IPGPT: Solving Integer Programming Problems with Sequence to Contrastive Multi-Label Learning

Shufeng Kong[1,2], Caihua Liu[3] and Carla Gomes[2]

[1]Sun Yat-sen University, Zhuhai Campus, China
[2]Cornell University, Ithaca, U.S.A
[3]Guilin University of Electronic Technology, Guilin, China

Abstract

Integer Programming (IP) is an essential class of combinatorial optimization problems (COPs). Its inherent NP-hardness has fostered considerable efforts towards the development of heuristic strategies. An emerging approach involves leveraging data-driven methods to automatically learn these heuristics. For example, using deep (reinforcement) learning to recurrently reoptimize an initial solution with Large Neighborhood Search (LNS) has demonstrated exceptional performance across numerous applications. A pivotal challenge within LNS lies in identifying an optimal subset of variables for reoptimization at each stage. Existing methods typically learn a policy to select a subset, either by maintaining a fixed cardinality or by decomposing the subset into independent binary decisions for each variable. However, such strategies overlook the modeling of LNS's sequential processes and fail to explore the correlations inherent in variable selection. To overcome these shortcomings, we introduce IPGPT, an innovative model that reimagines policy learning as a sequence-to-multi-label classification (MLC) problem. Our approach uniquely integrates a tailored Decision Transformer encoder, incorporating a causal transformer (GPT2) to capture the sequential nature of LNS. Additionally, we employ an MLC decoder with contrastive learning to exploit the correlations in variable selection. Our extensive experiments confirm that IPGPT significantly surpasses the performance of current state-of-the-art baselines and exhibits excellent generalization to larger instances.

Keywords

Integer Programming, contrastive learning, causal transformer, Multi-Label Learning

1. Introduction

IP has found applications in production planning [1], scheduling [2], scientific discovery [3], and telecommunications networks [4], among many others. It is well-known that IP is NP-complete [5] and many efforts have been devoted to designing effective heuristics to find near-optimal solutions [6]. Historically, such algorithms were designed largely manually, requiring a careful understanding of the underlying structure within specific classes of optimization problems.

Due to the recent success of deep learning (DL) and reinforcement learning (RL), there has been an increasing interest in automatically learning heuristics for COPs from training data [7]. Existing works often leverage machine learning (ML) to output solutions directly from input instances, configure hyperparameters of COP algorithms, or learn a local decision policy for search frameworks such as branch&bound (B&B), local branching (LB), or LNS. Among them, we are particularly interested in learning to iteratively reoptimize an initial solution with LNS [8, 9, 10, 11]. These approaches are attractive because we can leverage existing state-of-the-art commercial IP solvers such as Gurobi or SCIP as a generic black-box subroutine and thus benefits from the cutting-edge technologies of such commercial IP solvers.

In this paper, we focus on boosting the performance of LNS, though our method can also be applied to boost the performance of other local search algorithms such as LB. A key challenge of LNS is to select a promising variable subset to reoptimize based on the current solution. Since the selection choice is combinatorial, finding an optimal subset is also computationally hard. Song et al. [9] learn to select fixed, predefined variable subsets with imitation learning and RL. Wu et al. [8] learn to select arbitrary variable subsets with RL by factorizing the selection of a variable subset into elementary selections on each variable separately. Similarly, Sonnerat et al. [10] learn to predict the probability of selecting a variable independently of other variables using imitation learning and Nair et al. [11] use RL to learn a policy that selects one variable at a time. Nevertheless, all of these works miss modeling the sequential processes of LNS and also do not exploit correlations of variable selection [1]. To address these limitations, we propose to model the policy learning as a sequence to a multi-label classification problem, which jointly models the selection of variables as well as the sequential processes of LNS.

Our contributions are threefold: (1) we give a new an-

STRL'23: Second International Workshop on Spatio-Temporal Reasoning and Learning, August 2023, Macao, S.A.R, China
sk2299@cornell.edu (S. Kong); caihua.liu@guet.edu.cn (C. Liu); gomes@cs.cornell.edu (C. Gomes)

[1]We have reserved an in-depth discussion of the related work for the appendix due to space constraints.

gle of sequence to multi-label classification for learning an effective local decision policy for LNS; (2) we materialize this idea by providing a novel model to seamlessly integrate a customized decision transformer encoder to model the sequential processes of LNS and an MLC decoder with contrastive learning to exploit correlations of variable selection; (3) we conduct extensive experiments on various benchmarks and the results show that our model significantly outperforms state-of-the-art baselines.

2. Preliminaries

2.1. Integer Program

An integer program (IP) is a problem of optimizing a linear function over points in a polyhedral set: $\arg\min_x\{\mu^T x | Wx \leq b; x \geq 0; x \in \mathbb{Z}^n\}$, where $x \in \mathbb{Z}^n$ is a vector of n decision variables; $\mu \in \mathbb{R}^n$ denotes the vector of objective coefficients; the incidence matrix $W \in \mathbb{R}^{m \times n}$ and vector $b \in \mathbb{R}^m$ together define m linear constraints.

2.2. LNS and Its Markov Decision Process Formulation

Given an initial assignment of values to the decision variables in an IP instance, LNS iteratively refines this assignment by selecting a subset of decision variables, relaxing their values, and solving a subproblem that aims to optimize the objective function while respecting the instance's constraints. LNS aims to explore a complex solution neighborhood and gradually improve its current solution until a certain termination condition is met [12]. A key challenge of LNS is how to define a good solution neighborhood, namely, one needs to decide which variable subset to reoptimize given the current solution. Obviously, such a decision problem is combinatorial, and many works devote to constructing effective heuristics for it [13, 14, 15]. In this work, we are particularly interested in the recent trend of learning-based approaches, where data-driven methods are applied to learn the heuristics automatically [9, 8, 10]. To this end, the LNS framework can be formulated as a *Markov Decision Process* (MDP) $(\mathcal{S}, \mathcal{A}, P, R)$:

- $\mathcal{S}$ is a set of states. A state $s_t \in S$ describes the current status of the LNS process in step t, which normally includes the static IP instance information (e.g., variables, constraints, and objectives) and the dynamic solving statistics (e.g., the incumbent solution);
- $\mathcal{A}$ is a set of all candidate variable subsets for reoptimization. A variable subset $a_t \in \mathcal{A}$ is also called an *action* of an agent that is executed in step t;
- $P(s_t, a_t)$ is the transition function to return the next state. Let x_t be the solution with state s_t, a smaller sub-IP is first generated by keeping the values of non-selected variables in x_t and reoptimizing the remainder, and then the next state s_{t+1} is obtained by updating s_t with the new solution to the sub-IP: $x_{t+1} = \arg\min_x\{\mu^T x | Wx \leq b; x \geq 0; x \in \mathbb{Z}^n; x^i = x_t^i, \forall x^i \notin a^t\}$;
- $R(s_t, a_t)$ is the reward function to return the change of objective values, which is defined as $r_t = R(s_t, a_t) = \mu^T(x_t - x_{t+1})$. Let T be the step limit, the *cumulative rewards* from step t of an episode is defined as $R_t = \sum_{k=t}^{T} \gamma^{k-t} r_k$ with a discount factor $\gamma \in [0, 1]$.

A *policy* is a (potentially probabilistic) mapping $\pi : \mathcal{S} \to \mathcal{A}$. The goal of RL-based algorithms for solving IPs is to find a policy function to maximize the expected cumulative reward $\mathbb{E}[R_1]$ over all episodes, i.e., the expected improvement over initial solutions. However, existing RL-based algorithms for IP solving train a policy by either temporal difference (TD) learning [16], policy gradient [17], or behavior cloning [18], all of which miss modeling sequential processes of LNS explicitly. Furthermore, RL-based algorithms may suffer from various issues, such as the need for bootstrapping to propagate returns in TD-learning can cause stability problems, the discounting future rewards can induce undesirable short-sighted behaviors, policy gradient is known to be sample inefficient, and behavior cloning can suffer from cascading errors [19, 20, 21]. To circumvent these disadvantages, we propose to learn a policy with decision transformers, which seeks to benefit from modeling sequential processes of LNS and better generalization.

