GNN&GBDT-Guided Fast Optimizing Framework for Large-scale Integer Programming

March 13, 2024

Contents

Problem introduction

The latest two-stage optimization framework based on graph neural network (**GNN**) and large neighborhood search (**LNS**) is the most popular framework in solving large-scale integer programs (**IPs**). However, the framework can not effectively use the embedding spatial information in GNN and still highly relies on large-scale solvers in LNS, resulting in the scale of IP being limited by the ability of the current solver and performance bottlenecks.

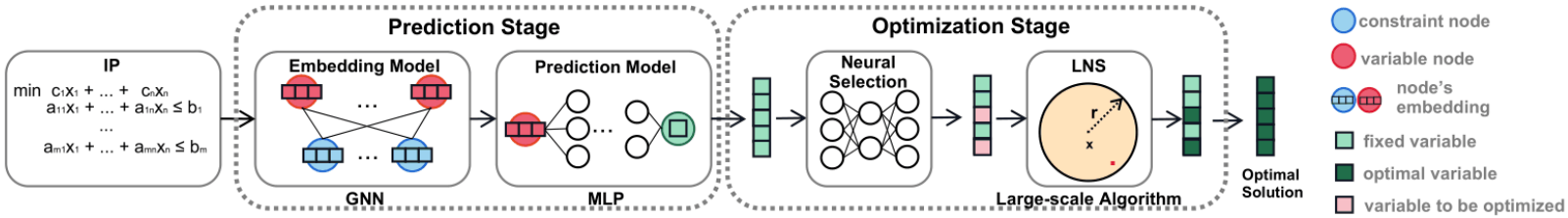


Figure 1. The two-stage framework of prediction-and-optimization. The prediction stage adopts a prediction model combining a graph neural network(GNN) based embedding model and a multi-layer perceptron(MLP) based prediction model. The optimization stage adopts a large neighborhood search strategy under the guidance of machine learning.

Problem introduction

To handle these issues, this paper presents a GNN&GBDT-guided fast optimizing framework for large-scale IPs that only uses a small-scale optimizer to solve large-scale IPs efficiently. Specifically, the proposed framework can be divided into three stages:

- Multitask GNN Embedding to generate the embedding space.
- GBDT Prediction to effectively use the embedding spatial information.
- Neighborhood Optimization to solve large-scale problems fast using the small-scale optimizer.

Integer Programs

Integer Programs (IPs) are a type of problem of maximizing or minimizing a linear expression subject to a number of linear constraints, where all decision variables are restricted to take integer values.Formally, an integer program can be defined as the following.

$$\begin{aligned} \min_{x} \quad & c^T x, \\ subject \quad to \quad & Ax \leq b, l \leq x \leq \mu, x \in z^n, (1) \end{aligned}$$

- where n is the number of decision variables, with $c, l, u \in R^n$ being their coefficient, lower bound and upper bound. $A \in R^{m \times n}$ and $b \in R^m$ denote the linear constraints on x.

Bipartite Graph Representation

The Bipartite Graph Representation of IPs was proposed by Gasse in $2019^{[1]}$ (Gasse et al., 2019), which can realize the lossless encoding of the original IP into a bipartite graph as the input of the Graph Neural Network, described in Figure 2.

