Learning Branching Policies for MILPs with Proximal Policy Optimization

Abdelouahed Ben Mhamed[1], **Assia Kamal-Idrissi**[1], **Amal El Fallah Seghrouchni**[1, 2]

[1]Ai movement - International Artificial Intelligence Center of Morocco,
University Mohammed VI Polytechnic, Rabat, Morocco
[2]Lip6, Sorbonne University, Paris, France

Abstract

Branch-and-Bound (B&B) is the dominant exact solution method for Mixed Integer Linear Programs (MILP), yet its exponential time complexity poses significant challenges for large-scale instances. The growing capabilities of machine learning have spurred efforts to improve B&B by learning data-driven branching policies. However, most existing approaches rely on Imitation Learning (IL), which tends to overfit to expert demonstrations and struggles to generalize to structurally diverse or unseen instances. In this work, we propose Tree-Gate Proximal Policy Optimization (TGPPO), a novel framework that employs Proximal Policy Optimization (PPO), a Reinforcement Learning (RL) algorithm, to train a branching policy aimed at improving generalization across heterogeneous MILP instances. Our approach builds on a parameterized state space representation that dynamically captures the evolving context of the search tree. Empirical evaluations show that TGPPO often outperforms existing learning-based policies in terms of reducing the number of nodes explored and improving p-Primal-Dual Integrals (PDI), particularly in out-of-distribution instances. These results highlight the potential of RL to develop robust and adaptable branching strategies for MILP solvers.

1 Introduction

Mixed Integer Linear Programming (MILP) is a foundational mathematical optimization framework for addressing complex decision-making problems characterized by both continuous and discrete variables. The inclusion of integer constraints induces non-convexity in the feasible solution space, transforming the search for optimal solutions into a combinatorial NP-hard problem. MILP has been extensively applied in various domains, including logistics and routing optimization (Matai, Singh, and Mittal 2010) and energy systems planning (Ren and Gao 2010). This broad applicability has motivated sustained research efforts to improve solution methodologies, particularly in reducing computational complexity and accelerating solving times for large-scale real-world instances.

Branch-and-Bound (B&B) is a canonical exact algorithm to solve discrete optimization problems, particularly integral to MILP. The method iteratively constructs a search tree where each node represents a subproblem defined by a partial assignment of integer variables. Central to B&B are two operations: (1) branching, which partitions the feasible region of a subproblem into disjoint subsets, and (2) bounding, which computes primal and dual bounds to evaluate the subproblem's potential for improving the incumbent solution. Subproblems whose bounds exceed the current best objective value are pruned, a critical mechanism for mitigating combinatorial explosion and ensuring computational tractability (Land and Doig 1960).

Recent advances in Machine Learning (ML) have led to innovative methodologies to improve the efficiency of B&B algorithms for MILP. In particular, Tree-aware branching transformers (Lin et al. 2022) have emerged as a paradigm for learning adaptive branching policies that generalize across MILP instances, building on earlier efforts to parameterize B&B search trees for improved generalization (Zarpellon et al. 2021). Reinforcement Learning (RL) frameworks, such as those that employ tree-structured Markov Decision Processes (MDPs), have further demonstrated the feasibility of learning branching rules from scratch without expert supervision (Scavuzzo et al. 2022). Currently, advances in hierarchical representation learning have introduced transformer architectures tailored to tree-structured data, enabling the capture of long-range dependencies within B&B search trees through multi-head attention mechanisms (Lin et al. 2022). These architectures often integrate bi-directional propagation strategies to synthesize local node features with global tree context, thereby improving the fidelity of learned node representations (Lin et al. 2022), (Zhang et al. 2025). Collectively, these works underscore the potential of data-driven approaches to address the combinatorial challenges inherent in B&B.

To address the limitations of existing imitation-based branching strategies and advance the development of generalizable policies, we propose an RL framework based on Proximal Policy Optimization (PPO). Our approach enables an agent to interact directly with the B&B solver dynamically exploring the state-space and learning to select high value branching decisions that minimize search tree complexity.

2 Related Work

2.1 Imitation Learning and Hybrid Frameworks

Traditional B&B algorithms for MILP rely on handcrafted heuristics for variable selection, which often exhibit suboptimal performance and poor scalability. Early work by He, Daumé, and Eisner (2014) demonstrated the feasibility of imitation learning to guide node selection in B&B trees, achieving faster solve times than commercial solvers like Gurobi. However, such methods inherit biases from expert demonstrations and generalize poorly to unseen instances. Recent hybrid frameworks aim to mitigate these limitations; for example Zhang, Banitalebi-Dehkordi, and Zhang (2022) propose a hybrid framework that integrates imitation learning, PPO, and Monte Carlo Tree Search (MCTS) to optimize variable selection in B&B algorithms for solving MILPs. Imitation learning bootstraps the PPO agent by mimicking strong branching heuristics, accelerating training, and reducing initial exploration inefficiencies. PPO refines the policy via stable actor-critic updates, while MCTS enhances global optimality through modified upper confidence bounds and lookahead simulations, refined via cross-entropy loss. Evaluated on benchmarks like set covering, combinatorial auctions, capacitated facility location, and maximum independent set, the method outperforms traditional heuristics and prior ML approaches in node reduction and solving times. However, limitations include potential biases from expert heuristics, high computational overhead from MCTS simulations limiting scalability, restricted evaluation to synthetic datasets questioning real-world generalization, and lingering sample inefficiency in sparse-reward settings, suggesting avenues for hybrid RL enhancements.

2.2 Autonomous Reinforcement Learning for Branching

To address imitation learning's limitations, later studies focused on fully autonomous RL frameworks. Qu et al. (2022) introduces a Double Deep Q-Network (DDQN) based framework that leverages the demonstration data of strong branching heuristics to accelerate initial offline training, a prioritized storage mechanism to dynamically balance demonstration and self-generated data for improved policy quality, and a superior Q-network to enhance training robustness against large state-action spaces. However, limitations include the reliance on value-based DDQN, which may still suffer from overestimation biases and high variance in gradient estimation due to the expansive action spaces in MILP, potentially leading to suboptimal exploration and local optima, as evidenced by the ablation studies showing instability without the superior network; additionally, the off-policy nature and discrete action focus could limit adaptability to more dynamic or continuous branching scenarios.

2.3 Hybrid RL with Classical Optimization

Beyond pure RL, recent work integrates classical optimization techniques to enhance the performance of B&B. Parjadis et al. (2021) integrates RL with Decision Diagrams (DD) to enhance bounding mechanisms in B&B algorithms for combinatorial optimization, specifically using Q-learning augmented by Graph Neural Networks (GNN) to learn high-quality variable orderings for approximate DD, which yield tighter bounds and reduce the size of the search tree in the maximum independent set problem compared to traditional heuristics. However, limitations arise from the computational overhead of GNN forward passes, which inflate solving times despite fewer nodes, requiring hybrid approaches with greedy heuristics and caching that can compromise bound tightness; moreover, the off-policy Q-learning framework may encounter overestimation biases, high variance in large state-action spaces, and challenges in exploration, leading to suboptimal policies or instability during training.

