Learning to Be Lazy

Laio Seman
Eduardo Camponogara

June 2024

This dissertation introduces a novel "Learning to Be Lazy" methodology, aimed at improving the efficiency of Mixed-Integer Linear Programming (MILP) solvers. The approach involves predicting lazy constraints, which are not immediately necessary for solving the initial relaxation of an MILP problem but may become critical at later stages. By identifying such constraints in advance, computational resources can be saved, and the solving process can be expedited. The proposed model leverages Graph Neural Networks (GNNs) and multitask learning to classify constraints and determine confidence levels for their laziness. The findings will be implemented in modern programming languages and tested on real-world optimization problems, with performance comparisons against state-of-the-art solvers.

Keywords: MILP, Graph Neural Networks, Lazy Constraints, Multitask Learning.

1 Introduction

Mixed-Integer Linear Programming (MILP) is a powerful optimization tool used across various domains, including logistics, finance, and engineering. However, solving MILP problems efficiently remains a significant challenge due to the combinatorial explosion of feasible solutions. Lazy constraints, which are constraints deferred during the initial relaxation but may become critical later, offer a potential solution to improve solver efficiency. The proposed SatGNN framework aims to predict these lazy constraints and their confidence levels, leveraging the power of Graph Neural Networks (GNNs) and multitask learning.

The "Learning to Be Lazy" methodology, which aims not only to predict whether a particular constraint is "lazy" or not but also to determine a confidence level for this classification. A lazy constraint is one that is not immediately necessary for solving the initial relaxation of a MILP problem but may become critical at later stages or in specific branches of the solution tree. Identifying such constraints in advance can save computational resources and speed up the solving process.

Our model employs multitask learning within the same GNN framework to perform this classification and to generate the associated confidence interval:

$$\text{lazyGNN} : \mathbb{R}^4 \longrightarrow [0,1] \times \{0,1\}, \tag{1}$$

$$f_{v_{\text{con}}} \longmapsto (\text{Confidence}_{\text{laziness}}, \text{Classification}_{\text{laziness}}) = \text{lazyGNN}(f_{v_{\text{con}}}) \tag{2}$$

Here, $\text{Confidence}_{\text{laziness}}$ serves as a slack variable, providing a more nuanced decision-making process. The $\text{Classification}_{\text{laziness}}$ is a binary indicator that marks a constraint as lazy (1) or not (0), based on whether the confidence level crosses a pre-defined threshold.

The multitask training phase of lazyGNN optimizes for both the classification of constraints into lazy or non-lazy categories and the confidence level associated with each prediction (by means of the predicted slackness). The objective function includes terms for the binary cross-entropy loss for the laziness classification and a suitable loss function for the confidence level.

This methodology offers several advantages. Firstly, it allows the optimization solver to focus on more relevant constraints, thereby speeding up the optimization process. Secondly, the use of a slack variable, or confidence level, allows for a more nuanced decision-making process, enabling the model to be more adaptable to different problem instances.

2 Objectives

This dissertation aims to advance the state-of-the-art in MILP solvers by incorporating a "Learning to Be Lazy" methodology within a GNN framework. The specific objectives are:

- Develop a GNN model that predicts lazy constraints and their confidence levels for MILP problems.
- Implement the lazyGNN model in a modern programming language, such as Python with PyTorch or TensorFlow.
- Apply the model to representative MILP problems across various domains, including logistics and engineering.
- Compare the performance of the lazyGNN-enhanced solver against state-of-the-art MILP solvers.

3 Methodology

Graph Neural Networks are an advanced form of neural network architectures specifically designed to handle graph-structured data. The motivation behind GNNs is to generalize the concepts from Convolutional Neural Networks (CNNs), which have been exceptionally successful with image and grid-like data, to data structures that are inherently non-grid-like, such as graphs. Therefore, the primary input to a GNN consists of a graph, coupled with features associated with its nodes. This input graph undergoes an iterative process of feature update, which is conceptually similar to how a convolution operation works in CNNs. This similarity has led to the process being termed as graph "convolutions."

Specifically, a GNN layer can be broken down into two main sequential operations:

- **Message Computation:** In this phase, for every node u in the graph, a "message" is computed. This message, denoted as $\boldsymbol{m}_u^{(l)}$, is a result of applying the function $M_l(\cdot)$ on the node's features from the previous layer:

$$\boldsymbol{m}_u^{(l)} = M_l(\boldsymbol{h}_u^{(l-1)}), \quad \forall u \in V \tag{3}$$

- **Node Feature Update:** Once the messages for all nodes are computed, the next step is to update the node features using these messages. The updated feature for a node v, $\boldsymbol{h}_v^{(l)}$, is computed by taking the node's previous features, aggregating the messages from its neighbors, and then applying the update function $U_l(\cdot)$:

$$\boldsymbol{h}_v^{(l)} = U_l\left(\boldsymbol{h}_v^{(l-1)}, \texttt{Aggregation}\left(\left\{\boldsymbol{m}_u^{(l)} : u \in \mathcal{N}(v)\right\}\right)\right) \tag{4}$$

  where $\mathcal{N}(v)$ is the set of neighboring nodes of v.

There have been various models in the literature that define specific implementations of the aforementioned operations. Notable examples include the work by Kipf and Welling (2017) and Hamilton et al. (2017).

MILP problems can be represented and fed into GNNs by constructing bipartite graphs. In these graphs, one set of nodes represents the variables in the MILP problem, while the other set represents the constraints. The edges in this bipartite graph are then determined based on the relationships dictated by the MILP's incidence matrix, denoted as A. Furthermore, the initial features of these nodes can be derived from the weights associated with the nodes and edges, which are given by elements like c_j and $A_{i,j}$.

To illustrate with an example, let us consider a hypothetical MILP problem defined with $\boldsymbol{c} = [1,2,3]^T$, $A = [[1,2,0],[0,1,-1],[3,0,1]]$, and $\boldsymbol{b} = [2,1,4]^T$. Based on this, a bipartite graph can be constructed. This graph would have two sets of nodes: V_{var} representing variables $\{x_1, x_2, x_3\}$ and V_{con} representing constraints $\{C_1, C_2, C_3\}$. The edges in this graph, denoted as E, are then determined by looking at the non-zero elements present in matrix A.

4 References

- Kipf, T. N., & Welling, M. (2017). Semi-Supervised Classification with Graph Convolutional Networks. arXiv preprint arXiv:1609.02907.
- Hamilton, W. L., Ying, R., & Leskovec, J. (2017). Inductive Representation Learning on Large Graphs. In Advances in Neural Information Processing Systems (pp. 1024-1034).