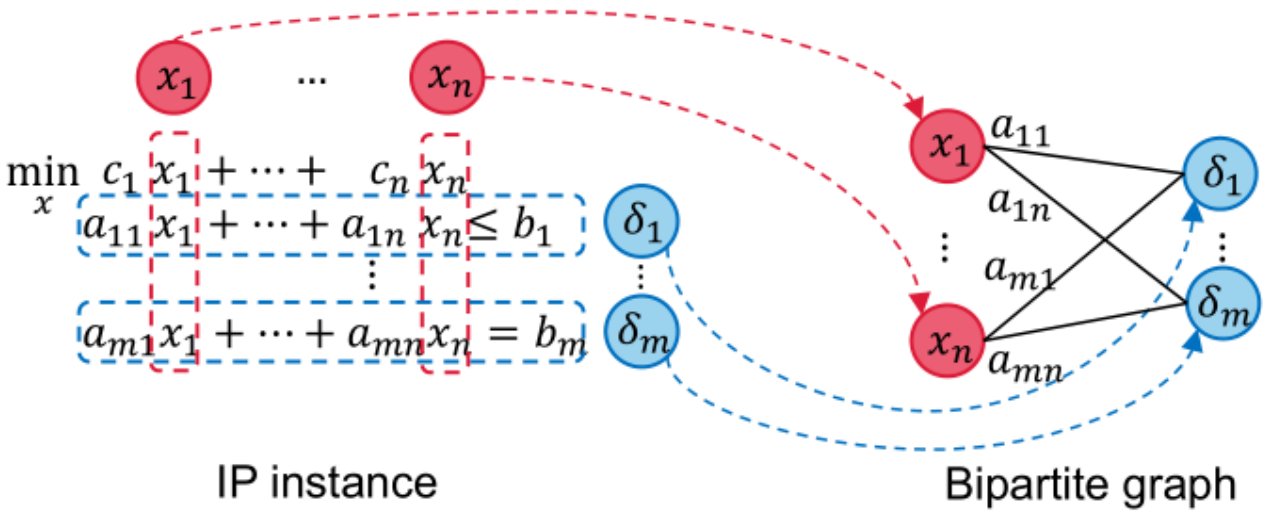


Figure 2. Transforming an IP instance to a bipartite graph. The set of n decision variables nodes $\{x_1, \ldots, x_n\}$ and the set of m constraint nodes $\{\delta_1, \ldots, \delta_m\}$ form the left set and right set of nodes of the bipartite graph.

Bipartite Graph Representation

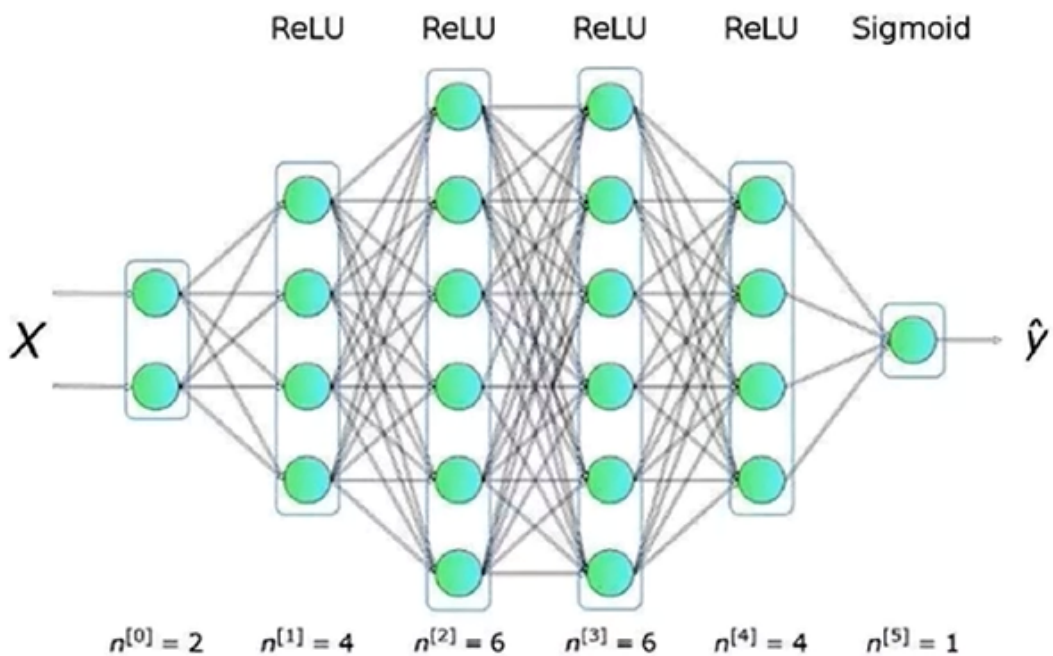
- The left set of n variable nodes in the bipartite graph represents the decision variables, while the right set of m constraint nodes represents the linear constraints.
- An edge (i, j) with edge weight a_{ij} connecting the left node i and right node j represents that the i-th decision variable appears in the j-th constraint, and the coefficient is a_{ij} .
- In the classic bipartite graph representation application, the feature selection of nodes and edges usually only depends on the coefficients in formula (1), such as upper bound l , lower bounds r, etc.