2.4 Open Challenges and Our Contribution

Despite notable progress in hybrid and autonomous RL for branching, three obstacles persist. (i) Short-horizon bias. Policies bootstrapped from strong branching can inherit short-horizon preferences for immediate bound improvements, which need not correlate with global search efficiency (e.g. total tree size). Value-based agents trained off-policy are also prone to Q-overestimation under distribution shift. (ii) Sample inefficiency under sparse, delayed rewards. MILP solving unfolds over $10^3 - 10^5$ branching decisions, where suboptimal early actions compound and informative signals (gap closure or PDI) arrive primarily at fathoming or timeout. (iii) Runtime overhead. Look-ahead (e.g., MCTS) and heavy GNNs can tighten bounds but increase per-node latency, stressing fixed wall-clock budgets (e.g., 3600 s).

Contributions. We advance learning-to-branch with a fully on-policy Tree-Gate Proximal Policy Optimization (TGPPO) architecture targeting B&B tree size reduction. *(i) Stability without expert imprinting.* The PPO clipped-surrogate objective mitigates off-policy estimation bias, enabling stable training directly on solver-generated trajectories. This avoids imprinting from hand-crafted expert traces and retains stable gradients. *(ii) Architecture for variable-arity decisions.* We propose a permutation-equivariant Tree-Gate Transformer that conditions attention via multiplicative gates driven by local tree statistics. This improves credit assignment for the variable-sized candidate sets. *(iii) Throughput at scale.* A lightweight encoder and batched rollout engine ensure competitive wall-clock times without resorting to heavy MCTS or GNN modules.

Evaluation protocol. We use nested cross-validation with a dual criterion: (a) explored nodes for runs completed within a 1-hour budget, and (b) the PDI otherwise, capturing both rapid completion and anytime progress. *Summary of findings.* Across held-out benchmarks, TGPPO surpasses prior learning-based branchers on 72% of test instances (fewer nodes), while a clear, quantified gap to expert-designed heuristics such as RELPSCOST in SCIP[1] still remains.

[1]SCIP (Solving Constraint Integer Programs) (Gleixner et al. 2018) is a leading non-commercial, extensible MILP solver framework, which we use as the B&B environment for our agent.

3 Optimization and Learning Frameworks

3.1 Branch-and-Bound

We consider a general MILP problem, formulated as:

$$\begin{aligned} \min \quad & c^T x \\ \text{s.t.} \quad & Ax \leq b, \\ & x_i \in \mathbb{Z}, \quad \forall i \in \mathcal{I}, \\ & x_j \in \mathbb{R}, \quad \forall j \in \mathcal{J}. \end{aligned} \tag{1}$$

Here, x is the decision vector with integer variables x_i ($i \in \mathcal{I}$) and continuous variables x_j ($j \in \mathcal{J}$). The objective is defined by $c \in \mathbb{R}^n$, and constraints by $A \in \mathbb{R}^{m \times n}$ and $b \in \mathbb{R}^m$.

The B&B algorithm solves this MILP by exploring a search tree. At each node, it solves the Linear Programming (LP) relaxation (Eq. (1) without integrality). If the solution x^* is not integer-feasible, the algorithm *branches* on a fractional variable $x_i^* \in \mathcal{I}$ by creating two subproblems with the added constraints $x_i \leq \lfloor x_i^* \rfloor$ and $x_i \geq \lceil x_i^* \rceil$. The LP solution of a subproblem provides a lower bound. A node is *pruned* if its lower bound exceeds the best-known integer-feasible solution (the incumbent). This process repeats recursively until the search space is exhausted, guaranteeing optimality.

3.2 Proximal Policy Optimization

PPO (Schulman et al. 2017) is a state-of-the-art RL algorithm that improves traditional policy gradient methods by optimizing a surrogate objective function while maintaining stable and efficient updates. PPO belongs to the class of policy-based RL algorithms, where an agent learns a parameterized policy $\pi_\theta(a \mid s)$ that maps states s to a probability distribution over actions a, with parameters θ optimized to maximize the expected cumulative reward.

A key challenge in policy optimization is to ensure a balance between policy improvement and stability. Large updates to the policy can lead to performance collapse, while overly conservative updates can slow learning. To address this, PPO introduces a clipped surrogate objective that constrains the policy update within a trust region.

The PPO objective employs an Actor-Critic architecture, where:

- The *actor* network $\pi_\theta(a|s)$ selects branching actions (variables) based on the current state of the solver.
- The *critic* network $V_\phi(s)$ estimates the expected cumulative reward (the value function).

$$\begin{aligned} \mathcal{L}(\theta, \phi) = \mathbb{E}_{(s,a)\sim\pi_{\theta_{\text{old}}}} \Bigg[\min \Big(& r_\theta(s,a) A_\phi^{\pi_{\theta_{\text{old}}}}(s,a), \\ & \text{clip}\left(r_\theta(s,a), 1-\epsilon, 1+\epsilon\right) A_\phi^{\pi_{\theta_{\text{old}}}}(s,a) \Big) \Bigg] \\ & - \lambda_1 \mathbb{E}_s \left[\left(V_\phi(s) - R(s) \right)^2 \right] + \lambda_2 \mathcal{H}(\pi_\theta(\cdot|s)) \end{aligned} \tag{2}$$

where:

- $r_\theta(s,a) = \frac{\pi_\theta(a|s)}{\pi_{\theta_{\text{old}}}(a|s)}$ is the probability ratio between the new and old policies.
- $A_\phi^{\pi_{\theta_{\text{old}}}}(s,a)$ is the advantage estimate, typically computed using Generalized Advantage Estimation (GAE) (Schulman et al. 2015).
- $R(s)$ is the empirical discounted return (the value target), used as the target for training the critic $V_\phi(s)$.
- $\mathcal{H}(\pi_\theta(\cdot|s))$ is the policy entropy that encourages exploration.
- λ_1, λ_2 are weighting coefficients for the critic loss and entropy bonus, respectively.

4 Methodology

Our methodology builds upon the feature representations introduced in prior works (Zarpellon et al. 2021; Lin et al. 2022), while proposing a novel RL framework based on PPO to acquire a generalized branching policy. Following (Zarpellon et al. 2021), we adopt hand-crafted features. Let $\mathcal{C}_t$ be the set of candidate variables at step t. The state is then represented by a matrix of candidate features, $\mathbf{C}_t \in \mathbb{R}^{|\mathcal{C}_t| \times 25}$, and a vector of local tree state features, $Tree_t \in \mathbb{R}^{61}$. Together, these features encapsulate the evolving context of the B&B process. In contrast to static deep neural network architectures or graph-based enhancements employed in T-BranT (Lin et al. 2022), we model branching as a sequential decision-making task and train an online PPO agent to refine variable selection through direct interaction with the B&B environment.

4.1 Policy Architecture

Our policy follows an actor-critic design tailored to branching in MILP search trees. Each decision state comprises (i) a set of candidate branching variables with heterogeneous, instance-dependent cardinality L, and (ii) a tree context capturing both the current node state and local MIP features. The architecture in Figure 1 is built around three principles: (1) permutation-equivariance over candidates; (2) contextualization of candidates by the current tree state; and (3) capacity control via a gated multi-layer reduction that adapts the effective network width to the current state. Concretely, both actor and critic share the same front-end: linear embeddings for variables and tree context, a Transformer encoder that operates over the candidate set with key-padding masks, and a light bi-directional matching module that fuses local tree information into the candidate stream. The heads then diverge: the actor produces a categorical distribution over candidates, while the critic aggregates masked candidate features into a scalar state-value.

