MAESTRO: AN EFFICIENT HIERARCHICAL REINFORCEMENT LEARNING FOR BRANCH-AND-CUT ORCHESTRATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Branch-and-cut solvers achieve strong performance on integer programming but face an inherent trade-off between search progress and the memory consumed as the LP relaxation is repeatedly tightened, branched, and re-solved. We introduce MAESTRO, a hierarchical reinforcement learning framework that learns to orchestrate solver behavior by deciding *when* to branch, *when* to add cuts, and *when* to invoke cut pruning, within a unified Manager–Worker architecture. The Manager makes high-level strategic decisions from a global graph embedding, while the Worker performs context-aware cut selection using shared embeddings from a bipartite graph neural network. We train this hierarchy in three phases (Worker pre-training, Manager training with frozen Worker, joint fine-tuning) using Proximal Policy Optimization with a difficulty curriculum. Rather than optimizing for time or memory in isolation, MAESTRO explicitly models and jointly optimizes their interaction. Across a diverse suite of benchmark instances, we show that MAESTRO achieves 100% solve rates on easy and medium Set Cover instances while using only 53–106 MB of memory compared to SCIP's 2–3 GB, representing a 29–40$\times$ reduction. On medium instances, MAESTRO solves 16% faster than SCIP (60.5s vs. 72.4s), while on hard instances it achieves a 60% solve rate, whereas SCIP solves none within comparable time budgets. When measured by Resource Cost – the product of time and memory that captures the total computational footprint – MAESTRO achieves 39–40$\times$ better efficiency than SCIP across difficulty levels. These results enable practical deployment in memory- and latency-constrained settings such as edge devices, embedded systems, and dense multi-tenant environments where traditional solvers' multi-gigabyte footprints are prohibitive.

1 INTRODUCTION

Integer Programming (IP) underlies a vast array of real-world applications, from supply chain optimization Pochet & Wolsey (2006) and scheduling Song et al. (2022) to network design Wolsey (2020). The branch-and-cut algorithm remains the dominant paradigm for solving IPs, combining systematic enumeration (branching) with constraint tightening (cutting planes). Despite decades of refinement, modern solvers like SCIP Gamrath et al. (2020) and Gurobi still face fundamental challenges: solving efficiency depends critically on *when* to branch versus add cuts, *which* variable to branch on, and *which* cuts to select from potentially thousands of candidates. These operations are tightly coupled: aggressive cut generation can dramatically tighten the LP relaxation but consume memory and slow down LP re-solves, while premature branching may lead to exponential tree growth. Traditional solvers rely on hand-crafted heuristics that fail to adapt to instance-specific structure.

Beyond classical server-based deployments, modern optimization increasingly arises in resource-constrained and distributed settings such as embedded planning systems, edge devices, and multi-tenant cloud services Shuvo et al. (2022). In these contexts, solver behavior must balance computational progress with strict memory and latency budgets, motivating learning-based orchestration that explicitly accounts for system-level constraints rather than optimizing solver components in isolation.

Recent advances in machine learning for combinatorial optimization have shown promise in learning individual solver components: GNNs for variable branching (Gasse et al., 2019; Khalil et al.,

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.

2017), neural scoring for cut selection (Tang et al., 2020; Paulus et al., 2022), and learned node selection (He et al., 2014). While these methods demonstrate gains on their respective tasks, they share three fundamental limitations: (i) *Isolated component optimization* – branching, cut selection, and cut management are treated as independent subproblems, ignoring their interdependencies; (ii) *Resource inefficiency as collateral damage* – a cut selector trained for bound improvement adds cuts aggressively, unaware that this bloats memory and slows iterations; (iii) *Local GNN representations* – prior GNN-based methods (Gasse et al., 2019; Nair et al., 2020) use embeddings to score individual decisions but do not leverage them for cross-component communication, preventing cuts from being selected in a manner informed by the broader solving trajectory.

This work asks: *Can we learn a unified policy that coordinates branching and cutting decisions while maintaining efficient resource utilization?* Our key insight is that effective solver control requires not just better individual decisions but *strategic orchestration* – learning when each action type is most productive given the current problem state.