Graph Neural Network

In IPs, a GNN is often used for model learning and neighborhood aggregation after bipartite graph representation.Formally, let ε denote the set of edges in the bipartite graph,a k layer **GNN** could be written as below.

$\widehat{h}_v^k = f_2^k\left(\left\{h_v^{k-1}, f_1^k\left(\left\{h_u^{k-1} : (u,v) \in \varepsilon\right\}\right)\right\})$,(2)

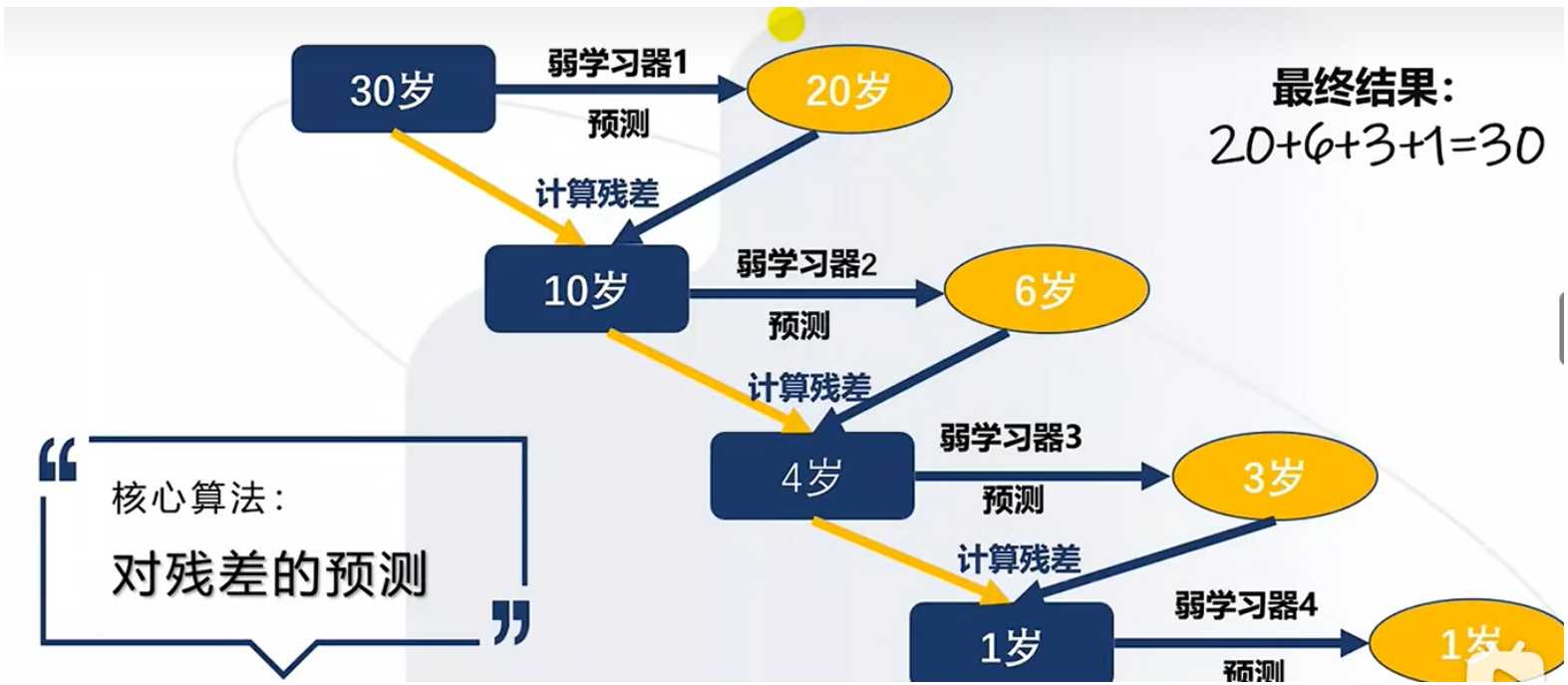
where h_v^k denotes the hidden state of node v in the k-th layer.The function f_1^k combines the hidden value of the $(k-1)$ -th layer of the neighbors to get the aggregation information, and function f_2^k combines the hidden value of the current point ε and the aggregation information of its neighbors.