Let t index the current B&B node. The SCIP solver exposes three feature blocks:

$$\mathbf{C}_t \in \mathbb{R}^{|\mathcal{C}_t| \times d_c}, \ \mathbf{n}_t \in \mathbb{R}^{d_n}, \ \mathbf{m}_t \in \mathbb{R}^{d_m},$$

where each row $\mathbf{c}_{t,i}$ of $\mathbf{C}_t$ (with $d_c = 25$) summarises candidate variable i, while $\mathbf{n}_t$ and $\mathbf{m}_t$ (with $d_n = 8$, $d_m = 53$) encode the current node and MIP search tree features, respectively.

Linear embeddings: We project every feature block into a common d_h-dimensional space:

$$\tilde{\mathbf{c}}_{t,i} = \mathbf{W}_c \, \text{LN}(\mathbf{c}_{t,i}), \qquad \tilde{\mathbf{t}}_t = \mathbf{W}_t \, \text{LN}[\mathbf{n}_t; \mathbf{m}_t], \tag{3}$$

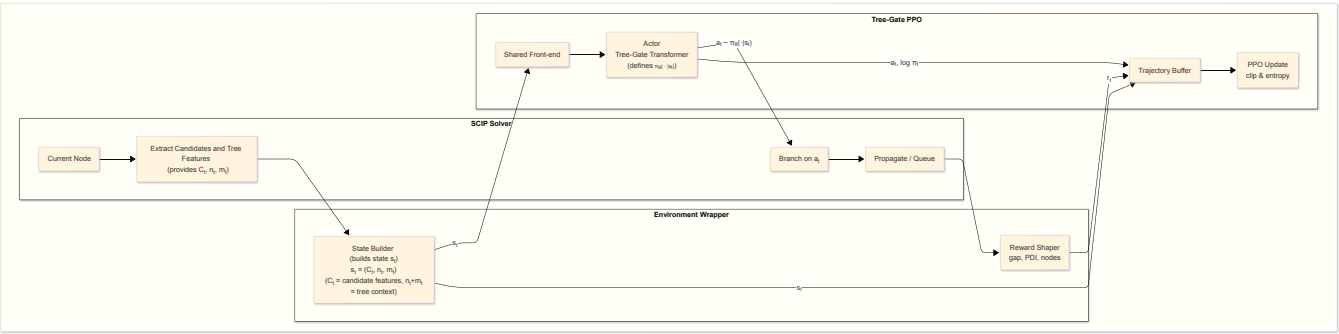

Figure 1: The Tree–Gate PPO agent interacts with SCIP through an environment wrapper: candidate features and local tree features are encoded, the actor chooses a branching variable, SCIP executes the branch, and the environment returns rewards. Trajectories accumulate in a buffer and are periodically used for PPO optimisation, closing the learning loop.

with $\mathbf{W}_c \in \mathbb{R}^{d_h \times d_c}$ and $\mathbf{W}_t \in \mathbb{R}^{d_h \times (d_n + d_m)}$ learned parameters and $\mathrm{LN}(\cdot)$ denoting layer normalization.

Candidate–tree fusion: By concatenating each embedded candidate with the embedded tree context and reprojecting via $\mathbf{W}_g \in \mathbb{R}^{d_h \times 2d_h}$,

$$\mathbf{z}_{t,i}^{(0)} = \mathbf{W}_g \, [\tilde{\mathbf{c}}_{t,i}; \tilde{\mathbf{t}}_t],$$

we obtain the matrix $\mathbf{Z}_t^{(0)} \in \mathbb{R}^{|\mathcal{C}_t| \times d_h}$, which is permutation-equivariant in $|\mathcal{C}_t|$.

Self-attention encoder: A N-layer Transformer encoder maps $\mathbf{Z}_t^{(0)}$ to $\mathbf{Z}_t^{(N)}$ while respecting padding masks $\mathbf{M}_{\text{pad}} \in \{0,1\}^{|\mathcal{C}_t|}$:

$$\mathbf{Z}_t^{(N)} = \mathrm{Transformer}^{(N)}\big(\mathbf{Z}_t^{(0)}, \mathbf{M}_{\text{pad}}\big).$$

Bi-directional tree–candidate matching: Following Lin et al. (2022), a BiMatching network refines candidate representations via soft mutual attention:

$$\begin{aligned}
\alpha_{t,i} &= \mathrm{softmax}_i\big((\mathbf{W}_{t1}\tilde{\mathbf{t}}_t)^\top \mathbf{z}_{t,i}^{(N)}\big), \\
\beta_{t,i} &= \mathrm{softmax}_i\big((\mathbf{W}_{c1}\mathbf{z}_{t,i}^{(N)})^\top \tilde{\mathbf{t}}_t\big), \\
\mathbf{e}_t &= \textstyle\sum_i \alpha_{t,i}\mathbf{z}_{t,i}^{(N)}, \quad \mathbf{d}_{t,i} = \beta_{t,i}\tilde{\mathbf{t}}_t, \\
\mathbf{r}_{t,i} &= \sigma(\mathbf{W}_3\mathbf{e}_t + \mathbf{W}_4\mathbf{d}_{t,i}) \odot \mathbf{e}_t \\
&\quad + \big(1 - \sigma(\mathbf{W}_3\mathbf{e}_t + \mathbf{W}_4\mathbf{d}_{t,i})\big) \odot \mathbf{d}_{t,i}
\end{aligned} \tag{4}$$

yielding refined candidate tensor $\mathbf{R}_t = [\mathbf{r}_{t,1}, \ldots, \mathbf{r}_{t,|\mathcal{C}_t|}]^\top$.

Tree-gated branching head (Actor). A hierarchical multi-layer perceptron of depth K reduces each $\mathbf{r}_{t,i}$ to a scalar logit $\ell_{t,i}$ under the multiplicative control of the tree vector:

$$\begin{aligned}
\mathbf{g}^{(k)} &= \sigma(\mathbf{U}_k\tilde{\mathbf{t}}_t), \; k = 1{:}K, \\
\mathbf{q}_{t,i}^{(k)} &= f_k\big(\mathbf{q}_{t,i}^{(k-1)} \odot \mathbf{g}^{(k)}\big), \; \mathbf{q}_{t,i}^{(0)} = \mathbf{r}_{t,i}, \\
\ell_{t,i} &= \mathbf{q}_{t,i}^{(K)} \in \mathbb{R}.
\end{aligned} \tag{5}$$

The branching policy is the categorical distribution $\pi_\theta(i \mid s_t) = \mathrm{softmax}_i(\ell_{t,i})$.

Algorithm 1: Tree–Gate PPO (TGPPO)

Require: Instance set $\mathcal{D}$, horizon H, epochs E
Initialise policy θ, value ϕ
$\mathcal{B} \leftarrow \emptyset$ ▷ trajectory buffer
for all episodes **do**
  Sample MILP $\mathcal{I} \sim \mathcal{D}$; reset solver
  for $t \leftarrow 1$ **to** H **do**
    Extract features $(\mathbf{C}_t, \mathbf{n}_t, \mathbf{m}_t)$
    $a_t \sim \pi_\theta(\cdot \mid s_t)$
    Execute a_t, obtain reward r_t, next state s_{t+1}
    $\mathcal{B} \leftarrow \mathcal{B} \cup (s_t, a_t, r_t, s_{t+1})$
    if s_{t+1} is terminal **or** $|\mathcal{B}| = H$ **then**
      Compute advantages $\hat{A}$ and returns via GAE
      for $u \leftarrow 1$ **to** E **do**
        Sample mini-batch $\mathcal{M} \subset \mathcal{B}$
        Update (θ, ϕ) with clipped loss Eq. (2)
      end for
      $\mathcal{B} \leftarrow \emptyset$
      break ▷ start next episode
    end if
  end for
end for

Value head (Critic). A permutation-invariant state vector $\bar{\mathbf{r}}_t = \frac{1}{\sum_i (1 - M_{\text{pad},i})} \sum_i (1 - M_{\text{pad},i})\, \mathbf{r}_{t,i}$ is concatenated with $\tilde{\mathbf{t}}_t$ and projected through a two-layer MLP, followed by the same tree-gated reduction, producing the scalar value estimate $V_\phi(s_t)$.

