Learning to Select Nodes in Branch-and-Bound via Tree-Aware Contrastive Learning

Anonymous authors
Paper under double-blind review

Abstract

Branch-and-bound (BnB) search is inherently hierarchical, yet learning to select its next node remains largely based on node preferences or tree-wide state representations. The central challenge is to turn this hierarchy into decision-relevant supervision. We introduce TreeCL, a contrastive framework that derives node-selection preferences from optimal-path consistency: if a reference optimum remains feasible at a node, it remains feasible at every ancestor. For competing nodes, their lowest common ancestor identifies the shared search context and anchors a contrast between branches that retain or exclude the reference optimum. This converts a structural property of BnB into an ancestor-conditioned selection rule, without requiring the reference solution at inference. Label-preserving branching-sequence augmentation further expands supervision from limited oracle trajectories. Experiments across five families of mixed-integer linear programming problems show significant gains in search efficiency over the evaluated learning-based baselines, transfer to larger instances, and fewer explored nodes on real-world instances.

Keywords: Mixed integer linear programming; branch and bound; node selection; contrastive learning; learning for optimization

1 Introduction

Mixed integer linear programming (MILP) is a fundamental formulation for NP-hard optimization (Floudas & Lin, 2005; Ren & Gao, 2010; Paschos, 2014). Branch and bound (BnB) (Land & Doig, 1960) is a dominant algorithmic framework for MILPs. It conducts a tree-based search that recursively decomposes the original problem into subproblems and prunes suboptimal regions. Node selection is crucial in BnB, as it shapes the search trajectory and affects efficiency by prioritizing promising subproblems (Achterberg, 2007).

Traditional BnB node selection approaches are heuristic-based, relying on problem features like lower bounds and feasibility (Achterberg et al., 2008). They are fast and easy to implement, but may lead to suboptimal search trajectories due to greedy decisions and lack generalization due to the reliance on specific features. Learning-based approaches, e.g., imitation learning (He et al., 2014; Song et al., 2018; Labassi et al., 2022) and reinforcement learning (Mattick & Mutschler, 2024; Zhang et al., 2025), aim to address these issues by learning expert trajectories or environment feedback from a distribution of target problem instances (Bengio et al., 2021). Among the latter, Mattick & Mutschler (2024) learn a policy over active nodes from a graph representation of the search tree, while Zhang et al. (2025) use a tripartite graph to encode tree-wide information for node selection. They are particularly effective when a priori computational resources allow offline training, which in turn enables online inference on many problem instances from the target domain.

However, a central challenge remains: learning node preferences that respect the hierarchical structure of BnB. This challenge has two aspects. First, a candidate's relevance depends on the branching decisions inherited from its ancestors. Node-wise scoring leaves this dependence implicit. DeepBound (Liu et al., 2026) learns pairwise preferences between nodes on and off a reference optimal path through cross-node feature fusion, but does not condition

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

the comparison on the candidates' shared ancestor. Thus, even a direct comparison between nodes need not capture the structural context in which their search paths diverge. Second, absolute cues such as node depth and bound values can change with instance scale and structure. Policies that rely on these cues may therefore transfer poorly to larger instances. This motivates learning from relations that remain valid across BnB search trees.

To address these challenges, we propose TreeCL, a contrastive node-selection framework grounded in optimal-path consistency. During training, TreeCL contrasts on-path and off-path candidates relative to their lowest common ancestor. The resulting ancestor-conditioned similarity directly implements node comparison at inference, without requiring a reference optimal solution. We further augment branching sequences while preserving optimal-path labels to improve sample efficiency. Our contributions are:

- Tree-structured supervision from optimal-path consistency. We identify optimal-path consistency: because branching constraints monotonically refine feasible regions, any ancestor of a node containing a reference optimum must also contain that optimum. This property turns ancestor–descendant relations into tree-structured supervision. It ties node preference to ancestral context rather than isolated, scale-dependent cues.
- Contrastive learning for structure-aware node selection. We reformulate node selection as learning compatibility between a candidate and its ancestral context, rather than an isolated score. Using the lowest common ancestor (LCA) as an anchor, InfoNCE pulls an on-path node closer and pushes an off-path node away. The resulting similarity defines a structure-aware selection criterion.
- Label-preserving augmentation for sample-efficient learning. We perturb recorded branching sequences while preserving whether the reference optimum remains feasible. Alternative branch variables and label-consistent directions yield additional contrastive triplets from existing oracle trajectories, without a separate solver run for each synthetic triplet. This enriches structural supervision when expert data are limited.

We validate TreeCL on MILP benchmark problems. Experiments show that it significantly reduces the number of explored nodes and improves solving time compared with strong baselines, and transfers more robustly to larger instances. On real-world problems, it solves more CORLAT instances and explores fewer nodes than the evaluated learning-based baselines. Moreover, the data augmentation ensures strong performance under limited expert supervision.

The remainder of the paper is organized as follows. Section 2 reviews related work. Section 3 describes the proposed method. Section 4 presents experimental results. Section 5 concludes the paper.

2 Background and Related Work

2.1 Mixed Integer Linear Programming

MILP refers to an optimization problem with a linear objective and linear constraints, where some variables are required to be integers and others are continuous. The standard form is

$$\begin{aligned} \min \quad & \mathbf{c}^T\mathbf{x}, \\ \text{s.t.} \quad & \mathbf{A}\mathbf{x} \le \mathbf{b}, \\ & x_i \in \mathbb{Z}, i \in I, \quad x_j \in \mathbb{R}, j \notin I, \end{aligned} \tag{1}$$

where $\mathbf{x} \in \mathbb{R}^n$ denotes decision variables, and $\mathbf{A} \in \mathbb{R}^{m\times n}$, $\mathbf{b} \in \mathbb{R}^m$ define the constraints.

2.2 Branch and Bound and Its Node Selection

BnB (Land & Doig, 1960) is a tree-based framework for solving MILPs. It recursively decomposes the problem into subproblems, with each node representing a subproblem defined by additional bound constraints. Branching creates child nodes by imposing disjoint

bounds on a fractional variable, and the algorithm proceeds by branching, node selection, and pruning.

Traditional BnB node selection approaches are heuristic-based, relying on handcrafted rules derived from problem features. Common strategies include best estimate search (Bénichou et al., 1971; Forrest et al., 1974), best first search (Hart et al., 1968; Achterberg, 2007), and depth first search (Dakin, 1965). While these rules are simple and computationally efficient, they often yield suboptimal trajectories due to their greedy nature and generalize poorly across problem instances.

Learning-based node selectors are trained from expert decisions or solver feedback; a broader review is given by Scavuzzo et al. (2024). He et al. (2014) used DAGGER (Ross et al., 2011) to imitate a diving oracle. Song et al. (2018) used retrospective imitation to learn preferences along search paths. Yilmaz & Yorke-Smith (2021) learned which child to explore during SCIP plunging and studied both heuristic and exact search. Building on the bipartite MILP representation of Gasse et al. (2019), Labassi et al. (2022) used a GNN for pairwise node comparison.

Other approaches guide node selection through predicted solutions or learn multiple solver decisions jointly. Khalil et al. (2022) predicted variable biases from near-optimal solutions and used them to guide node selection and warm starts. Cantürk et al. (2024) extended this approach with uncertainty-guided node selection and primal heuristics. Du et al. (2025) jointly learned node and branching-variable selection. For TSP, Sciandra et al. (2026) used graph-convolutional scores to guide both decisions. Liu et al. (2024) instead learned symbolic node-selection rules that support interpretable, inexpensive inference.