2.3. Decision Transformer

Decision transformer (DT) [21] abstracts the decision-making process in RL as a sequence modeling problem and attempts to learn a return-conditioned state-action mapping. The return-conditionality means that given a history of return-state-action tokens, such that the first token represents the desired return at the current state, the DT predicts the action required to achieve this desired return. In this paper, we follow the convention of the original DT and define return, g_t, as the non-discounted rewards-to-go: $g_t = \sum_t^T r_t$. DT takes as input a sequence of three-tokens: $(\langle g_{t-K}, s_{t-K}, a_{t-K}\rangle, \cdots, \langle g_t, s_t, a_t\rangle)$, where $K \leq T$ is the context length. Each token is then encoded into an embedding and added by a positional encoding. Let $(\langle z_{g_{t-K}}, z_{s_{t-K}}, z_{a_{t-K}}\rangle, \cdots, \langle z_{g_t}, z_{s_t}, z_{a_t}\rangle)$ be the corresponding sequence of embeddings, and it is fed into a

causal transformer to produce another sequence of embeddings $(\langle z^h_{g_{t-K}}, z^h_{s_{t-K}}, z^h_{a_{t-K}}\rangle, \cdots, \langle z^h_{g_t}, z^h_{s_t}, z^h_{a_t}\rangle)$. A decoder takes as input $z^h_{s_t}$ and outputs $\hat{a}_t$. During training, a suitable loss function is applied to penalize the difference between the prediction $\hat{a}_t$ and label a_t. During inference, after specifying a target return based on desired performance and the environment starting state, DT generates actions autoregressively. The actions are executed and the target return is subtracted by the achieved rewards to obtain the next states. The process of generating actions and applying them to obtain the next return-to-go and state is repeated until episode termination.

3. Solving IPs with Sequence to Contrastive Multi-Label Classification

Instead of learning to select fixed, predefined variable subsets [9], Wu et al. [8] factorizes the combinatorial action space $\mathcal{A}$ into elementary actions on each dimension (i.e. variables), where $a^i_t \in \{1, 0\}$ denotes the elementary action of whether selecting x^i for reoptimization in step t, and a^i_t is 1 if x^i is selected and 0 otherwise. Therefore, any action can be expressed as $a_t = \cup^n_{i=1} a^i_t$ and the action selection problem can be converted into n separated binary classification problems. The policy for action selection is factorized by

$$\pi(a_t|s_t) = \prod_{i=1}^{n} \pi^i(a^i_t|s_t), \tag{1}$$

which expresses the probability of selecting an action as the product of probabilities of selecting its elements. However, such an action space factorization limits the class of policies that can be learned and it also fails to explore the correlations between elementary actions. To address these limitations, *we propose to model the policy learning as a sequence to multi-label classification problem, which jointly models the selection of multiple elementary actions as well as the sequential processes of LNS*, i.e.,

$$\pi(a_t|s_t) = p_\theta(a_t|h_\phi(Q(t, K))), \tag{2}$$

where $Q(t, K)$ denotes the function to return the last K sequence of return-state-action tokens from steps $t - K$ to t, i.e., $(\langle g_{t-K}, s_{t-K}, a_{t-K}\rangle, \cdots, \langle g_t, s_t, \cdot\rangle)$; $h_\phi(\cdot)$ denotes the sequence encoder parameterized by ϕ(NN); and the MLC decoder parameterized by θ(NN) takes as input state embedding $z^h_{s_t}$ produced by h_ϕ and outputs action distribution $p_\theta(a_t|z^h_{s_t})$. Effective implementations of the sequence encoder and MLC decoder are crucial to this work.

3.1. A Novel Model IPGPT for Solving IPs

In this section, we propose a novel model IPGPT for the problem given in equation (2) based on the causal transformer and contrastive learning.

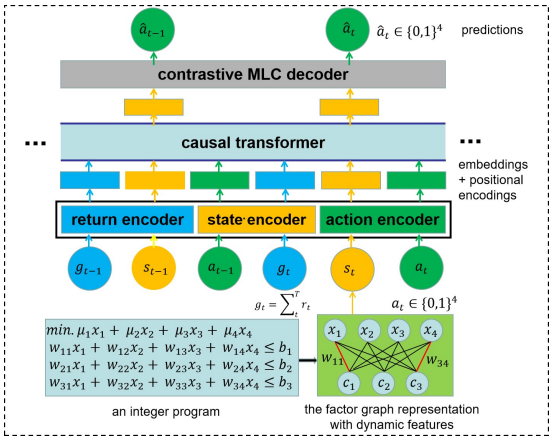


Figure 1: The architecture of our model IPGPT: it consists of several token encoders to produce latent token embeddings, a causal transformer to capture dependence between token embeddings, and a contrastive MLC decoder to exploit correlations between label categories. Here we use a small IP with 4 variables and 3 constraints to show the full pipeline of our model. The problem is first translated into a factor graph G, and G is associated with dynamic factor-node features that describe the states of MDP and are encoded into state embeddings in different steps. Similarly, returns g_t and actions a_t are also encoded into latent embeddings. We use a GCN as state encoder and two simple MLPs as return and action encoders respectively. Each token embedding is further added with its relative positional encoding. The sequence of embeddings is fed into the causal transformer to produce another sequence of embeddings. Finally, the contrastive MLC decoder takes as inputs the state embeddings and outputs action predictions.

3.1.1. Factor Graph Representation

An IP instance can be represented by a *factor graph* [22] which is a bipartitle grah $\mathcal{G} = (\mathcal{V}, \mathcal{C}, \mathcal{E})$ consistring of variable-nodes $\mathcal{V} = \{v_1, \cdots, v_n\}$ and factor-nodes $\mathcal{C} = \{c_1, \cdots, c_m\}$. Variable nodes correspond to the variables and factor nodes correspond to the constraints in the IP. An edge $e_{ij} \in \mathcal{E}$ between v_i and c_j is established only if the j-th constraint contains the i-th variable. The variable nodes are associated with a feature matrix $V \in \mathbb{Z}^{n \times d_v}$, where d_v is the number of features for each variable node. The features of each variable-node v_i include two parts: (1) *static* features: a one-hot vector indicates the node type and the objective coefficient μ_i of x_i; (2) *dynamic* features: the current solution of x_i in step t and the incumbent solution of x_i. Note that

the dynamic features are used to describe the states of MDP in different steps. The factor nodes are also associated with a feature matrix $C \in \mathbb{Z}^{m \times d_c}$, where d_c is the number of features for each factor node. The features of each factor-node c_i only include static features: a one-hot vector indicates the node type and the value b_i at the right-hand-side (RHS) of the i-th constraint. Finally, the weight matrix of edges is exactly the incidence matrix.

3.1.2. Model Architecture

Fig. 1 gives the overall architecture of our IPGPT and it consists of a customized DT encoder and a contrastive MLC decoder. Our encoder is only composed of several customized token encoders and a causal transformer without the linear decoder of DT [21].