Contributions. We present MAESTRO, a hierarchical RL framework with three contributions:

1. **Unified three-action orchestration**[1]**.** We jointly learn when to branch, when to add cuts, and when to trigger cut pruning within a single Manager policy. (Cut removal itself is a heuristic pruning routine; see Appendix A.12.)
2. **Hierarchical Manager–Worker architecture for IP.** A Manager makes strategic decisions (which action type) and a Worker executes tactical cut selection (which cuts), enabling long-horizon coordination separately from local cut-quality assessment.
3. **Context-aware cut selection via shared GNN embeddings.** Unlike prior cut selectors that score cuts using only local features (efficacy, sparsity, parallelism), our Worker conditions on global graph embeddings from the Manager's bipartite GNN, allowing cut selection to account for the current solving state, active constraints, and variable relationships.

In summary, MAESTRO reframes IP solver control as a hierarchical RL problem over three fundamental actions: branching, cut addition, and pruning invocation. The Manager conditions on a bipartite GNN encoding of the current LP state, capturing variable–constraint relationships that inform strategic timing. When cut selection is chosen, a specialized Worker policy scores candidate cuts using both local features (efficacy, sparsity) and global context from the Manager's embeddings. We train this hierarchy with PPO (Schulman et al., 2017) in three phases (Worker pre-training, Manager training with frozen Worker, joint fine-tuning) over a difficulty curriculum. The decomposition improves credit assignment and yields interpretable policies whose action distributions reveal learned solving strategies.

2 Related Work

Learning for IP Solving. The application of machine learning to IP solving has gained significant traction. Early work focused on learning branching policies through imitation learning Khalil et al. (2017); Alvarez et al. (2017). Gasse et al. (2019) demonstrated that GNNs can effectively encode the bipartite structure of IP instances for branching decisions. Recent work has extended this to cut selection: Tang et al. (2020) introduced RL for cutting-plane selection, while Paulus et al. (2022) proposed imitation-learning approaches. Wang et al. (2024) developed a hierarchical sequence model for cut selection. Unlike these approaches that treat components independently, we learn the coordination between strategic decisions (when to cut vs. branch) and tactical execution (which cuts to select).

GNNs for Combinatorial Optimization. GNNs have become the dominant architecture for learning over IP structure (Cappart et al., 2023). The bipartite variable-constraint graph (Gasse et al., 2019) naturally captures problem structure, with message passing propagating information between variables and constraints. Prior work uses these embeddings to score individual decisions – branches (Gasse et al., 2019) or cuts (Huang et al., 2022) – but **does not leverage GNN representations for cross-component communication**. We extend this paradigm by using the Manager's graph embeddings as context for the Worker's cut selection, enabling globally-informed tactical decisions.

[1] Hence the name MAESTRO.

Hierarchical Reinforcement Learning. Hierarchical RL Sutton et al. (1999); Dietterich (2000) decomposes complex tasks into manageable sub-problems. The options framework Sutton et al. (1999) and feudal networks Vezhnevets et al. (2017) provide principled approaches to temporal abstraction. Our Manager-Worker architecture draws inspiration from these frameworks but adapts them to the unique structure of IP solving, where the hierarchy is over decision *types* (strategic vs. tactical) rather than temporal scales. The Manager selects *what* to do (branch/cut/prune); when cutting is selected, the Worker determines *which* cuts to add or remove.

3 Problem Formulation

Definition 3.1 (Integer Program)**.** An Integer Program (IP) seeks $\min_{\mathbf{x}}\{\mathbf{c}^\top \mathbf{x} : \mathbf{A}\mathbf{x} \leq \mathbf{b}, x_j \in \mathbb{Z}\ \forall j \in \mathcal{I}\}$, where $\mathbf{c} \in \mathbb{R}^n$, $\mathbf{A} \in \mathbb{R}^{m\times n}$, $\mathbf{b} \in \mathbb{R}^m$, and $\mathcal{I} \subseteq \{1, \ldots, n\}$ indexes integer variables.