Recent reinforcement-learning methods represent the search tree directly. Mattick & Mutschler (2024) encoded it as a directed graph and learned a policy over active leaves. Zhang et al. (2025) used a tripartite graph to represent tree-wide information and trained a node-selection policy with delayed search feedback.

DeepBound (Liu et al., 2026) also uses optimal-path labels. It pairs oracle and non-oracle nodes and learns their preferences through cross-node feature fusion. TreeCL conditions the comparison on the candidates' lowest common ancestor and learns candidate-to-ancestor compatibility through a contrastive objective.

2.3 Related Work on Contrastive Learning

Self-supervised learning has emerged as a powerful paradigm for representation learning, where models are trained on automatically constructed pretext tasks without requiring manual labels (Jing & Tian, 2020). Among various self-supervised learning approaches, contrastive learning has achieved remarkable success in visual understanding (He et al., 2020; Chen et al., 2020; Yang et al., 2023) and graph representation learning (You et al., 2020; Tong et al., 2021; Wang et al., 2022). By contrasting positive (semantically similar) and negative (dissimilar) pairs, it induces representations where similar items are pulled together and dissimilar ones pushed apart.

Recent works have explored contrastive learning in optimization. Mulamba et al. (2020) introduced a contrastive loss in decision-focused learning for combinatorial problems with uncertain inputs. The optimal and non-optimal solutions were treated as positives and negatives, respectively. Duan et al. (2022) employed contrastive pretraining to Boolean satisfiability, inspired by its success in computer vision. Huang et al. (2023) integrated contrastive learning with large neighborhood search, identifying near-optimal solutions as positive to guide neighborhood prediction.

Existing contrastive methods for combinatorial optimization construct positive and negative samples from solution quality, instance augmentations, or search neighborhoods (Mulamba et al., 2020; Duan et al., 2022; Huang et al., 2023). They do not target the ancestry of competing BnB nodes. Existing node selectors use optimal-path labels, direct pairwise preferences, or global tree representations, but do not condition a contrastive objective on the candidates' shared ancestor. TreeCL uses optimal-path consistency to construct such supervision and learns an ancestor-conditioned rule for node comparison.

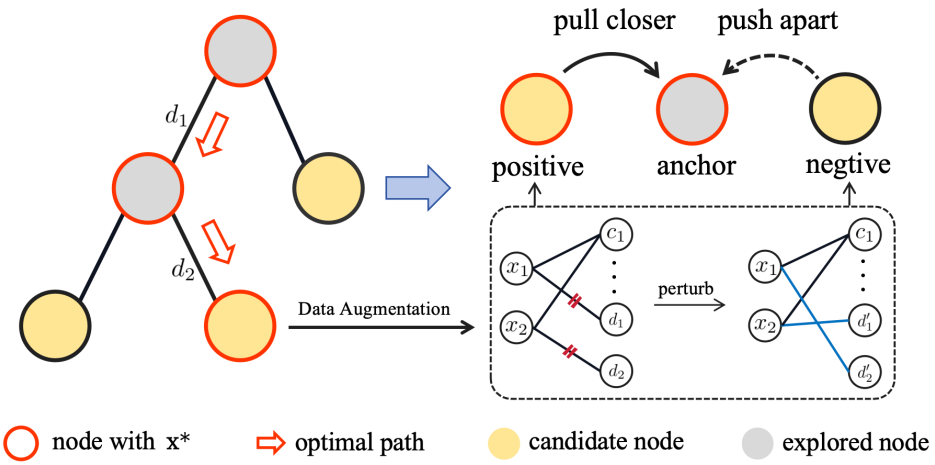


Figure 1: Overview of TreeCL. A reference optimal solution $x^\star$ induces optimal-path consistency in the BnB tree, based on which TreeCL builds LCA-anchored contrastive pairs and performs node selection with augmented structural supervision.

3 TreeCL

The BnB tree is more than a representation of solver state: its ancestry encodes a monotone relationship between search regions. TreeCL turns this relationship into supervision for node selection and learns preferences from where competing branches diverge. Figure 1 illustrates the resulting framework.

3.1 Tree-Structured Supervision via Optimal-Path Consistency

We use a simple structural property of BnB to derive supervision. Along a root-to-leaf path, branching constraints monotonically refine the feasible region. Let $\mathcal{F}_v$ denote the feasible region of node v, and let $\mathbf{x}^\star$ be a globally optimal solution. If $\mathbf{x}^\star \in \mathcal{F}_v$ for a descendant node v, then for any ancestor node v^{anc} on the path to v, we also have $\mathbf{x}^\star \in \mathcal{F}_{v^{\text{anc}}}$. We refer to this property as optimal-path consistency.

Optimal-path consistency suggests a relational training signal. Consider an ancestor node v^{anc} and a set of candidate nodes generated in its subtree. Nodes whose feasible regions contain the reference optimum should be more compatible with v^{anc} than nodes that exclude it. This motivates learning a similarity function H such that a compatible pair (v^{anc}, v^+) has higher similarity than an incompatible pair (v^{anc}, v^-).

Formally, we define a binary label $y_{\mathbf{x}^\star}(v) \in \{0, 1\}$, where $y_{\mathbf{x}^\star}(v) = 1$ if $\mathbf{x}^\star \in \mathcal{F}_v$ and 0 otherwise. Thus, on-path and off-path labels refer to this reference solution; an off-path node may contain a different optimal solution. For each anchor ancestor v^{anc}, we sample a positive node v^+ with $y_{\mathbf{x}^\star}(v^+) = 1$ and a negative node v^- with $y_{\mathbf{x}^\star}(v^-) = 0$. The training objective encourages

$$H\big(v^{\text{anc}}, v^+\big) \;>\; H\big(v^{\text{anc}}, v^-\big)\,, \tag{2}$$

where $H(\cdot, \cdot)$ is a similarity function defined on learned embeddings. Note that a MILP instance may have multiple globally optimal solutions. In this work, we use as reference a single optimal solution $x^\star$ returned by SCIP in the first solve. Node labels are then defined with respect to this reference solution, rather than by enumerating all optima. This keeps offline annotation tractable and maintains a fair supervision budget across methods.

3.2 Implement Tree-Structured Supervision with Contrastive Learning

Contrastive learning is well suited for learning similarity functions, making it a natural choice for learning H in equation 2. In our setting, a candidate and its ancestor form a positive

pair if the candidate contains the reference optimum, and a negative pair if it excludes that solution. The model is trained to assign higher similarity to pairs on the reference optimal path.

3.2.1 Data Collection

Like other learning-based node selection models, our model is trained offline on a distribution of target problem instances and deployed online to solve new instances from the target domain. Thus, we begin by collecting training data. Specifically, data collection requires solving each MILP twice, i.e., first to obtain a reference optimal solution, and then to simulate a guided BnB traversal to identify nodes containing that solution. For each branching step t of each instance, we construct data points using a diving oracle. Each data point is a triplet $(v_t^+, v_t^{-,k}, (v_t^{-,k})^{\text{anc}})$, $k = 1, 2, \cdots, |\mathcal{V}_t| - 1$. Here, $\mathcal{V}_t$ denotes the candidate node set at t; $v_t^+ \in \mathcal{V}_t$ is a positive candidate node containing the reference solution; $v_t^{-,k} \in \mathcal{V}_t \setminus \{v_t^+\}$ is the kth negative node excluding it; $(v_t^{-,k})^{\text{anc}}$ is the lowest common ancestor of v_t^+ and $v_t^{-,k}$ in the BnB tree. The lowest common ancestor is not necessarily their immediate parent, although it may coincide with one in special cases. We use the lowest common ancestor instead of the root as the anchor, as it provides a more localized context that better preserves the branching path relevant to the contrast.

