A self-supervised joint embedding predictive approach for bipartite graphs to solve linear programs

Joachim Verschelde
Computer Science Department,
Open Universiteit,
The Netherlands

Daniel Stanley Tan
Computer Science Department,
Open Universiteit,
The Netherlands

Stefano Bromuri
Computer Science Department,
Open Universiteit,
The Netherlands

***Abstract*—Linear programs (LPs) are typically solved with iterative algorithms such as simplex or interior-point methods, which enforce feasibility and optimality but treat each instance independently. End-to-end neural solvers instead aim to map LPs directly to solutions in a single forward pass, yet most require large datasets of optimal solutions or rely on classical solvers during training. We present a fully self-supervised framework that predicts primal–dual LP solutions without ground-truth supervision. First, we introduce BiJEPA, a joint-embedding predictive pretraining objective for bipartite variable–constraint graphs. By predicting global embeddings from masked local views, BiJEPA enables large-scale pretraining on unsolved LPs. Second, we fine-tune a bipartite graph neural network (GNN) using a differentiable residual loss based solely on LP structure via the Karush–Kuhn–Tucker conditions. Across four LP families, our method learns an end-to-end solver with low optimality gaps, demonstrating that optimization structure can replace external supervision when training neural LP solvers.**

I. Introduction

Mathematical optimization—especially linear programming (LP) and mixed-integer programming—supports resource allocation, scheduling, routing, and large-scale decision-making across science and industry. Classical solvers based on simplex, interior-point methods, and branch-and-bound offer strong guarantees [1], [2], but typically solve each instance independently. In many applications, related instances recur with shared structure (e.g., daily logistics, repeated planning, or low-latency embedded decision-making), motivating approaches that exploit experience across instance distributions [3]. Neural methods provide a complementary route: instead of re-solving each instance, a model can map problem data directly to candidate solutions. Existing ML–optimization work largely falls into two categories. ML-augmented solvers accelerate components of classical pipelines (e.g., branching or cut selection) while keeping the solver in the loop [4], [5]. End-to-end approaches instead predict solutions directly from instance features [6], [7], enabling fast inference but raising the question of how to obtain reliable learning signals. Many methods require large supervised datasets of solved instances or rely on penalty-based objectives that may not enforce feasibility or optimality and still depend on solvers for repair or verification [5].

This paper explores the problem of learning to solve LPs without solution labels. Optimization problems already contain rich internal structure—objectives, constraints, and optimality conditions—that can serve as supervision. This aligns with the broader AI theme of learning from environmental structure rather than external labels, in the case of LPs the Karush–Kuhn–Tucker (KKT) conditions characterize optimal primal–dual solutions under mild assumptions [8]. Training a model to output variables satisfying these conditions allows the LP itself to provide the learning signal. We propose a fully self-supervised pipeline combining two complementary signals. First, we introduce BiJEPA, a graph adaptation of Joint Embedding Predictive Architectures for the bipartite variable–constraint structure of an LP [9]. BiJEPA pretrains a bipartite GNN encoder by generating masked views of an instance and predicting a global embedding from heavily masked local contexts. Because this objective depends only on the LP description, it enables large-scale pretraining on unsolved instances and yields permutation-invariant representations transferable across families and sizes. Second, we train a solver head using a differentiable KKT residual loss, inspired by KKT-Net objectives [10], which penalizes violations of feasibility, complementary slackness, and stationarity. This stage remains label-free: the loss is computed solely from (A, b, c) and the network outputs, without requiring solver-generated solutions. This work makes the following contributions: we propose BiJEPA for LP graphs, a self-supervised pretraining objective for bipartite LP graphs that learns transferable, permutation-invariant representations by predicting global embeddings from masked local views; we train bipartite GNN solvers using a differentiable KKT residual objective, enabling unsupervised learning of primal–dual solutions and evaluating the approach across multiple LP families and sizes, including an analysis of the benefits of BiJEPA pretraining in low-compute fine-tuning regimes.

