[RE] Exact Combinatorial Optimization with Graph Convolutional Neural Networks

Lourdes Crivelli and Audrey-Anne Guindon
HEC Montréal

Introduction

In our reproducibility paper we study the application of graph convolution neural network (GCNN) model to learn branch-and-bound (B&B) variable selection for mixed-integer linear programs (MILP). Our goal is to implement the architecture described in the paper, run the experiments, provide insights and suggestions for replicating results, and analyze the outcomes obtained in comparison with the ones reported by the authors of the paper.

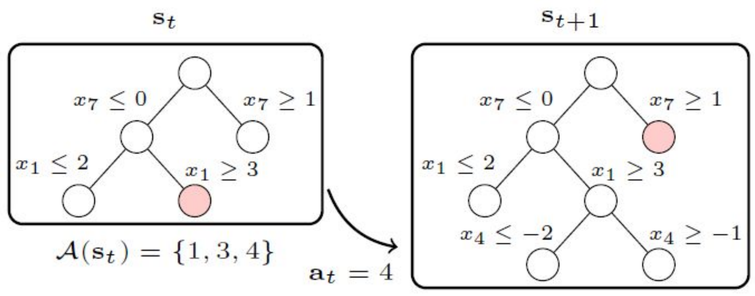
In order to assess the reproducibility of the paper and validate its conclusions, the main questions we will answer are:

- Can we validate the reported results?
- Does GCNN improve on other machine learning methods used to learn branching rules?
- How can the proposed model be further improved?

Background

MILP can be solved by applying the B&B algorithm. A key step in the B&B algorithm is selecting a fractional variable to branch on, which can have a very significant impact on the size of the resulting search tree. Strong branching computes the expected bound improvement for each candidate variable before branching, which requires the solution of two LPs for every candidate. Though computational expensive, it has been proven to produce the smallest search trees.

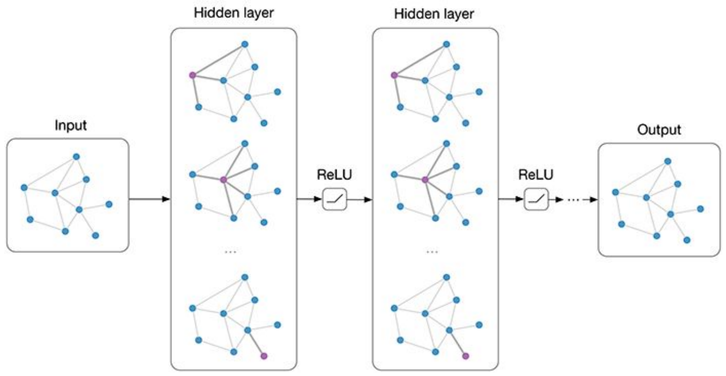
Most selection policies tend to rely on a set of hard-coded heuristics, designed specifically for a particular subtype of NP-hard problem. Since most trees and MILP have different sizes and structures, these rules fail to generalize and have limited applications.



The sequential decisions made during B&B can be assimilated to a Markov decision process. Consider the solver to be the environment, and the brancher the agent. At the t^{th} decision the solver is in a state s_t, the brancher then selects a variable among all fractional variables, according to a policy $\pi(a_t \mid s_t)$. Each episode amount to solving a MILP instance.

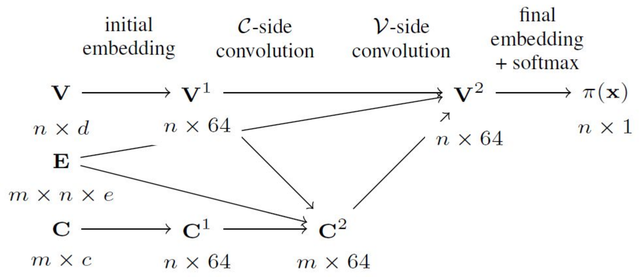
Instead of relying on strong branching, an imitation approach is used. The strong branching method is run on a training set, recording the state-action pairs. Next, a policy is learnt by picking the actions that minimize the cross-entropy loss. This approach is called behavioural cloning.

GCNN Structure

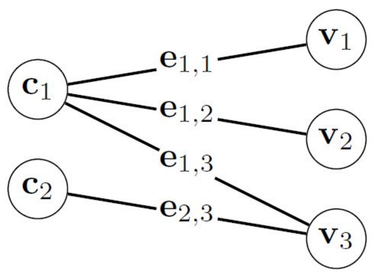


Important properties of GCNN:

1. they are well-defined no matter the input graph size;
2. their computational complexity is directly related to the density of the graph;
3. they are permutation-invariant, that is they will always produce the same output no matter the order in which the nodes are presented.



Our bipartite graph is split into two, one showing the relationship between variables to constraints and the other from constraints to variables. The result is a new bipartite graph, but with information about the neighbours. Though it is common to normalize each convolution (by the number of neighbours) it was avoided in favour of preserving information.



The initial weights are set to (x-β)/σ taking the mean (β) and standard deviation(σ) of x, allowing to generalize for new datasets.

Experiments

Within the experiments described in the paper, we focus on replicating the precise settings for three NP-Hard benchmarks, namely, a set-covering, combinatorial auction, and facility location problem.

NP-Hard Problem	Train	Test : Easy	Test: Medium	Test: Hard
Set Covering	500 x 1000	500 x 1000	1000 x 1000	2000 x 1000
Combinatorial Auction	100 x 500	100 x 500	200 x 1000	300 x 1500
Capacitated Facility Location	100 x 500	100 x 100	200 x 100	400 x 100

100,000 branching samples were generated from 10,000 training instances and 20,000 samples from 2,000 validation instances.

Although the authors compared their method to other machine learning methods, and notably, to MILP solvers, we focused only on replicating the benchmarks using the GCNN model.

Reproducibility Details

The paper was clearly written, implementation details were well documented, and the authors provided their code making it easier to replicate the results reported. However, we still encountered problems during the replication.

The process of generating the 120,000 samples (state-action pairs) was the most time consuming aspect of the reproducibility. The data generation process took from 12 to 35 hours even when using multiple CPUs. The resulting datasets were of 4GB which made them difficult to work with. The time it took to generate training instances speaks to the importance of this paper in addressing solver efficiency.

Furthermore, the version of full strong branching, *vanillafullstrong*, developed by the authors was removed from subsequent versions of *PySCIPOpt*, meaning the package was no longer up to date. A similar issue arose with the implementation of *tensorflow*'s contrib, which is no longer supported.

Future Work

- Our next step is to test the resulting algorithm on the validation set. Due to time and computation constraints, we will be focusing on reproducing the combinatorial auction NP-hard problem.
- We would recommend updating the original code, so that it works with the newer versions of *tensorflow* and *PySCIPOpt*. The author's code is well written and made it easy to reproduce the results; therefore, it would be ideal that it work in most environments.

1. Gasse, M., Chételat, D., Ferroni, N., Charlin, L., Lodi, A. (2019) Exact Combinatorial Optimization with Graph Convolutional Neural Networks. *arXiv:1906.01629, 2019.*
2. KiPF , T. (2008) Graph Convolutional Networks. Retrieved on 25 Nov. 2019 from: https://tkipf.github.io/graph-convolutional-networks/