3.2.2 Data Augmentation

We augment recorded oracle comparisons by perturbing the branching sequences below their lowest common ancestor. At each step, an alternative variable is sampled from the recorded LP branching candidates. Branch directions on the positive path are chosen to retain the reference optimum. The first divergent decision on the negative path takes the opposite branch, ensuring that the reference optimum is excluded. We construct each augmented graph by copying the ancestor graph and updating the affected variable-bound features. This generates additional labeled training triplets from the recorded traversal without a separate solver run for each synthetic triplet.

3.2.3 Model Architecture

Following prior work Gasse et al. (2019); Labassi et al. (2022); Scavuzzo et al. (2022), we represent each BnB subproblem as a bipartite graph of variables and constraints, with coefficient-weighted edges. Variable features encode types, objective coefficients, and bounds; constraint features encode senses and right-hand sides. Multi-layer perceptrons (MLPs) embed these features, a three-layer bipartite GNN propagates them, and mean pooling over variable embeddings yields the node representation. Feature definitions and GNN update rules are detailed in Appendix A.

3.2.4 Training

To align the node embeddings with the tree-structured optimality signal in the BnB process, we adopt a contrastive learning objective. Each node is encoded by a bipartite graph encoder ϕ_θ to obtain its embedding $\mathbf{v}_{\text{avg}}$. The similarity between two nodes v_a and v_b with graph representations G_a and G_b is then defined via cosine:

$$H(v_a, v_b) = \text{cos_sim}(\phi_\theta(G_a), \phi_\theta(G_b)) = \frac{\mathbf{v}_{\text{avg},a}^\top \mathbf{v}_{\text{avg},b}}{\|\mathbf{v}_{\text{avg},a}\| \, \|\mathbf{v}_{\text{avg},b}\|}. \tag{3}$$

We use a two-candidate InfoNCE objective (Oord et al., 2018) to learn H from the triplets constructed in Sections 3.2.1 and 3.2.2. Let $\mathcal{D}$ denote the training set. The loss is

$$\mathcal{L}(\theta) = \frac{-1}{|\mathcal{D}|} \sum_{(v^+, v^-, v^{\text{anc}}) \in \mathcal{D}} \log \quad \frac{\exp\left(H(v^{\text{anc}}, v^+)/\tau\right)}{\exp\left(H(v^{\text{anc}}, v^+)/\tau\right) + \exp\left(H(v^{\text{anc}}, v^-)/\tau\right)}. \tag{4}$$

where v^+ contains the reference optimal solution $\mathbf{x}^\star$, v^- excludes it, v^{anc} is their lowest common ancestor in the BnB tree, and τ is a temperature hyperparameter. The triplets are constructed as described in Section 3.2.1.

The loss directly matches the desired pairwise preference. For one triplet, define the score margin $\Delta = H(v^{\text{anc}}, v^{+}) - H(v^{\text{anc}}, v^{-})$. Its contribution to equation 4 is $\ell(\Delta) = \log(1 + \exp(-\Delta/\tau))$. Since $\tau > 0$,

$$\frac{d\ell}{d\Delta} = -\frac{1}{\tau\big(1 + \exp(\Delta/\tau)\big)} < 0. \tag{5}$$

Thus, increasing the positive-minus-negative similarity relative to their common ancestor strictly decreases the loss. This establishes the alignment between the training objective and the node comparator, without assuming that the learned, parameterized similarity recovers an optimal ranking.

4 Experiments

4.1 Setup

4.1.1 Benchmarks

We evaluate TreeCL on three MILP benchmarks: fixed charge multicommodity network flow (FCMCNF) (Hewitt et al., 2010), maximum satisfiability (MAXSAT) (Ansótegui & Gabàs, 2017), and generalized independent set (GISP) (Colombi et al., 2017). Each has in-distribution test instances and larger transfer instances. We also evaluate on the real-world CORLAT (Gomes et al., 2008) and MIK (Atamtürk, 2003) datasets. Instance generation, sizes, and dataset statistics are given in Appendix B.

4.1.2 Baselines

We compare against the following baselines:

- Oracle: an expert diving policy constructed by first solving the MILP to obtain the optimal solution and then, during the BnB process, always selecting the active node whose feasible region contains this optimal solution.
- Estimate: the widely used heuristic based on best estimate search.
- SCIP Gamrath et al. (2020): SCIP's default node selection, which augments Estimate with a diving heuristic.
- SVM He et al. (2014): an imitation learning method based on the support vector machine.
- RankNet Song et al. (2018): a MLP trained with pairwise ranking loss.
- L2C Labassi et al. (2022): a GNN-based pairwise node-selection method.

TRGNN (Zhang et al., 2025) and DeepBound (Liu et al., 2026) are closely related node-selection methods but are not included in our quantitative comparison. We could not locate author-released implementations, and their reported results use different solver configurations or benchmark families; we therefore avoid comparisons across incompatible experimental settings.

4.1.3 Metrics

We report the following evaluation metrics Labassi et al. (2022); Zhang et al. (2024):

- Time: total solving time in seconds.
- Nodes: number of nodes in the final BnB tree.
- Wins: A "win" is counted for a method on a given instance if it achieves the smallest solving time among all compared methods.
- Solved: number of instances solved with time limit of 3600 seconds.

Time and Nodes are averaged using the shifted geometric mean with a shift of 1, following the common MILP evaluation practice. For unsolved instances, we take SCIP's final node count and time to include them in the averaged metrics.

4.1.4 Implementation

Training. We generate 1000 instances per benchmark for training. This results in 16285, 41299, and 42511 training triplets for FCMCNF, MAXSAT, and GISP, respectively. To report imitation accuracy after training, we generate 100 held-out test instances per benchmark, yielding 3019, 4868, and 5236 test triplets, respectively. We use a separate set of 50 in-distribution test instances and 50 larger transfer instances per benchmark to evaluate solver performance and generalization. No validation set is used for model selection.

For the real-world datasets, we use 1,000 CORLAT instances for training, 100 held-out instances for testing imitation accuracy, and a separate 50 instances for testing solver performance. For MIK, we use 72 of the 90 instances for training and reserve the remaining 18 for testing. The CORLAT and MIK training sets contain 143,593 and 6,578 samples, respectively.