II. Related Work

A. ML based solvers for Linear Programs (LPs)

In recent times, several works emerged focusing on combining neural networks with KKT conditions. Wu et al. [11]

used the KKT conditions explicitly as (boundary) conditions for a first-order ODE (dynamic system). The idea is that this dynamic system converges, as $t \to \infty$, to a state that satisfies the KKT conditions. It is claimed that the solution can thus be found. However, for each LP, a new neural network must be trained, which makes the method very slow and computationally intensive. Chen et al. [12] provide an attempt at modeling an LP as a bipartite GNN. Their goal is to generally study solution methods for LPs via bipartite GNNs, where the main challenge remains how to determine reliably the objective values and solutions from the network. Qian et al. [13] provide an explanation of how GNNs can be used to solve LPs. Concretely, the Message Passing steps in the GNN are viewed as the steps of an interior-point algorithm.

B. ML based approaches for Mixed Integer Linear Programs (MILPs)

Scavuzzo [14] reviewed the current literature on the use of Machine Learning for MILP and categorized the approaches into three main groups:

1) Primal heuristics: A GNN is used to solve a (sub)part of the problem, corresponding to stable variables. The sub-problem with the remaining variables is then solved by a solver (usually SCIP). It has been established that for many binary MILPs, in numerous feasible solutions, certain variables consistently take the same values in all (or in most) feasible solutions. These are called stable variables. Several GNN-based approaches predict stable variables, after which the remaining subproblem is passed to a solver. Ding et al. [15] use a supervised attention-based GNN trained with binary cross-entropy to identify these variables, followed by a SCIP local-branching step. Nair et al. [4] extend this idea with a two-part architecture—Neural Diving and Neural Branching—where the former is a refined variant of Ding et al.'s primal heuristic with an adjusted probability loss. The Neural Branching GNN aims to predict which variables will undergo Full Strong Branching, so that the root search tree remains as small as possible. Imitation learning is used here, following the expert policy from SCIP. With respect to the contribution discussed in this paper, these approaches are only semi end-to-end in the sense that part of the MILP is determined by the neural network, but a solver is still needed to complete the solution.

2) Branch and Bound: These methods are not end-to-end but guide solver decisions with branch and bound. Gasse et al. [16] use a bipartite GNN trained via imitation learning to predict branching variables from SCIP expert data. Zarpellon et al. [17] instead model the search tree itself, claiming better generalization across problem classes. Lin et al. [18] follow this tree-based approach using a transformer with multitask learning, supervised by SCIP.

C. End-to-End Methods for both LP and MILPs

End-to-end ML methods for LPs and MILPs try to solve the problem by specifying an appropriate loss or approach to the solution. Tang et al. [6] propose a network that solves the LP relaxation and then applies a constraint layer to project the output onto integer values, training with an objective plus a penalty loss for inequalities. Lee et al. [7] instead use reinforcement learning, where actions adjust variable values; their method first rewards reducing constraint violations to reach feasibility, and then rewards improving the objective while maintaining feasibility.

III. Methods

A. Graph Neural Networks

We use GNNs to encode optimization instances of varying sizes. MLPs flatten the problem data into a single fixed-length vector, instead GNNs represent each instance as a graph whose nodes and edges reflect the problem's structure as can be seen in figure 1. A typical example in the LP context is the bipartite graph representation, which treats decision variables and constraints as two distinct sets of nodes.

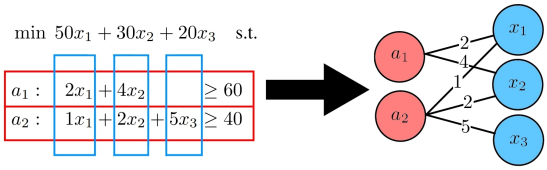

Fig. 1: A small LP (left) with its coefficient matrix A in the middle. Rows (constraints) become red nodes a_1, a_2, while columns (variables) become blue nodes x_1, x_2, x_3. Edges represent nonzero A_{ij}.

To construct the bipartite graph, one set of nodes corresponds to the constraints (rows of A) and the other set corresponds to the decision variables (columns). An edge exists between a constraint node and a variable node if and only if the corresponding coefficient in A is nonzero. Each node then holds features (e.g., objective coefficients c_j, right-hand sides b_i, or bounds), and during message passing, constraints pass information to variables and variables pass information back to constraints. This two-way flow of information helps the network learn to minimize violations, guide feasibility, and push solutions toward optimality. Since GNNs process nodes based on their connectivity rather than relying on a specific node ordering, the architecture is invariant under permutations. This is especially useful for LPs, where the labeling or order of constraints and variables should not affect the solution. Whereas MLP inputs must be ordered in some way (and shuffling the input could produce different outputs), GNNs aggregate messages from neighbors in a symmetric fashion (e.g., summation or averaging), ensuring the model's output remains consistent regardless of how the problem is indexed.