4.2 Reward Signal Design

All three rewards are instance-normalized by the node, gap and PDI statistics obtained with SCIP's default branching rule RELPSCOST; this guarantees scale-robust credit assignment across easy and hard MILPs. Briefly: **H1** penalizes node expansions relative to the baseline and adds a status-dependent speed-up bonus; **H2** log-scales the penalty and introduces a pace term plus gap/PDI shaping; **H3** adapts the weights of those components to the problem's difficulty, favoring gap closure on very large instances. The mixing co-

efficients were fixed *a priori* to prioritize node reduction and deliberately *not* included in the hyper-parameter search, avoiding over-tuning while keeping the optimization space tractable.

Since H3 was selected as the optimal signal by our hyper-parameter search (see Section 5.3), we present its core formulation here. It computes a difficulty index $d \in [0, 1]$ based on the baseline node count B and uses it to define adaptive weights for node efficiency (w_{nodes}), gap closure (w_{gap}), PDI (w_{pdi}), and pace (w_{pace}). The step reward is a clipped, weighted sum of these components, supplemented by a progress term q_t:

$$r_t^{\text{H3}} = \text{clip}_{[-1,1]}\Big(w_{\text{nodes}}\, e_t + w_{\text{pace}}\, p_t - w_{\text{gap}}\, g_t' + w_{\text{pdi}}\, d_t + w_{\text{prog}}\, q_t\Big) \tag{6}$$

where e_t measures log-scaled node efficiency relative to the baseline, p_t is a pace-keeping term, g_t' penalizes the current gap, and d_t rewards PDI reduction. This is combined with a large terminal reward R_T^{H3} based on the final solver status. The precise definitions of all components and the H1/H2 formulations are detailed in Appendix.

4.3 Training Procedure

The learning pipeline comprises two phases: *(i)* hyper–parameter selection via nested cross-validation using OPTUNA (Akiba et al. 2019), and *(ii)* final policy training with the configuration chosen in Phase 1. Because the public benchmark contains only 25 MILP instances, we run each instance under five independent SCIP seeds (seed $\in \{0, \ldots, 4\}$). The random permutations induced by these seeds act as inexpensive, solver-level *data augmentation* that leaves the underlying combinatorial structure unchanged, yet increases experiential diversity five-folds.

Phase 1: Hyper-parameter Selection.

Search space. The hyper-parameters and their ranges appear in Table 1. Exploration is performed with the TPE sampler provided by OPTUNA (Akiba et al. 2019).

Nested cross-validation. We adopt a two-level $(k_{\text{outer}}, k_{\text{inner}}) = (5, 2)$ design. For every candidate configuration $\boldsymbol{\theta} \in \mathcal{H}$ we minimise, on the inner folds, a composite score that balances the number of explored nodes N (measured on runs that finish within the time limit) and the PDI (measured on *all* runs):

$$\min_{\theta \in \mathcal{H}} \frac{1}{k_{\text{in}}} \sum_{i=1}^{k_{\text{in}}} \big(0.6\, \text{SGM}_i(N) + 0.4\, \text{SGM}_i(\text{PDI})\big),$$

where the *shifted geometric mean* (SGM) is defined as

$$\text{SGM}(x) = \exp\Big[\tfrac{1}{m} \sum_j \ln(x_j + 100)\Big] - 100,$$

and the $+100$ shift prevents the logarithm from exploding on small counts. Stratification over the (instance, seed) pairs preserves the baseline difficulty distribution across all folds.

Pruning rule. We enable Optuna's *median pruner*: a trial is stopped early whenever its running composite score fails to improve the best-so-far median across the inner folds for three consecutive iterations, thereby discarding unpromising settings quickly.

Table 1: Hyperparameter search space for Tree–Gate PPO.

Parameter	Range / Set
Hidden size d	$\{64, 128, 256, 384\}$
Transformer layers L	$\{2, 3, 4, 5, 6\}$
Attention heads h	$\{2, 4, 8\}$
Dropout δ	$[0, 0.3]$
Actor LR α_{act}	$[10^{-6}, 3\times10^{-4}]$
Critic LR α_{crt}	$[10^{-6}, 3\times10^{-4}]$
PPO clip ϵ	$[0.05, 0.3]$
Entropy weight β	$[10^{-5}, 10^{-2}]$
GAE γ	$[0.92, 0.999]$
GAE λ	$[0.8, 0.99]$
Minibatch size b	$\{32, 64, 128, 256, 512\}$
Epochs E	$\{1, 2, 3, 4, 5, 6\}$
Reward	$\{H1, H2, H3\}$

Selection criterion. The configuration achieving the lowest outer-fold composite score is selected for Phase 2.

Phase 2: Final Policy Training. With hyper-parameters fixed, the agent is retrained on the full $25 \times 5 = 125$ augmented problems for 500 complete episodes. An episode corresponds to one B&B run up to optimality or the 3600-second cutoff. Seeds are reshuffled each epoch to prevent over-fitting to a fixed permutation.

5 Experiments

5.1 Datasets

Following Lin et al. (2022), we train and evaluate on the exact same curated benchmark drawn from the public MILP libraries MIPLIB 3/2010/2017 and CORAL[2]. Lin et al. (2022) ran SCIP 6.0.1 with the default RELPSCOST rule on every publicly available instance and kept only those that (i) are feasible and bounded, (ii) achieve a finite primal gap within a 7200 s limit, and (iii) contain no more than 200 000 variables, discarding the rest. The resulting collection comprises 25 training instances and 66 testing instances as shown in table 2; the test set is further split into 33 easy and 33 hard problems according to whether all policies (except random) solve them within 3600 s. Using the same dataset ensures strict comparability with prior learning-based branching work while covering a heterogeneous mix of set–covering, facility-location, auction, and scheduling models.

5.2 Settings

Our experimental framework utilizes SCIP 6.0.1 as the underlying MILP solver, interfaced via a customized version of PYSCIPOPT (Maher et al. 2016) to access the solver's internal state and decision points required for training the PPO policy. This customization builds on prior work by Zarpellon et al. (2021), which demonstrated the feasibility of integrating ML with B&B solvers through parameterized search trees. To isolate the impact of branching variable selection (BVS) and align with our proof-of-concept study design,

[2]All instances are distributed under the CC BY–SA 4.0 licence.

Table 2: List of training and test instances selected from MIPLIB and CORAL.

