CS225 Project Proposal: ML for Branching

Ya-Chi Chu
Department of Mathematics
SUNet ID: ycchu97

Yingxi Li
Department of MS&E
SUNet ID: yingxi

1 Introduction

Mixed Integer Linear Programming (MILP) is an important class of mathematical optimization problems which are widely used in multiple application domains. MILPs are typically tackled by the branch-and-bound (B&B) algorithm, which recursively partitions the solution space into a search tree and computes relaxation bounds along the way to prune subtrees that provably cannot contain an optimal solution. The process requires make sequential decisions on *node selection* and *variable selection*. The former selects the next node in the search tree to evaluate the relaxation bound; and the latter selects the variable by which to partition the search space at the node. State-of-the-art MILP solvers typically make these decisions according to hand-crafted heuristics, which are carefully designed to minimize the average solving time on a representative set of MILP instances. In recent years, a line of works propose to use machine learning (ML) methods to learn the strategies for node selection and variable selection. In this project, we focus on the approach using graph neural network (GNN) model.

One practical challenge that MILP could be used to model and address is the school zoning problem. This problem concerns designing geographic boundaries to split a city into multiple zones, each containing several schools, where students would have access to schools within the zone where they reside. It is a highly challenging problem due to its multi-objective nature: zones should pertain to socioeconomic diversity, racial diversity, enough capacity for its residents, balanced sizes, and relatively compact shapes. The current approach is to formulate all objectives as linear constraints, which creates a huge MILP: the number of constraints exponentially grows as the number of neighborhood units increases. Such MILP is currently impossible to solve directly using commercial solvers. We suspect one main reason could be the inefficient branching strategy of solvers.

In this project, we aim to use ML to learn a more efficient branching scheme for the B&B algorithm, thereby speeding up the resolution of large-scale MILPs, such as the school zoning problem. First, we would like to train the GNNs for B&B on synthetic general MILP instances and evaluate the branching performance on school district MILP instances of different scales. Second, we propose to train the GNNs on partial school zones extracted from the large-scaled school zoning problem and evaluate the performance on the original large-scale school zoning problem.

2 Backgrounds

Mixed Integer Linear Programming Given a matrix $A \in R^{m \times n}$, vectors $b \in R^m$ and $c \in R^n$, a mixed integer linear program is an optimization problem in the following form:

$$z^* = \min\{c^T x | Ax \leq b, x \in \mathbb{R}^n, x_j \in \mathbb{Z} \quad \forall j \in M \subset 1, ..., n\} \tag{1}$$

A relaxation of a MILP is defined as a linear program acquired by relaxing the integral constraints of a MILP. The linear relaxation of 1 would be the following:

$$\tilde{z}^* = \min\{c^T x | Ax \leq b, x \in \mathbb{R}^n\} \tag{2}$$

Preprint. Under review.

Branch-and-Bound Solving MILP to optimality is a hard problem because of the nonconvex nature of the problem and the fact that complete enumeration cannot be affordable. The branch-and-bound algorithm allows us to cover all possible solutions by explicitly evaluation a small subset of them. The key idea is to use a divide-and-conquer strategy: divide the feasible solution domain into easier-to-solve sub-problems, and prune the part of the search tree using the solution of the sub-problems as relaxation bounds. Let X be the optimal solution of the relaxation problem 2, and f be the objective value function. We briefly describe it in Algorithm 1.

Algorithm 1 Branch-and-Bound(X, f)

Require: $L = \{X\}$ ▷ Input
Ensure: $\hat{x}$ ▷ Optimal solution

Initialize $\hat{z}$ with a large value
Initialize an empty priority queue Q
Insert initial node X into Q
while Q is not empty **do**
    Extract subproblem S with highest priority from Q, solve S to get a solution x_S
    if x_S is feasible to 1 and $f(x_S) < \hat{z}$ **then**
        $\hat{x} = x_S$, $\hat{z} = f(x_S)$
        fathomed by integrality
    else
        fathomed by infeasibility
    end if
    if $f(x_S) > \hat{z}$ **then**
        fathomed by bound, discard S
    else
        Branch S into children nodes
        Insert children nodes into Q
    end if
end while
return $\hat{x}$

In Algorithm 1, it's important to note that a sub-problem (or node within the search tree) can be "fathomed," effectively pruning all of its descendant branches, under three conditions: first, if the sub-problem proves infeasible, yielding no viable solution; second, if the sub-problem yields a solution that satisfactorily meets the constraints outlined in 1 (referred to as the "incumbent" solution); or third, if the sub-problem yields a solution with an objective value inferior to the current best "incumbent" solution. Understanding these conditions underscores the significance of the node selection and variable selection order in determining the solving time of a problem.