In TreeCL, each of the three MLP blocks consists of a LayerNorm, a linear layer, and a ReLU activation; the GCN is three-layer and the activation σ is also ReLU. The temperature τ in equation 4 is set to 0.07, following the default setting in He et al. (2020). We report sensitivity to this choice in Table 10. The model is trained using Adam Kingma & Ba (2014) with a batch size of 64 and a learning rate of 5×10^{-3}. For the augmentation strategy in Section 3.2.2, recorded branching candidates and the reference solution determine the synthetic variable-bound features and their labels. We test two variants of our model: (i) TreeCL, trained on the full dataset generated, and (ii) TreeCL-aug, trained with only 10% of the original data and 90% synthetic data generated via the data augmentation strategy to assess the effectiveness of the augmentation. SVM, RankNet, and L2C are faithfully reproduced using the code from Labassi et al. (2022), without additional tuning or modification. All methods are implemented in SCIP 7.0.3 with default parameters. The experiments are conducted on a workstation equipped with an Intel(R) Xeon(R) Gold 6348 CPU @ 2.60GHz and a single NVIDIA GeForce RTX 3090 GPU. Model training is performed on the GPU, while inference is executed on a single CPU thread with a fixed random seed to ensure reproducibility.

Inference. We use the learned similarity function H to define a node comparison strategy for BnB. Specifically, our model replaces SCIP's Gamrath et al. (2020) default NODECOMP function, which compares two candidate nodes. Since ranking all candidates at each selection step is computationally expensive, SCIP maintains a priority queue of candidates. When new nodes are created, they are inserted into the queue based on NODECOMP's pair-wise comparison outcome. The NODESELECT function then selects the next node to explore according to the queue. We redefine NODECOMP using our model as

$$\text{NODECOMP}(v_a, v_b, v^{\text{anc}}) = \begin{cases} -1 & \text{if } H(v_a, v^{\text{anc}}) > H(v_b, v^{\text{anc}}); \\ 1 & \text{if } H(v_b, v^{\text{anc}}) > H(v_a, v^{\text{anc}}); \\ 0 & \text{otherwise,} \end{cases} \tag{6}$$

where v_a and v_b are candidate nodes and v^{anc} is their lowest common ancestor. The returned value follows SCIP's convention, i.e., -1 prefers v_a, 1 prefers v_b, and 0 indicates no preference. This allows integrating tree-aware contrastive node selection into existing solvers with minimal overhead. The code will be publicly released upon acceptance.

4.2 In-Distribution Results

We compare the proposed TreeCL against baselines across the three benchmarks to evaluate its effectiveness and efficiency. Bold numbers in tables denote the best results (except Oracle).

Table 1 shows that TreeCL leads the evaluated non-oracle methods on FCMCNF and MAXSAT in solve time, explored nodes, and wins. On GISP, it matches SCIP's mean solve time and records the most wins among learning-based methods, although SCIP wins more instances and SVM explores slightly fewer nodes.

The results support the value of structural supervision, while showing that the gains vary by problem family. On CORLAT, TreeCL solves 49 of 50 instances, tying SCIP and exceeding

Table 1: Comparison of TreeCL with baselines on the standard test data.

Methods	FCMCNF			MAXSAT			GISP		
	Time (s)	Nodes	Wins	Time (s)	Nodes	Wins	Time (s)	Nodes	Wins
Oracle	3.38 ± 1.42	14.99 ± 4.13	-	5.28 ± 1.72	102.16 ± 2.48	-	3.68 ± 1.25	98.00 ± 3.29	-
Estimate	3.60 ± 1.48	21.40 ± 5.32	8	6.87 ± 1.69	176.58 ± 2.33	3	4.02 ± 1.25	218.06 ± 2.47	6
SCIP	4.04 ± 1.44	40.74 ± 4.57	11	7.99 ± 1.48	147.09 ± 1.96	2	3.82 ± 1.19	184.27 ± 1.96	23
SVM	3.49 ± 1.43	19.98 ± 5.05	7	6.23 ± 1.74	150.13 ± 2.60	3	3.99 ± 1.25	180.58 ± 2.78	2
RankNet	3.65 ± 1.48	20.96 ± 5.22	2	6.26 ± 1.75	142.13 ± 2.64	1	4.11 ± 1.28	203.49 ± 2.66	0
L2C	3.65 ± 1.50	21.24 ± 5.47	1	6.20 ± 1.90	138.03 ± 2.79	4	4.04 ± 1.27	184.31 ± 2.77	1
TreeCL	3.43 ± 1.49	19.25 ± 5.07	21	5.28 ± 1.78	105.68 ± 2.50	37	3.82 ± 1.25	181.69 ± 2.77	18

Table 2: Comparison of TreeCL with baselines on real-world datasets.

Methods	CORLAT			MIK		
	Time(s)	Nodes	Solved	Time(s)	Nodes	Solved
Oracle	3.20 ± 2.64	58.46 ± 16.66	50	11.70 ± 1.53	362.27 ± 19.89	18
Estimate	4.86 ± 4.04	98.05 ± 28.59	48	13.44 ± 1.39	618.58 ± 12.63	18
SCIP	4.09 ± 3.48	109.47 ± 29.51	49	15.79 ± 1.59	629.40 ± 13.39	18
SVM	10.26 ± 7.53	188.32 ± 42.44	46	15.37 ± 1.58	597.06 ± 15.68	18
RankNet	9.60 ± 8.14	198.15 ± 51.20	43	17.01 ± 1.72	741.52 ± 16.05	18
L2C	25.15 ± 15.36	156.34 ± 33.50	42	22.36 ± 1.76	601.55 ± 13.94	18
TreeCL	7.07 ± 4.32	120.82 ± 25.67	49	16.43 ± 1.46	581.89 ± 12.88	18

Note: Time(s) and Nodes are computed on the intersection of instances solved by all methods.

Table 3: Comparison of TreeCL with baselines on transfer data.

Methods	FCMCNF			MAXSAT			GISP		
	Time (s)	Nodes	Wins	Time (s)	Nodes	Wins	Time (s)	Nodes	Wins
Oracle	16.13 ± 1.72	74.59 ± 4.46	-	7.40 ± 1.49	160.33 ± 2.00	-	18.21 ± 1.53	1062.20 ± 1.81	-
Estimate	19.14 ± 1.93	122.37 ± 5.14	8	9.97 ± 1.55	247.01 ± 1.84	0	20.26 ± 1.68	1434.95 ± 1.84	1
SCIP	21.28 ± 1.84	178.45 ± 4.30	8	10.75 ± 1.34	170.92 ± 1.66	1	15.64 ± 1.48	1533.48 ± 1.85	37
SVM	18.30 ± 1.84	132.77 ± 4.73	12	8.80 ± 1.60	225.32 ± 2.06	2	18.53 ± 1.51	1221.79 ± 1.69	4
RankNet	19.76 ± 2.00	112.60 ± 5.50	3	9.50 ± 1.57	238.33 ± 2.11	1	18.89 ± 1.52	1215.05 ± 1.73	0
L2C	19.91 ± 1.90	120.17 ± 4.94	0	9.31 ± 1.70	233.10 ± 2.23	2	18.80 ± 1.52	1196.42 ± 1.75	0
TreeCL	18.19 ± 1.84	114.16 ± 4.57	19	7.04 ± 1.49	155.93 ± 2.02	44	17.31 ± 1.51	1194.70 ± 1.72	8

the other learning-based methods (Table 2). On the instances solved by all methods, it explores fewer nodes than the learning-based baselines on both CORLAT and MIK, but its inference cost increases wall-clock time relative to heuristic methods. Further analyses are in Appendix D.

TreeCL-aug is trained with 10% recorded data and 90% label-preserving augmentations. Appendix C compares it with fully supervised baselines; it remains competitive and achieves the most wins on FCMCNF and GISP.

