Light-EvoOPT: A Lightweight Evolutionary Optimization Framework for Ultralarge-Scale Mixed Integer Linear Programs

Huigen Ye, *Graduate Student Member, IEEE*, Hua Xu, *Member, IEEE*,
and Carlos A. Coello Coello, *Fellow, IEEE*

***Abstract*—Machine-learning-(ML)-based optimization frameworks emerge as a promising technique for solving large-scale mixed integer linear programs (MILPs), as they can capture the mapping between problem structures and optimal solutions to expedite their solution process. However, existing solution frameworks often suffer from high model computation costs, incomplete problem reduction, and reliance on large-scale solvers, leading to performance bottlenecks in ultralarge-scale problems with complex constraints. To address these issues, this article proposes Light-EvoOPT, a Lightweight Evolutionary Optimization Framework for Ultralarge-Scale MILPs, which can be divided into four stages: 1) problem Formulation for problem division to reduce model computational costs; 2) model-based initial solution prediction for predicting and constructing the initial solution using a small-scale training dataset; 3) problem reduction for both variable and constraint reduction; and 4) evolutionary optimization for current solution improvement employing a lightweight optimizer. Experiments on four benchmark datasets with tens of millions of variables and constraints and a real-world problem show that the proposed framework based on the sole use of a lightweight optimizer, trained on only one-thousandth of the scale of ultralarge-scale problems, is able to outperform state-of-the-art ML-based frameworks and advanced solvers (e.g., Gurobi) within a specified computational time, validating the feasibility and effectiveness of our proposed ML-based evolutionary optimization framework for ultralarge-scale MILPs.**

***Index Terms*—Evolutionary optimization, graph neural network (GNN), mixed integer program, ultralarge-scale.**

I. Introduction

MIXED Integer Linear Programs (MILPs) entail tackling linear optimization problems wherein certain or all the decision variables are subject to integer constraints [1]. Real-world applications often involve solving ultralarge-scale problems, such as those arising in power grid systems [2], Internet networks [3], and airport scheduling [4], which contain millions of decision variables and constraints. These problems are critical for optimizing resource allocation and operational efficiency in modern industries but pose significant computational challenges due to the exponential growth of the search space.

In numerous instances, there exists a need to concurrently solve a substantial volume of MILPs that are homogeneous in structure and share analogous combinatorial configurations. However, the sheer scale and complexity of these problems often exceed the capabilities of traditional solvers, especially when constraints are intricate. Under these circumstances, machine learning (ML)-based optimization architectures possess the capability to discern and leverage the correlations between these configurations and their corresponding solution values. By predicting high-quality initial feasible solutions and guiding dimensionality reduction, ML approaches effectively narrow down the search space, thereby accelerating the solution process, which enhances the efficiency of solving these problems. Consequently, ML-based optimization frameworks emerge as a highly promising avenue in this realm [5], [6].

In a groundbreaking endeavor to solve MILPs through an ML-based approach, Gasse et al. [7] introduced a novel concept: a lossless graph representation of MILPs utilizing bipartite graphs. This approach significantly enhances the efficiency of solving MILPs by employing a Graph Neural Network (GNN) model to adaptively learn the variable selection strategy crucial for the branch-and-bound method [8]. The challenge with the branch-and-bound algorithm is its search space, which grows exponentially with an increase in the dimensionality of the decision variables within MILPs [9], leading to a surge in computational demands, especially for large-scale problems. Aiming to address this computational challenge, Nair et al. [10] introduced the neural diving method. This technique simplifies the problem by fixing the majority of the decision variables based on preliminary solution predictions derived from GNN. Consequently, this method effectively transforms large-scale MILPs into more manageable, smaller-scale problems focused on a subset of the remaining decision variables, thereby significantly reducing the complexity associated with the decision variables in MILPs. Building on these advancements,

Received 19 August 2024; revised 1 November 2024 and 7 January 2025; accepted 4 March 2025. Date of publication 12 March 2025; date of current version 2 February 2026. A preliminary conference version of this paper appeared in ICLR 2024 [3]. This article was approved by Associate Editor R. Cheng. *(Corresponding author: Hua Xu.)*

Huigen Ye and Hua Xu are with the State Key Laboratory of Intelligent Technology and Systems, Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China (e-mail: yhg23@mails.tsinghua.edu.cn; xuhua@tsinghua.edu.cn).

Carlos A. Coello Coello is with the Department of Computer Science, CINVESTAV-IPN (Evolutionary Computation Group), Mexico City 07300, Mexico, on leave from the School of Engineering and Sciences, Tecnologico de Monterrey, Monterrey, Mexico (e-mail: carlos.coellocoello@cinvestav.mx).

Digital Object Identifier 10.1109/TEVC.2025.3550668

Sonnerat et al. [11] developed NeuralLNS, which pioneers a neural-based approach for neighborhood selection. This method strategically selects search neighborhoods to refine the initial solutions generated by Neural Diving. Despite these innovations, Neural Diving struggles to fully capitalize on the embedded spatial information, and NeuralLNS's dependence on large-scale solvers introduces performance bottlenecks, with solution quality inherently limited by the capabilities of existing solvers. To overcome these challenges, Ye et al. [12] proposed an advanced optimization framework that, respectively, employs Multitask GNN, gradient boosting decision tree (GBDT), and Neighborhood Optimization to generate the embedding space, to effectively use the embedding spatial information, and to improve the current solution by means of a lightweight optimizer.

While the GNN&GBDT-guided framework has shown impressive results in real-world scenarios, it is not without facing significant challenges. First, representing MILPs as complete graphs complicates the training process and demands substantial computational efforts, particularly for large-scale MILPs. Second, the efficacy of GNNs hinges on the availability of ultralarge-scale MILP instances that are closely matched in size to the training dataset, requiring considerable computational and storage resources throughout the training period. Third, the strategy of problem reduction is confined to decision variables alone, overlooking the benefits of integrating constraint reduction, leading to a constrained scope of problem reduction. Overall, when applied to ultralarge-scale MILPs featuring intricate constraints, the effectiveness of this method is significantly diminished.

To tackle the outlined challenges, we introduce here Light-EvoOPT, a lightweight evolutionary optimization framework tailored for addressing ultralarge-scale MILPs. This framework is structured around four meticulously designed phases.

1) *Problem Formulation:* Initially, the MILP is modeled as a bipartite graph. Subsequently, to mitigate computational demands, the problem undergoes partitioning, achieved through integrating the FENNEL graph partitioning algorithm and the innovative Graph-Bert subgraph partitioning concept, aiming at a significant reduction in computational complexity.
2) *Model-Based Initial Solution Prediction:* In this phase, the advanced edge-aggregated graph attention (EGAT) network is further enhanced with semi-convolution techniques and SelectiveNet. This network is trained on a small-scale dataset of problems with a homogeneous structure, tailored for predicting and assembling the initial solutions for each of the partitioned subproblems.
3) *Problem Reduction:* Following the initial solution prediction, a generalized confidence threshold approach is employed to reduce the dimensionality in decision variable space. Simultaneously, constraints are reduced using a K-nearest neighbor (KNN) strategy, which identifies and removes redundant constraints. This dual reduction process ensures a streamlined problem scale, facilitating a more efficient optimization in subsequent steps.
4) *Evolutionary Optimization:* The final phase leverages adaptive constraints partition (ACP) and active constraint updates to guide the neighborhood search, coupled with an innovative individual crossover strategy that respects the problem's partitioned and reduced structure. This phase is pivotal in iteratively refining the solution, employing a lightweight optimizer to achieve iterative enhancements.

Each of these phases is integral to our proposed Light-EvoOPT, ensuring it stands as a powerful solver for the efficient and effective resolution of ultralarge-scale MILPs, markedly improving upon existing methodologies with its unique combination of strategies and lightweight design.

To illustrate the effectiveness and efficiency of Light-EvoOPT, we undertook comprehensive testing through extensive experiments across four benchmark MILPs of ultralarge-scale, featuring ten million scale decision variables and constraints, alongside a real-world case study. The experimental results are benchmarked against a range of other competitive solvers for comparison. The findings underscore a remarkable superiority of our proposed framework, which leverages a lightweight optimizer and a small-scale training dataset, standing out against both contemporary optimization frameworks and established large-scale solvers such as Gurobi and SCIP, particularly in the realm of ultralarge-scale MILPs.

A further detailed analysis reveals that Light-EvoOPT excels in significantly diminishing the computational complexity of the model and in executing effective problem reduction. These outcomes robustly validate the framework's high effectiveness and efficiency. The contributions of our work are multifaceted, offering valuable advancements in the optimization field, which can be summarized as follows.

1) We introduce an innovative lightweight evolutionary optimization framework explicitly designed for tackling ultralarge-scale MILPs using only small-scale training data and a lightweight optimizer. The proposed approach encompasses problem formulation, model-based initial solution prediction, problem reduction, and evolutionary optimization. Each of these components plays a crucial role in minimizing the computational complexity of the model and enhancing its capability for dimensionality reduction.
2) We embark on pioneering efforts by employing ultralarge-scale standard MILPs of ten-million scale for the first time, benchmarking our proposed framework against state-of-the-art ML-based optimization frameworks and advanced solvers. We effectively showcase the capability of our framework to solve ultralarge-scale MILPs with lightweight optimizers and small-scale datasets. This comparison provides foundational insights into the efficient resolution of extensive MILPs with constrained computational resources.
3) We explore a model self-optimization strategy within ML-based optimization frameworks, focusing on the integration of model pruning to refine the framework's efficiency and scalability. Specifically, by reducing 20% of the network parameters, the pruned variant of Light-EvoOPT achieves comparable or improved performance

on ultralarge-scale MILPs, with a significant reduction in computational overhead. This novel self-optimization approach demonstrates the potential of pruning techniques to streamline the optimization process without compromising solution quality. Moreover, it lays a foundation for further advancements in lightweight optimization for ultralarge-scale problems.

The remainder of this article is organized as follows. Section II reviews the most relevant previous related work on MILPs and ML-based optimization frameworks. Section III details our proposed Light-EvoOPT framework and its four core stages. Section IV presents the experimental settings and the results obtained. Finally, Section V concludes this article and provides some potential paths for future research.

II. Previous Related Work

A. Mixed Integer Linear Programs

MILP represent a category of optimization problems characterized by a linear objective function subject to multiple linear constraints, with the stipulation that some or all of the decision variables are constrained to integer values. Formally, an MILP is structured as follows [13]:

$$\min_{x} \; c^T x, \text{ subject to } Ax \le b, l \le x \le u, x_i \in \mathbb{Z}, i \in \mathbb{I} \tag{1}$$

where x represents the decision variables, with the dimensionality denoted by $n \in \mathbb{Z}$, and $l, u, c \in \mathbb{R}^n$ correspond to the lower bounds, upper bounds, and coefficient values of the decision variables, respectively. The matrix $A \in \mathbb{R}^{m\times n}$ and the vector $b \in \mathbb{R}^m$ define the linear constraints of the problem. The set $\mathbb{I} \subseteq \{1, 2, \ldots, n\}$ denotes the indices of variables that are constrained to be integer values. A solution to the MILP is considered feasible if the decision variable vector $x \in \mathbb{R}^n$ satisfies all the constraints specified in (1). Among feasible solutions, the one that minimizes the objective function value is deemed optimal [14]. Furthermore, if the removal of a constraint alters the optimal solution of the MILP, then that constraint is considered as an active constraint, and vice versa for redundancy constraints [15], [16].

B. Bipartite Graph Representation

Gasse's proposed MILP bipartite graph representation [7] achieves a lossless translation of the MILP into a graph format, serving as input for the neural embedding network [10]. This process is depicted in detail in Fig. 1. The n decision variables in MILP are represented as the set of variable nodes on the right side of the bipartite graph, while the m linear constraints are represented as the set of constraint nodes on the left side. An edge connecting a variable node and a constraint node indicates the presence of the corresponding variable in that constraint.