Train (25): 30n20b8, air04, air05, cod105, comp21-2idx, dcmulti, eil33-2, istanbul-no-cutoff, l152lav, lseu, misc03, neos20, neos21, neos-476283, neos648910, pp08aCUTS, rmatr100-p10, rmatr100-p5, rmatr200-p5, roi5alpha10n8, sp150×300d, stein27, supportcase7, swath1, vpm2
Test (66): aflow40b, app1-2, atlanta-ip, bab5, bc1, bell3a, bell5, biella1, binkar10_1, blend2, dano3_5, fast0507, harp2, map10, map16715-04, map18, map20, mik-250-20-75-4, mine-166-5, misc07, msc98-ip, mspp16, n2seq36q, n3seq24, neos11, neos12, neos-1200887, neos-1215259, neos13, neos18, neos-4722843-widden, neos-4738912-atrato, neos-480878, neos-504674, neos-504815, neos-512201, neos-584851, neos-603073, neos-612125, neos-612162, neos-662469, neos-686190, neos-801834, neos-803219, neos-807639, neos-820879, neos-829552, neos-839859, neos-892255, neos-950242, ns1208400, ns1830653, nu25-pr12, nw04, opm2-z7-s2, p0201, pg, pigeon-10, pp08a, rail507, roll3000, rout, satellites1-25, seymour1, sp98ir, unitcal_7

we adopt a streamlined solver configuration following established BVS benchmarking practices (Linderoth and Savelsbergh 1999). Specifically, we disable primal heuristics and provide the known optimal solution value as an initial cutoff, ensuring the solver focuses exclusively on tree-search efficiency rather than primal bound discovery (Gamrath and Schubert 2018). Although this represents a simplification of a full solver run, this "expert" setting is a standard benchmarking practice to isolate the performance of branching rules, as done in (Lin et al. 2022; Zarpellon et al. 2021). We analyze the impact of this choice in our limitations (6). A one-hour time limit per instance balances computational feasibility with solution quality assessment.

5.3 Training

After the nested cross-validation described in 4.3, the best hyper-parameter configuration was found to be: *actor_lr* $= 2.4 \times 10^{-4}$, *critic_lr* $= 1.2 \times 10^{-4}$, hidden size $d_{\text{model}} = 256$, $L = 5$ transformer layers, $H = 8$ attention heads, dropout 0.05, PPO clip coefficient 0.16, entropy bonus 3.0×10^{-3}, $\gamma = 0.97$, $\lambda_{\text{GAE}} = 0.92$, mini-batch size 256, three optimization epochs per update, and the difficulty–adaptive reward H3. We then retrained a fresh policy with these values on the full training dataset.

Training proceeds for 500 complete episodes. Each episode consists of a full B&B run (initialized with a new seed) and is followed by three gradient-descent epochs over the collected trajectories. Both actor and critic are optimized with AdamW ($\beta_1 = 0.9$, $\beta_2 = 0.999$); the PPO objective includes the above clip coefficient and an entropy regularizer of 3.0×10^{-3}. All experiments are implemented in PyTorch (Paszke et al. 2019) and executed on 40-core CPU nodes.

5.4 Results and Analysis

Protocol and metrics. Each test instance (5.1) is evaluated under five independent seeds (0–4), yielding $33 \times 5 = 165$ runs for the *easy* subset and the same for the *hard*. We record (i) explored nodes (NNODES) and (ii) (PDI). On easy instances we compare NNODES; on hard instances, we compare PDI (lower is better).

Table 3: Per–instance dominance of TGPPO. Entries are the % of test instances for which TGPPO improves over the baseline (SGM over five seeds).

Baseline	% win (Nnodes)	% win (PDI)
TBRANT (Lin et al. 2022)	78.8	90.62
BRANT (Lin et al. 2022)	72.7	71.87
LTBRANT (Lin et al. 2022)	72.7	71.87
TREE (Zarpellon et al. 2021)	72.7	68.75
RANDOM	90.9	93.75
PSCOST	66.7	78.12
RELPSCOST	18.2	46.87

Aggregates. We report two complementary views: (1) an *overall* shifted geometric mean

$$\text{SGM}(x_{1:m}) = \exp\Big[\tfrac{1}{m}\sum_{j=1}^{m}\ln(S + x_j)\Big] - S,$$

computed across all instance–seed pairs (with S=100 for NNODES and S=0 for PDI); and (2) a *per-instance* comparison: first compute the SGM over the five seeds of an instance, then compare TGPPO head-to-head against each baseline (a win is counted if TGPPO's SGM is strictly smaller).

Head-to-head dominance. Table 3 shows the fraction of instances on which TGPPO dominates each baseline (SGM over seeds). TGPPO surpasses the prior state-of-the-art learner TBRANT on 78.8% of instances by NNODES and 90.6% by PDI; see Fig. 2, where points below the dashed diagonal indicate TGPPO wins. It also dominates other learning branchers (BRANT, LTBRANT, TREE) on roughly 73% by NNODES and $\approx$ 69–72% by PDI. Among the classical rules, PSCOST is most competitive (wins 66.7% / 78.1% for NNODES/PDI), while RELPSCOST is harder to beat in nodes (18.2%) but is still outperformed in PDI nearly half the time (46.9%). These outcomes indicate consistent per-instance superiority of TGPPO, even though some instances still favor hand-crafted rules.

Overall comparisons (easy: NNODES; hard: PDI). We analyze within-instance ranks with a Friedman omnibus test (Friedman 1937) and Nemenyi post-hoc (Nemenyi 1963) (single-step adjusted p). The easy subset shows significant differences across the five learning policies (χ^2=22.215, df=4, p=1.82×10^{-4}), with TGPPO significantly better ranked than BRANT, LTBRANT, and TREE, and statistically tied with TBRANT. On the hard subset, ranks by PDI also differ significantly (χ^2=18.05, df=4, p=0.00121); TGPPO is significantly better ranked than BRANT, LTBRANT, and TBRANT, and marginal vs. TREE. For directionality on raw values, one-sided paired Wilcoxon tests favor TGPPO on easy instances (wins on 24–26 of 33; median node reductions 80–142; Holm-adjusted $p \in [0.09, 0.11]$) and on hard

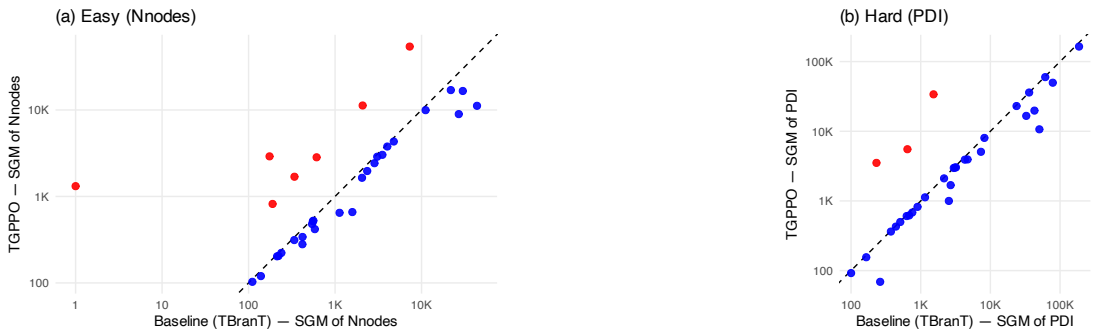

Figure 2: Head-to-head scatter of TGPPO vs. TBRANT (log–log): points below the dashed diagonal ($y < x$) indicate TGPPO wins; (a) comparison of the easy instances using Nnodes, (b) comparison of the hard instances using PDI

Table 4: Compact omnibus and post-hoc: Friedman test on within-instance ranks and Nemenyi p (single-step) for TGPPO vs. others. Bold $p<.05$.