B. BiJEPA: Joint Embedding Predictive Pretraining for Bipartite LP Graphs

In our task to learn bipartite graphs, we introduce a pretraining stage inspired by Joint Embedding Predictive Architectures (JEPAs) [9]. The goal is to learn node representations that capture the structural relationships between constraints and variables in the bipartite graph, before training the solver heads, so fewer instances are needed to get good results.

1) Graph views via masking: Let an LP instance be represented as a bipartite graph $G = (V_c, V_v, E)$, where constraint nodes $i \in V_c$ carry features u_i (e.g., right-hand side or derived statistics), variable nodes $j \in V_v$ carry features v_j (e.g., objective coefficient and bounds-related features), and edges $(i, j) \in E$ carry edge features e_{ij} corresponding to the nonzero coefficient A_{ij}. To form multiple views of the same instance, we mask a random subset of constraint nodes, variable nodes, and edges. Importantly, we do not remove nodes (i.e., we do not sample subgraphs), so node identities and ordering remain consistent across views. Masking is implemented by replacing the corresponding embedded features with learnable mask tokens:

$$\tilde{h}_i = \begin{cases} t_c, & \text{if } i \text{ is masked (constraint)} \\ h_i, & \text{otherwise} \end{cases}$$
$$\tilde{h}_j = \begin{cases} t_v, & \text{if } j \text{ is masked (variable)} \\ h_j, & \text{otherwise} \end{cases} \tag{1}$$

and the same goes for masked edges with token t_e. This preserves the bipartite connectivity and makes view-to-view prediction well-defined at the node level. We generate two types of views: (i) global views that are lightly masked (high-information, used as targets), and (ii) local views that are heavily masked (low-information, used as contexts).

2) Node embedding and predictive alignment: Let f_θ be the bipartite GNN encoder. For a masked view $G^{(k)}$, we compute node embeddings $Z^{(k)} = f_\theta(G^{(k)}) \in \mathbb{R}^{N \times D}$ where $N = |V_c| + |V_v|$ and $Z^{(k)}$ is formed by concatenating the constraint-node and variable-node embeddings (this matches our implementation, where constraint embeddings and variable embeddings are projected and concatenated). Following the JEPA principle of predicting in embedding space (rather than reconstructing raw inputs), we use the average embedding of the global views as a target representation. Given n_g global views, the target center is $\bar{Z} = \frac{1}{n_g} \sum_{g=1}^{n_g} Z^{(g)}$.

For each view (global or local), we encourage its embeddings to match $\bar{Z}$. We use a mean-squared error in embedding space:

$$\mathcal{L}_{\text{pred}} = \frac{1}{|\mathcal{V}|} \sum_{k \in \mathcal{V}} \frac{1}{N} \left\| Z^{(k)} - \bar{Z} \right\|_F^2 \tag{2}$$

where $\mathcal{V}$ denotes the set of all views (global + local).

3) Collapse prevention via distribution regularization: Predictive alignment alone can lead to representation collapse, where the model produces degenerate solutions and all embeddings become nearly constant. Past work prevents this collapse with mechanisms such as stop–gradient targets or teacher–student updates [9]. Balestriero et al. introduce a representation regularizer $\mathcal{L}_{\text{sig}}$ (SigReg) that promotes isotropic, non-collapsed embeddings by encouraging the empirical embedding distribution to match a reference distribution (standard normal) under random one-dimensional projections. The combined BiJEPA loss uses only one scalar hyperparameter $\lambda \in [0, 1]$ to trade off prediction alignment and regularization strength. The final BiJEPA pretraining objective is

$$\mathcal{L}_{\text{BiJEPA}} = (1 - \lambda)\, \mathcal{L}_{\text{pred}} + \lambda\, \mathcal{L}_{\text{sig}} \tag{3}$$

where $\lambda \in [0, 1]$ balances predictive alignment and representation regularization. This pretraining stage uses only LP instances (no optimal solution labels) and produces an encoder that is subsequently used by the solver trained with a KKT-based loss

C. KKT Loss Representation for End-to-End LP Solving

After pretraining, we train the model to output primal and dual variables by minimizing a differentiable surrogate of the Karush–Kuhn–Tucker (KKT) conditions [8]. Under mild regularity conditions, any primal-dual solution satisfying KKT is guaranteed to be optimal. By minimizing a loss function that measures violation of KKT conditions, we provide a signal to our network: the loss is zero if and only if the output represents an optimal solution for the given LP. This allows us to train the network in an unsupervised way. The KKT objective residual becomes the loss signal so we have no need for explicit solution labels. This enables end-to-end learning without requiring ground-truth optimal solutions as labels.