4.3 Transfer Results

Table 3 evaluates TreeCL on larger instances than those used in training. It achieves the lowest solve time among non-oracle methods on FCMCNF and MAXSAT, with 19 and 44 wins, respectively. On GISP, SCIP remains faster, while TreeCL is the fastest learning-based method.

On FCMCNF, RankNet explores slightly fewer nodes than TreeCL but solves more slowly. This illustrates that search-tree size alone does not determine wall-clock performance.

Appendix C.1 reports TreeCL-aug on larger transfer instances. Despite using only 10% recorded training data, it remains competitive with fully supervised learning-based baselines, particularly on GISP.

Table 4: Ablation study on test sets.

Methods	FCMCNF			MAXSAT			GISP		
	Time (s)	Nodes	Wins	Time (s)	Nodes	Wins	Time (s)	Nodes	Wins
TreeCL-10%	3.76 ± 1.51	26.76 ± 5.72	9	6.83 ± 1.73	175.38 ± 2.41	6	4.50 ± 1.24	286.83 ± 2.22	2
TreeCL-aug	3.49 ± 1.52	21.23 ± 5.38	23	6.58 ± 1.79	132.94 ± 2.64	5	3.77 ± 1.24	206.67 ± 2.46	28
TreeCL	3.43 ± 1.49	19.25 ± 5.07	18	5.28 ± 1.78	105.68 ± 2.50	39	3.82 ± 1.25	181.69 ± 2.77	20

4.4 Ablations

Table 4 compares TreeCL trained on all recorded data, TreeCL-10% trained on one tenth, and TreeCL-aug trained on one tenth plus label-preserving augmentations.

TreeCL-aug reduces both solve time and node count relative to TreeCL-10% on all three benchmarks. On GISP, it also achieves more wins than full-data TreeCL (28 versus 20) and a slightly lower mean solve time. On MAXSAT, full-data TreeCL remains strongest, indicating that augmentation does not fully replace recorded supervision.

Transfer-set ablations in Appendix C.2 show that augmentation improves over training on 10% of the original data alone. Further analyses of imitation accuracy, temperature sensitivity, and optimality gaps are provided in Appendix D.

5 Conclusion

This paper has studied learning-based node selection in branch-and-bound for mixed-integer linear programming. We argue that the BnB tree structure provides relational signals that are under-utilized by node-centric scoring formulations. Based on an optimal-path consistency property, we have proposed TreeCL, a tree-aware contrastive learning framework that learns node representations by pulling on-path nodes closer to their ancestors while separating off-path nodes in the embedding space. We have also introduced a branching-sequence-based augmentation strategy to improve sample efficiency and robustness.

Experiments on multiple MILP benchmarks demonstrated that TreeCL reduces the number of explored nodes and improves solving time compared with strong heuristic and learning-based baselines, and exhibits better transfer to larger instances. Future work includes designing richer tree-context encoders, extending the contrastive signal to incorporate incumbent updates and pruning dynamics, and combining the learned node selection rule with complementary learned components such as branching and pruning.

Declaration of generative AI and AI-assisted technologies in the manuscript preparation process

During the preparation of this work, the authors used ChatGPT (OpenAI) to assist with language refinement, readability improvement, and the organization of manuscript content. After using this tool, the authors carefully reviewed and edited the manuscript as needed and take full responsibility for the content of the published article.

References

Tobias Achterberg. Constraint integer programming. 2007.

Tobias Achterberg, Timo Berthold, Thorsten Koch, and Kati Wolter. Constraint integer programming: A new approach to integrate cp and mip. In International Conference on Integration of Artificial Intelligence and Operations Research Techniques in Constraint Programming, pp. 6–20, 2008.

Carlos Ansótegui and Joel Gabàs. Wpm3: an (in) complete algorithm for weighted partial maxsat. Artificial Intelligence, 250:37–57, 2017.

Alper Atamtürk. On the facets of the mixed–integer knapsack polyhedron. Mathematical Programming, 98(1):145–175, 2003.

Ramón Béjar, Alba Cabiscol, Felip Manyà, and Jordi Planes. Generating hard instances for maxsat. In 2009 39th International Symposium on Multiple-Valued Logic, pp. 191–195, 2009.

Yoshua Bengio, Andrea Lodi, and Antoine Prouvost. Machine learning for combinatorial optimization: a methodological tour d'horizon. European Journal of Operational Research, 290(2):405–421, 2021.

Michel Bénichou, Jean-Michel Gauthier, Paul Girodet, Gerard Hentges, Gerard Ribière, and Olivier Vincent. Experiments in mixed-integer linear programming. Mathematical Programming, 1:76–94, 1971.

Furkan Cantürk, Taha Varol, Reyhan Aydoğan, and Okan Örsan Özener. Scalable primal heuristics using graph neural networks for combinatorial optimization. Journal of Artificial Intelligence Research, 80:327–376, 2024. doi: 10.1613/JAIR.1.14972.

Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations. In International Conference on Machine Learning, pp. 1597–1607, 2020.

Antonia Chmiela, Elias B Khalil, Ambros Gleixner, Andrea Lodi, and Sebastian Pokutta. Learning to schedule heuristics in branch and bound. In Advances in Neural Information Processing Systems, pp. 24235–24246, 2021.

Marco Colombi, Renata Mansini, and Martin Savelsbergh. The generalized independent set problem: Polyhedral analysis and solution approaches. European Journal of Operational Research, 260(1):41–55, 2017.

Robert J Dakin. A tree-search algorithm for mixed integer programming problems. The Computer Journal, 8(3):250–255, 1965.

Shuhan Du, Junbo Tong, and Wenhui Fan. Learning efficient branch-and-bound for solving mixed integer linear programs. Applied Soft Computing, 172:112863, 2025. doi: 10.1016/j.asoc.2025.112863.

Haonan Duan, Pashootan Vaezipoor, Max B Paulus, Yangjun Ruan, and Chris Maddison. Augment with care: Contrastive learning for combinatorial problems. In International Conference on Machine Learning, pp. 5627–5642, 2022.

Paul Erdos, Alfréd Rényi, et al. On the evolution of random graphs. Publ. math. inst. hung. acad. sci, 5(1):17–60, 1960.

Christodoulos A Floudas and Xiaoxia Lin. Mixed integer linear programming in process scheduling: Modeling, algorithms, and applications. Annals of Operations Research, 139: 131–162, 2005.

JJH Forrest, JPH Hirst, and JOHN A Tomlin. Practical solution of large mixed integer programming problems with umpire. Management Science, 20(5):736–773, 1974.

Gerald Gamrath, Daniel Anderson, Ksenia Bestuzheva, Wei-Kun Chen, Leon Eifler, Maxime Gasse, Patrick Gemander, Ambros Gleixner, Leona Gottwald, Katrin Halbig, et al. The scip optimization suite 7.0. 2020. Zuse Institute Berlin.

Maxime Gasse, Didier Chételat, Nicola Ferroni, Laurent Charlin, and Andrea Lodi. Exact combinatorial optimization with graph convolutional neural networks. In Advances in Neural Information Processing Systems, pp. 15580–15592, 2019.

Carla P Gomes, Willem-Jan Van Hoeve, and Ashish Sabharwal. Connections in networks: A hybrid approach. In International conference on integration of artificial intelligence (AI) and operations research (OR) techniques in constraint programming, pp. 303–307. Springer, 2008.

