Solving MIP using NN (Google)

- Preliminary

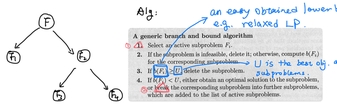
1. Branch and Bound. (General Description)

Consider the following problem:

$\min c^T x \quad \text{s.t. } x \in F.$

We aim to solve it in a "divide and conquer" way.

$\min c^T x \quad \text{s.t. } x \in F_i, \ i = 1, 2, \dots, k.$



Alg:

A generic branch and bound algorithm

1. Select an active subproblem F_i.
2. If the subproblem is infeasible, delete it; otherwise, compute $b(F_i)$, the lower bound for the corresponding subproblem.
3. If $b(F_i) \geq \bar{U}$, delete the subproblem.
4. If $b(F_i) < \bar{U}$, either obtain an optimal solution to the subproblem, or break the corresponding subproblem into further subproblems, which are added to the list of active subproblems.

an easy obtained lower bound. e.g. relaxed LP.

$\bar{U}$ is the best obj among those of subproblems

Two problems:

①: how to select F_i. (Best First, DFS) ②: how to split.

Suppose x^* is the optimal solution of some relaxed LP for F_i.

①. choose some integer variable x_j ▷ "how to choose?" Learn it in this paper.

②. creating two subproblems (splitting two nodes) by adding constrains: $x_i \leq \lfloor x_i^* \rfloor$; $x_i \geq \lceil x_i^* \rceil$

2. Primal Heuristic

Aiming to find a feasible variable assignment, providing an upper bound for the optimal value. (primal bound). There are two typical ways to use in MIP:

①. Provide a variable assignment on integer variables and solve the resulting LP.

②. Combining with branch and bound, finding a feasible assignment of the unfixed variables from a given node in the search tree.

Diving:

Diving is a kind of primal heuristic and can be combined in the procedure of B & B.

> Pruning suboptimal branches is an important part of Branch-And-Bound-algorithms. This helps keeping the Branch-And-Bound-tree small as well as the number of computing steps and hence, the solving time and the required memory. A branch can be pruned if the objective value of its LP-optimum is not smaller than the one of the incumbent solution.
>
> Therefore, it is of high importance to discover feasible solutions as early as possible in order to achieve a good performance of the Branch-And-Bound-process. Start heuristics aim at finding a feasible solution early in the Branch-And-Bound-process.

We have known there are some policies to select node to branch in B & B. Diving corresponds to the DFS (depth first) policy, aiming to get a feasible solution a.s.a.p. So it's a kind of primal heuristic.

- Main Points.

1. Neural Diving.

① Basic idea:

Train a generative model (random but deterministic) over the assignments to integer variables.

② Supervised Learning: Data collected by SCIP.

③ Details:

Given a parametric generative model $P_\theta(x|M)$, the input is an MIP instance $M = (A, b, c)$, the output is the assignment on integer variables x, represented by the energy function:

$$P_\theta(x|M) = \frac{\exp(-E(x;M))}{Z} \quad \text{where} \quad E(x;M) = \begin{cases} c^T x & \text{if feasible} \\ \infty & \text{otherwise.} \end{cases}$$

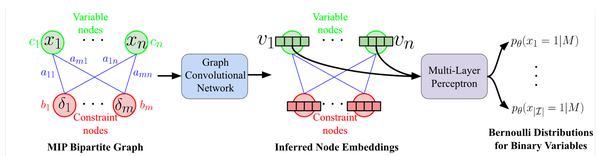
Given dataset $D_{train} = \{(x_i, M_i)\}_{i=1}^N$, $x_i = \{x_{ij}\}_{j=1}^{N_i}$ (N_i assignments).

using MLE to optimize:

$$L(\theta) = -\sum_{i=1}^{N} \sum_{j=1}^{N_i} w_{ij} P_\theta(x_{ij}|M_i)$$

where $w_{ij} = \frac{\exp(-c_i^T x_{ij})}{\sum_j \exp(-c_i^T x_{ij})}$ to balance the bias of the sampling.

④. Parametrization Method:



Naive Bayes assump. $P_\theta(x|M) = \prod_{d \in \mathcal{I}} P_\theta(x_d|M)$ (Also has Auto-Regressive Version).

Binary Case: $t_d = MLP(v_d; \theta)$. $P_\theta(x_d = 1|M) = \text{sigmoid}(t_d)$.

General Case:

represent an integer in the binary style: n_b → some integer

"train the most n_b significant bits"

During inference:

If $\lceil \log_2(\text{card}(\mathcal{Z})) \rceil > n_b$: n_b predictions. $\mathcal{Z}$.

else: $\lceil \log_2(\text{card}(\mathcal{Z})) \rceil$ predictions.