1) LP form and network outputs: We consider the linear program

$$\min_{x \in \mathbb{R}^n} \; c^\top x \quad \text{s.t.} \quad Ax \le b \tag{4}$$

where $A \in \mathbb{R}^{m \times n}$, $b \in \mathbb{R}^m$, and $c \in \mathbb{R}^n$. In our bipartite representation, each variable node j produces a scalar prediction $\hat{x}_j$ (primal variable), and each constraint node i produces a scalar prediction $\hat{\lambda}_i$ (dual multiplier). To enforce dual feasibility, we parameterize $\hat{\lambda}$ with a nonnegative softplus activation, so $\hat{\lambda} \ge 0$ by construction.

2) Differentiable KKT residuals: For inequality constraints $Ax \le b$, the KKT conditions consist of: (i) primal feasibility $Ax \le b$, (ii) dual feasibility $\lambda \ge 0$, (iii) complementary slackness $\lambda_i (b_i - a_i^\top x) = 0$ for all i, and (iv) stationarity of the Lagrangian, $c + A^\top \lambda = 0$.

Primal feasibility. Let the constraint violation vector be $g_p = A\hat{x} - b \in \mathbb{R}^m$ We penalize inequality violations with the squared ReLU penalty:

$$\mathcal{L}_{\text{p}} = \frac{1}{m} \left\| \text{ReLU}(g_p) \right\|_2^2 \tag{5}$$

Dual feasibility. The dual variables are parameterized to be nonnegative using a softplus activation, so no extra penalty is needed. But for completion if $\hat{\lambda}$ were unconstrained, we again could use the squared ReLU penalty:

$$\mathcal{L}_{\text{d}} = \frac{1}{m} \left\| \text{ReLU}(-\hat{\lambda}) \right\|_2^2 \tag{6}$$

Complementary slackness. Let the slack be $s = b - A\hat{x}$. Complementary slackness is enforced by

$$\mathcal{L}_{\text{c}} = \frac{1}{m} \|\hat{\lambda} \odot s\|_2^2 \quad s \in \mathbb{R}^m \tag{7}$$

where $\odot$ denotes element-wise multiplication. Intuitively, if a constraint is inactive ($s_i > 0$), then the loss encourages $\hat{\lambda}_i \to 0$, while for active constraints ($s_i \approx 0$), $\hat{\lambda}_i$ is free to take values required by stationarity.

Stationarity. The Lagrangian gradient residual is $r_s = c + A^\top \hat{\lambda} \in \mathbb{R}^n$. So to enforce stationarity, the loss becomes:

$$\mathcal{L}_{\mathrm{s}} = \frac{1}{m} \left\| c + A^\top \hat{\lambda} \right\|_2^2 \tag{8}$$

3) KKT loss: We combine the losses into:

$$\mathcal{L}_{\mathrm{KKT}(\alpha_1,\alpha_2,\alpha_3,\alpha_4)} = \alpha_1 \mathcal{L}_{\mathrm{p}} + \alpha_2 \mathcal{L}_{\mathrm{d}} + \alpha_3 \mathcal{L}_{\mathrm{c}} + \alpha_4 \mathcal{L}_{\mathrm{s}} \tag{9}$$

The loss is minimized when feasibility, complementary slackness, and stationarity are simultaneously satisfied. Under standard regularity assumptions, any primal-dual pair $(\hat{x}, \hat{\lambda})$ achieving $\mathcal{L}_{\mathrm{KKT}} \approx 0$ corresponds to an optimal solution of (4) [8], [10].

D. Dataset generation

To train and evaluate our GNN encoder, we need a large collection of feasible, bounded LP instances with varying sizes, together with a consistent graph representation. We therefore generate instances from 4 combinatorial optimization classes and convert them into LPs by relaxation.

1) Problem families and size control: We used Ecole [19] to generate instances from the following problem classes: Independent Set (IS), Set Cover (SC), Combinatorial Auction (CA), Capacitated Facility Location (CFL). Across all families, we generate instances for multiple size settings (in our experiments, the number of variables $n \in \{10, 50, 200\}$), and we store each instance on disk as a `.lp` file.