Gradient Boosting Decision Tree

For a given data set containing n examples and m features $D = \{x_i, y_i\}$ where $|D| = n, x_i \in R^m$, and $y_i \in R$, GBDT tries to use K regression trees to fit the data set. The final prediction result of GBDT is the weighted sum of T regression trees-based model prediction

$$\hat{y}_i = \phi(x_i) = \sum_{t=1}^{T} f_t(x_i)$$



Gradient Boosting Decision Tree

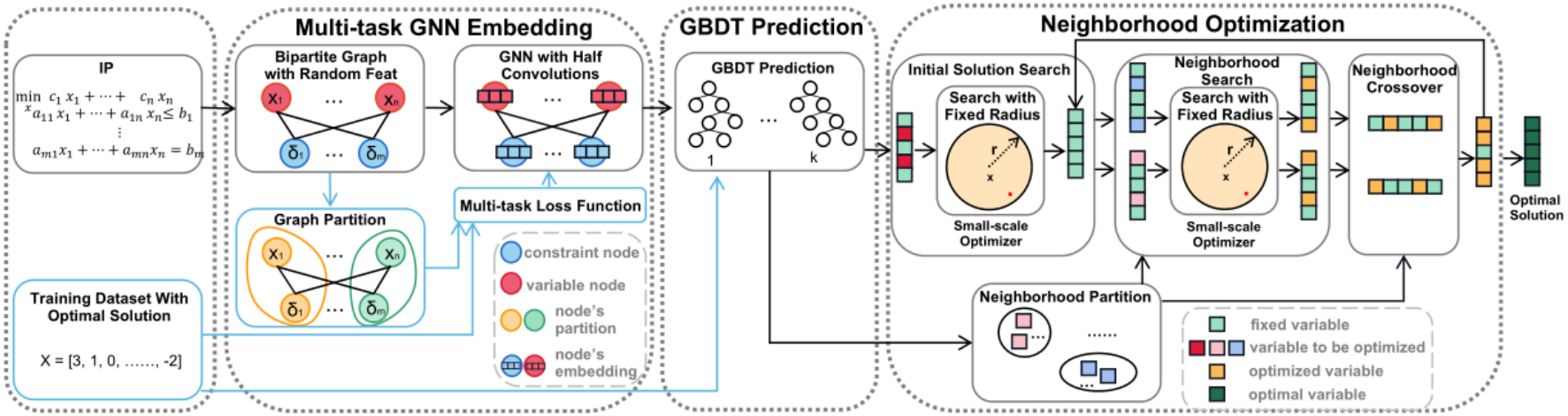


Figure 3. An overview of GNN&GBDT-Guided Fast Optimizing Framework for Large-scale Integer Programming. The blue line indicates that it is only used during training, while the black line indicates that it is used during both training and testing. In the stage of Multi-task GNN Embedding, the IP is represented as a bipartite graph, followed by employing a graph partition algorithm(FENNEL) to divide the bipartite graph into blocks. Then a multi-task GNN with half convolutions is used to learn the embedding of decision variables, where the loss function is a metric for both optimal solution and graph partition. In the stage of GBDT prediction, the GBDT is used to predict the optimal value of the decision variable in the IP through the variable embedding. In the stage of Neighborhood Optimization, some decision variables are fixed as the rounding results of the predicted values of GBDT and a search with fixed radius is used to find an initial solution. Then, under the guidance of the neighborhood partition, neighborhood search and neighborhood crossover are used iteratively to improve the current solution.