Peter E Hart, Nils J Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. IEEE Transactions on Systems Science and Cybernetics, 4(2):100–107, 1968.

He He, Hal Daumé III, and Jason Eisner. Learning to search in branch-and-bound algorithms. In Advances in Neural Information Processing Systems, pp. 3293–3301, 2014.

Kaiming He, Haoqi Fan, Yuxin Wu, Saining Xie, and Ross Girshick. Momentum contrast for unsupervised visual representation learning. In IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 9726–9735, 2020.

Mike Hewitt, George L Nemhauser, and Martin WP Savelsbergh. Combining exact and heuristic approaches for the capacitated fixed-charge network flow problem. INFORMS Journal on Computing, 22(2):314–325, 2010.

Taoan Huang, Aaron M Ferber, Yuandong Tian, Bistra Dilkina, and Benoit Steiner. Searching large neighborhoods for integer linear programs with contrastive learning. In International Conference on Machine Learning, pp. 13869–13890, 2023.

Longlong Jing and Yingli Tian. Self-supervised visual feature learning with deep neural networks: A survey. IEEE transactions on pattern analysis and machine intelligence, 43 (11):4037–4058, 2020.

Elias B. Khalil, Christopher Morris, and Andrea Lodi. Mip-gnn: A data-driven framework for guiding combinatorial solvers. In Proceedings of the AAAI Conference on Artificial Intelligence, volume 36, pp. 10219–10227, 2022. doi: 10.1609/aaai.v36i9.21262.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980, 2014.

Abdel Ghani Labassi, Didier Chételat, and Andrea Lodi. Learning to compare nodes in branch and bound with graph neural networks. In Advances in Neural Information Processing Systems, pp. 32000–32010, 2022.

AH Land and AG Doig. An automatic method of solving discrete programming problems. 28:497–520, 1960.

Hongyu Liu, Haoyang Liu, Yufei Kuang, Jie Wang, and Bin Li. Deep symbolic optimization for combinatorial optimization: Accelerating node selection by discovering potential heuristics. In Proceedings of the Genetic and Evolutionary Computation Conference Companion, pp. 2067–2075, 2024. doi: 10.1145/3638530.3664128.

Ruizhi Liu, Liming Xu, Xulin Huang, Jingyan Sui, Shizhe Ding, Boyang Xia, Chungong Yu, and Dongbo Bu. Designing faster mixed integer linear programming algorithm via learning the optimal path, 2026. URL https://arxiv.org/abs/2601.16056.

Alexander Julian Mattick and Christopher Mutschler. Reinforcement learning for node selection in branch-and-bound. Transactions on Machine Learning Research, 2024.

Maxime Mulamba, Jayanta Mandi, Michelangelo Diligenti, Michele Lombardi, Victor Bucarey, and Tias Guns. Contrastive losses and solution caching for predict-and-optimize. arXiv preprint arXiv:2011.05354, 2020.

Aaron van den Oord, Yazhe Li, and Oriol Vinyals. Representation learning with contrastive predictive coding. arXiv preprint arXiv:1807.03748, 2018.

Vangelis Th Paschos. Applications of combinatorial optimization. 2014.

Hongbo Ren and Weijun Gao. A milp model for integrated plan and evaluation of distributed energy systems. Applied Energy, 87(3):1001–1014, 2010.

Stéphane Ross, Geoffrey Gordon, and Drew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In Proceedings of the fourteenth international conference on artificial intelligence and statistics, pp. 627–635, 2011.

Lara Scavuzzo, Feng Yang Chen, Didier Chételat, Maxime Gasse, Andrea Lodi, Neil Yorke-Smith, and Karen Aardal. Learning to branch with tree mdps. In Advances in Neural Information Processing Systems, pp. 18514–18526, 2022.

Lara Scavuzzo, Karen Aardal, Andrea Lodi, and Neil Yorke-Smith. Machine learning augmented branch and bound for mixed integer linear programming. Mathematical Programming, 2024. doi: 10.1007/s10107-024-02130-y.

Lorenzo Sciandra, Roberto Esposito, Andrea Grosso, Laura Sacerdote, and Cristina Zucca. Graph convolutional branch and bound. European Journal of Operational Research, 334 (3):798–809, 2026. doi: 10.1016/j.ejor.2026.03.036.

Jialin Song, Ravi Lanka, Albert Zhao, Aadyot Bhatnagar, Yisong Yue, and Masahiro Ono. Learning to search via retrospective imitation. arXiv preprint arXiv:1804.00846, 2018.

Zekun Tong, Yuxuan Liang, Henghui Ding, Yongxing Dai, Xinke Li, and Changhu Wang. Directed graph contrastive learning. In Advances in Neural Information Processing Systems, pp. 19580–19593, 2021.

Yuyang Wang, Jianren Wang, Zhonglin Cao, and Amir Barati Farimani. Molecular contrastive learning of representations via graph neural networks. Nature Machine Intelligence, 4(3):279–287, 2022.

Chuanguang Yang, Zhulin An, Helong Zhou, Fuzhen Zhuang, Yongjun Xu, and Qian Zhang. Online knowledge distillation via mutual contrastive learning for visual recognition. IEEE Transactions on Pattern Analysis and Machine Intelligence, 45(8):10212–10227, 2023.

Kaan Yilmaz and Neil Yorke-Smith. A study of learning search approximation in mixed integer branch and bound: Node selection in scip. AI, 2(2):150–178, 2021. doi: 10.3390/ai2020010.

Yuning You, Tianlong Chen, Yongduo Sui, Ting Chen, Zhangyang Wang, and Yang Shen. Graph contrastive learning with augmentations. In Advances in Neural Information Processing Systems, pp. 5812–5823, 2020.

Changwen Zhang, Wenli Ouyang, Hao Yuan, Liming Gong, Yong Sun, Ziao Guo, Zhichen Dong, and Junchi Yan. Towards imitation learning to branch for mip: A hybrid reinforcement learning based sample augmentation approach. In International Conference on Learning Representations, 2024.

Sijia Zhang, Shuli Zeng, Shaoang Li, Feng Wu, and Xiangyang Li. Learning to select nodes in branch and bound with sufficient tree representation. In International Conference on Learning Representations, 2025.

A Model Architecture Details

Following prior work Gasse et al. (2019); Labassi et al. (2022); Scavuzzo et al. (2022), we represent each node as a bipartite graph $G = (E, V, C)$. Here, $E \in \mathbb{R}^{|V|\times|C|}$ encodes the connectivity between variables and constraints; an edge (p, q) exists if the pth variable appears in the qth constraint, and the edge value is the coefficient $A_{q,p}$ from the constraint matrix A in equation 1. $V \in \mathbb{R}^{|V|\times d_V}$ contains variable node attributes, including type (binary, integer, continuous), objective coefficients, and bounds. $C \in \mathbb{R}^{|C|\times d_C}$ contains constraint node attributes, including constraint type $(\leq, =, \geq)$ and right-hand side value.

With the bipartite graph representation $G = (E, V, C)$, we encode each node using the encoder ϕ_θ. In ϕ_θ, the raw graph representation is first mapped into a latent space using three multi-layer perceptrons (MLPs):

$$\mathbf{E} = \mathrm{MLP}_E(E), \quad \mathbf{V} = \mathrm{MLP}_V(V), \quad \mathbf{C} = \mathrm{MLP}_C(C), \tag{7}$$

where $\mathbf{E} \in \mathbb{R}^{|C|\times|V|}$, $\mathbf{V} \in \mathbb{R}^{|V|\times d}$, and $\mathbf{C} \in \mathbb{R}^{|C|\times d}$ are the edge, variable, and constraint embeddings, respectively.