2) Relaxation to a standardized LP form: The generated instances originate as MILPs (binary/integer variables and potentially additional constructs). Since our method focuses on LP solving with a KKT-based loss, we convert every generated model into a linear program and enforce a consistent minimize objective direction by applying an LP relaxation routine that removes integrality constraints and by transforming maximization problems in minimization ones through negating all objective coefficients and flipping the model sense obtaining a problem with the same form as in equation (4). This ensures that all components (graph construction, KKT loss computation, and metrics) operate on a single canonical LP format.

3) Bipartite graph extraction and feature design: Each LP file is converted into a bipartite graph as described in Section III-A. We parse the `.lp` file with a modeling toolkit and construct a set of variable nodes V_v (one for each decision variable), a set of constraint nodes V_c (one per linear constraint with at least one nonzero coefficient), and an edge set E that contains an edge (i, j) whenever $A_{ij} \neq 0$. Each edge stores the signed coefficient value A_{ij} as its attribute. To ensure deterministic indexing across runs and consistent alignment between solver outputs and graph nodes, variables are ordered by name and constraints are ordered first by their number of nonzeros and then by a stable identifier.

a) Variable-node features.: For each variable x_j, we compute a 6-dimensional base feature vector $v_j = [c_j, \mathrm{avgcoef}(j), \deg(j), \mathrm{maxcoef}(j), \mathrm{mincoef}(j), \mathbb{I}\{j \in \mathbb{Z}\}$ where $\deg(j)$ is the number of incident nonzeros, and the coefficient statistics are computed over the incident edges.

b) Constraint-node features.: For each constraint node i, we compute $u_i = \left[\mathrm{avgcoef}(i),\ \deg(i),\ \mathrm{rhs}(i),\ \mathrm{sense}(i)\right]$ where $\mathrm{sense}(i) \in \{0, 1, 2\}$ encodes $\leq$, $\geq$, and $=$ constraints, respectively.

c) Positional features: Although message passing is permutation invariant, in practice it can be beneficial to provide a weak, deterministic "identity" signal. We therefore add a positional encoding to variable nodes by lexicographically ranking variables using their base features (objective coefficient, coefficient statistics, degree, and integrality flag). We then encode each variable's rank quantile as a fixed-length binary fraction (12 bits) and append these bits to v_j. This yields optimization variable feature vectors of dimension $6{+}12$ when positional features are enabled.

d) Instance-wise normalization.: To keep feature magnitudes comparable across sizes and families, we apply per-instance min–max normalization to node features (feature-wise). When positional bits are enabled, we normalize only the first 6 real-valued features and keep the bit features binary.

4) Sparse $\leq$-form for KKT residual computation: In addition to the bipartite graph used by the GNN, we construct a sparse matrix representation (A, b, c) used by the KKT loss in Section III-C. Since our KKT formulation assumes inequalities of the form $Ax \leq b$, we convert the parsed constraints into a unified $\leq$-only system:

- Equalities are split into two inequalities ($a^\top x \leq b$ and $-a^\top x \leq -b$).
- $\geq$ constraints are multiplied by -1 to obtain $\leq$ constraints.
- Finite variable bounds are added as additional inequality rows ($x_j \leq u_j$ and $-x_j \leq -\ell_j$).

Storing A in sparse COO format allows efficient computation of $A\hat{x}$ and $A^\top \hat{\lambda}$ via scatter operations, matching the sparse bipartite structure used during message passing.

a) Stored artifacts.: For each instance we store (i) the relaxed `.lp` model, (ii) a serialized graph bundle containing sparse A, bipartite edge indices/attributes, and node features, and (iii) for `val`/`test`, the solver-produced optimal objective and primal solution. This design makes data loading independent of the original modeling stack and enables fast iteration during training and evaluation.

IV. Experimental Results

A. Baselines for comparison

MLP baseline (MLP-BL) [10] This baseline follows the fully-connected approach of KKT Nets: the LP instance is represented as dense tensors (A, b, c) Training is driven by the KKT-based loss.

GNN baseline (GNN-BL) This is a variant of our bipartite-graph solver trained from scratch without any self-supervised pretraining. Each LP is encoded as a bipartite graph with constraint nodes and variable nodes, and edges corresponding to nonzeros of A. All parameters (numeric embeddings, message passing layers, and prediction heads) are optimized end-to-end using the same KKT-based loss used across all variants.