Formally, let h_x^i, h_δ^i, and $h_{(i,j)}$ denote the feature selection of the ith variable node, jth constraint node, and edge (i, j), respectively. The classical feature selection policy can be expressed as follows:

$$h_x^i = (c_i, l_i, u_i, t_i, d_i)$$
$$h_\delta^j = \left(b_j, \mathfrak{o}_j\right)$$
$$h_{(i,j)} = \left(a_{ij}\right) \tag{2}$$

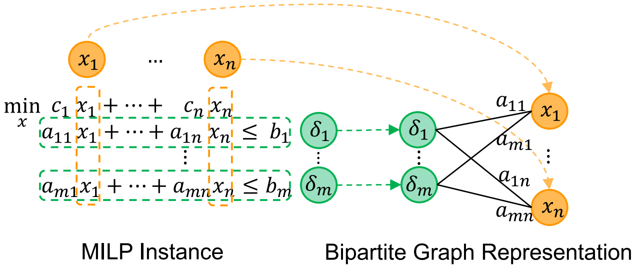
x_1
···
x_n
$\min_x \; c_1 x_1 + \cdots + c_n x_n$
$a_{11}x_1 + \cdots + a_{1n}x_n \le b_1$
$a_{m1}x_1 + \cdots + a_{mn}x_n \le b_m$
δ_1
δ_m
a_{11}
a_{m1}
a_{1n}
a_{mn}
x_1
x_n
MILP Instance
Bipartite Graph Representation

Fig. 1. Conversion of an MILP instance into a bipartite graph. This graph consists of two sets of nodes: the set $\{\delta_1, \ldots, \delta_m\}$ containing m constraint nodes on the left side of the graph, and the set $\{x_1, \ldots, x_n\}$ comprising n decision variable nodes on the right side.

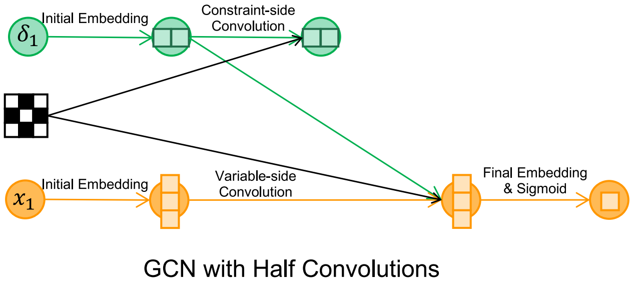
δ_1
Initial Embedding
Constraint-side Convolution
x_1
Initial Embedding
Variable-side Convolution
Final Embedding & Sigmoid
GCN with Half Convolutions

Fig. 2. Architecture of GCN with only half-convolutions consists of a single layer. The layers in GCN can be segmented into two successive passes: one from variables to constraints and another from constraints to variables.

where c_i, l_i, u_i, and t_i denote the coefficient, lower bound, upper bound, and type of the ith decision variable, respectively; $d_i \in \{0, 1\}$ indicates whether the ith decision variable is restricted to take integer values; b_j and $\mathfrak{o}_j$ refer to the value and symbol of the jth constraint; and a_{ij} denotes the weight of edge (i, j).

C. FENNEL Graph Partition Algorithm

For graphs at the million-level scale, the streaming graph partitioning algorithm is commonly utilized. Two primary heuristics exist for streaming graph partitioning. On the one hand, newly arrived vertices are assigned to the cluster with the largest number of neighbors. On the other hand, they can be assigned to the cluster with the fewest non-neighbors. FENNEL introduces an innovative approach by combining these two heuristics.

Formally, for a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ comprising n vertices and m edges, partitioning it into k blocks, FENNEL aggregates vertices with dense edges into a single block while segregating vertices with sparse edges are placed into separate blocks. Specifically, it systematically evaluates each newly arrived vertex v in the graph one by one and computes $\delta_g(v, \mathcal{P}_i)$ for each block $\mathcal{P}_i$ that satisfies $|\mathcal{P}_i| < \nu(n/k)$, as follows:

$$\delta_g(v, \mathcal{P}_i) \leftarrow |\mathcal{P}_i \cap N(v)| - \alpha\gamma|\mathcal{P}_i|^{\gamma-1} \tag{3}$$

where $N(v)$ denotes the neighbor node set of v, ν and γ are preset parameters related to block balancing and minimum cut, and $\alpha = \sqrt{k}(m/n^{3/2})$ denotes the balance between the two types of heuristics.

D. Graph Convolutional Networks

In MILPs, utilizing bipartite graph representations, a graph convolutional network (GCN) [17] is employed to learn neural embeddings and to facilitate model-based initial solution prediction. Formally, denoting $\mathcal{E}$ as the edges in a bipartite graph, a k-layers GCN can be expressed as follows:

$$h_v^k = f_2^k\left(\left\{h_v^{(k-1)}, f_1^k\left(\left\{h_u^{(k-1)} : (u, v) \in \mathcal{E}\right\}\right)\right\}\right) \tag{4}$$

where h_v^k denotes the hidden state of node v in the kth layer. f_1^k aggregates hidden values from the neighbors in the previous $(k-1)$th layer to acquire aggregation information, while f_2^k blends the hidden value of the current node v with the aggregation information from its neighbors.

For the bipartite graph representation, a GCN with two interleaved half-convolutional layers has been shown to yield superior performance [7], [18]. Formally, denoting $\mathcal{V}_x$ as the set of n variable nodes and $\mathcal{V}_\delta$ as the set of m constrained nodes, the k-layer half-convolutional GCN can be expressed as follows:

$$h_{\delta_j}^k = f_\delta^k\left(\left\{h_{\delta_j}^{(k-1)}, \sum_{(x_i,\delta_j)\in\mathcal{E}} g_\delta^k\left(\left\{h_x^{(k-1)}, h_{\delta_j}^{(k-1)}\right\}\right)\right\}\right), \delta_j \in \mathcal{V}_\delta$$
$$h_{x_i}^k = f_x^k\left(\left\{h_{x_i}^{(k-1)}, \sum_{(x_i,\delta_j)\in\mathcal{E}} g_x^k\left(\left\{h_{x_i}^{(k-1)}, h_{\delta_j}^k\right\}\right)\right\}\right), x_i \in \mathcal{V}_x \tag{5}$$

where h_x^k, h_δ^k, g_x^k, and g_δ^k serve as the aggregation functions. An illustration of a single-layer GCN with a half-convolutional layer is presented in Fig. 2.

E. Edge Aggregated Graph Attention Network

The current GCN model discussed in Section II-D faces two primary challenges. First, it relies solely on a fixed strategy for aggregating node information. Second, it fails to fully integrate edge features, resulting in suboptimal performance, especially in graphs with edge weights. To address these issues, Gong and Cheng [19] proposed the edge aggregated graph attention network (EGAT), which introduces an attention mechanism for graph neighborhoods and fully leverages edge information to enhance the learning of neural embeddings and to improve model-based initial solution prediction. Formally, denoting $\mathcal{E}$ as the edges in a bipartite graph, an EGAT with k layers can be expressed as follows:

$$h_v^k = \sigma\left(\left\{\alpha_{uv}^k\left(h_u^{k-1}, \mathcal{E}_{uv}^{k-1}\right) : (u, v) \in \mathcal{E}\right\}, g^k\left(h_v^{k-1}\right)\right) \tag{6}$$

where h_v^k represents the hidden state of node v in the kth layer, while $\mathcal{E}uv^k$ denotes the hidden state of the edges connecting nodes u and v in the kth layer. Here, σ represents a nonlinear activation function, and g^k is the transformation that maps the node's features from the input space to the output space. Specifically, $g^k(hv^{k-1}) = Wh_v^{k-1}$, where W is a transformation parameter matrix. In addition, α_{uv}^k is the attention coefficient, which is a function of h_u^{k-1}, h_v^{k-1} and $\mathcal{E}_{uv}^{k-1}$, which is defined as follows:

$$\alpha_{uv}^k = \mathrm{DS}\left(\exp\left\{L\left(a^T\left[Wh_u^{k-1}||a^T\left[Wh_v^{k-1}\right]\right)\right\}\mathcal{E}_{uv}^{k-1}\right) \tag{7}$$

where L is the LeakyReLU activation function [20]; W represents a transformation parameter matrix, and $||$ represents the concatenation operation. Finally, the attention coefficients serve as new edge features for the subsequent layer, defined as $\mathcal{E}_{uv}^k = \alpha_{uv}^k$. Moreover, DS is the doubly stochastic normalization operator [21], aiming to prevent an undesired increase in the magnitude of the output features due to this multiplication. Formally, denoting the raw edge features as $\hat{\mathcal{E}}$, the normalized features $\mathcal{E}$ can be calculated through DS which is denoted as follows:

$$\tilde{\mathcal{E}}_{ijp} = \frac{\hat{\mathcal{E}}_{ijp}}{\sum_{k=1}^{N} \hat{\mathcal{E}}_{ikp}} \tag{8}$$

$$\mathcal{E}_{ijp} = \sum_{k=1}^{N} \frac{\tilde{\mathcal{E}}_{ikp}\tilde{\mathcal{E}}_{jkp}}{\sum_{v=1}^{N} \tilde{\mathcal{E}}_{vkp}} \tag{9}$$

where $\mathcal{E}_{ij} \in \mathbb{R}^P$ represents the P-dimensional feature vector of the edge connecting the ith and jth nodes; $\mathcal{E}_{ijp}$ denotes the pth channel of the edge feature in $\mathcal{E}_{ij}$. In addition, when the ith and jth points are not contiguous, $\mathcal{E}_{ijp} = 0$ for any p.

F. Network Pruning

The recent rise of large-scale deep neural networks underscores the need for deep neural compression [22]. From among a wide variety of network compression methods, pruning [23] has emerged as highly effective and practical. The objective of network pruning is to eliminate redundant parameters from a given network, thereby reducing its size and potentially accelerating inference speed. As a prominent pruning algorithm, unstructured pruning [24] has shown significant success across numerous network compression tasks.

Unstructured pruning typically begins by learning connectivity through conventional network training. Subsequently, it prunes connections with small weights–those falling below a predetermined threshold–thus removing them from the network. Following this, the network undergoes retraining to determine the final weights for the remaining sparse connections. Ultimately, fine-tuning is employed to refine the compressed network structure, resulting in reduced parameter sizes without substantial performance degradation.

III. Our Proposed Light-EvoOPT

This section introduces Light-EvoOPT, which is our proposed evolutionary lightweight optimization framework designed to solve ultralarge-scale MILPs using a lightweight optimizer and a small training dataset. Light-EvoOPT can be divided into four stages: 1) problem formulation (Section III-A); 2) model-based initial solution prediction (Section III-B); 3) problem reduction (Section III-C); and 4) evolutionary optimization (Section III-D). The overall architecture of the proposed framework is illustrated in Fig. 3.