Then, a GNN is applied, in which each layer alternates between updating constraint and variable embeddings. The update rule is defined as

$$\begin{aligned} \mathbf{C}' &= \sigma\left(\mathbf{W_1}\mathbf{C} + \mathbf{W_2}\sum_{p\in\mathcal{N}(q)} \mathbf{E}_{q,p}\cdot\mathbf{V}_p\right), \\ \mathbf{V}' &= \sigma\left(\mathbf{W_1}\mathbf{V} + \mathbf{W_2}\sum_{q\in\mathcal{N}(p)} \mathbf{E}_{q,p}\cdot\mathbf{C}_q\right), \end{aligned} \tag{8}$$

where $\mathbf{W_1}, \mathbf{W_2} \in \mathbb{R}^{d\times d}$ are shared learnable weights, σ is a nonlinear activation, $\mathcal{N}(q)$ denotes the set of variables connected to the qth constraint, and $\mathcal{N}(p)$ denotes the set of constraints connected to the pth variable.

Finally, average pooling is applied over the variable embeddings

$$\mathbf{v}_{\text{avg}} = \frac{1}{|\mathcal{V}|}\sum_{p=1}^{|\mathcal{V}|}\mathbf{V}'_p, \tag{9}$$

where $\mathbf{v}_{\text{avg}} \in \mathbb{R}^d$ is the final embedding of the node. As constraint information has been integrated into $\mathbf{V}'$, no additional aggregation is needed.

B Benchmark Details

We evaluate TreeCL on three widely-used NP-hard MILP benchmarks, following the experimental setup of Labassi et al. (2022). Each benchmark contains training instances, held-out instances for testing imitation accuracy and solver performance, and larger transfer instances. The training and in-distribution test instances share the same size distribution. The benchmarks are:

- Fixed charge multicommodity network flow (FCMCNF) (Hewitt et al., 2010): generated using (Chmiela et al., 2021)'s code. We use instances with 20 nodes and 30 commodities for training and in-distribution testing, and 30 nodes and 45 commodities for transfer.
- Maximum satisfiability (MAXSAT) (Ansótegui & Gabàs, 2017): generated via the method in (Béjar et al., 2009). Numbers of nodes in training and in-distribution test instances are sampled from $[60, 70]$, and transfer instances from $[80, 100]$.
- Generalized independent set (GISP) (Colombi et al., 2017): generated using (Chmiela et al., 2021)'s code. Numbers of nodes in training and in-distribution test instances are sampled from $[60, 70]$, and transfer instances from $[70, 80]$.

All above problems rely on an underlying graph structure, generated by Erdős–Rényi random graphs (Erdos et al., 1960). The edge probability is 0.33 for FCMCNF, and 0.6 for MAXSAT and GISP.

In addition, we also evaluate on two real-world datasets:

- CORLAT (Gomes et al., 2008): a real-world dataset for constructing a wildlife corridor for grizzly bears in the Northern Rockies region.
- MIK (Atamtürk, 2003): a set of MILP problems with knapsack constraints.

Table 5 summarizes the statistics of the MILPs used in the experiments, including the number of instances and average problem sizes in terms of binary, integer, and continuous variables, as well as constraints.

Table 5: Statistics of MILP problem instances.

Metric	FCMCNF		MAXSAT		GISP		CORLAT	MIK
	Test	Transfer	Test	Transfer	Test	Transfer	Test	Test
Avg Binary	64.48	115.20	1426.00	1722.92	64.80	90.80	100.00	75.00
Avg Integer	0.00	0.00	0.00	0.00	0.00	0.00	0.00	300.00
Avg Continuous	1418.56	3456.00	0.00	0.00	623.66	1229.84	366.00	11.67
Avg Constraints	394.48	715.20	1361.16	1647.92	1247.66	2458.70	484.92	311.67
Num Instances	50	50	50	50	50	50	50	18

C Augmentation Results

Table 6 evaluates the effectiveness of our data augmentation strategy under limited expert supervision. Specifically, TreeCL-aug is trained on only 10% of the real data and 90% synthetic data, while all baselines are trained on 100% real data. Despite using far less supervision, TreeCL-aug achieves competitive performance across all benchmarks, outperforming most heuristic and learning-based baselines on FCMCNF and GISP, and close to the best-performing methods on MAXSAT. Notably, TreeCL-aug achieves 19 wins on FCMCNF and GISP, surpassing or matching fully supervised approaches. These results highlight the effectiveness of our augmentation strategy in addressing data scarcity, enabling the model to maintain low solve times and compact BnB trees even with limited oracle guidance. More broadly, this suggests that our method benefits not merely from more data, but from informative structural perturbations guided by optimality transitivity. The ability to generalize from synthetic data further supports the claim that TreeCL captures the intrinsic structure of the BnB process, rather than overfitting to raw feature-label patterns.

Table 6: Comparison of TreeCL-aug with baselines on standard test data.

Methods	FCMCNF			MAXSAT			GISP		
	Time (s)	Nodes	Wins	Time (s)	Nodes	Wins	Time (s)	Nodes	Wins
Oracle	3.38 ± 1.42	14.99 ± 4.13	-	5.28 ± 1.72	102.16 ± 2.48	-	3.68 ± 1.25	98.00 ± 3.29	-
Estimate	3.60 ± 1.48	21.40 ± 5.32	8	6.87 ± 1.69	176.58 ± 2.33	6	4.02 ± 1.25	218.06 ± 2.47	6
SCIP	4.04 ± 1.44	40.74 ± 4.57	11	7.99 ± 1.48	147.09 ± 1.96	3	3.82 ± 1.19	184.27 ± 1.96	18
SVM	3.49 ± 1.43	19.98 ± 5.05	9	6.23 ± 1.74	150.13 ± 2.60	12	3.99 ± 1.25	180.58 ± 2.78	3
RankNet	3.65 ± 1.48	20.96 ± 5.22	2	6.26 ± 1.75	142.13 ± 2.64	7	4.11 ± 1.28	203.49 ± 2.66	2
L2C	3.65 ± 1.50	21.24 ± 5.47	1	6.20 ± 1.90	138.03 ± 2.79	18	4.04 ± 1.27	184.31 ± 2.77	2
TreeCL-aug	3.49 ± 1.52	21.23 ± 5.38	19	6.58 ± 1.79	132.94 ± 2.64	4	3.77 ± 1.24	206.67 ± 2.46	19

C.1 Transfer Results with Limited Supervision

To further assess the generalization capability of our method under limited data, we evaluate TreeCL-aug, trained with only 10% real data and 90% augmented data, on the transfer sets. Table 7 presents the results. Despite with only 10% supervision, TreeCL-aug achieves strong performance across all benchmarks, demonstrating its ability to generalize to larger and more complex instances.

Table 7: Comparison of TreeCL-aug with baselines on transfer data.