Multi-task GNN Embedding

- Then a graph partition algorithm is used to divide the bipartite graph into blocks.
- Further, a multi-task GNN with half convolutions is used to learn the embedding of decision variables, where the loss function is a metric for both optimal solution and graph partitions.

Multi-task GNN Embedding

Formally, let $h_x^i, h_\delta^j, h_{(i,j)}$ denotes the feature selection of the $i-$th variable node, $j-$th constraint node and edge (i,j).

$$h_x^i = (c_i, l_i, u_i, \xi),$$

$$h_\delta^j = (b_j, o_j, \xi),$$

$$h_{(i,j)} = a_{ij},$$

where c_i, l_i, u_i denotes the coefficient, lower bound and upper bound of the i-th decision variable,b_j, o_j denotes the value and symbol of the j-th constraint,a_{ij} denotes the weight of edge (i,j) and $\xi \sim U(0,1)$ denotes the uniform random feat between 0 and 1

Graph Partition Algorithm

FENNEL is a stream algorithm, which means that it checks each node υ in the graph one by one and calculates $\delta_g(\upsilon, p_i)$ with each block p_i. The calculation method is as follows.

$$\delta_g(\upsilon, p_i) \leftarrow |p_i \cap N(\upsilon)| - \alpha\gamma|p_i|^{\gamma-1},$$

where $N(\upsilon)$ denotes the neighbor node set of υ,α,γ are preset parameters related to block balancing and minimum cut.

GBDT prediction

The GBDT is used to predict the optimal value of the decision variables in the IP through the variable embeddings, and generating the guidance for the initial solution search and neighborhood partition in the next stage.

Variable Value Prediction

A training data set $D = \left\{h_x^K, Opt_i\right\}_{i=1}^N$ is used to construct GBDT. The trained GBDT can predict the variables value in the optimal solution through the weighted accumulation of the prediction results of each regression tree that can be rewritten as follows according to formula

$$\hat{y_i} = \phi\left(h_{x_i}^K\right) = \sum_{t=1}^T f_t\left(h_{x_i}^K\right),$$

And the prediction loss and embedding space partition of each regression tree will be used to guide the initial solution search and neighborhood partition in the next stage.

Neighborhood Optimization

- In the stage of neighborhood optimization, for solving a large-scale IP with n decision variables, a variable proportion $\in (0, 1)$ needs to be defined first, which means that an optimizer that can solve a small-scale IP with αn decision variables can be used to solve the corresponding large-scale IP.
- In order to realize this stage, a search with a fixed radius is used to search an initial solution first.
- Then under the guidance of neighborhood partition, neighborhood search and neighborhood crossover are used iteratively to improve the current solution.

Initial Solution Search

- Given the predicted value $\hat{y}_i$ and the prediction loss P_i of each decision variable, the decision variables are sorted according to the prediction loss from small to large. After that, the first $(1 - \alpha n)$ decision variables are fixed, while the remaining variables are searched with a fixed radius. The details are shown in Algorithm 1.

Numerical experiments

Algorithm 1 Initial Solution Search

Input: The number of decision variables n, predicted value $\hat{y}$, prediction loss $\mathcal{P}$, variable proportion α

Init: Initial Solution $\mathcal{X} = \{\}$

$\mathcal{X} \leftarrow \hat{y}$

Sort the decision variables in ascending order of $\mathcal{P}$

$\alpha_{set} = \alpha$

repeat

  $\mathcal{F} \leftarrow$ The first $(1 - \alpha_{set})n$ decision variables ▷Fixed