The branch-and-cut algorithm maintains a search tree of LP relaxations. At each node it obtains LP solution $\mathbf{x}^*_{\text{LP}}$ and bound z^*_{LP}; if $\mathbf{x}^*_{\text{LP}}$ is integer-feasible the node is pruned, otherwise the solver may *branch* (partition the feasible region on a fractional variable), *add cuts* (valid inequalities $\boldsymbol{\alpha}^\top \mathbf{x} \leq \beta$ tightening the relaxation), or *prune cuts* (remove inactive constraints to reduce LP size).

Definition 3.2 (Bipartite IP Graph)**.** Following Gasse et al. (2019), we represent an IP as $\mathcal{G} = (\mathcal{V}, \mathcal{C}, \mathcal{E})$ with variable nodes $\mathcal{V} = \{v_1, \ldots, v_n\}$, constraint nodes $\mathcal{C} = \{c_1, \ldots, c_m\}$, and edges $\mathcal{E} = \{(v_j, c_i) : W_{ij} \neq 0\}$ weighted by the corresponding non-zero coefficient.

We formulate IP solving as a hierarchical Markov Decision Process (MDP) $\mathcal{M} = (\mathcal{S}, \mathcal{A}, P, R, \gamma)$.

Definition 3.3 (State Space)**.** The state $s_t \in \mathcal{S}$ at decision step t encodes the complete solving context:

$$s_t = \left(\mathcal{G}_t, \mathbf{x}^*_t, z^*_{\text{LP},t}, z^*_{\text{IP},t}, \text{gap}_t, n_t, d_t, \mathcal{C}_{\text{pool},t}\right) \tag{1}$$

Here, $\mathcal{G}_t$ denotes the bipartite graph representation of the current LP state (Definition 3.2), while $\mathbf{x}^*_t \in \mathbb{R}^n$ is the current LP relaxation solution. The bounds $z^*_{\text{LP},t}, z^*_{\text{IP},t} \in \mathbb{R}$ represent the LP relaxation objective value and the best known integer-feasible objective, respectively, from which we derive the optimality gap $\text{gap}_t = (z^*_{\text{IP},t} - z^*_{\text{LP},t})/|z^*_{\text{IP},t}|$ measuring the relative distance to proven optimality. The search progress is captured by $n_t \in \mathbb{N}$, the number of branch-and-bound nodes explored, and $d_t \in \mathbb{N}$, the current depth in the search tree. Finally, $\mathcal{C}_{\text{pool},t}$ denotes the active cut pool containing valid inequalities generated through separation that currently participate in the LP relaxation.

Definition 3.4 (Hierarchical Action Space)**.** The Manager selects a meta-action from $\mathcal{A}^{\text{meta}} = \{\textsc{Branch}, \textsc{SelectCuts}, \textsc{PruneCuts}\}$, capturing fundamentally different strategies: BRANCH partitions the feasible region on a fractional integer-typed variable; SELECTCUTS tightens the LP via valid inequalities; PRUNECUTS removes inactive cuts to shrink the LP (Manager invokes the action; the removal procedure itself is heuristic). Given a^M, the sub-action space is

$$\mathcal{A}^{\text{sub}}(a^M) = \begin{cases} \mathcal{V}_{\text{frac}} \cap \mathcal{I} & a^M = \textsc{Branch} \\ \{\mathcal{C}_{\text{sel}} \subseteq \mathcal{C}_{\text{cand}} : |\mathcal{C}_{\text{sel}}| \leq B\} & a^M = \textsc{SelectCuts} \\ \{\text{age-based}, \text{efficacy-based}\} & a^M = \textsc{PruneCuts} \end{cases} \tag{2}$$

The branching set is restricted to integer-typed variables since SCIP rejects branching on continuous variables, which matters in mixed-integer relaxations such as Facility Location. The full action is the tuple $a_t = (a^M_t, a^{\text{sub}}_t)$.