TABLE I: Optimality gap (%) in the Combinatorial Auction problem after E fine-tuning epochs for $n = 10$.

Method	E=1	2	5	10	20	30	50
MLP-BL	12.8 ± 8.2	10.3 ± 7.8	35.6 ± 16.9	29.1 ± 12.9	27.5 ± 12.1	28.8 ± 11.8	22.4 ± 10.8
GNN-BL	22.2 ± 16.4	20.1 ± 15.8	18.1 ± 14.3	17.3 ± 13.2	18.2 ± 14.1	16.8 ± 13.1	16.9 ± 13.4
GNN-FE	24.0 ± 17.9	20.3 ± 15.9	19.6 ± 15.4	19.2 ± 15.2	19.0 ± 14.9	18.9 ± 15.1	19.0 ± 15.1
GNN-FF	19.6 ± 14.9	19.5 ± 14.8	17.9 ± 14.5	17.5 ± 14.1	17.1 ± 13.4	16.7 ± 13.2	17.0 ± 13.3

TABLE II: Optimality gap (%) in the Combinatorial Auction problem after E fine-tuning epochs for $n = 50$. .

Method	E=1	2	5	10	20	30	50
MLP-BL	36.2 ± 18.6	10.2 ± 8.5	31.2 ± 9.1	17.4 ± 6.2	12.4 ± 6.5	13.1 ± 5.8	11.1 ± 5.2
GNN-BL	31.3 ± 17.9	32.0 ± 17.6	20.8 ± 14.0	22.0 ± 13.7	11.7 ± 8.4	9.1 ± 7.0	9.3 ± 7.0
GNN-FE	33.5 ± 19.4	15.0 ± 11.2	24.8 ± 13.3	20.8 ± 12.6	17.3 ± 11.5	15.9 ± 10.9	18.6 ± 11.6
GNN-FF	15.5 ± 11.7	11.9 ± 9.4	11.2 ± 8.6	21.3 ± 10.5	11.3 ± 8.1	9.6 ± 7.3	9.9 ± 7.3

GNN frozen encoder (GNN-FE) We initialize the bipartite GNN encoder from a BiJEPA pretraining checkpoint (Section III-B) and then freeze the entire encoder (no gradient updates). Only the downstream prediction heads that map embeddings to $(\hat{x}, \hat{\lambda})$ are trained with the KKT loss. This isolates the benefit of pretraining from any further feature adaptation.

GNN full finetune (GNN-FF) We initialize the encoder from the same BiJEPA checkpoint as GNN-FE but finetune all parameters (encoder + prediction heads) using the downstream KKT loss. Compared to GNN-FE, this allows the encoder to adapt its features to the specific structure of the KKT objective and typically yields the fastest convergence in early training.

B. Experimental Setup

We consider instances of three sizes: $n \in \{10, 50, 200\}$ variables. Results are reported on a held out test set as the mean value of the relative optimality gap, as defined below:

$$\text{Gap}(\hat{x}, \hat{\lambda}) = \frac{2\left|\, c^\top \hat{x} + b^\top \hat{\lambda} \right|}{|\, c^\top \hat{x}\,| + \left|\, b^\top \hat{\lambda}\,\right| + \varepsilon} \tag{10}$$

where c is the objective function $\hat{x}$ is the predicted primal variable vector $\hat{\lambda}$ is the predicted dual variable vector and b is the right-hand-side.

C. Compute-limited fine-tuning: budget sweep

We train all GNNs and MLPs jointly on the four problem families, but focus evaluation on the Combinatorial Auction instance, where GNN methods require more fine-tuning epochs than MLPs. To assess the data-efficiency gains from BiJEPA, we fine-tune each model for $E \in 1, 2, 5, 10, 30, 50$ epochs under identical settings. Tables I–III reveal a clear distinction between low- and high-budget regimes. With limited fine-tuning, the pretrained full-finetuning model (GNN-FF) provides strong warm starts on larger instances: for n=50 it attains 11.20% at E=5 vs. 20.82% for GNN-BL (Table II), and for n=200 it reaches $\approx 11.8\%$ by $E \in 5, 10$ while GNN-BL remains at 18–26% (Table III). With enough training, however, the baseline continues improving and eventually achieves the best final performance on larger sizes (e.g., 9.09% at E=30 for n=50 and 7.08% at E=50 for n=200). The frozen-encoder variant (GNN-FE) plateaus early, indicating that updating only the prediction heads is insufficient to further reduce gaps. if fine-tuning compute is limited, BiJEPA pretraining yields strong early performance on medium/large instances, achieving low double-digit gaps within ≤ 5 epochs for n=50 and ≤ 10 for n=200. With sufficient downstream data, longer training allows the baseline to match or surpass pretrained models and obtain the best final gaps.