Token Encoders: Each token is first encoded into an embedding and added by a positional encoding. For return and action tokens, two simple multilayer perceptrons (MLPs) are used as return and action encoders respectively. Positional encodings are produced by another simple MLP which takes as input a single scalar t. Each state token s_t is represented by a factor graph as introduced in section 3.1.1, and we use a graph convolutional network (GCN) [23] as the state encoder. A single graph convolution layer is detailed below

$$\begin{aligned} C^{(k+1)} &= C^{(k)} + \sigma\left(\mathrm{LN}\left(W V^{(k)} H_v^{(k)}\right)\right), \\ V^{(k+1)} &= V^{(k)} + \sigma\left(\mathrm{LN}\left(W^T C^{(k+1)} H_c^{(k)}\right)\right), \end{aligned} \tag{3}$$

where $H_v^{(k)}, H_c^{(k)} \in \mathbb{R}^{d_h \times d_h}$ are trainable weight matrices in the k-th layer; $V^{(k)} \in \mathbb{R}^{n \times d_h}$ and $C^{(k)} \in \mathbb{R}^{m \times d_h}$ are embeddings for variable-nodes and factor-nodes respectively in the k-th layer; LN and $\sigma(\cdot)$ denote layer normalization and Tanh activation function respectively. The initial embeddings $V^{(0)}$ and $C^{(0)}$ are linear projections of the raw feature matrices V and C respectively. In this paper, all MLP encoders only have two layers, and the embeddings' dimensions d_h are set to be 128; the GCN encoder consists of two convolution layers and a mean pooling layer.

Causal Transformer: Causal transformer [24] is an architecture to efficiently model sequences that consist of stacked self-attention layers with residual connections. In our model, each layer receives a sequence of $L = 3K$ token embeddings $\{z_i\}_{i=1}^{L}$, and outputs L embeddings $\{z_i^h\}_{i=1}^{L}$, preserving the input dimensions. Specifically, each token embedding z_i is mapped to a key z_i^k, a query z_i^q, and a value z_i^v via linear functions, and the output z_i^h is given by

$$z_i^h = \sum_{j=1}^{i} \alpha_{ij} z_j^v, \quad \alpha_{ij} = \frac{\exp(z_i^q \cdot z_j^k)}{\sum_{j'=1}^{i} \exp(z_i^q \cdot z_{j'}^k)}. \tag{4}$$

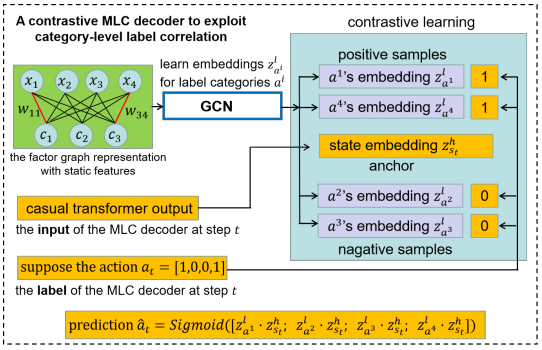


Figure 2: The architecture of our contrastive MLC decoder. Firstly, a GCN takes as input an IP instance and learns embeddings for label categories respectively, and labels within the same category share the same embedding. Secondly, we use the state embedding in step t as an input feature whose inner products with label embeddings are used to produce prediction $\hat{a}_t$. Lastly, a contrastive loss is designed to pull together the feature embedding and positive label embeddings, while separating the feature embedding from the negative label embeddings. Note that a label embedding is positive only if its label is 1. An example with label $[1, 0, 0, 1]$ is shown.

In this work, we adopt the causal transformer GPT2 [25] to learn and reason about sequences and we defer the other architecture details to the original paper.

Contrastive MLC Decoder: Recall that an elementary action $a_t^i \in \{0, 1\}$ (a.k.a. a label) denotes whether or not to select variable x^i for reoptimization in step t. A label vector $a_t = \cup_{i=1}^{n} a_t^i \in \{0, 1\}^n$ denotes the selected action given state s_t. Different from eq. (1) which approximates $\pi(a_t|s_t)$ with n separated binary classification problems, we propose to approximate $\pi(a_t|s_t)$ with an MLC decoder that finds a mapping from $z_{s_t}^h$ to a_t, where $z_{s_t}^h$ is a state embedding generated by the causal transformer and served as an input feature for our MLC decoder. *A key aspect of learning a policy with an MLC module is that we can exploit the correlations between elementary actions, which is missing in those existing ML-boosted IP solvers* [8, 9, 10].

We propose to exploit category-level label correlation with contrastive learning based on the MLC model GMVAE [26]. GMVAE assumes that the number of labels is fixed and label embeddings are shared across all samples. This is not applicable to our case since different instances may have a different number of decision variables, i.e., actions from different instances may have different cardinalities. Alternatively, we learn category-level label embeddings for each IP instance with a shared GCN, and the embeddings are only shared across samples within each instance. Fig. 2 gives the architecture of our MLC decoder. We denote a^i the i-th category of labels $\{a_j^i\}_{j=t-K}^{t}$ collected from steps $[t - K, t]$. Our idea is as follows: (1) we learn an embedding $z_{a^i}^l$ for

each label category a^i such that labels within the same category share the same embedding. Since the number of label categories is exactly the number of variables in an IP instance, we use the GCN described in equations (3) to take as input an IP instance and output node embeddings, and we use the variable-node embeddings as label category embeddings respectively; (2) we use the state embedding $z^h_{s_t}$ of each LNS step as an input feature whose inner products with label embeddings correspond to feature-label similarity and can be used for prediction; (3) we use contrastive learning to capture correlations between label categories by pulling similar categories' embeddings together. Sepcifically, let $I \equiv \{1, \cdots, n\}$ and we define the positive label set of a_t as $P(a_t) \equiv \{i \in I | a^i_t = 1\}$. Given a sequence of return-state-action tokens $(\langle g_{t-K}, s_{t-K}, a_{t-K}\rangle, \cdots, \langle g_t, s_t, a_t\rangle)$, our decoder is designed to optimize the following contrastive loss function:

$$\mathcal{L}_{CL} = \frac{1}{K} \sum_{j=t-K}^{t} \frac{1}{|P(a_j)|} \sum_{i \in P(a_j)} -\log \frac{z^h_{s_j} \cdot z^l_{a^i}}{\sum_{i' \in I} z^h_{s_j} \cdot z^l_{a^{i'}}}, \tag{5}$$

where state embeddings $z^h_{s_j}$ for $j \in [t-K, t]$ are generated by the causal transformer and are computed as in eq. (4).

For example, if in most of the actions a_t, labels a^i_t and a^j_t often appear together (i.e., they both equal 1), contrastive learning will implicitly pull their embeddings together. In other words, if two labels do co-appear often, their label embeddings would become similar. On the other hand, if they never co-occur or only co-appear occasionally, their connections are not significant and our decoder will not optimize for their similarity.

3.2. Training Algorithm

Our model will be trained with supervised learning. Given a set of training IP instances, we first collect a dataset of sequences of return-state-action tokens that are generated by the MDP with some expert policy: $\mathcal{D} = \{(\langle g_{t-K}, s_{t-K}, a_{t-K}\rangle_j, \cdots, \langle g_t, s_t, a_t\rangle_j)\}^N_{j=1}$, where $K \leq T$ is the length of each sequence. For each sequence $q \in \mathcal{D}$, our model will take as input q and generate a set of action predictions $\{\hat{a}_j\}^t_{j=t-K}$, and we will also collect the set of labels from q, $\{a_j\}^t_{j=t-K}$. A supervised cross-entropy loss for each sequence is given by

$$\mathcal{L}_{CE} = \frac{1}{K} \sum_{j=t-K}^{t} \frac{1}{n} \sum_{i=1}^{n} a^i_j \log \hat{a}^i_j + (1-a^i_j) \log(1-\hat{a}^i_j). \tag{6}$$

The final objective function to minimize is given by

$$\mathcal{L} = \mathcal{L}_{CL} - \beta \mathcal{L}_{CE}, \tag{7}$$

where β is a trade-off weight. The model is trained with Adam [27] and optimized with $\mathcal{L}$. We sample mini-batches of sequence length K from the dataset $\mathcal{D}$ and the model is trained with GPUs in parallel. However, similar to DT, our model will generate action predictions autoregressively during testing. We refer the reader to section 2.3 for more details.