A. Problem Formulation

In Light-EvoOPT, the large-scale MILP is initially represented in the form of a *Bipartite Graph*. Subsequently, the FENNEL *Graph Partition* algorithm [25] is employed to

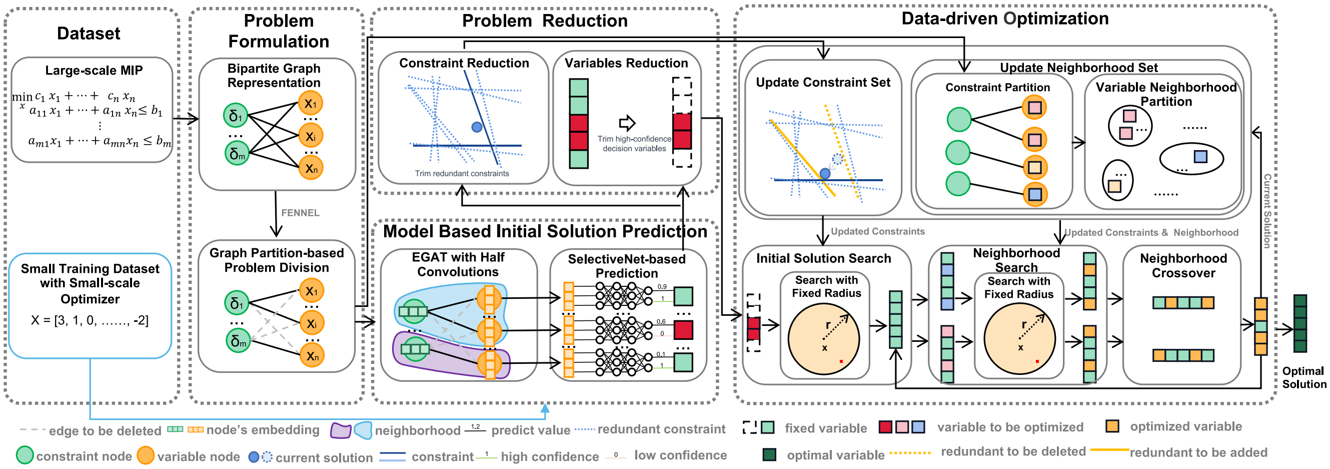

Fig. 3. Overview of LightEvoOPT: The blue line indicates its exclusive use during training, while the black line refers to its application during both training and testing. First, the MILP is represented as a bipartite graph, and the FENNEL graph partitioning algorithm is employed to divide the problem, thereby reducing computational costs. Second, utilizing the divided graph representations, an EGAT combined with SelectiveNet, trained on a small-scale dataset, is employed to predict and construct the initial solution for the original large-scale MILP. Then, leveraging the predicted solution, confidence thresholds for decision variables and a KNN strategy for constraints are introduced to reduce the dimensionality of the problem. Finally, based on problem division and reduction, adaptive constraints partitioning and active constraint updating guide neighborhood search and individual crossover, iteratively improving the current solution through a lightweight optimizer.

partition the graph into multiple blocks. Following these steps, all the subgraphs obtained from the graph partition serve as inputs for feature-embedding neural networks.

1) Bipartite Graph Representation: Building upon the classic bipartite graph representation introduced in Section II-B, we further enhance the feature selection policy to improve embedding ability. By integrating the random feat strategy [26], the reliability of predicting MILP feasibility, optimal objective values, and optimal solutions is remarkably strengthened when facing some particular MILPs called foldable [26]. Moreover, the conventional classification feature selection strategy, relying solely on a single integer, often fails to capture classification information adequately, resulting in weak feature neural embedding within the feature embedding neural network. To address this limitation, we employ a one-hot [27] strategy to represent classification features, thereby amplifying the influence of classification information on the neural embedding outcomes. Formally, let h^i_x, h^j_δ, and $h_{(i,j)}$ denote the feature selection of the ith variable node, jth constraint node, and edge (i, j), respectively. Based on (2), the refined feature selection policy can be expressed as follows:

$$\begin{aligned} h^i_x &= (c_i, l_i, u_i, d_i, t_i, \xi) \\ h^j_\delta &= \left(b_j, \mathfrak{o}'_j, \xi\right) \\ h_{(i,j)} &= \left(a_{ij}\right) \end{aligned} \tag{10}$$

where $\xi \sim U(0, 1)$ represents a random feat sampled from a uniform distribution between 0 and 1, and $\mathfrak{o}'_j$ is the one-hot representation of the type of constraint in the jth constraint node, with $\mathfrak{o}'_j$ defined as $\{\geq \rightarrow [1, 0, 0], \leq \rightarrow [0, 1, 0], = \rightarrow [0, 0, 1]\}$.

2) Graph Partition-Based Problem Division: In Light-EvoOPT, the bipartite graph representation of the MILP problem is partitioned into smaller, manageable subgraphs to tackle the challenges associated with solving ultralarge-scale problems. As discussed in Section III-A, the entire MILP is first modeled as a bipartite graph, where one set of nodes represents the variables, and the other set represents the constraints. This bipartite graph is then partitioned using the FENNEL algorithm [25], which is both computationally efficient and effective in minimizing disruptions to the graph's topological structure. The primary challenges that arise when solving ultralarge-scale MILPs involve the massive computational and memory requirements associated with solving a single, monolithic problem. Instead of directly feeding the entire bipartite graph representation into a neural network for feature embedding, we partition the graph using FENNEL into numerous low-correlation subgraphs, where each subgraph represents a smaller subproblem. These subproblems are then sequentially processed by the feature-embedded neural network, transforming a large-scale optimization problem into the parallel solution of multiple smaller and more tractable problems.

The partitioning process proceeds as follows: First, the FENNEL algorithm is applied to the bipartite graph to partition it into subgraphs with minimal interconnections. This ensures that the subgraphs are of approximately equal size, which is crucial for balancing the computational load between different subproblems. Each subgraph is then processed sequentially by the feature-embedded neural network. By solving these smaller subproblems in parallel or sequentially, Light-EvoOPT can efficiently handle tens of millions of variables and constraints without overwhelming computational resources. The use of FENNEL ensures that the partitioning process is efficient in both time and memory usage, while the low-correlation nature of the subgraphs minimizes the loss of important structural information during partitioning. This allows Light-EvoOPT to maintain high solution accuracy and scalability, making it particularly well-suited for solving ultralarge-scale MILPs. This graph partition-based problem division strategy represents a key innovation of our framework, offering a scalable solution to the challenges posed by large-scale optimization

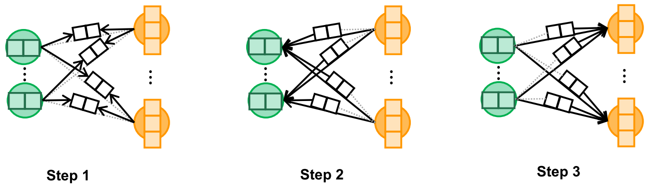

Fig. 4. Information transition flow in the EGAT with half-convolutions involves consecutive steps as follows: Step 1, transforming variable nodes and constraint nodes information to the edge; Step 2, transforming the variable nodes and edge information to constraint nodes; Step 3, transforming constraint nodes and edge information to the variable node.

problems. By breaking down a complex problem into smaller, independent subproblems, Light-EvoOPT achieves significant improvements in both computational efficiency and solution accuracy.

B. Model-Based Initial Solution Prediction

Given a graph representation consisting of multiple small-scale subgraphs derived from a large-scale MILP, the *EGAT with Half-convolutions* learns neural embeddings of the decision variables. Subsequently, the *SelectiveNet-based Prediction* network predicts the initial values of the corresponding decision variables in the MILP based on these neural embeddings. Finally, the predicted initial solutions will guide the subsequent stage of problem reduction.

1) EGAT With Half-Convolutions: Based on the GNN with half-convolutions [18] (Section II-D) and EGAT [19] (Section II-E), we propose an EGAT with a multilayer half-convolutions structure. This combination aims to enhance the learning of neural embeddings for decision variables in each divided subgraph further. Formally, based on (5) and (6), considering $\mathcal{E}$ as the edges in the bipartite graph, a k-layer EGAT with a multilayer half-convolutions structure can be represented as follows:

$$\begin{aligned}
\alpha_{x_i\delta_j}^{k} &= \mathrm{DS}\left(\exp\left\{L\left(a^T\left[Wh_{x_i}^{k-1}||a^T\left[Wh_{\delta_j}^{k-1}\right]\right)\right\}\mathcal{E}_{x_i\delta_j}^{k-1}\right)\\
h_{\delta_j}^{k} &= \sigma\left(\left\{\alpha_{x_i\delta_j}^{k}\left(h_x^{(k-1)}, \mathcal{E}_{x_i\delta_j}^{k-1}\right) : (x_i, \delta_j) \in \mathcal{E}\right\}, g^k\left(h_{\delta_j}^{k-1}\right)\right)\\
h_{x_i}^{k} &= \sigma\left(\left\{\alpha_{x_i\delta_j}^{k}\left(h_{\delta_j}^{k}, \mathcal{E}_{x_i\delta_j}^{k-1}\right) : (x_i, \delta_j) \in \mathcal{E}\right\}, g^k\left(h_{x_i}^{k-1}\right)\right)\\
\mathcal{E}_{x_i\delta_j}^{k} &= \alpha_{x_i\delta_j}^{k} \qquad (11)
\end{aligned}$$

where $h_{\delta_j}^k$ and $h_{x_i}^k$ represent the hidden state of constraint node δ_j and variable node x_i in the kth layer, respectively. $\mathcal{E}_{x_i\delta_j}^k$ and $\alpha_{x_i\delta_j}^k$ denote the hidden state and attention coefficient of the edge connecting variable node x_i and constraint node δ_j in the kth layer, respectively. σ denotes a nonlinear activation function, and g^k represents a transformation that maps node features from the input space to the output space. The information transition flow of EGAT with half-convolutions is illustrated in Fig. 4.

2) SelectiveNet-Based Prediction: Using the neural embedding of decision variables from EGAT, a SelectiveNet [28] predicts the initial value of each corresponding decision variable for every subdivided small-scale MILP. For binary variables x_i, alongside the probability that the decision variable equals 1, denoted as $p(x_i|M)$, SelectiveNet introduces an additional output $y_i \in \{0, 1\}$ determining whether x_i should be predicted ($y_i = 1$) or not.

For general integer variables or even real variables, depending on the required accuracy, the neural embedding of a decision variable needs to pass through multiple independent SelectiveNet models in parallel. For the ith decision variable, the output of the jth SelectiveNet can represent the probability that the jth binary bit is 1 and whether the jth binary bit should be predicted or not, represented as $p(x_{ij}|M)$ and y_{ij}. The value of the bit is determined as follows:

$$x_{ij} = \begin{cases} \lfloor p\left(x_{ij}|M\right) + 0.5\rfloor, & y_{ij} = 1 \\ \text{Unpredictable}, & y_{ij} = 0. \end{cases} \qquad (12)$$

For a conditionally independent model, when combined with Focal Loss [29], we define the predicted value loss $\mathcal{L}_v$ and the predicted proportion loss $\mathcal{L}_p$ as follows:

$$\begin{aligned}
\mathcal{L}_v &= \frac{-\sum_{i,j} y_{ij} w_{ij}\left(1 - p(x_{ij}|M)\right)^{\gamma} \log p\left(x_{ij}|M\right)}{\sum_{i,j} y_{ij}}\\
\mathcal{L}_p &= \Phi\left(C - \frac{1}{|\sum_{i,j} 1|}\sum_{i,j} y_{ij}\right) \qquad (13)
\end{aligned}$$

where $\mathcal{L}_v$ represents the prediction loss, with $\gamma \geq 0$ being the focusing parameter and w_{ij} the class balancing parameter. This term helps to improve the accuracy of the predicted variables, especially for harder samples. On the other hand, $\mathcal{L}_p$ is the prediction proportion loss, where C is a coverage threshold that defines the desired relative frequency of the predicted variables. The function $\Phi(a) = \max(0, a)^2$ serves as a quadratic penalty term, discouraging the network from predicting too many or too few variables. This forces the network to focus on positions where it is confident, preventing it from wasting capacity on uncertain predictions.

The total loss function is then defined as

$$\mathcal{L} = \beta\mathcal{L}_v + (1 - \beta)\mathcal{L}_p \qquad (14)$$

where β is a hyperparameter that controls the tradeoff between the prediction accuracy and the coverage of predicted variables.