Definition 3.5 (Hierarchical Policy)**.** The joint policy $\pi = (\pi^M_\theta, \pi^W_\psi)$ has Manager $\pi^M_\theta : \mathcal{S} \to \Delta(\mathcal{A}^{\text{meta}})$ selecting strategic meta-actions and Worker $\pi^W_\psi : \mathcal{S} \times \mathcal{C}_{\text{cand}} \to \Delta(\mathcal{C}_{\text{cand}})$ selecting cut subsets when $a^M = \textsc{SelectCuts}$. Define $V^\pi(s) = \mathbb{E}_{\tau\sim\pi}[\sum_{t=0}^\infty \gamma^t r_t \mid s_0 = s]$, $Q^\pi(s,a) = \mathbb{E}_{\tau\sim\pi}[\sum_{t=0}^\infty \gamma^t r_t \mid s_0 = s, a_0 = a]$, and the advantage $A^\pi(s,a) = Q^\pi(s,a) - V^\pi(s)$.

Definition 3.6 (Reward Function)**.** We use a composite reward balancing progress and efficiency:

$$r_t = \underbrace{\lambda_{\text{gap}}\Delta\text{gap}_t}_{\text{progress}} - \underbrace{\lambda_{\text{time}}\Delta t}_{\text{time}} + \underbrace{\lambda_{\text{solve}}\mathbf{1}[\text{solved}]}_{\text{bonus}} - \underbrace{\lambda_{\text{mem}}\Delta m_t}_{\text{memory}} \tag{3}$$

with $(\lambda_{\text{gap}}, \lambda_{\text{time}}, \lambda_{\text{solve}}, \lambda_{\text{mem}}) = (1.0, 0.005, 1.0, 0.001)$.

Problem 3.7 (Strategic Orchestration for IP)**.** Find (θ, ψ) maximizing $\mathbb{E}_{\tau\sim\pi}[\sum_{t=0}^T \gamma^t r_t]$ with $\gamma = 0.99$, where trajectories τ are generated by the joint policy $\pi = (\pi^M_\theta, \pi^W_\psi)$.

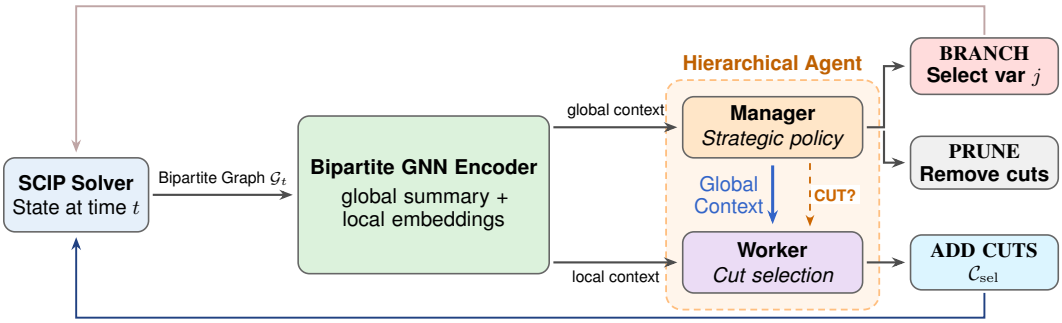


Figure 1: **MAESTRO Architecture.** The **Bipartite GNN** encodes MILP state into global ($\mathbf{h}^g$) and node embeddings. The **Manager** (meta-controller) selects BRANCH, PRUNE, or delegates CUT to the specialized **Worker**. The Worker uses node features enriched by the Manager's global context (highlighted blue arrow) for informed cut selection. Actions update the solver state for the next iteration via clean feedback loops (top: Branch/Prune, bottom: Cuts).

4 THE MAESTRO FRAMEWORK

The architecture of MAESTRO comprises three tightly integrated components (Figure 1): (1) a bipartite GNN encoder that learns structural representations, (2) a Manager agent for strategic meta-decisions, and (3) a Worker agent for context-aware cut selection. The key architectural insight is that the GNN serves dual purposes: providing the Manager with global state representation and furnishing the Worker with contextual embeddings for informed cut selection.

4.1 BIPARTITE GRAPH NEURAL NETWORK ENCODER

We encode the IP instance using a bipartite GNN that propagates information between variable and constraint nodes, capturing the structural relationships essential for both strategic and tactical decisions.