4. Experiments

We conduct experiments on four diverse NP-hard COP benchmarks, including minimum vertex cover (MVC), maximum cut (MC), set covering (SC), and combinatorial auction (CA). We follow the experimental settings of [9] and [8].

4.1. Datasets and Experimental Setup

Datasets. MVC and MC are graph optimization problems; SC and CA are general IPs. For MVC, we use the Erdős-Rényi (ER) model [28] to generate random graphs with 1000 nodes and edge probability 0.15. For MC, we use the Barabasi-Albert (BA) model [29] to generate random graphs with 500 nodes and an average degree of 4. For SC, we generate instances with matrices having 5000 rows and 1000 columns following the procedure in [30], where each entry $B_{ij} \in \{0, 1\}$ represents whether the i-th element in the universe belongs to the j-th set. For CA, we use the Combinatorial Auction Test Suit (CATS) [31] with arbitrary relationships to generate instances with 2000 items and 4000 bids. For each problem type, we generate 100, 20, and 50 instances for training, validation, and testing, respectively. For each training instance, we use LNS with a random policy to run on it for 100 steps and set a time limit of 2s for solving the sub-IP in each step with Gurobi. We run it 30 times and collect the trajectory with the best objective values for training IPGPT, and we randomly sample 20 sequences of return-state-action tokens with length $K = 25$ from each trajectory. Therefore, our dataset for training IPGPT includes 2000 sequences of three tokens.

Initialization. LNS starts with a feasible initial solution. For MVC, MAXCUT, SC, and CATS, we initialize a feasible solution by including all vertices in the cover set, randomly partitioning all vertices into two complementary sets, including all sets in the set cover, and accepting no bids, respectively. The initialization process does not incur additional computational costs in our experiments.

Implementation and Hyperparameters. Return, action, and position encoders are all simple MLPs with 2 layers and 128 hidden neurons. The state encoder is a GCN with 2 convolution layers and a mean pooling layer. We use the GPT2 as our casual transformer. Dimensions of all hidden embeddings are set to 128. We set the batch

Methods	MVC-1000		MC-500		SC-1000		CA-2000	
	Obj.±Std.%	Gap%	Obj.±Std.%	Gap%	Obj.±Std.%	Gap%	Obj.±Std.%	Gap%
Gurobi	482.2 ± 0.8	5.45	-863.9 ± 3.8	3.66	554.9 ± 8.3	3.10	-111668 ± 2.0	2.50
FT-LNS	470.0 ± 0.4	2.80	-866.2 ± 1.7	3.41	564.1 ± 8.4	4.81	-110041 ± 1.6	3.92
RL-LNS	469.0 ± 0.5	2.57	-878.0 ± 1.6	2.10	551.9 ± 8.3	2.55	-111787 ± 2.6	2.40
LB-SRMRL	472.4 ± 0.7	3.30	-859.1 ± 2.3	4.20	560.9 ± 7.3	4.22	-110741 ± 3.1	3.31
IPGPT	$\mathbf{457.1 \pm 0.8}$	**0**	$\mathbf{-896.8 \pm 1.4}$	**0**	$\mathbf{538.2 \pm 5.3}$	0	$\mathbf{-114535 \pm 1.8}$	**0**

Table 1
A comparison of IPGPT and the state-of-the-art baselines on 4 diverse benchmarks. The time limit is set to 200s. Each result is averaged over 5 runs. The gap is the ratio of objective difference w.r.t. the best result. The best results are shown in **bold**.

Methods	MVC-2000		MC-1000		SC-2000		CA-4000	
	Obj.±Std.%	Gap%	Obj.±Std.%	Gap%	Obj.±Std.%	Gap%	Obj.±Std.%	Gap%
Gurobi	392.5 ± 1.3	7.50	-1784.7 ± 1.0	4.23	295.7 ± 7.9	1.86	-212890 ± 1.8	4.90
FT-LNS	390.5 ± 1.1	6.96	-1767.8 ± 1.0	5.14	303.3 ± 8.0	4.48	-211324 ± 2.1	5.60
RL-LNS	375.8 ± 2.1	2.93	-1831.0 ± 0.9	1.74	295.4 ± 7.8	1.76	-216650 ± 1.7	3.23
LB-SRMRL	395.2 ± 1.9	8.24	-1765.6 ± 1.5	5.25	301.4 ± 7.2	3.82	-209420 ± 2.1	6.45
IPGPT	$\mathbf{365.1 \pm 1.1}$	**0**	$\mathbf{-1863.5 \pm 0.8}$	**0**	$\mathbf{290.3 \pm 7.1}$	**0**	$\mathbf{-223870 \pm 1.7}$	**0**

Table 2
Generalization to larger instances with a double number of variables. The time limit is set to 500s.

size and the number of training epochs to 128 and 100, respectively, for all experiments. Our model was implemented with the Pytorch deep learning framework and the whole model was trained using the Adam optimizer [27] with a learning rate of 0.0001 and a weight decay ratio of 0.01 in an end-to-end fashion. All experiments were carried out on a machine with a 4.2 GHz quad-core Intel i7 CPU, 16 GB RAM, and an Nvidia RTX 3090 24GB GPU card. Further implementation details can be found in the appendix.

Baselines. We compare our method with four baselines: (1) Gurobi (version 9.5) with default settings: a leading state-of-the-art IP solver; (2) FT-LNS: the best-performing LNS version by [9], which applies imitation learning to mimic the best demonstrations; (3) RL-LNS: the current state-of-the-art learning-based LNS method for solving IPs [8], which uses deep RL to learn LNS policy via action factorization to represent all potential variable subsets; (4) LB-SRMRL: the best-performing LB version by [32], which uses a regression model and RL to learn a hybrid model to predict and adapt the neighborhood size for the LB heuristic. We follow the default settings of these learning-based baselines and further fine-tune them on our datasets to get the best hyperparameters. For more details of the settings of these baselines, we refer the reader to their original papers.

Evaluation Metrics. The performances of different algorithms are compared in two measures: (1) the objective of solutions returned by different algorithms within a time limit; (2) the gap between solutions, namely, the ratio of objective difference w.r.t. the best result.

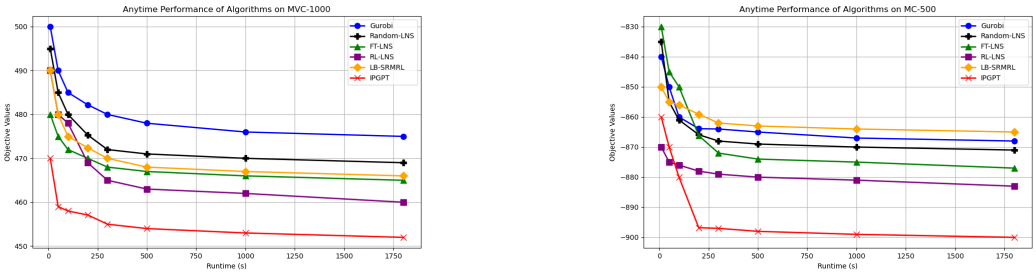
4.2. Experimental Results

A comparison of IPGPT and other state-of-the-art baselines on 4 diverse benchmarks is given in Table 6. All learning-based algorithms, including our IPGPT, call Gurobi to solve sub-IPs with a time limit of 2s at every step. We can observe that LB-SRMRL is not comparable to other algorithms. RL-LNS remains the most competitive baseline and consistently outperforms both Gurobi and FT-LNS. Our IPGPT consistently outperforms RL-LNS and Gurobi on all benchmarks by averaged ratios of 2.41% and 3.68%, respectively. Overall, these results suggest that our approach can reliably offer substantial improvements over state-of-the-art solvers.