The practical significance of these two terms is as follows: $\mathcal{L}_v$ ensures that the network focuses on making accurate predictions for the decision variables, while $\mathcal{L}_p$ acts as a constraint to force the network to prioritize predictions only for variables where it has high confidence. This balance allows the model to concentrate its limited capacity on positions where it can make the most reliable predictions, rather than attempting to predict every variable.

C. Problem Reduction

Given the predicted initial solution of the MILP, the generalized confidence threshold method adaptively fixes high-confidence decision variables to achieve *Variable Reduction*. Subsequently, the KNN strategy is employed for *Constraint Reduction* to identify active constraints. The unfixed decision variables and KNN constraints jointly guide the initial solution search and the optimization process.

1) Variable Reduction: Aiming to reduce variable dimensionality efficiently, Light-EvoOPT proposes a generalized confidence threshold method based on SelectiveNet and the confidence threshold method specifically designed for binary variables [18]. Specifically, we redefined the additional output of SelectiveNet y_{ij} as $y_{ij} \in [0, 1]$, denoting the confidence value. Based on the confidence value obtained from the model predictions, the total confidence value f_i of the decision variable x_i which contains c binary bits can be represented as follows:

$$f_i = \sqrt[c]{\prod_{j=1}^{c} y_{ij}}. \tag{15}$$

Subsequently, the confidence values of the decision variables are arranged in descending order. Utilizing the specified reduction ratio of $k\%$, the top $k\%$ of the decision variables are set to their predicted values. The remaining decision variables constitute a fresh small-scale optimization problem, which is then introduced into the initial solution search of the subsequent stage. This process accomplishes the dimensionality reduction of decision variables for the large-scale MILP. Due to the large parameters of the initial solution prediction network and the high graphic memory requirement of computational resources, we further try to prune EGAT (see Section II-F), and we will detail the effect before and after pruning in the experimental section.

2) Constraint Reduction: In scenarios where integer constraints are not under consideration, the feasible domain defined by the linear constraints in the MILP exhibit a remarkable property: convexity. Within this convex space, the optimal solution of the MILP often lies near a cluster of active constraints [30]. Leveraging this insight, the predicted initial solution–derived from a model tailored to approximate the optimal solution–serves as a valuable estimate of the true optimum. By utilizing this estimate, we can effectively gauge the proximity of each constraint hyperplane to the predicted solution.

To implement constraint reduction, we employ a KNNs approach. Each linear constraint in the MILP can be viewed as a hyperplane, and we calculate the distance from the current solution point to each hyperplane. The K nearest hyperplanes—i.e., the ones closest to the current solution—are considered active constraints and are retained in the MILP formulation. These active constraints are likely to be the most crucial in determining the optimal solution, while the more distant constraints are treated as redundant and are pruned. The distance between the current solution and a constraint hyperplane is computed by measuring the perpendicular distance, allowing us to quantitatively assess the relevance of each constraint. This process is not static; rather, it is dynamically updated as the solution process progresses. As described in Section III-D3, the set of active constraints is continuously maintained and adjusted based on the updated solution. This ensures that the optimization process always focuses on the most relevant constraints, significantly reducing the complexity of the problem while maintaining solution quality. Through this dynamic KNN-based constraint reduction mechanism, we are able to effectively alleviate search pressure and improve the efficiency of solving large-scale MILPs.

Algorithm 1 Initial Solution Search

```
Input: The number of decision variables $n$, predicted value $\hat{x}$, prediction loss $f$, variable proportion $\alpha$ representing the fixed radius
Init: Initial Solution $\mathcal{X} = \{\}$
$\mathcal{X} \leftarrow \hat{x}$
Sort the decision variables in ascending order of $f$
$\alpha_{set} = \alpha$
repeat
    $\mathcal{F} \leftarrow$ The first $(1 - \alpha_{set})n$ decision variables   ▷Fixed
    $\mathcal{U} \leftarrow$ The last $\alpha_{set}$ decision variables   ▷Unfixed
    $\mathcal{F}', \mathcal{U}' \leftarrow$ REPAIR($\mathcal{F}, \mathcal{U}, \mathcal{X}$)
    if $|\mathcal{U}'| > \alpha n$ then
        $\alpha_{set} = \eta * \alpha_{set}$
    end if
until $|\mathcal{U}'| \leq \alpha n$
$\mathcal{X} \leftarrow$ SEARCH($\mathcal{F}', \mathcal{U}', \mathcal{X}$)
Return: $\mathcal{X}$
```

D. Evolutionary Optimization

Starting with the predicted initial solution and following the problem reduction, we address the reduced subproblem initially to derive the *Initial Solution* for the complete MILP. Subsequently, leveraging *Neighborhood Set Updating* alongside active *Constraint Set Updating*, iterative *Evolutionary Optimization* processes of neighborhood search and individual crossover enhance the current solution iteratively. Finally, upon reaching a predetermined wall-clock time or meeting specific conditions, the current solution is presented as the final optimization result.

1) Initial Solution Search: For the large-scale MILP with n decision variables, we first define a coefficient $\alpha \in (0, 1)$ to denote that the lightweight optimizer can solve small-scale MILP containing at most αn decision variables. Following this, an initial solution search method is employed for the reduced small-scale MILP obtained from the dimensionality reduction stage. Given the predicted value $\hat{x}_i$ and the prediction confidence value f_i for each decision variable, the decision variables are arranged in ascending order based on their prediction losses. For the predefined coefficient $\alpha \in (0, 1)$, which denotes that the lightweight optimizer can solve small-scale MILP containing at most αn decision variables, the first $(1 - \alpha)n$ decision variables are held constant. In contrast, the remaining variables are explored within a predetermined fixed radius. The specific steps are outlined in Algorithm 1, where $\eta \in (0, 1)$ is a reduction coefficient used to expand the fixed proportion.

Specifically, the SEARCH($\mathcal{F}, \mathcal{U}, \mathcal{X}$) function in Algorithms 1, 3, and 4 implements the fixed radius search strategy. This function optimizes the variables that are free (those not in $\mathcal{F}$) using a restricted MILP solver, such as SCIP, while keeping the remaining variables fixed at their current values. The function can be written as follows:

$$\min_{x \notin \mathcal{F}} \quad c^T x$$
$$\text{subject to} \quad Ax \leq b, l \leq x \leq u, x \in \mathbb{Z}^n$$

Algorithm 2 REPAIR Algorithm

```
Input: The set of fixed variables $\mathcal{F}$, the set of unfixed variables $\mathcal{U}$, the current solution $\mathcal{X}$
$\{A, b, l, u\} \leftarrow$ The coefficient of the given MILP
$n \leftarrow$ the number of decision variables
$m \leftarrow$ the number of constraints
for $i = 1$ to $m$ do
  $\mathcal{N} \leftarrow 0$
  for $j = 1$ to $n$ do
    if The $j^{th}$ decision variable $\in \mathcal{F}$ then
      $\mathcal{N} \leftarrow \mathcal{N} + \mathcal{X}_j * A_{i,j}$
    else
      $x_{ij} = \begin{cases} \mathcal{N} \leftarrow \mathcal{N} + l_j * A_{i,j}, & A_{ij} > 0 \\ \mathcal{N} \leftarrow \mathcal{N} + u_j * A_{i,j}, & A_{ij} \leq 0 \end{cases}$
    end if
  end for
  if $\mathcal{N} > b_i$ then
    for $j = 1$ to $n$ do
      if The $j^{th}$ decision variable $\in \mathcal{F}$ then
        Remove the $j^{th}$ decision variable from $\mathcal{F}$
        Append the $j^{th}$ decision variable into $\mathcal{U}$
        $\mathcal{N} \leftarrow \mathcal{N} - \mathcal{X}_j * A_{i,j}$
        $x_{ij} = \begin{cases} \mathcal{N} \leftarrow \mathcal{N} + l_j * A_{i,j}, & A_{ij} > 0 \\ \mathcal{N} \leftarrow \mathcal{N} + u_j * A_{i,j}, & A_{ij} \leq 0 \end{cases}$
        if $\mathcal{N} \leq b_i$ then
          BREAK
        end if
      end if
    end for
  end if
end for
Return: $\mathcal{F}, \mathcal{U}$
```

$$x_i = \mathcal{X}_i \quad \forall x_i \in \mathcal{F} \tag{16}$$

where $\mathcal{F}$ refers to the set of decision variables fixed to their current solution values as defined in Algorithm 1. The remaining variables, which fall within the fixed radius determined by α, are optimized in this search process.

However, due to the possibility of incorrect predictions, the Mixed-Integer MILP corresponding to (16) may become infeasible, resulting in the failure of the current solution. To address this issue, we introduce a REPAIR Algorithm designed to examine and rectify constraints that are inherently infeasible by removing the fixation of certain illegal variables associated with these infeasible constraints. In particular, when dealing with a given MILP alongside a set of fixed variables denoted as $\mathcal{F}$, the REPAIR algorithm systematically iterates through each constraint in the MILP. For each constraint under consideration, the algorithm assesses whether it is inevitably infeasible based on the upper and lower bounds of the unfixed variables. If the algorithm determines that the constraint is indeed infeasible, it proceeds to release the fixation of specific decision variables associated with that constraint, aiming to restore feasibility. The intricate steps of this process are elucidated in Algorithm 2. Finally, the optimal solution obtained by the lightweight optimizer is recombined with the reduced decision variables to form an initial feasible solution.

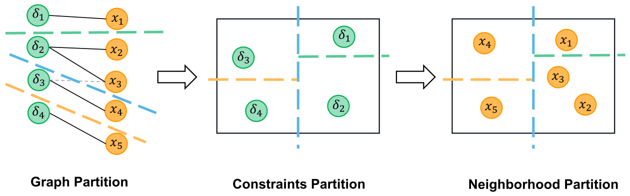

Fig. 5. Use of the partitioning result of the bipartite graph as a neighborhood partition. The decision variables contained in the constraints in the partitioned blocks form the corresponding neighborhood.

Algorithm 3 ACP

```
Input: The set of subgraphs $\mathcal{G} = \{(\Delta_1, \mathcal{E}_1), \ldots, (\Delta_k, \mathcal{E}_k)\}$, the coefficient $\alpha \in (0, 1)$ to denote that the ablity of lightweight optimizer.
Init: Neighborhood Set $N = \{\}$
$\mathcal{N}_{now} \leftarrow \{\}$
for $\mathcal{G}_{sub} = (\Delta_{sub}, \mathcal{E}_{sub}) \in \mathcal{G}$ do
  $\mathcal{N}_{new} \leftarrow \mathcal{N}_{now}$
  $X_{sub} \leftarrow$ variables appearing in the constraint set $\Delta_{sub}$
  $X_{choose} \leftarrow X_{sub} \backslash \{x | x \in \mathcal{N}_{sub}, \forall \mathcal{N}_{sub} \in (N \cup \mathcal{N}_{now})\}$
  if $|\mathcal{N}_{now} \cup X_{choose}| \leq n * \alpha$ then
    $\mathcal{N}_{now} \leftarrow \mathcal{N}_{now} \cup X_{choose}$
  else
    $N \leftarrow N \cup \mathcal{N}_{now}$
    $N_{now} \leftarrow X_{choose}$
  end if
end for
Return: Neighborhood Set $N$
```

2) Neighborhood Set Update: At each optimization iteration, the creation of a fresh neighborhood becomes imperative to navigate the landscape of local optima effectively. Building upon the subgraph partitioning derived from the FENNEL algorithm, we employ the ACP [31] algorithm to further harness the bipartite graph's potential. This synergy yields refined partitioning results for both the constraint and variable sets, as depicted in Fig. 5. Initially, leveraging the FENNEL algorithm, the bipartite graph G undergoes segmentation into a collection of subgraphs denoted as $\mathcal{G} = \{(\Delta_1, \mathcal{E}_1), \ldots, (\Delta_k, \mathcal{E}_k)\}$, where Δ_k represents the constraints within each subgraph and $\mathcal{E}_k$ characterizes the edge connectivity. Subsequently, employing ACP's two-step selection methodology–first constraints, then variables–we strategically identify decision variable nodes interconnected by the set of constraint points within each subgraph. This deliberate selection process enhances the cohesion among decision variables within the same neighborhood, thereby augmenting the efficacy of neighborhood search operations. The detailed pseudo-code of the algorithm is shown in Algorithm 3.