Methods	FCMCNF			MAXSAT			GISP		
	Time (s)	Nodes	Wins	Time (s)	Nodes	Wins	Time (s)	Nodes	Wins
Oracle	16.13 ± 1.72	74.59 ± 4.46	-	7.40 ± 1.49	160.33 ± 2.00	-	18.21 ± 1.53	1062.20 ± 1.81	-
Estimate	19.14 ± 1.93	122.37 ± 5.14	8	9.97 ± 1.55	247.01 ± 1.84	9	20.26 ± 1.68	1434.95 ± 1.84	1
SCIP	21.28 ± 1.84	178.45 ± 4.30	10	10.75 ± 1.34	170.92 ± 1.66	5	15.64 ± 1.48	1533.48 ± 1.85	37
SVM	18.30 ± 1.84	132.77 ± 4.73	13	8.80 ± 1.60	225.32 ± 2.06	11	18.53 ± 1.51	1221.79 ± 1.69	4
RankNet	19.76 ± 2.00	112.60 ± 5.50	3	9.50 ± 1.57	238.33 ± 2.11	3	18.89 ± 1.52	1215.05 ± 1.73	0
L2C	19.91 ± 1.90	120.17 ± 4.94	4	9.31 ± 1.70	233.10 ± 2.23	8	18.80 ± 1.52	1196.42 ± 1.75	0
TreeCL-aug	19.59 ± 2.08	124.96 ± 5.19	12	8.87 ± 1.63	207.66 ± 2.09	14	17.17 ± 1.51	1182.28 ± 1.72	8

On GISP, TreeCL-aug achieves the lowest solve time and node count among the learning-based methods in Table 7. On MAXSAT, it records the most wins among the methods compared in that table. On FCMCNF, RankNet explores fewer nodes than TreeCL-aug (112.60 versus 124.96) but solves more slowly (19.76 versus 19.59 seconds). Thus, a smaller search tree does not necessarily imply a shorter solve time.

In summary, the results demonstrate that our data augmentation strategy not only reduces dependence on expert-labeled data, but also improves model generalization by exposing the learner to a wider range of decision structures. This leads to more effective node selection, better scaling to large instances, and stronger resilience to distribution shifts.

C.2 Transfer-Set Ablation

Table 8 reports the ablation results on the transfer sets. Across all benchmarks, the augmented model (TreeCL-aug) significantly outperforms TreeCL-10%, achieving lower solving times and fewer explored nodes. In GISP, for instance, TreeCL-aug reduces the node count from 1323.70 to 1182.28 and improves the solve time by over 3 seconds. These results confirm that the augmentation strategy not only improves in-distribution generalization, but also strengthens the model's ability to handle out-of-distribution, larger-scale problems.

Although the fully trained model (TreeCL) achieves the best performance overall, TreeCL-aug remains competitive, especially on GISP, where its solve time (17.17s) even slightly improves over TreeCL (17.31s). This observation suggests that data augmentation may inject structural diversity into training, which helps the model generalize better to complex or underrepresented structures. Moreover, the win counts show that TreeCL-aug closes much of the gap with the fully trained TreeCL (e.g., 15 vs. 29 wins on GISP), while substantially outperforming TreeCL-10% (6 wins). This demonstrates that effective augmentation can compensate for the lack of labeled data, yielding a scalable and generalizable node selection policy without requiring full supervision.

In summary, these results highlight that our augmentation strategy not only enhances performance but also promotes generalization to unseen problems, which is key for practical deployment where labeled data is limited.

Table 8: Ablation study on transfer sets.

Methods	FCMCNF			MAXSAT			GISP		
	Time (s)	Nodes	Wins	Time (s)	Nodes	Wins	Time (s)	Nodes	Wins
TreeCL-10%	21.22 ± 1.92	166.66 ± 4.40	10	8.93 ± 1.55	209.23 ± 2.10	5	20.62 ± 1.52	1323.70 ± 1.68	6
TreeCL-aug	19.59 ± 2.08	124.96 ± 5.19	14	8.87 ± 1.63	207.66 ± 2.09	10	17.17 ± 1.51	1182.28 ± 1.72	15
TreeCL	18.19 ± 1.84	114.16 ± 4.57	26	7.04 ± 1.49	155.93 ± 2.02	35	17.31 ± 1.51	1194.70 ± 1.72	29

D Additional Results

We report imitation accuracy on held-out test instances in Table 9. This measures how often each method selects the same node as the diving Oracle after training. While RankNet and L2C achieve high imitation accuracy, Table 1 shows inferior search performance. This

indicates that they can generalize in mimicking Oracle decisions, yet node-wise modeling does not translate such imitation into effective search. In comparison, TreeCL achieves both high accuracy and strong performance in terms of effectiveness and efficiency. These results confirm that modeling optimal-path consistency signals, rather than fitting feature-label correlations, leads to more effective and generalizable models.

Table 9: Imitation accuracy on held-out test instances.

Methods	FCMCNF	MAXSAT	GISP	CORLAT	MIK
SVM	0.926	0.934	0.965	0.973	0.862
RankNet	0.976	0.959	0.978	0.994	0.954
L2C	0.945	0.960	0.978	0.999	0.856
TreeCL	0.951	0.987	0.971	0.996	0.907

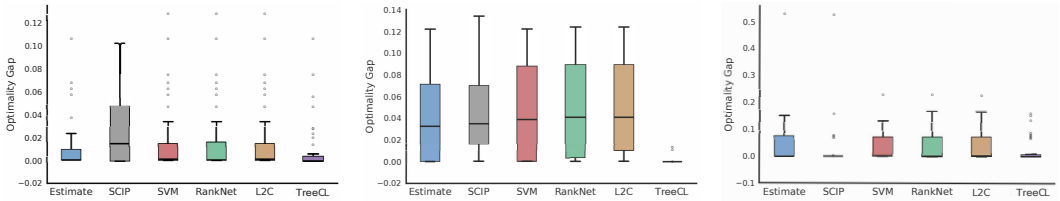
We examine the effect of the InfoNCE temperature in Table 10. Accuracy is measured on the held-out imitation-accuracy test triplets, whereas Nodes is measured on the separate solver-performance test instances.

Table 10: Test-set imitation accuracy (Acc) and node counts of TreeCL under different temperature values (τ); the two metrics use separate test sets.

τ	FCMCNF		MAXSAT		GISP	
	Acc	Nodes	Acc	Nodes	Acc	Nodes
0.03	0.951	21.51 ± 5.65	0.985	110.82 ± 2.52	0.978	176.68 ± 2.72
0.05	0.950	18.99 ± 5.54	0.983	105.77 ± 2.41	0.985	226.31 ± 2.54
0.07	0.951	19.25 ± 5.07	0.987	105.68 ± 2.50	0.971	181.69 ± 2.77
0.10	0.943	20.71 ± 5.49	0.984	106.86 ± 2.51	0.965	202.98 ± 2.74

FCMCNF and MAXSAT remain stable across the tested values, while GISP is more sensitive to τ. No temperature dominates on every benchmark. We therefore use $\tau = 0.07$ as a competitive default in the main experiments.

We further report the mean optimality gap of each method, which measures the relative difference between the best-found feasible solution and the optimal one. As a standard indicator of solution quality and convergence, a smaller gap implies that the method converges faster and achieves better performance within the time limit. As shown in Figure 2, TreeCL consistently achieves smaller optimality gaps across benchmarks, highlighting its superior convergence efficiency and solution quality compared to all baselines.



(a) FCMCNF (b) MAXSAT (c) GISP

Figure 2: Optimality gap across benchmark problems over Test sets.