We also compare the generalization ability of all algorithms to solve large IPs. To this end, we generate two sets of testing instances following the same settings as in section 4.1 but double and quadruple the number of variables respectively. Note that we only generate 50 testing instances for each problem type without considering training and validation. We test all (trained) models on these new instances and summarize results in Tables 7 and 3. We can observe that the advantage of our IPGPT still preserves on larger problem instances

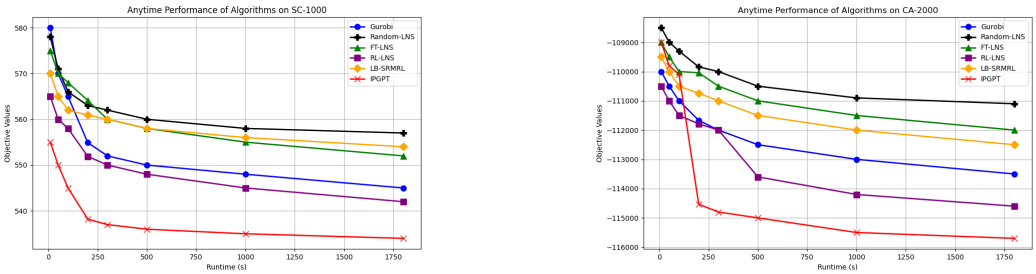
Methods	MVC-4000		MC-2000		SC-4000		CA-8000	
	Obj.±Std.%	Gap%	Obj.±Std.%	Gap%	Obj.±Std.%	Gap%	Obj.±Std.%	Gap%
Gurobi	278.3 ± 0.9	3.34	-3574.4 ± 0.8	3.42	175.4 ± 7.0	1.74	-422291 ± 1.2	3.61
FT-LNS	279.2 ± 1.7	3.68	-3526.2 ± 0.8	4.72	175.2 ± 6.6	1.62	-431234 ± 0.9	1.57
RL-LNS	273.6 ± 2.1	1.60	-3612.5 ± 0.7	2.39	$\mathbf{172.4 \pm 7.1}$	**0**	-432980 ± 0.7	1.17
LB-SRMRL	275.6 ± 2.2	2.34	-3505.1 ± 0.9	5.29	177.1 ± 7.2	2.73	-415631 ± 0.5	5.13
IPGPT	$\mathbf{269.3 \pm 1.9}$	**0**	$\mathbf{-3700.9 \pm 1.0}$	**0**	173.6 ± 6.1	0.70	$\mathbf{-438091 \pm 0.6}$	**0**

Table 3
Generalization to larger instances with a quadruple number of variables. The time limit is set to 500s.



(a) MVC-1000

(b) MC-500



(c) SC-1000

(d) CA-2000

Figure 3: Anytime Performance Comparison of Gurobi, FT-LNS, RL-LNS, LB-SRMRL, and IPGPT on Four IP benchmarks. Runtimes are up to 30 minutes.

compared to baselines. Specifically, Table 7 shows that IPGPT still consistently outperforms all baselines on the 4 benchmarks when the instance size is doubled; IPGPT outperforms RL-LNS and Gurobi by averaged ratios of 2.06% and 4.56% respectively. On the other hand, Table 3 shows that IPGPT outperforms all baselines on 3 out of 4 benchmarks when the instance size is quadrupled; IPGPT outperforms RL-LNS and Gurobi by averaged ratios of 1.12% and 3.03% respectively. In summary, our IPGPT learned on small instances generalizes well to larger instances, with a persistent advantage over other methods.

4.3. Anytime Performance

We further showcase the anytime performance of various algorithms, including random LNS in this experiment, to facilitate an easier comparison between random LNS and IPGPT across four benchmarks, as illustrated in Figure 3. Our observations indicate that: (1) IPGPT significantly outperforms other baselines with a noteworthy margin. (2) Even with extended time limits, IPGPT's advantage persists. (3) Interestingly, even though IPGPT is trained on trajectories derived from random LNS, it can generate remarkably superior trajectories during testing, leading to substantially improved performance compared to random LNS. This mirrors the findings in [21], where the Decision Transformer (DT) can generate optimal trajectories during test time, even when trained on random walk data for the shortest pathfinding problem.

4.4. Additional Experiments with SCIP

Our framework can integrate any ILP solver to enhance incumbent solutions. We primarily conducted experiments with Gurobi, given its status as a leading ILP solver. Additionally, we also present results utilizing SCIP (v6.0.1) as an alternative ILP solver. By employing the same settings as detailed in Section 5.1 and applying them to the four benchmarks, we display the results in Table 6. These outcomes align with those observed when using Gurobi as the ILP solver, albeit with SCIP exhibiting a notably lower performance compared to Gurobi.

4.5. Testing on Real-World Instances in MIPLIB

We follow the experimental settings for real-world instances in MIPLIB as described in [8]. We exclude "easy" instances with relatively small sizes, as well as instances where Gurobi cannot find any feasible solutions within a 3600-second time limit. Consequently, we choose 35 representative "hard" or "open" instances containing only integer variables. Within these instances, the number of variables ranges from 150 to 393,800 (averaging 49,563),

Methods	MVC-1000		MC-500		SC-1000		CA-2000	
	Obj.±Std.%	Gap%	Obj.±Std.%	Gap%	Obj.±Std.%	Gap%	Obj.±Std.%	Gap%
SCIP	552.2 ± 0.5	18.98	-793.9 ± 2.8	9.39	604.3 ± 3.2	9.40	-100514 ± 2.1	10.75
FT-LNS	490.2 ± 0.6	5.62	-836.3 ± 1.2	4.55	585.2 ± 6.4	5.96	-107141 ± 1.7	4.87
RL-LNS	480.0 ± 0.5	3.43	-847.5 ± 1.3	3.27	575.6 ± 6.2	4.22	-108787 ± 2.2	3.41
LB-SRMRL	492.4 ± 0.9	6.10	-820.3 ± 1.9	6.38	580.8 ± 5.3	5.16	-107741 ± 3.0	4.34
IPGPT	$\mathbf{464.1 \pm 0.7}$	**0**	$\mathbf{-876.2 \pm 1.2}$	**0**	$\mathbf{552.3 \pm 5.3}$	0	$\mathbf{-112626 \pm 1.5}$	**0**

Table 4
Results with SCIP. The time limit is set to 200s. Each result is averaged over 5 runs. The gap is the ratio of objective difference w.r.t. the best result. The best results are shown in **bold**.

and the number of constraints varies from 301 to 850,513 (averaging 96,778). We use the datasets in section 4.1 to train our model and evaluate our model (with Gurobi as the repair solver) on this realistic dataset, in the style of *active search* [33, 8, 34] on each instance. Our findings indicate that, with a 3600-second time limit, IPGPT surpasses both solvers on 20 of the 35 instances and exhibits comparable performance on 10 of the 35 instances. More details are provided in the appendix.

5. Conclusion

Addressing large-scale IP problems presents a formidable challenge. An increasingly prevalent approach for the automated design and tuning of IP solvers leverages data-driven methodologies. This paper concentrates on enhancing learning-based LNS approaches, given their ability to conveniently utilize any existing solver as a subroutine. We introduce IPGPT, a novel approach that models policy learning as a sequence to an MLC problem. It seamlessly integrates a customized decision transformer encoder, encompassing a causal transformer, to model the sequential processes of LNS, and an MLC decoder with contrastive learning to exploit correlations in variable selection. Furthermore, we carry out comprehensive experiments on four diverse benchmarks and real-world instances. The results suggest that our IPGPT approach consistently delivers substantial improvements over state-of-the-art solvers and exhibits excellent generalization capabilities for larger instances.