3) Constraint Set Update: Given the potential discrepancy between the initially predicted and optimal solutions, the initial forecast of active constraints may exhibit bias. Therefore, as each optimization iteration progresses and the current solution gradually converges toward optimality, Light-EvoPOT recalculates a fresh set of KNN constraints. These updated constraints refine the estimates of the active constraints, leveraging the current solution as the new reference point. Moreover, recognizing that the reliability of active constraint predictions improves throughout optimization, we adopt a progressive strategy. Initially, we select a larger value of K

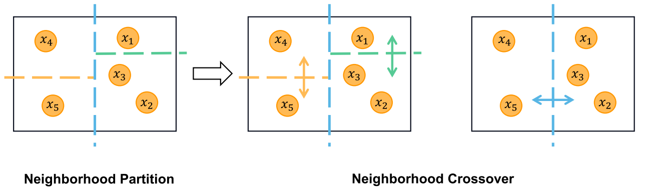

Fig. 6. Use of the hierarchical crossover strategy for neighborhood merging and for the corresponding individual crossover.

Algorithm 4 Neighborhood Search

```
**Input:** The set of decision variables $X$, the number of decision variables $n$, predicted value $\hat{y}$, prediction loss $f_i$, variable proportion $\alpha$, neighborhood $N_{now}$, current solution $\mathcal{X}$
**Init:** Neighborhood search solution $\mathcal{X}' = \{\}$
Sort the decision variables in $N_{now}$ in descending order of $f_i * |\phi_i - \hat{y}_i|$
$\mathcal{N} \leftarrow$ The first $\alpha n$ 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
$\mathcal{X}' \leftarrow$ SEARCH($\mathcal{F}, \mathcal{U}, \mathcal{X}$)
**Return:** $\mathcal{X}'$
```