2). Combining with the classic solver:

Only choose a subset of integer variables to assign:

$x_1, x_2, x_3, \dots, x_{|\mathcal{I}|}$

↓(1) use NN; ↓(1) use NN; ↓(0) not use; ↓(0) not use

Train a new binary classifier to determine which variable should be assigned by NN. [Selective Net]

Loss: $\ell_{selective}(\theta, x, M) = \frac{-\sum_{d \in \mathcal{I}} \log P_\theta(x_d|M) \cdot y_d}{\sum_{d \in \mathcal{I}} y_d} + \lambda \left(c - \frac{1}{|\mathcal{I}|} \sum_{d \in \mathcal{I}} y_d\right)^2$

preset coverage rate. In practice, train several models w.r.t. different c.

$L_{selective}(\theta) = \sum_{i,j} w_{ij} \ell_{selective}(\theta, x_{ij}, M_i)$.

"In experiments with the conditionally independent model we use one sample per model, and generate different sub-MIPs by using models trained with different coverage thresholds for γ."

2. Neural Branching.

① Basic idea: Learn to choose which variable to branch on

② Imitation Learning (Supervised learning)

③ Details:

i) Collect data from Full Strong Branch (FSB) policy, producing smaller search trees.

$\{x_1, \dots, x_c\}$: integer candidates. ADMM solves 2c LPs $\{(OPT_i^{up}, OPT_i^{down}), i = 1, \dots, c\}$

$s_i = (OPT_i^{up} - OPT + \varepsilon)(OPT_i^{down} - OPT + \varepsilon)$ $\quad p_i^{expert} = \frac{s_i}{\sum_j s_j}$

directly select one ↓ obj val at this node.

$t_c = MLP(v_c, \phi)$ $\quad P_\phi(x_c|M) \sim \text{softmax}(\{t_j, j = 1, \dots, c\})$ $\quad L(\phi) = \sum_{j=1}^{c} p_j^{expert} \log P_\phi(x_j|M)$.

3. Joint Evaluation. Neural Diving + Neural Branching.

An MIP instance: $\mathcal{I}$

$\mathcal{I} \to ND \to \begin{cases} \text{assigned} & \{x_i, i \in \mathcal{I}^a\} \\ \text{unassigned} & \{x_i, i \in \mathcal{I}^u\} \end{cases}$ fixed / unfixed → Neural Branching → $\begin{cases} \{x_i, i \in \mathcal{I}^a\} \\ \{x_i, i \in \mathcal{I}^u\} \text{ solved by NB} \\ \{x_i, x_i \text{ is cont.}\}. \text{ LP}. \end{cases}$

4. Comments: Naive Learning + massive tricks + huge computation cost.

More like a technique report rather than a research paper.

其实这里玩了一个小小的文字游戏。**作为MIP求解器开发人员，一般不把一定时间内能拿到的Gap作为主要衡量标准。**因为这有一定的误导性。设想一类较特殊的整数规划问题，如可行性问题，它没有目标函数，只需要找到一组整数解即可完成。那么在找到整数解之前，其Gap就是100%，找到之后就是0%。如果某个启发式（或者割平面）算法，在开启和关闭的的情况下，分别可以于1小时和3小时找到可行解。则如果以两小时为观察点，则可以说在开启这项算法的前提下，实现的Gap提升就是无穷多倍，而若以半小时或者三个小时作为观察点，则Gap没有提升。鉴于DeepMind并未公布计算这些性能指标的原始数据，我们无法用MIP业内的公认方式来对它做出评价。一般来说，根据目前公认的测试标准，一般是在MIPLIB的问题集上，以两小时为限，考虑能求解的问题数量和平均求解时间进行比较。

对于特定的测试集取得惊人的性能提升并不意外，因为这正是机器学习擅长的地方：它可以捕捉同一类问题的特征结构，并且给出优化趋势的判断。如后文所述，我们自己在开发的过程中也有类似的经历。真正值得关注的是它在MIPLIB上的表现。MIPLIB 2017 由1000多个来自各行各业的实例构成，而MIPLIB2017 Benchmark则是其中挑选的240个结构各异的问题组成，在筛选的时候就充分的做到了差异化，因此它和电网优化和N N Verification等测试集有本质的区别。这也解释了在

To tackle this challenge, we built a training set for each attempted open instance which should represent the problem distribution of the target instance. More specifically, for a given target instance, we generated up to 500 similar instances through randomly applying a mix of the following manipulations: (1) Randomly dropping constraints of the target instance, and (2) fixing a random subset of variables to the previous incumbent. To generate each such instance, we applied 10 iterations of each step (1) and (2) on the original target instance.

ML can only capture the similarity on fixed dataset.