Consider this scenario: if we could identify the optimal set of nodes to branch to right from the outset of the B&B algorithm, we could leverage this as the "incumbent" solution, rapidly pruning the majority of alternative branches. It's evident why the strategic selection of nodes and variables is pivotal in optimizing problem-solving efficiency. This observation naturally motivates the exploration of machine learning techniques to develop a more intelligent search scheme.

3 Related works

GNN for CO Graph neural networks (GNN) emerge as a powerful framework for machine learning on graph-structured data Gilmer et al. (2017). Because many prominent CO problems involve graph structures, many attempts have been made to use GNN for CO. Some works focused on directly tackling CO problems and output solutions using GNN, while others focused on incorporating GNN in modules of the solver. Khalil et al. (2017), for example, uses GNN to approximate value function in the Q-learning formulation of CO.

GNN for Branch-and-Bound Even before GNN becomes popular, machine learning for B&B has already been proposed: Khalil et al. (2016) use ML to learn a variable-ranking rule that decides which variable to branch on next.

GNNs for variable selection. Gasse et al. (2019) propose a graph neural network (GNN) model for learning branch-and-bound variable selection policies, which leverages the natural variable-constraint bipartite graph representation of MILPs. Instead of training by reinforcement learning, Gasse et al. (2019) use imitation learning to learn directly from a strong expert branching rule. They demonstrate on four NP-hard problems that the branching policies learned by their GNN model outperforms previously proposed machine learning approaches for branching, and could also outperform the default branching strategy in the optimization solver.

GNNs for node selection. Khalil et al. (2022) propose a general framework using GNN to enhance the tasks of node selection and warm-starting in MILP solvers. Encoding the variable-constraint interactions of a given MILP as a bipartite graph, they leverage state-of-the-art graph neural network architectures to predict variable biases, i.e., component-wise averages of (near) optimal solutions, indicating how likely a variable will be set to 0 or 1 in (near) optimal solutions of binary MILPs. The predicted biases are then used to guide the solver, replacing the original heuristic rules.

4 Our Intended Approach

We have taken some initial steps of getting the code of Gasse et al. (2019) working and modifying its inputs. We plan to train the model for general MILP instances and test out its performance on the school choice problem. Then, we plan to dive into ways of extracting representative but smaller zoning instances from the initial school zoning problem, train the strong branching expert policies based on the optimal solutions of the smaller problems, and use the fine-tuned GCNN model to tackle the original school choice problem. In the end, if time permits, we will dive into the theoretical part of the learning to branch literature, such as Balcan et al. (2018), and see if we could derive some theoretical insight into the problem.

References

Maria-Florina Balcan, Travis Dick, Tuomas Sandholm, and Ellen Vitercik. Learning to branch. In *International conference on machine learning*, pages 344–353. PMLR, 2018.

Maxime Gasse, Didier Chételat, Nicola Ferroni, Laurent Charlin, and Andrea Lodi. Exact combinatorial optimization with graph convolutional neural networks. *Advances in neural information processing systems*, 32, 2019.

Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. In *International conference on machine learning*, pages 1263–1272. PMLR, 2017.

Elias Khalil, Pierre Le Bodic, Le Song, George Nemhauser, and Bistra Dilkina. Learning to branch in mixed integer programming. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 30, 2016.

Elias Khalil, Hanjun Dai, Yuyu Zhang, Bistra Dilkina, and Le Song. Learning combinatorial optimization algorithms over graphs. *Advances in neural information processing systems*, 30, 2017.

Elias B Khalil, Christopher Morris, and Andrea Lodi. Mip-gnn: A data-driven framework for guiding combinatorial solvers. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 10219–10227, 2022.