Algorithm 5 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
$\mathcal{F}', \mathcal{U}' \leftarrow$ REPAIR($\mathcal{F}, \mathcal{U}, \mathcal{X}''$)
**if** $|\mathcal{U}'| \le \alpha n$ **then**
    $\mathfrak{X} \leftarrow$ SEARCH($\mathcal{F}', \mathcal{U}', \mathcal{X}''$)
**end if**
**Return:** $\mathfrak{X}$
```

in the KNN algorithm, gradually narrowing down the set of active constraints as the optimization process proceeds. Following each iteration, K is updated to $K \times \eta$ until the predefined minimum reduction threshold is attained. Here, $\eta \in (0, 1)$ serves as the preset descent rate, facilitating the gradual reduction of constraints. Lastly, as changes occur in the active constraint set, the feasible solution region undergoes alterations as well. To ensure that the current solution remains within the bounds of the revised feasible solution range dictated by the updated active constraint set, we employ the REPAIR algorithm detailed in Algorithm 2. This mechanism effectively repairs the current solution, ensuring its adherence to the new constraints.

4) Evolutionary Optimization: Based on the initial solution obtained from the prediction, the efficient Evolutionary Optimization method can quickly improve the current solution. In ultralarge-scale optimization problems, large neighborhood search has a good solution and convergence performance. We integrate the idea of using a genetic algorithm on the basis of large-neighborhood search, and propose a strategy based on multiple large-neighborhood parallel search processes and hierarchical crossover of individual solutions obtained from the search processes in order to achieve fast iterative improvement of the initial feasible solutions. Based on the updated neighborhood set and updated active constraint set, the neighborhood search is executed in parallel in each neighborhood, utilizing a lightweight optimizer. Specifically, for the ith neighborhood N_i, Algorithm 4 shows the details in neighborhood search, where f_i, $\mathcal{X}_i$ and $\hat{y}_i$ denote the prediction loss defined in (15), the value in the current solution and the predicted value of the ith decision variable, respectively. Since the size of the neighborhood is limited to no more than αn, it is easy to fall into a local optimum 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 Fig. 6. Algorithm 5 outlines the specific procedure for crossing two neighborhoods, denoted as N_1 and N_2.

IV. Experiments

To validate the effectiveness and efficiency of Light-EvoOPT for ultralarge-scale MILPs, we compare it with respect to two types of baseline approaches on four widely used ultralarge-scale NP-hard standard MILP datasets: 1) set covering (SC, minimize) [32]; 2) minimum vertex cover (MVC, Minimize) [33]; 3) maximum independent set (MIS, maximize) [34]; and 4) mixed integer knapsack set (MIKS, Maximize) [35] and one real-world large-scale MILP in the Internet domain (Case Study, Maximize). Within the baseline approaches adopted there are state-of-the-art MILP solvers, including SCIP [36], CPLEX [37], and Gurobi [38]. The other one is the mainstream ML-based optimization framework based on GNN&GBDT (GBDT) [12]. And we also make a comparison with respect to the latest ML-based optimization framework Light-MILPopt (MILP) [3]. The details of the datasets, baselines and our detailed experimental setup are presented in Section IV-A.

To ensure a fair comparison, we employ multiple evaluation metrics to analyze the performance of all the methods discussed in this article, including a detailed comparison of solution effectiveness under the same running time (Section IV-B), an evaluation of solution efficiency under the same solution quality (Section IV-C), a convergence analysis (Section IV-D), and a training efficiency analysis (Section IV-E).

A. Experimental Settings

1) Dataset: For the four widely used NP-hard benchmark MILPs, the existing datasets cannot meet such large-scale data requirements, so we used data generators to produce both the training and the test datasets. Specifically, for the maximum independent set problem (MIS) or minimum vertex covering problem (MVC) with n decision variables and m constraints, we generated a random graph with n nodes and m edges to correspond to an MILP that meets the scale requirements. For

TABLE I
SIZE OF ONE REAL-WORLD CASE STUDY IN THE INTERNET DOMAIN AND FOUR WIDELY USED NP-HARD BENCHMARK MILPS

Problem	Scale	Number of Variables	Number of Constraints
SC (Minimize)	SC$_1$	200000	200000
	SC$_2$	2000000	2000000
	SC$_3$	20000000	20000000
MVC (Minimize)	MVC$_1$	100000	300000
	MVC$_2$	1000000	3000000
	MVC$_3$	10000000	30000000
MIS (Maximize)	MIS$_1$	100000	300000
	MIS$_2$	1000000	3000000
	MIS$_3$	10000000	30000000
MIKS (Maximize)	MIKS$_1$	200000	200000
	MIKS$_2$	2000000	2000000
	MIKS$_3$	20000000	20000000
Case (Maximize)	Case Study	2040000	100003

TABLE II
SETTINGS FOR THE FIXED EXECUTION TIME AND THE FIXED TARGET OBJECTIVE FUNCTION VALUE FOR ONE REAL-WORLD CASE STUDY IN THE INTERNET DOMAIN AND FOUR WIDELY USED NP-HARD BENCHMARK MILPS

Problem	Scale	Setting Runing Time	Setting Target Value
SC (Minimize)	SC$_1$	2000s	17121.5
	SC$_2$	12000s	166756.0
	SC$_3$	80000s	2013187.3
MVC (Minimize)	MVC$_1$	2000s	27337.8
	MVC$_2$	8000s	273014.6
	MVC$_3$	80000s	2739485.6
MIS (Maximize)	MIS$_1$	2000s	22621.7
	MIS$_2$	8000s	227074.5
	MIS$_3$	80000s	2247227.5
MIKS (Maximize)	MIKS$_1$	2000s	35067.8
	MIKS$_2$	12000s	355887.6
	MIKS$_3$	60000s	3306566.1
Case (Maximize)	Case Study	1000s	944086.4

the set covering problem (SC) with n decision variables and m constraints, we generated a random problem with n items and m sets where each set bid includes 4 items. For the mixed integer knapsack set (MIKS) with n decision variables and m constraints, we generated a random problem with n items and m dimension where each dimension includes 4 items and at least half of the items have integer constraints. For the optimal solution in the training data set, we allowed Gurobi to run for 8 h to find the approximate optimal solution. The decision variables and constraint scale of one case study in the Internet domain and four widely used NP-hard benchmark MILPs are shown in Table I, where SC denotes the SC. MVC denotes the MVC. MIS denotes the MIS. MIKS denotes the MIKS problem. Case denotes the real-world case study.

2) Baseline Approaches: In this article, two types of baseline approaches are employed. One type is the mainstream ML-based optimization framework based on GNN&GBDT (GBDT) [12] and the latest ML-based optimization framework Light-MILPopt (MILP) [3]. The other type of baseline approaches are the state-of-the-art MILP solvers, including SCIP(4.3.0) [36], CPLEX(22.1.1.0) [37] and Gurobi(10.0.1) [38]. Their scale-constrained versions are used as lightweight optimizerr for both the proposed framework and the latest ML-based optimization framework.

3) Environment: All the experiments were run on a PC with an Intel Xeon Platinum 8375C @ 2.90GHz CPU and four NVIDIA TESLA V100(32G) GPUs. Each scale of any Benchmark MILP was tested on five different instances, and the results shown are the average of the five results. To verify the effectiveness and efficiency of the proposed Light-EvoOPT, we not only compare the results obtained at the same fixed execution time, but we also compare the execution time required with a fixed target objective function value. The settings of the fixed execution time and of the fixed target objective function value for each MILP are shown in Table II.

B. Comparisons of Solution Effectiveness

Before delving into detailed comparisons of solution effectiveness, we first summarize the overall experimental trends observed. As shown in Table III, the performance of various solvers and frameworks, including traditional solvers (e.g., SCIP, Gurobi, and CPLEX) and learning-based methods (e.g., GNN&GBDT and Light-MILPopt), is compared. Our proposed Light-EvoOPT framework, represented by two versions, Ours (without pruning) and OursP (with pruning), consistently achieves superior or comparable results. The learning-based methods' results reflect their best performance, achieved by leveraging lightweight optimizers and parameters tuning, ensuring fair comparisons under the same execution time.

The experimental results demonstrate that Light-EvoOPT–particularly, the pruned variant, OursP–delivers high-quality solutions across diverse problem types and scales. Specifically, compared to traditional solvers such as SCIP, Gurobi, and CPLEX, Light-EvoOPT outperforms them on most problem instances, especially in large-scale cases where traditional solvers often face scalability challenges. Light-EvoOPT maintains computational efficiency and produces better solutions within the same execution time, highlighting its superior scalability. When compared to other learning-based methods like GNN&GBDT and Light-MILPopt, Light-EvoOPT also demonstrates significant advantages for more complex problem instances. OursP generally outperforms Ours in larger problems, where pruning streamlines optimization without sacrificing accuracy.

To further verify the effectiveness of Light-EvoOPT, we compared its results with respect to those of SCIP, Gurobi, GNN&GBDT, and Light-MILPopt under the same execution times. Light-EvoOPT employs a lightweight optimizer, limiting the variable proportion α to 30% and 50%, and uses only 0.1% of the size of ultralarge-scale benchmark MILPs as training data. The experimental results, shown in Tables IV and V, indicate that Ours-30%G (Light-EvoOPT with Gurobi, $\alpha = 30\%$, no pruning) and OursP-30%G (with pruning) significantly outperform SCIP and Gurobi. For instance, in large-scale SC problems, Light-EvoOPT achieves a clear advantage, while Gurobi and SCIP struggle with scalability. Similarly, Light-EvoOPT outperforms

TABLE III
COMPARISON OF OBJECTIVE VALUE RESULTS WITH BASELINE APPROACHES USING THE SAME EXECUTION TIME. AN UPWARD ARROW (↑) INDICATES THAT THE RESULT IS BETTER THAN THE BASELINE. A DASH (−) INDICATES THAT NO FEASIBLE SOLUTION WAS FOUND. AN ASTERISK (*) INDICATES AN OUT-OF-MEMORY OR OUT-OF-GPU-MEMORY ERROR DURING TRAINING. BOLDFACE IS USED TO DENOTE THE BEST RESULTS

	SCIP	Gurobi	CPLEX	GBDT	MILP	Ours	OursP
SC_1	25191.2	17934.5	16442.2	16728.8	16108.1	**15886.3↑**	15888.2↑
SC_2	385708.4	320240.4	164889.6	252797.2	160015.5	159367.1↑	**159309.1↑**
SC_3	9190301.1	3198747.6	9155020.6	*	1603278.9	1596880.3↑	**1596075.7↑**
MVC_1	31275.4	28151.3	27645.3	27107.9	26950.7	26845.8↑	**26842.0↑**
MVC_2	491042.9	283555.8	278194.1	271777.2	269571.5	**269129.6↑**	269216.4↑
MVC_3	4909318.0	2834161.3	5000068.0	*	2694126.4	2692249.4↑	**2690273.6↑**
MIS_1	18649.6	21789.0	21260.4	22795.7	22966.5	23082.7↑	**23088.5↑**
MIS_2	9104.3	216591.3	210089.5	227006.4	230432.9	**231033.7↑**	230977.8↑
MIS_3	90750.0	2165906.7	2100795.6	*	2299504.6	**2308329.3↑**	2307663.9↑
$MIKS_1$	29974.7	32960.0	32787.4	-	36125.5	36347.8↑	**36370.9↑**
$MIKS_2$	168289.9	329642.4	320604.2	-	362265.1	**363238.9↑**	363225.7↑
$MIKS_3$	919031.1	3229713.4	0.0	-	3515173.0	3621878.0↑	**3622450.3↑**
Case	924954.5	-	898538.3	-	980688.0	981748.3↑	**982274.8↑**

TABLE IV
COMPARISON OF OBJECTIVE VALUE RESULTS WITH BASELINE APPROACHES USING THE SAME EXECUTION TIME ADOPTING GUROBI AS THE LIGHTWEIGHT OPTIMIZER. BOLDFACE IS USED TO DENOTE THE BEST RESULTS

	GBDT-30%G	MILP-30%G	Ours-30%G	OursP-30%G	GBDT-50%G	MILP-50%G	Ours-50%G	OursP-50%G	Gurobi
SC_1	18487.6	17047.3	16127.2↑	**16125.6↑**	17503.4	16108.1	**15886.3↑**	15888.2↑	17934.5
SC_2	281021.2	163975.9	162082.0↑	**161758.3↑**	252797.2	160015.5	159367.1↑	**159311.6↑**	320240.4
SC_3	*	1667157.9	1633930.3↑	**1628124.9↑**	*	1603278.9	1596880.3↑	**1596075.7↑**	3198747.6
MVC_1	27700.8	27223.3	27097.3↑	**27078.5↑**	27329.9	26950.7	26845.8↑	**26842.0↑**	28151.3
MVC_2	281234.5	272579.5	**271539.7↑**	271610.2↑	274600.8	269571.5	**269129.6↑**	269216.4↑	283555.8
MVC_3	*	2724414.7	**2714601.2↑**	2715561.3↑	*	2694126.4	2692249.4↑	**2690273.6↑**	2834161.3
MIS_1	22115.9	22658.0	22842.9↑	**22850.7↑**	22530.1	22966.5	23082.7↑	**23088.5↑**	21789.0
MIS_2	210019.2	227305.4	**228714.3↑**	228444.6↑	215393.6	230432.9	**231033.7↑**	230977.8↑	216591.3
MIS_3	*	2267990.8	**2286245.8↑**	2281966.2↑	*	2299504.6	**2308329.3↑**	2307663.9↑	2165906.7
$MIKS_1$	-	35533.4	36039.5↑	**36105.6↑**	-	36108.2	36347.8↑	**36370.9↑**	32960.0
$MIKS_2$	-	357439.5	358346.6↑	**360230.3↑**	-	362265.1	**363238.9↑**	363225.7↑	329642.4
$MIKS_3$	-	3505202.2	3565188.2↑	**3567503.8↑**	-	3515173.0	3621878.0↑	**3622450.3↑**	3229713.4
Case	-	979797.8	**979898.8↑**	**979898.8↑**	-	980688.0	981748.3↑	**982274.8↑**	-

TABLE V
COMPARISON OF OBJECTIVE VALUE RESULTS WITH BASELINE APPROACHES USING THE SAME EXECUTION TIME ADOPTING SCIP AS THE LIGHTWEIGHT OPTIMIZER. BOLDFACE IS USED TO DENOTE THE BEST RESULTS

	GBDT-30%S	MILP-30%S	Ours-30%S	OursP-30%S	GBDT-50%S	MILP-50%S	Ours-50%S	OursP-50%S	SCIP
SC_1	17222.2	17121.5	16184.8↑	**16156.0↑**	16728.8	16147.2	15943.7↑	**15931.4↑**	25191.2