  $\mathcal{U} \leftarrow$ The last α_{set} decision variables ▷Unfixed

  $\mathcal{F}', \mathcal{U}' \leftarrow \text{REPAIR}(\mathcal{F}, \mathcal{U}, \mathcal{X})$

  if $|\mathcal{U}'| > \alpha n$ **then**

    $\alpha_{set} = \eta * \alpha_{set}$

  end if

until $|\mathcal{U}'| \leq \alpha n$

$\mathcal{X} \leftarrow \text{SEARCH}(\mathcal{F}', \mathcal{U}', \mathcal{X})$

Return: $\mathcal{X}$

Neighborhood Partition

Every iteration of neighborhood search and neighborhood crossover requires a new neighborhood partition. Since the embedding of variables with a strong correlation are close, the embedding space partition result of the regression tree in GBDT is directly used as the neighborhood partition. Figure 4 is an example of the neighborhood partition.

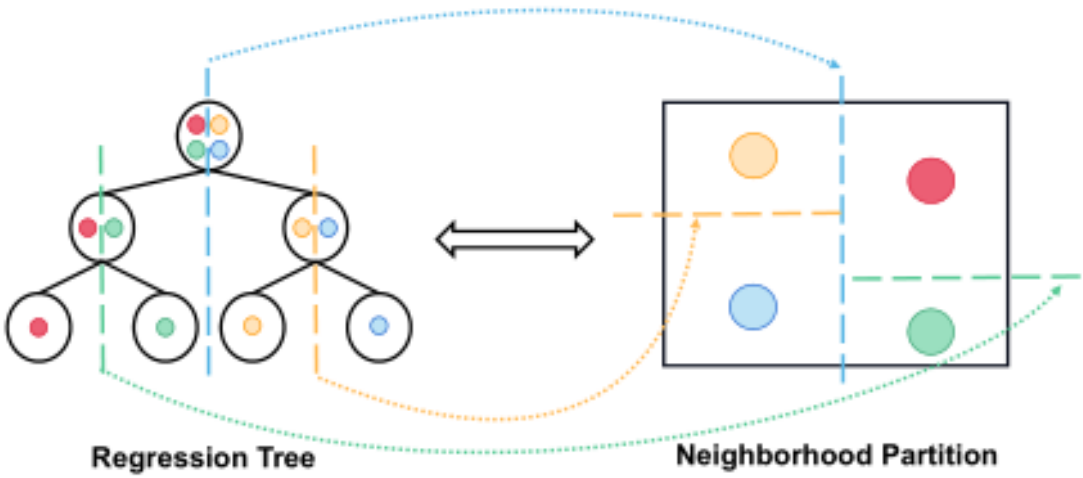


Figure 4. Using the partition result of the regression tree as the neighborhood partition. Different branches on the left regression tree correspond to the partitions of the right embedding space.

Neighborhood Search

Based on the neighborhood partition results, the current solution is used to explore neighborhoods in parallel. Specifically, for the i-th neighborhood N_i ,the details in neighborhood search are shown in Algorithm 2.

Algorithm 2 Neighborhood Search

Input: The set of decision variables X, the number of decision variables n, predicted value $\hat{y}$, prediction loss $\mathcal{P}$, variable proportion α, neighborhood N_{now}, current solution $\mathcal{X}$

Init: Neighborhood search solution $\mathcal{X}' = \{\}$

Sort the decision variables in N_{now} in descending order of $\mathcal{P}_i * |\phi_i - \hat{y}_i|$

$\mathcal{N} \leftarrow$ The first αn decision variables in N_{now}

$\mathcal{F} \leftarrow \{x \| x \in X \wedge x \notin N\}$ ▷Fixed

$\mathcal{U} \leftarrow \{x \| x \in X \wedge x \in N\}$ ▷Unfixed

$\mathcal{X}' \leftarrow \text{SEARCH}(\mathcal{F}, \mathcal{U}, \mathcal{X})$

Return: $\mathcal{X}'$

Neighborhood Crossover

Since the size of the neighborhood is limited to no more than αn , it is easy to fall into the local optima because the radius of the neighborhood search is too small. So neighborhood crossover is crucial. Based on the result of neighborhood partition, neighborhood crossover is carried out step by step, as shown in Figure 5.