TABLE III: Optimality gap (%) in the Combinatorial Auction problem after E fine-tuning epochs for $n = 200$.

Method	E=1	2	5	10	20	30	50
MLP-BL	45.0 ± 26.7	19.1 ± 3.6	15.5 ± 6.7	21.7 ± 7.6	19.8 ± 8.6	23.4 ± 8.9	21.6 ± 8.9
GNN-BL	19.1 ± 9.8	19.3 ± 10.4	25.9 ± 11.0	18.0 ± 9.4	17.4 ± 6.5	7.7 ± 4.9	7.1 ± 4.6
GNN-FE	21.7 ± 9.8	25.3 ± 7.7	20.6 ± 7.9	26.3 ± 7.7	19.7 ± 7.4	24.1 ± 7.5	22.4 ± 7.3
GNN-FF	19.3 ± 8.7	19.4 ± 8.5	11.8 ± 6.5	11.8 ± 6.6	14.1 ± 6.3	18.2 ± 6.6	16.9 ± 6.7

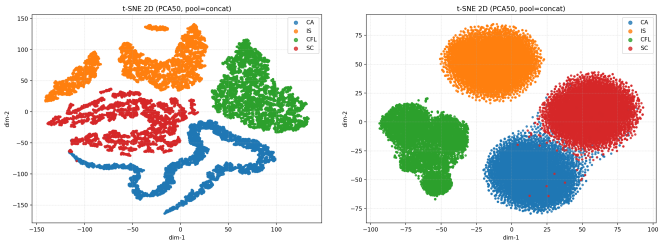

Fig. 2: t-SNE visualization (PCA-50 pre-reduction) of pooled not pretrained (left) and pretrained (right) embeddings. Points are colored by problem family (CA, IS, CFL, SC). The representation yields well-separated clusters across families.

D. Plotting embedding space

Figure 2 shows a 2D t-SNE projection of the pretrained graph embeddings (after a PCA reduction to 50 dimensions). We visualize pooled embeddings obtained by concatenating a constraint- and variable-pooled representation for each instance (pool=concat). The four LP families (CA, IS, CFL, SC) form clearly separated clusters, indicating that the pretrained encoder captures strong, problem-type specific structure in the latent space. You can see that the CA, IS and SC clusters have a spherical, Gaussian-like distribution, which is consistent with the effect of the SIGReg regularization. The visualization provides evidence that the learned representations are discriminative enough to separate different LP families and that BiJEPA does not suffer from representation collapse but instead pushes respresentations towards an isotropic distribution.

E. Warm-starting a classical LP solver

We additionally investigated whether the solution predicted by our learned solver can be used to warm-start a classical LP solver. Concretely, for each validation instance we first predict a primal solution $\hat{x}$ and provide these values as an initial point to the downstream solver. We compare against the solver's default initialization, and report wall-clock solve time and iteration counts (median/P50 and P90).

Unfortunately, in our current settings, warm-starting yields negligible improvements. Across the validation set, both the median and P90 solve times remain essentially unchanged

when initializing the solver from $(\hat{x}, \hat{\lambda})$ rather than using the solver's internal defaults. In absolute terms, the largest instances considered in our experiments are solved in approximately $\sim$20,ms regardless of initialization, and the tail behavior (P90) similarly shows no meaningful improvement.

This is due to the fact that our instances of size 200 are still relatively small-sized LPs for which modern solved have already been optimized. An important direction for future work will therefore be to identify relevant approaches towards scaling up the size of the problems to thousands of variables.

V. Limitations

The results and experiments presented in this contribution have a number of limitations. First of all the LP instances are relaxed versions of synthetically generated MILPs. Secondly, due to scalability limitations, we used relatively small models and problem instances. These limitations all present opportunities for future work as discussed in Section VI.