SC_2	261174.0	166756.0	**162716.5↑**	164383.2↑	268294.9	166966.9	163992.4↑	**159309.1↑**	385708.4
SC_3	*	2013187.3	1703925.6↑	**1701251.7↑**	*	2143953.6	1712892.2↑	**1688015.2↑**	9190301.1
MVC_1	27515.4	27337.8	27103.7↑	**27093.7↑**	27107.9	26956.8	26849.3↑	**26849.5↑**	31275.4
MVC_2	276306.9	273014.6	**271855.9↑**	272358.0↑	271777.2	269771.3	**269300.4↑**	269438.1↑	491042.9
MVC_3	*	2739485.6	2720567.2↑	**2718868.0↑**	*	2743856.4	2697426.5↑	**2693109.8↑**	4909318.0
MIS_1	22389.3	22621.7	22843.4↑	**22848.6↑**	22795.7	22963.6	23073.7↑	**23082.9↑**	18649.6
MIS_2	223349.8	227074.5	**228386.5↑**	228322.2↑	227006.4	230278.1	**230861.0↑**	230567.8↑	9104.3
MIS_3	*	2247227.5	**2282112.1↑**	2279376.9↑	*	2249585.2	2306204.4↑	**2306699.5↑**	90750.0
$MIKS_1$	-	35067.8	35998.3↑	**36004.2↑**	-	36125.5	36333.0↑	**36338.2↑**	29974.7
$MIKS_2$	-	355887.6	356921.2↑	**359686.4↑**	-	357483.8	362395.9↑	**363012.4↑**	168289.9
$MIKS_3$	-	3410694.6	3428798.2↑	**3456219.4↑**	-	3306566.1	3394443.9↑	**3434767.9↑**	919031.1
Case	-	944086.4	944217.6↑	**982247.2↑**	-	944166.1	944240.1↑	**981977.9↑**	924954.5

GNN&GBDT and Light-MILPopt in MVC and MIS problems, where GNN&GBDT often fails to handle large-scale instances efficiently.

Light-EvoOPT also proves advantageous compared to Light-MILPopt, achieving further improvements in most problems within fixed solution times. Remarkably, the pruned variant OursP achieves nearly identical or better performance compared to Ours, despite removing 20% of its network parameters. The lighter network structure in OursP further enhances efficiency and scalability, enabling it to outperform Ours on most large-scale and ultralarge-scale test problems.

The superior performance of Light-EvoOPT can be attributed to several key factors. First, its architecture focuses computational resources on the most critical problem components through constraint reduction and small-scale subgraph division. For example, in SC problems, reducing constraints cuts the time per optimization iteration to one-fifth of the original. Second, small-scale subgraphs improve generalization to larger, more complex problems by leveraging patterns from smaller instances. Finally, network pruning in OursP reduces computational overhead, simplifying the network without sacrificing solution quality. This balance between efficiency and accuracy underpins Light-EvoOPT's strong performance across diverse benchmarks.

In summary, Light-EvoOPT, and particularly its pruned variant OursP, shows exceptional scalability and efficiency in solving large-scale MILPs, maintaining a substantial lead over traditional solvers and learning-based methods.

C. Comparisons of Solution Efficiency

To further validate the efficiency of the Light-EvoOPT, we compare the running time of the proposed framework with respect to that of the baseline algorithms with fixed-solving results in medium-scale and large-scale test problems. Our experimental results are shown in Table VI (using a scale-constrained version of Gurobi as the lightweight optimizer) and Table VII (using a scale-constrained version of SCIP as the lightweight optimizer). It is evident that Light-EvoOPT significantly reduces the time required to obtain the same optimization results compared to all the baseline approaches on all MILPs.

Specifically, compared to the large-scale baseline solvers, the proposed framework can achieve the same results in only 0.5% of the time for the benchmark MILPs, including SC_1, MVC_1, MIS_1 and MIKS_1. Even more surprisingly, on the MIS_1, Light-EvoOPT only using a scale-limited version solver with variable proportion $\alpha = 50\%$ can achieve the same results as the state-of-the-art solver using only 0.1% of the time. Compared to the ML-based frameworks GNN&GBDT, our Light-EvoOPT can save more than 95% of the solution time on most MILPs to achieve the same results. On more than half of the benchmark MILPs, our Light-EvoOPT can save even more than 99% of the running time. Even when compared with the latest Light-MILPopt framework, although Light-EvoOPT and Light-MILPopt did not pull apart in the previous section's comparison, Light-EvoOPT can save more than 50% of the solution time on most MILPs to achieve the same results in most of the benchmark MILPs. It is interesting to note this fact for MVC_1 and MIS_1. In addition, although there is little difference among all methods in achieving the same optimization results within a specific time frame, there is a significant difference in their execution times to achieve a target value. We can also clearly see that the improvement in efficiency by introducing model pruning is very substantial, and even with only 20% of the model parameters subtracted. The pruned Light-EvoOPT achieves the best results by being more efficient than the original model on most problems.

This dramatic reduction in solution time is primarily due to the ability of Light-EvoOPT to effectively simplify the problem through variable and constraint reduction. By focusing on a limited but critical subset of variables, Light-EvoOPT avoids the exponential growth in complexity that typically hampers traditional solvers like SCIP and Gurobi when dealing with large-scale MILPs. For example, in problems like MIS_1, Light-EvoOPT achieves the target result in just 0.1% of the time required by Gurobi, demonstrating its exceptional efficiency in navigating the solution space with minimal computational burden. Furthermore, Light-EvoOPT's efficiency advantage over ML-based frameworks such as GNN&GBDT and Light-MILPopt is particularly remarkable. The lightweight optimizer, combined with the model pruning techniques, significantly reduces computational overhead while maintaining high solution quality. This is particularly evident in problems like MVC_1 and MIS_1, where Light-EvoOPT achieves the same results as Light-MILPopt yet in less than half the time, showcasing its superior ability to balance computational efficiency with optimization precision. Finally, the use of network pruning further amplifies the efficiency gains. By pruning 20% of the model parameters, the pruned Light-EvoOPT not only retains comparable accuracy but also reduces computation time dramatically. This is particularly beneficial for large-scale problems where the lighter network structure enables faster convergence. In cases like MIKS_3, the pruned version of Light-EvoOPT consistently outperforms the original model in terms of efficiency, highlighting the importance of pruning in enhancing the scalability and speed of the framework.

D. Analysis of Convergence

Convergence is an essential metric for evaluating the performance of optimization frameworks. To analyze the convergence of Light-EvoOPT, we record the trend of the objective value with the iteration time of the proposed framework and of the baseline algorithm on four standard benchmark MILPs, including SC, MVC, MIS and MIKS. The time-objective variation is visualized in Fig. 7. We can see that the proposed framework can obtain high-quality solutions for large-scale MILPs with only small-scale training data and a lightweight optimizer. Fig. 7 also illustrates that the convergence performance of Light-EvoPOT is not weaker than that of the state-of-the-art solver Gurobi as well as that of the state-of-the-art ML-based optimization framework.

The convergence performance of Light-EvoOPT is particularly impressive when considering that it operates on small-scale training data and uses a lightweight optimizer. This allows the framework to outperform or match the performance of more resource-intensive solvers such as Gurobi and ML-based frameworks like GBDT, which typically require extensive data preparation and training. As shown in Fig. 7, for problems such as SC_1, MVC_1, and MIS_1, Light-EvoOPT converges rapidly, reaching near-optimal solutions in fewer iterations compared to GBDT and even Gurobi. This rapid convergence highlights the effectiveness of the problem simplifications and variable reductions employed by Light-EvoOPT, which allows it to focus its computational effort on the most critical aspects of the MILP. Moreover, the stability of the convergence curves across different problem types further underscores the robustness of Light-EvoOPT. In problems like SC_2 and MIKS_2, the framework consistently achieves a smooth and steady reduction in the objective value, unlike some of the baseline methods (e.g., GBDT-30%G), which show erratic jumps or plateaus during the optimization process. This suggests that Light-EvoOPT is less prone to getting stuck in local optima or experiencing sudden increases in computational complexity, making it a more reliable choice for large-scale optimization problems.

Another key strength of Light-EvoOPT is its efficiency in handling complex constraints, as demonstrated in the MIKS problems. The framework quickly converges to high-quality solutions even in cases where the constraints are

TABLE VI

COMPARISON OF EXECUTION TIMES UNDER THE SAME TARGET VALUE ADOPTING GUROBI AS THE LIGHTWEIGHT OPTIMIZER UNDER THE TARGET SOLUTION. A GREATER-THAN SYMBOL (>) INDICATES THE INABILITY TO ACHIEVE THE TARGET OBJECTIVE FUNCTION IN SOME INSTANCES WITHIN THE MAXIMUM RUNNING TIME. BOLDFACE IS USED TO DENOTE THE BEST RESULTS

	GBDT-30%G	MILP-30%G	Ours-30%G	OursP-30%G	GBDT-50%G	MILP-50%G	Ours-50%G	OursP-50%G	Gurobi
SC$_1$	>30347.8s	1166.8s	218.0s↑	**194.1s↑**	5041.6s	177.8s	116.0s↑	**103.8s↑**	>60000s
SC$_2$	>60000s	5645.0s	4459.1s↑	**2912.8s↑**	>60000s	1795.4s	1300.1s↑	**1299.1s↑**	>60000s
SC$_3$	*	14787.3s	**13720.4s↑**	14345.7s↑	*	10164.0s	10677.5s	**9581.4s↑**	>80000s
MVC$_1$	>60000s	1475.3s	417.3s↑	**342.5s↑**	29320.5s	193.8s	**138.0s↑**	172.0s↑	>60000s
MVC$_2$	>60000s	6453.3s	**4020.9s↑**	4261.0s↑	21397.3s	1503.3s	**1164.2s↑**	1460.5s↑	>60000s
MVC$_3$	*	43687.1s	**30912.8s↑**	37392.2s↑	*	13896.6s	**12652.1s↑**	13379.5s↑	>80000s
MIS$_1$	>60000s	1487.3s	413.4s↑	**396.6s↑**	4227.1s	223.5s	124.7s↑	**62.5s↑**	>60000s
MIS$_2$	>60000s	7250.5s	**3663.2s↑**	4307.3s↑	27952.9s	2062.7s	**1190.2s↑**	1288.9s↑	>60000s
MIS$_3$	*	46518.7s	**23488.7s↑**	28306.9s↑	*	43748.0s	**15163.2s↑**	17688.0s↑	>80000s
MIKS$_1$	-	593.9s	238.1s↑	**221.3s↑**	-	160.5s	**117.2s↑**	121.6s↑	>60000s
MIKS$_2$	-	7941.9s	7408.6s↑	**3834.2s↑**	-	2137.8s	1800.5s↑	**1768.4s↑**	>60000s
MIKS$_3$	-	25735.3s	24788.1s↑	**17579.6s↑**	-	17976.7s	14023.6s↑	**10882.5s↑**	>60000s
Case	-	**511.5s**	568.6s	555.6s	-	506.9s	**476.6s↑**	542.2s	2584.7s

TABLE VII

COMPARISON OF EXECUTION TIMES UNDER THE SAME TARGET VALUE ADOPTING SCIP AS THE LIGHTWEIGHT OPTIMIZER UNDER THE TARGET SOLUTION. BOLDFACE IS USED TO DENOTE THE BEST RESULTS

	GBDT-30%S	MILP-30%S	Ours-30%S	OursP-30%S	GBDT-50%S	MILP-50%S	Ours-50%S	OursP-50%S	SCIP
SC$_1$	>48369.2s	1998.1s	285.7s↑	**233.5s↑**	587.6s	352.2s	223.5s↑	**191.1s↑**	>60000s
SC$_2$	>60000s	11823.0s	**4788.9s↑**	8471.0s↑	297.6s	11441.3s	9183.3s↑	**7851.1s↑**	>60000s
SC$_3$	*	73126.5s	**19269.6s↑**	23902.9s↑	*	>80000s	30584.5s↑	**19498.6s↑**	>80000s
MVC$_1$	>60000s	1951.6s	461.7s↑	**404.8s↑**	297.6s	203.1s	142.8s↑	**124.1s↑**	>60000s
MVC$_2$	>60000s	7967.2s	**4764.4s↑**	6412.9s↑	7570.5s	1815.3s	**1418.72s↑**	2159.5s	>60000s
MVC$_3$	*	78018.4s	40892.9s↑	**36067.2s↑**	*	>80000s	17209.2s↑	**16543.8s↑**	>80000s
MIS$_1$	>60000s	1951.6s	467.8s↑	**463.2s↑**	348.6s	225.9s	143.2s↑	**65.2s↑**	>60000s
MIS$_2$	>60000s	7967.2s	**4419.4s↑**	4561.3s↑	5920.7s	1945.7s	**1405.5s↑**	3029.1s↑	>60000s
MIS$_3$	*	81048.4s	**29329.2s↑**	31373.2s↑	*	74536.7s	14723.5s↑	**14454.8s↑**	>80000s
MIKS$_1$	-	1982.0s	288.5s↑	**258.9s↑**	-	194.9s	170.7s↑	**136.4s↑**	>60000s
MIKS$_2$	-	11980.4s	10911.7s↑	**4770.0s↑**	-	9576.1s	7055.9s↑	**2130.6s↑**	>60000s
MIKS$_3$	-	37384.1s	40764.8s	**33675.5s↑**	-	63558.0s	50200.0s↑	**36751.7s↑**	>60000s
Case	-	996.4s	937.8s↑	**544.2s↑**	-	776.2s	937.4s	**559.2s↑**	3097.0s

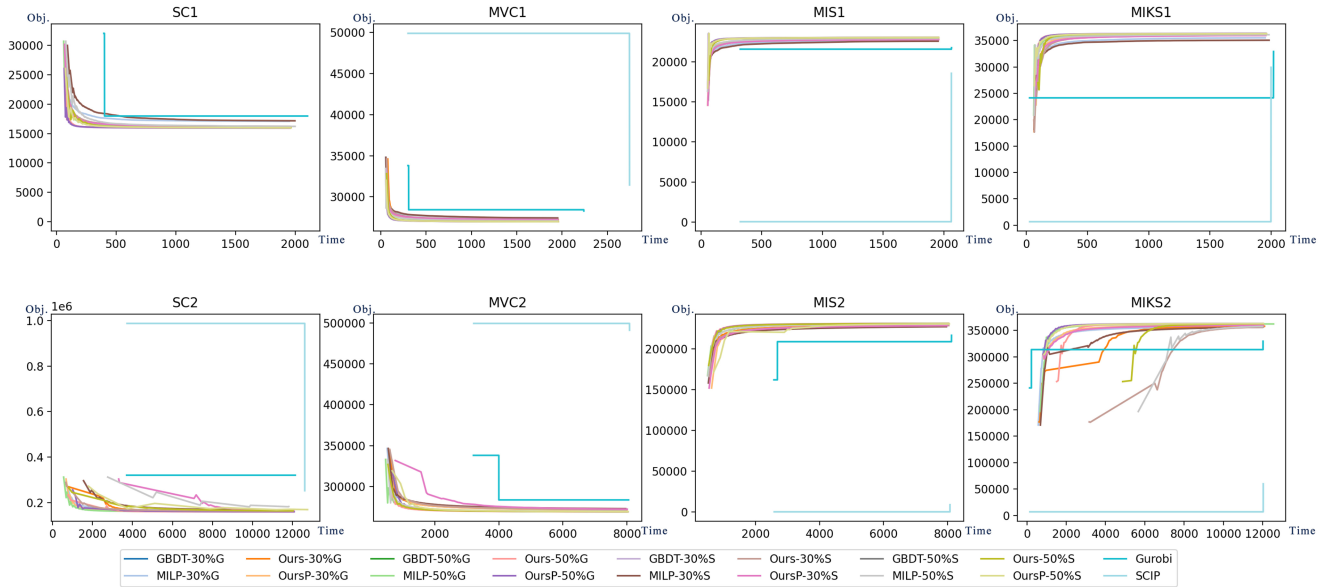

Fig. 7. Time-objective figure for the benchmark MLIPs. (SC$_1$, SC$_2$) The minimized SC problem. (MVC$_1$, MVC$_2$) The minimized MVC problem. (MIS$_1$, MIS$_2$) The maximized MIS problem. (MIKS$_1$, MIKS$_2$) The maximized MIKS problem. The x-axis represents the optimization time (seconds), and the y-axis represents the objective values for the corresponding problems.

more intricate and difficult to satisfy. This is especially evident in MIKS$_1$, where Light-EvoOPT outperforms Gurobi in terms of both convergence speed and final objective value. The ability to handle complex constraints efficiently is a direct result of Light-EvoOPT's architecture, which effectively balances computational effort between constraint satisfaction and objective optimization. Finally, the OursP variant, which incorporates network pruning, further enhances convergence performance. By reducing the number of parameters after initial training, the pruned model becomes more streamlined,

TABLE VIII
AVERAGE TRAINING TIMES FOR DIFFERENT ML-BASED METHODS WHEN SOLVING LARGE-SCALE PROBLEMS

Stage	GBDT	MILP	Ours	OursP
Data Preparation	308060s	20100s	20100s	20100s