Acknowledgments

The work of Caihua Liu is supported and funded by the Humanities and Social Sciences Youth Foundation, Ministry of Education of the People's Republic of China (Grant No.21YJC870009).

References

[1] J. Mula, R. Poler, J. P. García-Sabater, F. C. Lario, Models for production planning under uncertainty: A review, International journal of production economics 103 (2006) 271–285.

[2] W.-Y. Ku, J. C. Beck, Mixed integer programming models for job shop scheduling: A computational analysis, Computers & Operations Research 73 (2016) 165–173.

[3] D. Chen, Y. Bai, S. Ament, W. Zhao, D. Guevarra, L. Zhou, B. Selman, R. B. van Dover, J. M. Gregoire, C. P. Gomes, Automating crystal-structure phase mapping by combining deep learning with constraint reasoning, Nature Machine Intelligence 3 (2021) 812–822.

[4] S. Gollowitzer, I. Ljubić, Mip models for connected facility location: A theoretical and computational study, Computers & Operations Research 38 (2011) 435–449.

[5] R. M. Karp, Reducibility among combinatorial problems, in: Complexity of computer computations, Springer, 1972, pp. 85–103.

[6] H. A. Taha, Integer programming: theory, applications, and computations, Academic Press, 2014.

[7] Y. Bengio, A. Lodi, A. Prouvost, Machine learning for combinatorial optimization: a methodological tour d'horizon, European Journal of Operational Research 290 (2021) 405–421.

[8] Y. Wu, W. Song, Z. Cao, J. Zhang, Learning large neighborhood search policy for integer programming, Advances in Neural Information Processing Systems 34 (2021) 30075–30087.

[9] J. Song, Y. Yue, B. Dilkina, et al., A general large neighborhood search framework for solving integer linear programs, Advances in Neural Information Processing Systems 33 (2020) 20012–20023.

[10] N. Sonnerat, P. Wang, I. Ktena, S. Bartunov, V. Nair, Learning a large neighborhood search algorithm for mixed integer programs, arXiv preprint arXiv:2107.10201 (2021).

[11] V. Nair, M. Alizadeh, et al., Neural large neighborhood search, in: Learning Meets Combinatorial Algorithms at NeurIPS2020, 2020.

[12] D. Pisinger, S. Ropke, Large neighborhood search, in: Handbook of metaheuristics, Springer, 2010, pp. 399–419.

[13] S. Ropke, D. Pisinger, An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows, Transportation science 40 (2006) 455–472.

[14] L. Perron, P. Shaw, V. Furnon, Propagation guided large neighborhood search, in: International Conference on Principles and Practice of Constraint Programming, Springer, 2004, pp. 468–481.

[15] D. Dumez, F. Lehuédé, O. Péton, A large neighborhood search approach to the vehicle routing problem with delivery options, Transportation Research Part B: Methodological 144 (2021) 103–132.

[16] R. S. Sutton, A. G. Barto, Reinforcement learning: An introduction, MIT press, 2018.

[17] R. J. Williams, Simple statistical gradient-following algorithms for connectionist reinforcement learning, Machine learning 8 (1992) 229–256.

[18] F. Torabi, G. Warnell, P. Stone, Behavioral cloning from observation, in: Proceedings of the 27th International Joint Conference on Artificial Intelligence, 2018, pp. 4950–4957.

[19] L. P. Kaelbling, M. L. Littman, A. W. Moore, Reinforcement learning: A survey, Journal of artificial intelligence research 4 (1996) 237–285.

[20] S. Ross, D. Bagnell, Efficient reductions for imitation learning, in: Proceedings of the thirteenth international conference on artificial intelligence and statistics, JMLR Workshop and Conference Proceedings, 2010, pp. 661–668.

[21] L. Chen, K. Lu, A. Rajeswaran, K. Lee, A. Grover, M. Laskin, P. Abbeel, A. Srinivas, I. Mordatch, Decision transformer: Reinforcement learning via sequence modeling, Advances in neural information processing systems 34 (2021) 15084–15097.

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

[23] S. Zhang, H. Tong, J. Xu, R. Maciejewski, Graph convolutional networks: a comprehensive review, Computational Social Networks 6 (2019) 1–23.

[24] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, I. Polosukhin, Attention is all you need, Advances in neural information processing systems 30 (2017).

[25] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, I. Sutskever, et al., Language models are unsupervised multitask learners, OpenAI blog 1 (2019) 9.

[26] J. Bai, S. Kong, C. P. Gomes, Gaussian mixture variational autoencoder with contrastive learning for multi-label classification, in: International Conference on Machine Learning, PMLR, 2022, pp. 1383–1398.

[27] D. P. Kingma, J. Ba, Adam: A method for stochastic optimization, arXiv preprint arXiv:1412.6980 (2014).

[28] P. Erdős, A. Rényi, et al., On the evolution of random graphs, Publ. Math. Inst. Hung. Acad. Sci 5 (1960) 17–60.

[29] R. Albert, A.-L. Barabási, Statistical mechanics of complex networks, Reviews of modern physics 74 (2002) 47.

[30] E. Balas, A. Ho, Set covering algorithms using cutting planes, heuristics, and subgradient optimization: a computational study, Springer, 1980.

[31] K. Leyton-Brown, M. Pearson, Y. Shoham, Towards a universal test suite for combinatorial auction algorithms, in: Proceedings of the 2nd ACM conference on Electronic commerce, 2000, pp. 66–76.

[32] D. Liu, M. Fischetti, A. Lodi, Learning to search in local branching, in: Proceedings of the AAAI Conference on Artificial Intelligence, volume 36, 2022, pp. 3796–3803.

[33] I. Bello, H. Pham, Q. V. Le, M. Norouzi, S. Bengio, Neural combinatorial optimization with reinforcement learning, arXiv preprint arXiv:1611.09940 (2016).

[34] E. Khalil, H. Dai, Y. Zhang, B. Dilkina, L. Song, Learning combinatorial optimization algorithms over graphs, Advances in neural information processing systems 30 (2017).

[35] E. Larsen, S. Lachapelle, Y. Bengio, E. Frejinger, S. Lacoste-Julien, A. Lodi, Predicting solution summaries to integer linear programs under imperfect information with machine learning, arXiv preprint arXiv:1807.11876 (2018).

[36] V. Nair, S. Bartunov, F. Gimeno, I. von Glehn, P. Lichocki, I. Lobov, B. O'Donoghue, N. Sonnerat, C. Tjandraatmadja, P. Wang, et al., Solving mixed integer programs using neural networks, arXiv preprint arXiv:2012.13349 (2020).

[37] C. K. Joshi, T. Laurent, X. Bresson, An efficient graph convolutional network technique for the travelling salesman problem, arXiv preprint arXiv:1906.01227 (2019).

[38] Y. Deng, S. Kong, C. Liu, B. An, Deep attentive belief propagation: Integrating reasoning and learning for solving constraint optimization problems, arXiv preprint arXiv:2209.12000 (2022).

[39] Y. Razeghi, K. Kask, Y. Lu, P. Baldi, S. Agarwal, R. Dechter, Deep bucket elimination., in: IJCAI, 2021, pp. 4235–4242.

[40] Y. Deng, S. Kong, B. An, Pretrained cost model for distributed constraint optimization problems, in: Proceedings of the AAAI Conference on Artificial Intelligence, volume 36, 2022, pp. 9331–9340.

