *Cell tracking: Large* (b) *Graph Matching: Worms* (c) *MRF: Color-seg-n8*

Figure 5. Convergence plots averaged over all instances of a dataset. Lower curves depict increasing lower bounds while markers denote objectives of rounded primal solutions. The x-axis is plotted logarithmically.

- one order of magnitude faster than previous algo
- needs 3 times more iterations for Block Coordinate Ascent Method than previous algo

Outline

More parameters when updating λ

Rule 1: used at each step

Let $\omega_{ij} \in [0,1]$ and $\alpha_{ij} \geq 0$ with $\sum_{j \in \mathcal{J}_i} \alpha_{ij} = 1$.

$$\lambda_i^j = \lambda_i^j - \omega_{ij}(m_{ij}^1 - m_{ij}^0) + \sum_{k \in J_i} \alpha_{ij}(\overline{m}_{ik}^1 - \overline{m}_{ik}^0) \quad \forall j \in \mathcal{J}_i$$

Rule 2: used for initialization

Let $\theta \in \mathbb{R}^\lambda$.

$$\lambda_i^j = \lambda_i^j + \theta_{ij} - \frac{1}{|\mathcal{J}_i|} \sum_{k \in J_i} \theta_{ik}$$

Learn θ, α and ω!

ML pipeline

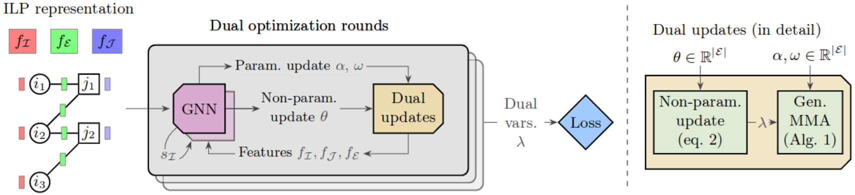


Figure 1: Our method for optimizing the Lagrangean dual (D). The dual problem is encoded on a bipartite graph containing features $f_{\mathcal{I}}$, $f_{\mathcal{J}}$ and $f_{\mathcal{E}}$ for primal variables, subproblems and dual variables resp. A graph neural network (GNN) predicts θ, α, ω for dual updates. In one dual update block (right), current set of Lagrange multipliers λ are first updated by the non-parametric update using θ. Afterwards parametric update is done via Alg. 1 using α, ω. The updated solver features f and LSTM cell states $s_{\mathcal{I}}$ are sent to the GNN in next optimization round. See Sec. 3.6 for further details.

- Loss $= LR(\lambda)$ ($\frac{\partial L}{\partial \lambda}$ is easy, it is x^* so can be done on GPU)
- It is possible to backpropagate!

GNN

- Graph convolution with attention mechanism (transformer based graph convolution scheme (Shi et al., 2021))
- LSTM with a state used in subsequent optimization rounds

Algorithm 3: Parameter prediction by GNN

Input: Primal variable features $f_{\mathcal{I}}$ and cell states $s_{\mathcal{I}}$, Subproblem features $f_{\mathcal{J}}$, Dual variable (edge) features $f_{\mathcal{E}}$, Set of edges $\mathcal{E}$.

$h_{\mathcal{J}} = \texttt{ReLU}\left(\texttt{LN}\left(\texttt{CONV}_{\mathcal{J}}\left(f_{\mathcal{I}}, f_{\mathcal{J}}, f_{\mathcal{E}}, \mathcal{E}\right)\right)\right)$ // Compute subproblems embeddings

$h_{\mathcal{I}} = \texttt{ReLU}\left(\texttt{LN}\left(\texttt{CONV}_{\mathcal{I}}\left(f_{\mathcal{I}}, [f_{\mathcal{J}}, h_{\mathcal{J}}], f_{\mathcal{E}}, \mathcal{E}\right)\right)\right)$ // Compute primal variable embeddings

$z_{\mathcal{I}}, s_{\mathcal{I}} = \texttt{LSTM}_{\mathcal{I}}(h_{\mathcal{I}}, s_{\mathcal{I}})$ // Compute output and cell state

$(\hat{\alpha}, \hat{\omega}, \theta) = \Phi\left([f_{\mathcal{I}}, h_{\mathcal{I}}, z_{\mathcal{I}}], [f_{\mathcal{J}}, h_{\mathcal{J}}], f_{\mathcal{E}}, \mathcal{E}\right)$ // Prediction per edge

$\alpha_{i\bullet} = \texttt{Softmax}(\hat{\alpha}_{i\bullet}), \forall i \in \mathcal{I}, \ \omega = \texttt{Sigmoid}(\hat{\omega})$ // Ensure non-decreasing obj., Prop. 1

return $\alpha, \omega, \theta, s_{\mathcal{I}}$

Learning pipeline

1 dual optimization round

1. GNN to predict α, ω, θ
2. Update λ with θ (rule 2)
3. Apply T iterations of block Coordinate Ascent Method with predicted α, ω (rule 1)

Training

- Perform at most R dual optimization rounds
- For each mini-batch
  - randomly select $r \in [R]$
  - run $r - 1$ without gradient tracking
  - backpropagate through the last round (3 last ones with LSTM) by computing the loss
  - Use two neural networks: one for first rounds ($< \frac{R}{2}$), the other for the last ones.

Learning pipeline

Inference

- 2nd neural network when improvement becomes too small ($< 1e^{-6}$) with the first one
- For efficiency, use GNN only every T iterations of block Coordinate Ascent Method

Results

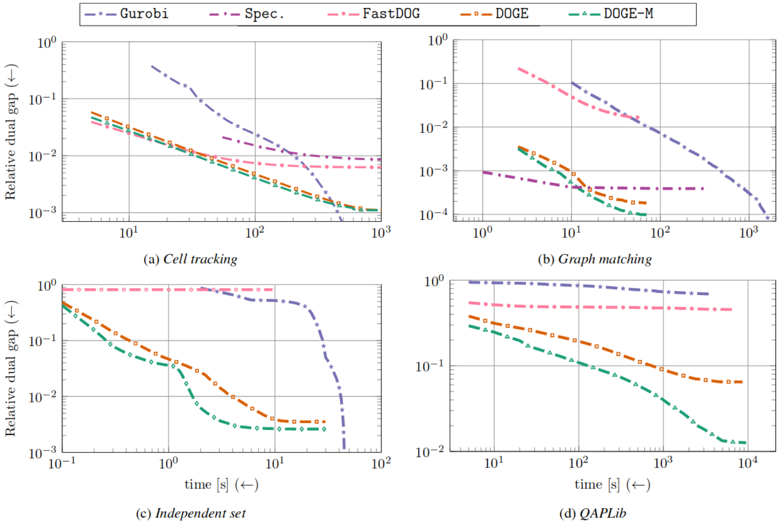


Figure 2: Convergence plots for $g(t)$ the relative dual gap to the optimum (or maximum suboptimal objective among all methods) of the relaxation (D). X-axis indicates wall clock time and both axes are logarithmic. The value of $g(t)$ is averaged over all test instances in each dataset.

Results

- One order of magnitude more accurate relaxation solutions wrt no learning
- On some datasets, no improvement by learning