GNN Training	25271s	229s	365s	365s
GBDT Training	45749s	-	-	-
Pruning	-	-	-	101s
Total Time	379080s	20329s	20465s	20566s

allowing for faster convergence without sacrificing solution quality. In Fig. 7, this is clearly seen in problems like MVC_2 and MIS_2, where the pruned model (OursP) consistently converges faster than the unpruned version (Ours), while maintaining comparable objective values. This demonstrates that network pruning not only improves computational efficiency but also accelerates the optimization process, making Light-EvoOPT an even more powerful tool for large-scale MILPs.

E. Analysis of Training Efficiency

Table VIII provides a comparison of training times for ML-based methods across key stages, including data preparation, network training, and pruning (for OursP). The results show that Light-EvoOPT, in both its original (Ours) and pruned (OursP) forms, achieves significantly reduced training times compared to GBDT. This efficiency is primarily due to Light-EvoOPT's ability to generalize from small-scale problem instances, which minimizes data preparation requirements and reduces overall training time.

GBDT's training process requires over 379 000 s, largely due to its extensive data preparation phase, which scales with the size of the target problem. In contrast, Light-EvoOPT maintains a total training time of just over 20 000 s, comparable to the MILPopt framework. Remarkably, the pruning process in OursP introduces only 101 additional seconds, a negligible overhead, while improving model efficiency and maintaining high performance. This highlights the practicality of OursP, as it achieves similar or better results with a lighter network and minimal retraining effort. The scalability of Light-EvoOPT is further underscored by its minimal GNN training time (365 s), which is orders of magnitude faster than GBDT's data preparation and training. This efficiency is particularly advantageous for large-scale problems with millions of decision variables and constraints, as the training time remains manageable even as problem size grows.

In summary, Light-EvoOPT, especially its pruned variant OursP, provides a clear advantage in training efficiency over GBDT and other ML-based methods. Its ability to generalize from small-scale data and leverage efficient pruning ensures competitive performance while keeping training times low, making it a scalable and practical option for solving large-scale optimization problems.

V. CONCLUSION AND FUTURE WORK

This article proposes *Light-EvoOPT*, a lightweight optimization framework for large-scale MILPs, and further explores the important role of model pruning for AI4Optimize. Light-EvoOPT uses graph partition-based problem division and EGAT with half-convolutions to efficiently predict initial MILP solutions with only a lightweight training dataset. Through variables and constraints reduction, Light-EvoOPT rapidly updates the current solution with a lightweight optimizer. Experimental evaluations on four standard large-scale MILPs and a real-world case study show that our framework outperforms SCIP, Gurobi, the GNN&GBDT-based optimization framework, and Light-MILPopt.

Despite its promising results, Light-EvoOPT has some limitations. First, the framework requires a large amount of small-scale training data, which could be impractical in certain scenarios. Second, it is currently limited to solving linear problems and cannot handle nonlinear constraints. Finally, the framework's ability to generalize is restricted to solving problems that are homogeneous with the training data, as it lacks the capability to generalize to significantly different problem types using the same network parameters. As part of our future work, we aim to address these limitations. We plan to develop data generators to produce synthetic examples for data augmentation, enhancing the training set. In addition, we will explore the use of hypergraph structures to extend the framework's applicability to nonlinear problems. Finally, we intend to introduce larger network architectures and leverage pretraining techniques to improve generalization capabilities, ultimately creating a base model that can be applied to a broader range of problems.

REFERENCES

[1] L. A. Wolsey, *Integer Programming*. Hoboken, NJ, USA: Wiley, 2020.

[2] M. I. A. Shekeew and B. Venkatesh, "Learning-assisted variables reduction method for large-scale MILP unit commitment," *IEEE Open Access J. Power Energy*, vol. 10, pp. 245–258, 2023.

[3] H. Ye, H. Xu, and H. Wang, "Light-MILPopt: Solving large-scale mixed integer linear programs with lightweight optimizer and small-scale training dataset," in *Proc. 12th Int. Conf. Learn. Represent.*, 2024, pp. 1–24.

[4] Z. Guo, C. S. Lai, P. Luk, and X. Zhang, "Techno-economic assessment of wireless charging systems for airport electric shuttle buses," *J. Energy Storage*, vol. 64, Aug. 2023, Art. no. 107123.

[5] Q. Han et al., "A GNN-guided predict-and-search framework for mixed-integer linear programming," 2023, *arXiv:2302.05636*.

[6] K. Deb, P. Fleming, Y. Jin, K. Miettinen, and P. M. Reed, "Key issues in real-world applications of many-objective optimisation and decision analysis," in *Many-Criteria Optimization and Decision Analysis: State-of-the-Art, Present Challenges, and Future Perspectives*. Cham, Switzerland: Springer, 2023, pp. 29–57.

[7] M. Gasse, D. Chételat, N. Ferroni, L. Charlin, and A. Lodi, "Exact combinatorial optimization with graph convolutional neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 32, 2019, pp. 1–13.

[8] E. L. Lawler and D. E. Wood, "Branch-and-bound methods: A survey," *Oper. Res.*, vol. 14, no. 4, pp. 699–719, 1966.

[9] S. Boyd and J. Mattingley, "Branch and bound methods," Notes EE364b, Stanford Univ., Stanford, CA, USA, 2007.

[10] V. Nair et al., "Solving mixed integer programs using neural networks," 2020, *arXiv:2012.13349*.

[11] N. Sonnerat, P. Wang, I. Ktena, S. Bartunov, and V. Nair, "Learning a large Neighborhood search algorithm for mixed integer programs," 2021, *arXiv:2107.10201*.

[12] H. Ye, H. Xu, H. Wang, C. Wang, and Y. Jiang, "GNN&GBDT-guided fast Optimizing framework for large-scale integer programming," in *Proc. Int. Conf. Mach. Learn.*, 2023, pp. 39864–39878.

[13] T. Achterberg, "Constraint integer programming," Ph.D. dissertation, Dept. Math. Natural Sci. Tech., Univ. Berlin, Berlin, Germany, 2007.

[14] A. Schrijver, *Theory of Linear and Integer Programming*. Hoboken, NJ, USA: Wiley, 1998.

[15] M. G. Bailey and B. E. Gillett, "Parametric integer programming analysis: A contraction approach," *J. Oper. Res. Soc.*, vol. 31, no. 3, pp. 257–262, 1980.
[16] K. G. Murty and F.-T. Yu, *Linear Complementarity, Linear and Nonlinear Programming*, vol. 3. Berlin, Germany: Heldermann, 1988.
[17] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," 2016, *arXiv:1609.02907*.
[18] T. Yoon, "Confidence threshold neural diving," 2022, *arXiv:2202.07506*.
[19] L. Gong and Q. Cheng, "Exploiting edge features for graph neural networks," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2019, pp. 9211–9219.
[20] J. Xu, Z. Li, B. Du, M. Zhang, and J. Liu, "Reluplex made more practical: Leaky ReLU," in *Proc. IEEE Symp. Comput. Commun. (ISCC)*, 2020, pp. 1–7.
[21] R. Zass and A. Shashua, "Doubly stochastic Normalization for spectral clustering," in *Proc. Adv. Neural Inf. Process. Syst.*, 2006, pp. 1–8.
[22] J. O. Neill, "An overview of neural network compression," 2020, *arXiv:2006.03669*.
[23] J. Lin, Y. Rao, J. Lu, and J. Zhou, "Runtime neural pruning," in *Proc. 31st Conf. Neural Inf. Process. Syst.*, vol. 30, 2017, pp. 1–11.
[24] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained Quantization and Huffman coding," 2015, *arXiv:1510.00149*.
[25] C. Tsourakakis, C. Gkantsidis, B. Radunovic, and M. Vojnovic, "Fennel: Streaming graph partitioning for massive scale graphs," in *Proc. 7th ACM Int. Conf. Web Search Data Min.*, 2014, pp. 333–342.
[26] Z. Chen, J. Liu, X. Wang, and W. Yin, "On representing linear programs by graph neural networks," in *Proc. 11th Int. Conf. Learn. Represent.*, 2023, pp. 1–29.
[27] C. Shen, Q. Wang, and C. E. Priebe, "One-hot graph encoder embedding," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 45, no. 6, pp. 7933–7938, Jun. 2022.
[28] Y. Geifman and R. El-Yaniv, "SelectiveNet: A deep neural network with an integrated reject option," in *Proc. Int. Conf. Mach. Learn.*, 2019, pp. 2151–2159.
[29] T.-Y. Lin, P. Goyal, R. B. Girshick, K. He, and P. Dollár, "Focal loss for dense object detection," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2017, pp. 2980–2988.
[30] T. P. Runarsson and X. Yao, "Search biases in constrained evolutionary optimization," *IEEE Trans. Syst., Man, Cybern., Part-C (Appl. Rev.)*, vol. 35, no. 2, pp. 233–243, May 2005.
[31] H. Ye, H. Wang, H. Xu, C. Wang, and Y. Jiang, "Adaptive constraint partition based optimization framework for large-scale integer linear programming (student abstract)," in *Proc. AAAI Conf. Artif. Intell.*, 2023, pp. 16376–16377.
[32] A. Caprara, P. Toth, and M. Fischetti, "Algorithms for the set covering problem," *Ann. Oper. Res.*, vol. 98, no. 1, pp. 353–371, 2000.
[33] I. Dinur and S. Safra, "On the hardness of approximating minimum vertex cover," *Ann. Math.*, vol. 162, no. 1, pp. 439–485, 2005.
[34] R. E. Tarjan and A. E. Trojanowski, "Finding a maximum independent set," *SIAM J. Comput.*, vol. 6, no. 3, pp. 537–546, 1977.
[35] A. Atamtürk, "On the facets of the mixed-integer knapsack polyhedron," *Math. Program.*, vol. 98, no. 1, pp. 145–175, 2003.
[36] T. Achterberg, "SCIP: Solving constraint integer programs," *Math. Program. Comput.*, vol. 1, pp. 1–41, Jul. 2009.
[37] *IBM ILOG CPLEX Optimization Studio, Version 12*, CPLEX, Bethesda, MD, USA, 1987.
[38] (Gurobi Optim., LLC, Beaverton, OR, USA). *Gurobi Optimizer Reference Manual*. 2023. [Online]. Available: https://www.gurobi.com

Huigen Ye (Graduate Student Member, IEEE) received the B.S. degree from the School of Computer Science and Engineering, Sun Yat-sen University, Guangzhou, China, in 2023. He is currently pursuing the Ph.D. degree with the State Key Laboratory of Intelligent Technology and Systems, Department of Computer Science and Technology, Tsinghua University, Beijing, China.

His research focuses on large-scale optimization, aiming to solve complex real-world problems by leveraging machine learning techniques to improve computational efficiency.

Hua Xu (Member, IEEE) received the Ph.D. degree in computer science from Tsinghua University, Beijing, China, in 2000.

He is currently a Tenured Associate Professor with the Department of Computer Science, Tsinghua University, Beijing, China. He has authored 30 top international conference papers as the first or corresponding author and 50 high-impact SCI international journal papers (35 in the first quartile, with 40 having an impact factor >5.0 and 2 highly cited in ESI), accumulating 2000 SCI citations and 8700 Google Scholar citations. He has authored 4 textbooks and 10 academic monographs, such as *Intelligent Evolutionary Optimization* (Elsevier, 2024), *Multimodal Sentiment Analysis* (Springer, 2023), and *Intent Recognition for Human Machine Interactions* (Springer, 2023). He holds 36 granted patents for inventions and 26 granted software copyrights. His contributions have been cited in top journals, such as *ACM Computing Surveys*, with citations from 6 domestic and international academicians and over 30 IEEE Fellows. His primary research focuses on intelligent optimization and human–machine natural interaction in the field of artificial intelligence, leveraging research platforms, such as the National Research Center for Information Science and Technology in Beijing.

Dr. Xu has previously been honored with the National Science and Technology Progress Award (Second Class), the Beijing Science and Technology Award (First Class), two Industry Association Science and Technology Awards (First Class), and three second and third-class awards at the provincial and ministerial levels. He has been invited to serve as an Editor-in-Chief for Elsevier's international journals *Intelligent Systems with Applications* and an Associate Editor for *Expert Systems with Applications* (1st quartile, SCI IF=8.665).

Carlos A. Coello Coello (Fellow, IEEE) received the Ph.D. degree in computer science from Tulane University, New Orleans, LA, USA, in 1996.

He is a Professor (CINVESTAV-3F Researcher) with the Department of Computer Science of CINVESTAV-IPN, Mexico City, Mexico. He has authored and co-authored over 550 technical papers and book chapters. He has also co-authored the book *Evolutionary Algorithms for Solving MultiObjective Problems* (Second Edition, Springer, 2007). His publications currently report over 77 600 citations in Google Scholar (his H-index is 106). His research interests include evolutionary multiobjective optimization and constraint-handling techniques for evolutionary algorithms.