[41] S. Hochreiter, J. Schmidhuber, Long short-term memory, Neural computation 9 (1997) 1735–1780.

[42] I. Sutskever, O. Vinyals, Q. V. Le, Sequence to sequence learning with neural networks, Advances in neural information processing systems 27 (2014).

[43] J. Devlin, M.-W. Chang, K. Lee, K. Toutanova, Bert: Pre-training of deep bidirectional transformers for language understanding, arXiv preprint

arXiv:1810.04805 (2018).
[44] Q. Li, H. Peng, J. Li, C. Xia, R. Yang, L. Sun, P. S. Yu, L. He, A survey on text classification: From traditional to deep learning, ACM Transactions on Intelligent Systems and Technology (TIST) 13 (2022) 1–41.
[45] J. Bai, Y. Du, Y. Wang, S. Kong, J. Gregoire, C. P. Gomes, Xtal2dos: Attention-based crystal to sequence learning for density of states prediction, in: NeurIPS 2022 AI for Science: Progress and Promises, 2022.
[46] J. Read, B. Pfahringer, G. Holmes, E. Frank, Classifier chains for multi-label classification, Machine learning 85 (2011) 333–359.
[47] M.-L. Zhang, Z.-H. Zhou, Ml-knn: A lazy learning approach to multi-label learning, Pattern recognition 40 (2007) 2038–2048.
[48] K. Bhatia, H. Jain, P. Kar, M. Varma, P. Jain, Sparse local embeddings for extreme multi-label classification, Advances in neural information processing systems 28 (2015).
[49] C.-K. Yeh, W.-C. Wu, W.-J. Ko, Y.-C. F. Wang, Learning deep latent space for multi-label classification, in: Thirty-first AAAI conference on artificial intelligence, 2017.
[50] J. Bai, S. Kong, C. Gomes, Disentangled variational autoencoder based multi-label classification with covariance-aware multivariate probit model, arXiv preprint arXiv:2007.06126 (2020).
[51] W. Zhao, S. Kong, J. Bai, D. Fink, C. Gomes, Hotvae: Learning high-order label correlation for multi-label classification via attention-based variational autoencoders, in: Proceedings of the AAAI Conference on Artificial Intelligence, volume 35, 2021, pp. 15016–15024.
[52] T. Berthold, Measuring the impact of primal heuristics, Operations Research Letters 41 (2013) 611–614.
[53] T. Achterberg, T. Berthold, G. Hendel, Rounding and propagation heuristics for mixed integer programming, in: Operations Research Proceedings 2011: Selected Papers of the International Conference on Operations Research (OR 2011), August 30-September 2, 2011, Zurich, Switzerland, Springer, 2012, pp. 71–76.
[54] E. Danna, E. Rothberg, C. L. Pape, Exploring relaxation induced neighborhoods to improve mip solutions, Mathematical Programming 102 (2005) 71–90.
[55] E. Rothberg, An evolutionary algorithm for polishing mixed integer programming solutions, INFORMS Journal on Computing 19 (2007) 534–541.
[56] G. Hendel, Adaptive large neighborhood search for mixed integer programming, Mathematical Programming Computation (2022) 1–37.

A. Ablation Study

To demonstrate the strength of IPGPT, we compare it with two variants: (1) IPGPT⊖MLC: a modified IPGPT where its MLC decoder is replaced with a linear decoder; (2)IPGPT⊖DT: a modified IPGPT where its casual transformer component is removed. The results are summarized in Table 5. We can observe that IPGPT outperforms the two variants consistently; our model's performance drop significantly if we do not consider modeling the sequential process of LNS (drop by 7.69% on average) or exploit correlations of variable selection (drop by 3.08% on average).

B. Additional Experiments with SCIP

Our framework can integrate any ILP solver to enhance incumbent solutions. We primarily conducted experiments with Gurobi, given its status as a leading ILP solver. Additionally, we also present results utilizing SCIP (v6.0.1) as an alternative ILP solver. By employing the same settings as detailed in Section 5.1 and applying them to the four benchmarks, we display the results in Table 6. These outcomes align with those observed when using Gurobi as the ILP solver, albeit with SCIP exhibiting a notably lower performance compared to Gurobi.

C. Testing on Real-World Instances in MIPLIB

We follow the experimental settings for real-world instances in MIPLIB as described in [8]. We exclude "easy" instances with relatively small sizes, as well as instances where Gurobi cannot find any feasible solutions within a 3600-second time limit. Consequently, we choose 35 representative "hard" or "open" instances containing only integer variables. Within these instances, the number of variables ranges from 150 to 393,800 (averaging 49,563), and the number of constraints varies from 301 to 850,513 (averaging 96,778). For each instance, we employ Gurobi to generate trajectories and train IPGPT using these trajectories. The trained IPGPT is then evaluated on the 35 instances, with expected returns set to be 10% lower than the best results returned by Gurobi and RL-LNS. The results are shown in Table 7. Our findings indicate that, with a 3600-second time limit, IPGPT surpasses both solvers on 20 of the 35 instances and exhibits comparable performance on 10 of the 35 instances.

Methods	MVC-1000		MC-500		SC-1000	
	Obj.±Std.%	Gap%	Obj.±Std.%	Gap%	Obj.±Std.%	Gap%
IPGPT⊖MLC	465.5 ± 0.6	1.84	-872.0 ± 1.5	2.77	550.2 ± 8.3	2.23
IPGPT⊖DT	472.6 ± 1.1	3.39	-879.1 ± 0.8	1.97	546.9 ± 6.3	1.62
IPGPT	$\mathbf{457.1 \pm 0.8}$	**0**	$\mathbf{-896.8 \pm 1.4}$	**0**	$\mathbf{538.2 \pm 5.3}$	0

Table 5
An ablation study on the casual transformer and MLC decoder components of IPGPT. Note that the experimental settings here follow that of Table 6.

Methods	MVC-1000		MC-500		SC-1000		CA-2000	
	Obj.±Std.%	Gap%	Obj.±Std.%	Gap%	Obj.±Std.%	Gap%	Obj.±Std.%	Gap%
SCIP	552.2 ± 0.5	18.98	-793.9 ± 2.8	9.39	604.3 ± 3.2	9.40	-100514 ± 2.1	10.75
FT-LNS	490.2 ± 0.6	5.62	-836.3 ± 1.2	4.55	585.2 ± 6.4	5.96	-107141 ± 1.7	4.87
RL-LNS	480.0 ± 0.5	3.43	-847.5 ± 1.3	3.27	575.6 ± 6.2	4.22	-108787 ± 2.2	3.41
LB-SRMRL	492.4 ± 0.9	6.10	-820.3 ± 1.9	6.38	580.8 ± 5.3	5.16	-107741 ± 3.0	4.34
IPGPT	$\mathbf{464.1 \pm 0.7}$	**0**	$\mathbf{-876.2 \pm 1.2}$	**0**	$\mathbf{552.3 \pm 5.3}$	0	$\mathbf{-112626 \pm 1.5}$	**0**

Table 6
Results with SCIP. The time limit is set to 200s. Each result is averaged over 5 runs. The gap is the ratio of objective difference w.r.t. the best result. The best results are shown in **bold**.