VI. Conclusion and Future Work

In this contribution We presented BiJEPA, a GNN solution trained with KKT loss and joint embeddings of a bipartite graph that represents variables and constraints of an optimization problem. Training with BiJEPA and KKT allow our GNNs to learn representations of optimization problems, and as a consequence a direct mapping to a feasible solution, even when the network is trained with heterogeneous problems at the same time. Future work can branch in multiple directions. One aspect that seems relevant to the optimization is how to sample the instances during the training of the GNN, a different batching and shuffling strategy may allow learning better weights with the GNN model. Finally, studying the properties of the manifold generated by the GNN, may allow to discover further terms to be added to the KKT conditions to speed up the learning process.

Acknowledgment

Large language models have been used to correct the grammar, summarize sentences and check the formalization, but except for this the content has been produced by humans.

References

[1] M. J. Todd, "The many facets of linear programming," *Mathematical Programming*, vol. 91, no. 3, pp. 417–436, 2002. DOI: 10.1007/s101070100261.

[2] A. M. Verweij, "Selected applications of integer programming: A computational study," Ph.D. dissertation, Utrecht University, Sep. 2000.

[3] J. Kotary, F. Fioretto, P. Van Hentenryck, and B. Wilder, "End-to-end constrained optimization learning: A survey," *arXiv preprint arXiv:2103.16378*, 2021.

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

[5] C. Qian and C. Morris, "Towards graph neural networks for provably solving convex optimization problems," *arXiv preprint arXiv:2502.02446*, 2025.

[6] B. Tang, E. B. Khalil, and J. Drgoňa, "Learning to optimize for mixed-integer non-linear programming," *arXiv preprint arXiv:2410.11061*, 2024.

[7] T.-H. Lee and M.-S. Kim, "Rl-milp solver: A reinforcement learning approach for solving mixed-integer linear programs with graph neural networks," *arXiv preprint arXiv:2411.19517*, 2024.

[8] H. W. Kuhn and A. W. Tucker, "Nonlinear programming," in *Berkeley Symposium on Mathematical Statistics and Probability, 1950*, J. Neyman, Ed., Berkeley and Los Angeles: University of California Press, 1951, pp. 481–492.

[9] M. Assran et al., "Self-supervised learning from images with a joint-embedding predictive architecture," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2023, pp. 15 619–15 629.

[10] S. Arvind, R. Pomaje, and R. V. Bhat, "Karush-kuhn-tucker condition-trained neural networks (kkt nets)," *arXiv preprint arXiv:2410.15973*, 2024.

[11] D. Wu and A. Lisser, "A deep learning approach for solving linear programming problems," *Neurocomputing*, vol. 520, pp. 15–24, 2023.

[12] Z. Chen, J. Liu, X. Wang, J. Lu, and W. Yin, "On representing linear programs by graph neural networks," *arXiv preprint arXiv:2209.12288*, 2022.

[13] C. Qian, D. Chételat, and C. Morris, "Exploring the power of graph neural networks in solving linear optimization problems," in *International Conference on Artificial Intelligence and Statistics*, PMLR, 2024, pp. 1432–1440.

[14] L. Scavuzzo, K. Aardal, A. Lodi, and N. Yorke-Smith, "Machine learning augmented branch and bound for mixed integer linear programming," *Mathematical Programming*, pp. 1–44, 2024.

[15] J.-Y. Ding et al., "Accelerating primal solution findings for mixed integer programs based on solution prediction," in *AAAI conference on artificial intelligence*, vol. 34, 2020, pp. 1452–1459.

[16] M. Gasse, D. Chételat, N. Ferroni, L. Charlin, and A. Lodi, "Exact combinatorial optimization with graph convolutional neural networks," *Advances in neural information processing systems*, vol. 32, 2019.

[17] G. Zarpellon, J. Jo, A. Lodi, and Y. Bengio, "Parameterizing branch-and-bound search trees to learn branching policies," *AAAI Conference on Artificial Intelligence*, vol. 35, no. 5, pp. 3931–3939, May 2021.

[18] J. Lin, J. Zhu, H. Wang, and T. Zhang, "Learning to branch with tree-aware branching transformers," *Knowledge-Based Systems*, vol. 252, p. 109 455, 2022.

[19] A. Prouvost, J. Dumouchelle, L. Scavuzzo, M. Gasse, D. Chételat, and A. Lodi, "Ecole: A gym-like library for machine learning in combinatorial optimization solvers," *arXiv preprint arXiv:2011.06069*, 2020.