Subset	χ^2 (df)	**p**	BRANT	LTBRANT	TBRANT	TREE
Easy (NNODES)	22.215 (4)	0.0001816	**0.0064**	**0.0027**	0.5000	**0.0011**
Hard (PDI)	18.05 (4)	0.001207	**0.0026**	**0.0105**	**0.0046**	0.0557

instances (wins on 22–29 of 33; median PDI reductions 82–111; adjusted $p \leq 0.0314$).

Practical takeaway. TGPPO reduces search effort on easy instances and, critically, yields markedly better PDI on hard instances where optimality is not reached within one hour, a regime of substantial practical interest.

6 Conclusion and Perspectives

This paper introduced TGPPO, an on-policy RL framework that learns branching policies for MILPs directly from solver interactions. The approach combines (i) a permutation–equivariant transformer encoder over candidate variables, (ii) multiplicative tree-gates that condition decisions on local node and tree statistics, and (iii) a PPO training loop with instance-normalized rewards and a nested cross-validation protocol for robust model selection. Empirically, TGPPO consistently improves search efficiency over prior learning-based branchers. On easy instances it reduces the number of explored nodes, while on hard instances (where runs may time out) it delivers markedly lower PDIs. Together, these results indicate that on-policy RL can be a competitive and stable alternative to imitation-driven methods for learning-to-branch, narrowing, though not erasing, the gap to strong, hand-crafted rules such as RELPSCOST.

Limitations and threats to validity. Our experimental protocol focuses on branching-variable selection under a streamlined solver configuration to isolate branching effects; this leaves end-to-end interactions with primal heuristics and other solver modules for future study. The training set is necessarily small at the instance level and curated for comparability with prior work, which may limit coverage of industrial distributions. Although our instance-normalized rewards improve stability, reward shaping remains a sensitive design choice. Finally, while our rollout engine provides solid CPU throughput, full system-level benchmarking on diverse hardware and wall-clock metrics (including memory footprint) is still required for operational deployment.

Perspectives. We see several promising directions to extend this work:

1. **End-to-end solver integration.** Re-enable primal heuristics, cuts, and default parameterizations to evaluate the net impact on wall-clock time, optimality gaps, and anytime behavior; formulate training as a multi-objective problem balancing nodes, PDI, and time.
2. **Representation learning.** Enrich the encoder with bipartite variable–constraint message passing or lightweight decision-diagram features; add memory over partial subtrees to better capture long-range dependencies without heavy graph stacks.
3. **Data- and compute-efficiency.** Explore offline pretraining on solver logs followed by on-policy fine-tuning (Behavior Cloning $\rightarrow$ PPO), and investigate advantage normalization, return decomposition, or truncated credit assignment tailored to deep search trees.
4. **Reward design and objectives.** Systematically study difficulty-adaptive rewards and risk-sensitive objectives (e.g., CVaR/quantile variants) to trade off mean performance and tail robustness on hard instances.

In sum, TGPPO advances learning-to-branch by coupling stable on-policy optimization with a tree-aware architecture that respects the symmetries of candidate sets and the context of the search. We expect that integrating richer representations, end-to-end objectives, and deployment-conscious systems design will further tighten the gap to expert heuristics and make RL-based branching practical for large, heterogeneous MILPs.

References

Akiba, T.; Sano, S.; Yanase, T.; Ohta, T.; and Koyama, M. 2019. Optuna: A Next-generation Hyperparameter Optimization Framework. *CoRR*, abs/1907.10902.

Friedman, M. 1937. The use of ranks to avoid the assumption of normality implicit in the analysis of variance. *Journal of the american statistical association*, 32(200): 675–701.

Gamrath, G.; and Schubert, C. 2018. Measuring the Impact of Branching Rules for Mixed-Integer Programming. In Kliewer, N.; Ehmke, J. F.; and Borndörfer, R., eds., *Operations Research Proceedings 2017*, 165–170. Cham: Springer International Publishing. ISBN 978-3-319-89920-6.

Gleixner, A.; Bastubbe, M.; Eifler, L.; Gally, T.; Gottwald, R.; Hendel, G.; Hojny, C.; Koch, T.; Lübbecke, M.; Maher, S.; Miltenberger, M.; Müller, B.; Pfetsch, M.; Puchert, C.; Rehfeldt, D.; Schlösser, F.; Schubert, C.; Serrano, F.; Shinano, Y.; Viernickel, J.; Walter, M.; Wegscheider, F.; Witt, J.; and Witzig, J. 2018. *The SCIP Optimization Suite 6.0*. ZIB Report. Zuse Institut Berlin.

He, H.; Daumé, H.; and Eisner, J. 2014. Learning to Search in Branch and Bound Algorithms. In Ghahramani, Z.; Welling, M.; Cortes, C.; Lawrence, N.; and Weinberger, K., eds., *Advances in Neural Information Processing Systems*, volume 27. Curran Associates, Inc.

Land, A. H.; and Doig, A. G. 1960. An Automatic Method of Solving Discrete Programming Problems. *Econometrica*, 28(3): 497–520.

Lin, J.; Zhu, J.; Wang, H.; and Zhang, T. 2022. Learning to branch with Tree-aware Branching Transformers. *Knowledge-Based Systems*, 252: 109455.

Linderoth, J. T.; and Savelsbergh, M. W. P. 1999. A Computational Study of Search Strategies for Mixed Integer Programming. *INFORMS Journal on Computing*, 11(2): 173–187.

Maher, S.; Miltenberger, M.; Pedroso, J. P.; Rehfeldt, D.; Schwarz, R.; and Serrano, F. 2016. PySCIPOpt: Mathematical Programming in Python with the SCIP Optimization Suite. In Greuel, G.-M.; Koch, T.; Paule, P.; and Sommese, A., eds., *Mathematical Software – ICMS 2016*, 301–307. Cham: Springer International Publishing. ISBN 978-3-319-42432-3.

Matai, R.; Singh, S. P.; and Mittal, M. L. 2010. Traveling Salesman Problem: An Overview of Applications, Formulations, and Solution Approaches. In Davendra, D., ed., *Traveling Salesman Problem*, chapter 1. Rijeka: IntechOpen.

Nemenyi, P. B. 1963. *Distribution-free multiple comparisons.* Princeton University.

Parjadis, A.; Cappart, Q.; Rousseau, L.-M.; and Bergman, D. 2021. Improving Branch-and-Bound Using Decision Diagrams and Reinforcement Learning. In Stuckey, P. J., ed., *Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, 446–455. Cham: Springer International Publishing. ISBN 978-3-030-78230-6.

Paszke, A.; Gross, S.; Massa, F.; Lerer, A.; Bradbury, J.; Chanan, G.; Killeen, T.; Lin, Z.; Gimelshein, N.; Antiga, L.; Desmaison, A.; Kopf, A.; Yang, E.; DeVito, Z.; Raison, M.; Tejani, A.; Chilamkurthy, S.; Steiner, B.; Fang, L.; Bai, J.; and Chintala, S. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In Wallach, H.; Larochelle, H.; Beygelzimer, A.; d'Alché-Buc, F.; Fox, E.; and Garnett, R., eds., *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc.