Instance	Gurobi	RL-LNS	IPGPT
a2864-99blp	-72	-72	**-85**
bab3	-655388.1120	-654912.9204	**-655412.3501**
bley-xs1noM	**3938322.37**	3975481.35	3958310.21
cdc7-4-3-2	-257	-276	**-280**
comp12-2idx	380	363	**352**
ds	**177**	319	189
ex1010-pi	239	238	238
graph20-80-1rand	-6	-6	-6
graph40-20-1rand	**-15**	-14	-14
neos-3426085-ticino	226	226	226
neos-3594536-henty	401948	402426	**401896**
neos-3682128-sandon	34666770.0	**34666765.12338**	34666770
ns1828997	133	128	**98**
nursesched-medium-hint03	115	131	115
opm2-z12-s8	-33269	-53379	**-55269**
pb-grow22	-46217.0	-46881.0	**-56782**
proteindesign121hz512p9	1499	1489	**1481**
queens-30	-39	-39	-39
ramos3	245	248	**216**
rmine13	-3493.781904	-3487.807859	**-3493.79821**
rmine15	-1979.559046	-5001.279118	**-5002.129874**
rococoC12-010001	34673	**35443**	35467
s1234	29	41	29
scpj4scip	133	134	**131**
scpk4	330	329	**325**
scpl4	279	281	**269**
sorrell3	-16	-16	-16
sorrell4	-23	-23	-23
sorrell7	-187	-188	**-190**
supportcase2	397	397	397
t1717	201342	195894	**185241**
t1722	117171	117978	**115983**
tokyometro	8479.5	8582.70	**8456.7**
v150d30-2hopcds	41	41	41
z26	-1083	-1171	**-1176**

Table 7
Results on MIPLIB. The best results are shown in **bold**. The time limit is set to 3600s.

D. Other Related Work

Learning to Optimize. Recently, there has been an increasing interest in applying ML to learn solving COPs. Broadly speaking, there are three categories of learning to optimize algorithms: **(1)** Learning to predict solutions from inputs. Larsen et al. [35] train a deep neural network (DNN) to predict the solution of a stochastic load planning problem. Nair et al. [36] propose neural diving to learn a DNN to generate multiple partial assignments for its integer variables, and the resulting smaller mixed integer programs (MIPs) for un-assigned variables are solved with an off-the-shelf MIP solver to construct high-quality joint assignments. Joshi et al. [37] learn a DNN by supervision to predict the probability of an edge to be in the traveling salesman problem (TSP) tour. A feasible tour is then generated by beam search. **(2)** Learning to configure COP algorithms. Liu et al. [32] learn to configure the search neighborhood size of LB in each step by using RL. Deng et al. [38] integrate belief propagation (BP), gated recurrent units (GRUs), and graph attention networks (GATs) within the message-passing framework to reason about dynamic weights and damping factors for composing new BP messages. **(3)** Learning alongside COP algorithms. Nair et al. [11] learn a DNN to make variable selection decisions in B&B to bound the objective value gap with a small tree. Deep Bucket Elimination (DBE) [39] uses DNNs to approximate the large bucket functions. Deng et al. [40] propose a pretrained cost model which predicts the optimal cost of a given partially instantiated COP. The predicted cost is then used to construct heuristics for various COP algorithms such as LNS and B&B. Our work belongs to the third category.

Sequence Learning. The recent rapid development of sequence modeling is largely due to the successful applications of DNNs, from LSTMs [41] and sequence-to-sequence models [42] to transformer architectures with self-attention [24]. Sequence learning aims to capture the temporal dependence of sequential data (text, speech, video, etc.), and it is widely used in NLP [43, 44]. There is also a recent attempt to apply sequence learning in scientific discovery such as using a causal transformer to model material property with sequential structures [45]. In light of this, it is also tempting to consider how such sequence models can lead to improved performance in LNS, which is also concerned with sequential processes. A causal transformer is an architecture to efficiently model sequences, which is the cornerstone of a decision transformer in RL [21]. However, little has been done to model the sequential processes of LNS with transformers; we will address this limitation by adopting a causal transformer (GPT2).

Multi-Label Classification. Multi-label classification (MLC) is a prediction task where each sample can have more than one labels. Unlike the single-label scenario, label correlations are prevalent in MLC. Early works capture the correlations through classifier chains [46], Bayesian inference [47], and dimensionality reduction [48]. Thanks to the huge capacity of DNNs, one can alleviate the laborious feature mapping and therefore focus on the loss function, feature-label and label-label correlation modeling [49, 50, 51]. It has been shown that contrastive learning can exploit label information effectively in a data-driven manner, and learn meaningful feature and label embeddings that capture the label correlations and enhance the predictive power [26]. However, current LNS algorithms fail to explore the correlations of variable selection in each step. In this work, we will formulate the policy learning of variable selection as an MLC problem and adopt contrastive learning to model category-level label correlations. To the best of our knowledge, we are the first to use sequence to multi-label learning to improve the performance of LNS, and thus enable us to benefit from modeling long sequences of LNS behaviors and exploiting correlations between variable selection simultaneously.

Primal Heuristics. Numerous primal heuristic algorithms have been proposed to enhance the efficiency of solving ILPs [52]. Primal heuristics span from simpler rounding heuristics [53] to more computationally demanding diving and large neighborhood search (LNS) heuristics, such as Relaxation Induced Neighborhood Search (RINS) [54]. LNS heuristics are improvement heuristics that solve auxiliary problems using the branch-and-bound technique.

RINS, a prominent LNS meta-heuristic, seeks to improve a given feasible MIP solution. By comparing the feasible solution with one obtained from relaxing integer variables, it identifies and eliminates variables with differing values between the two. The resulting sub-MIP is then solved using a MIP solver.

The solution-polishing heuristic [55] employs a variable-fixing neighborhood similar to RINS, while also integrating an evolutionary algorithm approach. Unlike RINS, this heuristic uses crossover and mutation operations to combine multiple solutions chosen from a pool of available feasible solutions. The crossover process fixes variables that have identical values across all selected solutions, while mutation is introduced by randomly fixing additional variables to refine already high-quality solutions.

Adaptive LNS [56] capitalizes on an ensemble of LNS algorithms for MIPs and employs a multi-armed bandit to adaptively switch among them during a MIP solve. Although our work does not focus on an ensemble approach, it could be incorporated as another ensemble member to enhance performance.

In contrast, learning-based LNS approaches, such as IPGPT, can be regarded as primal heuristics automatically learned through machine learning. These approaches

showcase significant potential by exploiting data-driven techniques, which ultimately result in improved performance and adaptability across a wide range of problem instances. This work is particularly interested in advancing the capabilities of learning-based LNS approaches.

E. Further Implementation Details

We implemented our GPT-2 using the Huggingface Transformers library. The hyperparameters we employed are as follows: (1) Number of layers: 3; (2) Number of attention heads: 1; (3) Embedding dimension: 128; (4) Nonlinearity function: ReLU; (5) Batch size: 128; (6) Context length K: 25; (7) Dropout ratio: 0.1; (8) Learning rate: 1e-4; (9) Gradient norm clipping: 0.25. We maintained other parameters at their default values. We trained the model from scratch and did not utilize any pre-trained weights.

Grid search is adopted for tuning. We tune learning rate from 0.00005 to 0.002 with interval 0.00005, dropout ratio from $[0.05, 0.1, 0.3, 0.5]$, weight decay from $[0, 0.01, 0.0001]$, β from $[0.1, 0.5, 1, 1.5, 2.0]$, token embedding size from $[64, 128, 256, 512]$, context length K from $[15, 20, 25, 30]$, batch size from $[64, 128, 256]$, Gradient norm clipping from $[0.15, 0.2, 0.25, 0.3, 0.35]$.

In the data collection process, we utilize random LNS with adaptive neighborhood size. The neighborhood size is initially set to 10% of the number of variables in the input problem instance. It is then adapted following the approach described in the paper by [10].

During testing, at each step t, the model generates action distributions $a_{t,i}$ for each dimension i autoregressively. We apply a threshold of 0.5 to convert these values into 1 or 0, representing the selection or non-selection of the corresponding variable x_i in step t.