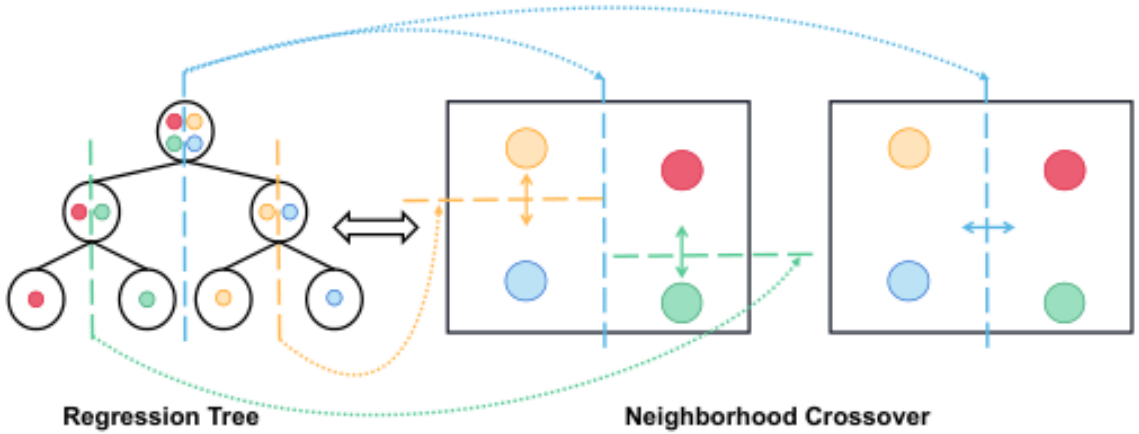


Figure 5. Carrying out neighborhood crossover step by step. First, the neighborhood partition corresponding to the second layer of branching of the regression tree is crossed by neighborhood crossover. Then the crossover over the neighborhood partition corresponding to the first layer of branching is similar to the second layer.

Neighborhood Crossover

Algorithm 3 Neighborhood Crossover

Input: The set of decision variables X, the number of decision variables n, neighborhood N_1, N_2, neighborhood search solution $\mathcal{X}'_1, \mathcal{X}'_2$
Init: Neighborhood crossover solution $\mathfrak{X} = \{\}$
$\mathcal{X}'' \leftarrow \{\}$
for $i = 1$ **to** n **do**
  if The i-th decision variables in N_1 **then**
    $\mathcal{X}''[i] \leftarrow \mathcal{X}'_1[i]$
  else
    $\mathcal{X}''[i] \leftarrow \mathcal{X}'_2[i]$
  end if
end for
$\mathcal{F} \leftarrow X$ ▷Fixed
$\mathcal{U} \leftarrow \emptyset$ ▷Unfixed
$\mathcal{F}', \mathcal{U}' \leftarrow \text{REPAIR}(\mathcal{F}, \mathcal{U}, \mathcal{X}'')$
if $|\mathcal{U}'| \leq \alpha n$ **then**
  $\mathfrak{X} \leftarrow \text{SEARCH}(\mathcal{F}', \mathcal{U}', \mathcal{X}'')$
end if
Return: $\mathfrak{X}$

Summary

- On top of it, by using a neighborhood search with a fixed search radius and small-scale neighborhood crossover, our framework can just use a small-scale optimizer to solve large-scale IPs efficiently.

Reference

[1] Gasse, M., Chételat, D., Ferroni, N., Charlin, L., and Lodi, A. Exact combinatorial optimization with graph convolutional neural networks. Advances in Neural Information Processing Systems, 32, 2019.

Think

The proposed framework is currently tailored for efficient solving of IPs. In the future,Can we will continue optimizing the proposed framework and try to make breakthroughs in the aspects of ultra-large-scale, multi-objective and mixed integer?

Thanks!