Qu, Q.; Li, X.; Zhou, Y.; Zeng, J.; Yuan, M.; Wang, J.; Lv, J.; Liu, K.; and Mao, K. 2022. An Improved Reinforcement Learning Algorithm for Learning to Branch. *CoRR*, abs/2201.06213.

Ren, H.; and Gao, W. 2010. A MILP model for integrated plan and evaluation of distributed energy systems. *Applied Energy*, 87(3): 1001–1014.

Scavuzzo, L.; Chen, F.; Chetelat, D.; Gasse, M.; Lodi, A.; Yorke-Smith, N.; and Aardal, K. 2022. Learning to Branch with Tree MDPs. In Koyejo, S.; Mohamed, S.; Agarwal, A.; Belgrave, D.; Cho, K.; and Oh, A., eds., *Advances in Neural Information Processing Systems*, volume 35, 18514–18526. Curran Associates, Inc.

Schulman, J.; Moritz, P.; Levine, S.; Jordan, M.; and Abbeel, P. 2015. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*.

Schulman, J.; Wolski, F.; Dhariwal, P.; Radford, A.; and Klimov, O. 2017. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

Zarpellon, G.; Jo, J.; Lodi, A.; and Bengio, Y. 2021. Parameterizing Branch-and-Bound Search Trees to Learn Branching Policies. *Proceedings of the AAAI Conference on Artificial Intelligence*, 35(5): 3931–3939.

Zhang, S.; Zeng, S.; Li, S.; Wu, F.; and Li, X. 2025. Learning to Select Nodes in Branch and Bound with Sufficient Tree Representation. In *The Thirteenth International Conference on Learning Representations*.

Zhang, T.; Banitalebi-Dehkordi, A.; and Zhang, Y. 2022. Deep Reinforcement Learning for Exact Combinatorial Optimization: Learning to Branch. In *2022 26th International Conference on Pattern Recognition (ICPR)*, 3105–3111.

A Reward Formulations

Notation. Let B denote the baseline node count from SCIP's RELPSCOST (per instance), n_t the cumulative nodes explored by our policy up to step t, and $\Delta n_t = n_t - n_{t-1}$. Let Δ^t_{gap} be the relative optimality gap at t, PDI^t the primal–dual integral, and $\tau_t = t/T_{\max} \in [0,1]$ the normalized wall time. At termination ($t{=}T$) the solver status $\text{stat}_T \in \{\text{OPTIMAL}, \text{TIMELIMIT}, \text{INFEASIBLE}, \text{UNBOUNDED}\}$. We use the numerically safe mappings $\tanh_s(x) = \tanh(sx)$ and $\text{ratio}_c(a,b) = \min(a/\max\{b, 10^{-12}\}, c)$.

H1: Baseline-Normalised Node Efficiency. Step penalty and terminal bonus:

$$r_t^{\text{H1}} = -\tanh_\alpha\left(\frac{\Delta n_t}{0.02\,B+1}\right), \quad \alpha{=}1, \tag{7}$$

$$R_T^{\text{H1}} = \begin{cases} 1+2s, & \text{OPTIMAL}, \\ 0.5+1.5s, & \text{INFEASIBLE/UNBOUNDED}, \\ 0.2s+0.6\,g_T+0.2\,d_T, & \text{TIMELIMIT}, \\ 0.2s, & \text{otherwise}, \end{cases} \tag{8}$$

where $s = \text{ratio}_c(B, n_T; 3)$, $g_T = \tanh_1(\Delta^0_{\text{gap}} - \Delta^T_{\text{gap}})$, and $d_T = \tanh_1\big((\text{PDI}^0 - \text{PDI}^T)/\text{PDI}^0\big)$.

H2: Log-Scaled Efficiency with Pace Shaping. With $\beta{=}1.5$, $\rho{=}0.7$:

$$e_t = \tanh_\beta\left(1 - \frac{\log(1+n_t)}{\log(1+B)}\right), \tag{9}$$

$$p_t = \tanh_\beta\left(\frac{B\tau_t^\rho - n_t}{B\tau_t^\rho + 1}\right). \tag{10}$$

$$g_t = \begin{cases} \tanh_1\big(\frac{\Delta^{t-1}_{\text{gap}} - \Delta^t_{\text{gap}}}{|\Delta^{t-1}_{\text{gap}}| + 10^{-9}}\big), & t > 0, \\ 0, & t = 0, \end{cases} \tag{11}$$

$$d_t = \tanh_1\left(\frac{\text{PDI}^{t-1} - \text{PDI}^t}{\text{PDI}^0}\right). \tag{12}$$

$$r_t^{\text{H2}} = \text{clip}_{[-1,1]}(0.5e_t + 0.2p_t + 0.2g_t + 0.1d_t)\,. \tag{13}$$

$$R_T^{\text{H2}} = \begin{cases} 1+2.5s, & \text{OPTIMAL}, \\ 0.7+2s, & \text{INFEASIBLE/UNBOUNDED}, \\ 0.4s+0.4g_T+0.2d_T, & \text{TIMELIMIT}, \\ 0.3s, & \text{otherwise}. \end{cases} \tag{14}$$

H3: Difficulty-Adaptive Reward. Define a difficulty index

$$d = \sigma\left(\frac{\log(1+B) - \log 2}{\log(1+10^6) - \log 2}\right) \in [0,1].$$

$$w_{\text{nodes}} = 0.55(1-d) + 0.25d, \tag{15}$$

$$w_{\text{gap}} = 0.10(1-d) + 0.30d, \tag{16}$$

$$w_{\text{pdi}} = 0.05(1-d) + 0.20d, \tag{17}$$

$$w_{\text{pace}} = 0.15(1-d) + 0.10d, \tag{18}$$

$$w_{\text{prog}} = 0.15. \tag{19}$$

$$r_t^{\text{H3}} = \text{clip}_{[-1,1]}\Big(w_{\text{nodes}}\,e_t + w_{\text{pace}}\,p_t - w_{\text{gap}}\,\tanh_{0.5}\Big(\frac{\Delta^t_{\text{gap}}}{\Delta^0_{\text{gap}} + 10^{-9}}\Big) + w_{\text{pdi}}\,d_t + w_{\text{prog}}\,q_t\Big). \tag{20}$$

$$R_T^{\text{H3}} = \begin{cases} 1+3s, & \text{OPTIMAL}, \\ 0.8+2s, & \text{INFEASIBLE/UNBOUNDED}, \\ 0.5s+0.3\,g_T+0.2\,d_T, & \text{TIMELIMIT}, \\ 0.3s, & \text{otherwise}. \end{cases} \tag{21}$$

Weight choice. The component weights in H2/H3 were fixed a priori to prioritize shrinking the search tree (node efficiency) while preserving anytime progress (gap/PDI). We deliberately *did not* tune these weights during hyper-parameter search to avoid expanding the search space and to keep the evaluation reproducible.

B Additional Results

Table 5: Number of nodes explored by each branching policy. Bold figures indicate that TGPPO outperforms TBRANT on the same instance.

Instance	**brant**	**lbrant**	**pscost**	**random**	**relpscost**	**tbrant**	**tgppo**	**tree**
ALL TEST	1602.91	1484.65	1822.01	4787.13	731.37	1318.33	1588.72	1671.27
aflow40b	21357.41	18275.65	19828.14	274377.17	9331.03	21795.26	**16971.64**	21745.75
bc1	4875.32	4391.09	4269.85	7420.14	2565.54	4771.82	**4315.00**	4988.66
bell3a	2235.47	1964.98	1502.46	2033.78	2025.22	2050.18	**1643.00**	2029.09
bell5	355.11	424.39	566.32	359.41	362.81	419.75	**280.04**	392.04
binkar10_1	3375.36	3274.07	3242.54	5090.92	2489.30	3072.18	**2879.40**	4273.29
blend2	515.78	525.50	621.53	1031.12	172.94	420.43	**340.00**	356.27
dano3_5	219.21	229.95	239.48	435.13	54.47	138.81	**119.92**	248.46
map20	2504.94	341.74	800.78	6389.80	227.48	337.59	**313.00**	4611.05
misc07	26380.91	25763.24	17822.84	39116.32	29191.05	26930.40	**8952.64**	28352.56
n2seq36q	1354.12	800.41	1444.75	729.64	1077.22	1126.69	**647.00**	740.33
neos-1200887	56066.21	61136.40	53279.03	995155.72	250472.28	43785.80	111624.42	48046.76
neos-1215259	28570.73	26758.50	7896.79	73239.39	893.60	2358.62	**1968.15**	2365.55
neos-504674	15322.44	18949.81	11840.86	84011.86	7740.03	7295.20	54057.68	18558.15
neos-504815	3817.37	4143.30	3751.79	16947.46	2527.60	3523.58	**3028.15**	3683.93
neos-512201	6406.57	4388.47	2816.04	13815.58	1508.07	2084.19	11264.13	3102.75
neos-584851	1488.80	1669.34	2011.74	2718.10	231.42	583.53	**418.62**	2327.39
neos-612125	122.74	122.70	178.99	547.69	69.25	110.57	**103.00**	115.91
neos-612162	241.40	247.21	290.31	911.83	138.31	221.57	**205.00**	197.78
neos-801834	605.95	716.27	1339.37	1282.85	230.62	553.97	**517.00**	670.64
neos-803219	33220.80	26894.64	30731.61	112534.17	12394.67	29982.03	**16575.41**	26737.87
neos-807639	12315.86	10683.13	5190.83	20090.11	3880.27	11128.42	**9945.08**	14668.51
neos-820879	352.41	398.22	927.56	3838.14	125.90	339.23	1690.00	408.96
neos-892255	878.62	746.42	736.23	1048.06	636.76	611.80	2835.74	1085.85
neos-950242	3392.59	3602.97	4728.49	29949.69	2618.27	2854.37	**2422.81**	2541.50
ns1208400	204.94	281.61	1187.38	1640.88	58.75	174.73	906.98	267.86
nu25-pr12	217.14	264.12	342.47	1658.41	21.39	189.39	819.00	204.81
nw04	1.00	1.00	71.08	135.64	1.00	1.00	1315.00	1.00
p0201	161.82	186.23	200.86	219.77	18.51	213.65	**203.00**	232.61
pg	553.12	575.58	565.24	652.29	134.59	561.26	**523.99**	570.56
pp08a	208.94	214.49	254.26	304.71	45.04	239.32	**222.51**	221.77
satellites1-25	917.57	700.71	3050.29	5051.30	530.28	1583.85	**659.70**	1625.83
sp98ir	3096.86	4080.53	5663.54	39208.94	1225.49	4016.84	**3776.87**	4425.60
unitcal_7	621.67	590.95	656.11	2429.35	197.43	542.50	**480.00**	1433.84

Table 6: PDI of each policy. Bold figures indicate that TGPPO outperforms TBRANT on the same instance.

Instance	**brant**	**lbrant**	**pscost**	**random**	**relpscost**	**tbrant**	**tgppo**	**tree**
ALL TEST	3347.67	2776.06	3879.65	10404.71	1981.85	2922.72	2945.33	2939.30
app1-2	48065.58	64566.58	17571.65	74204.23	48510.64	50486.20	**10662.62**	68145.71
atlanta-ip	27341.37	25261.71	33241.90	16611.90	16161.90	23019.96	25162.92	25152.92
bab5	2950.77	3300.29	3649.85	3515.74	2857.45	3104.14	**3017.28**	3161.85
bell4	389.60	359.27	415.42	751.32	338.24	373.05	**336.38**	386.70
fast0507	759.61	759.04	1008.48	4282.23	853.35	756.44	**682.07**	750.39
harp2	1068.41	0.00	75.20	253.58	26.20	165.24	**155.52**	106.20
map10	33678.51	8603.33	13714.21	33711.26	6860.22	8202.81	**8026.37**	29696.21
map16715-04	153711.00	68731.63	139907.50	194165.73	109656.59	61628.88	**59857.30**	183932.06
map18	249.33	1207.50	2670.84	14625.19	1734.81	1153.24	**1126.14**	3213.67
mik-250-20-75-4	342.36	374.06	102.46	3788.17	107.03	644.41	5522.93	256.99
mine-166-5	589.38	621.70	1418.08	9936.65	592.82	628.77	**608.52**	604.81
msc98-ip	3063.46	2561.40	3048.09	30005.12	2759.71	2982.23	**2971.23**	2171.01
mspp16	31422.96	92314.93	34335.58	38343.58	27227.17	32715.92	**1662.65**	35488.06
n2seq24	7441.40	7705.33	7222.47	6500.93	7383.59	7321.92	**5070.49**	7881.80
neos-4722843-widden	82183.73	87286.31	73739.80	120234.25	75606.13	78598.83	**64733.27**	71320.47
neos-4738912-atrato	6130.94	4726.75	3786.63	3270.19	1875.18	2531.34	**999.91**	1975.43
neos-480878	144.58	144.85	144.75	3461.05	68.94	231.52	3516.70	111.06
neos-603073	15230.76	6017.58	10356.21	19456.74	34982.95	4302.37	**3911.20**	18902.46
neos-662469	544.29	516.65	485.98	706.49	483.42	508.64	**498.46**	514.40
neos-686190	643.02	777.75	603.07	6630.59	770.69	683.12	**619.00**	670.46
neos-829552	10265.45	2906.35	6747.03	9010.09	3760.47	2687.66	**1685.03**	4602.34
neos-839859	337.65	384.93	277.96	7120.80	173.49	439.95	**428.35**	295.92
neos-892255	220186.70	187674.12	267356.75	257304.52	213861.48	187294.79	**164249.46**	173672.63
neos12	44087.85	53639.77	87728.51	92286.19	11128.05	42895.16	**19790.43**	15487.38
neos13	1368.44	1443.07	85810.06	84649.32	1444.75	1526.50	38838.92	1604.58
neos18	2089.19	1899.67	12441.87	18554.88	546.53	2162.18	**2107.20**	1359.88
ns1830653	5540.18	3964.29	10834.00	37564.13	3931.31	4706.58	**3938.23**	4473.86
pigeon-10	36002.07	36002.36	36002.12	36003.96	36002.22	36003.55	**36001.56**	36002.18
rail507	952.20	912.05	1174.57	4718.87	989.64	899.03	**821.88**	1043.87
roll3000	321.50	536.84	341.09	929.54	147.57	261.69	**69.01**	311.69
rout	1135.77	821.40	817.41	4624.89	132.84	3203.34	**3029.28**	846.96
seymour1	95.41	95.40	131.77	2120.43	95.94	100.11	**92.13**	96.09