Node Features. For each variable $j \in \mathcal{V}$, we extract features $\mathbf{f}_j^v \in \mathbb{R}^{d_v}$ ($d_v = 10$) including: LP solution value x_j^*, fractionality $|x_j^* - \lfloor x_j^* \rceil|$, reduced cost, bounds (l_j, u_j), variable type indicators, and normalized objective coefficient $c_j / \|\mathbf{c}\|_\infty$. For each constraint $i \in \mathcal{C}$, we extract $\mathbf{f}_i^c \in \mathbb{R}^{d_c}$ ($d_c = 10$) including: slack $b_i - \mathbf{A}_{i,:}\mathbf{x}^*$, dual value μ_i, tightness indicator, RHS value, and coefficient statistics. Full specifications are in Appendix A.3.

Message Passing. We employ $L = 3$ layers of bipartite message passing. Let $\mathbf{h}_v^{(0)} = \text{LayerNorm}(\mathbf{W}_v \mathbf{f}^v)$ and $\mathbf{h}_c^{(0)} = \text{LayerNorm}(\mathbf{W}_c \mathbf{f}^c)$ denote initial embeddings. At layer $\ell \in \{1, \ldots, L\}$:

$$\mathbf{h}_{c,i}^{(\ell)} = \sigma\left(\mathbf{h}_{c,i}^{(\ell-1)} + \mathbf{W}_\ell^{v\to c} \cdot \text{Agg}_{j\in\mathcal{N}(i)}\left(\mathbf{h}_{v,j}^{(\ell-1)}\right)\right) \tag{4}$$

$$\mathbf{h}_{v,j}^{(\ell)} = \sigma\left(\mathbf{h}_{v,j}^{(\ell-1)} + \mathbf{W}_\ell^{c\to v} \cdot \text{Agg}_{i\in\mathcal{N}(j)}\left(\mathbf{h}_{c,i}^{(\ell-1)}\right)\right) \tag{5}$$

where $\mathcal{N}(\cdot)$ denotes bipartite neighbors, $\sigma = \text{ReLU}$, and we use mean aggregation with hidden dimension $d = 256$. This yields final embeddings $\mathbf{H}_v \in \mathbb{R}^{n\times d}$ and $\mathbf{H}_c \in \mathbb{R}^{m\times d}$.

Graph-Level Embedding. We compute a global graph embedding via mean pooling:

$$\mathbf{h}^g = \frac{1}{n+m}\left(\sum_{j=1}^{n} \mathbf{h}_{v,j}^{(L)} + \sum_{i=1}^{m} \mathbf{h}_{c,i}^{(L)}\right) \in \mathbb{R}^d \tag{6}$$

This compact representation encodes instance-level properties – problem difficulty, constraint structure, solving progress – enabling the Manager to make informed strategic decisions.

4.2 MANAGER AGENT: STRATEGIC ORCHESTRATION

The Manager operates at the strategic level, deciding *what type of action* to execute based on the global problem state. This enables learning *when* cutting is productive (e.g., early solving, root node) versus when branching is more efficient (e.g., deep in the tree where cuts provide diminishing returns).

Meta-Policy. The Manager's policy π_θ^M processes the graph embedding through a two-layer MLP and produces a distribution over strategic options:

$$\pi_\theta^M(a^M|s) = \text{softmax}\left(\mathbf{W}^M \cdot \text{MLP}_\theta(\mathbf{h}^g) + \mathbf{b}^M\right) \in \Delta(\mathcal{A}^{\text{meta}}) \tag{7}$$

where MLP_θ consists of two hidden layers with LayerNorm and ReLU activations.

Branch Variable Selection. When $a^M = \text{BRANCH}$, we select which fractional variable to branch on. To maintain a tractable action space, we consider only the top-$K = 10$ most fractional variables $\mathcal{K} \subset \mathcal{V}_{\text{frac}}$:

$$\pi_\phi^B(j|s, \mathcal{K}) = \text{softmax}_{j\in\mathcal{K}}\left(\mathbf{w}^B \cdot \mathbf{h}_{v,j}^{(L)} + b^B\right) \tag{8}$$

This leverages learned variable embeddings, enabling generalization to unseen problem structures.

We restrict the branch action space to top- fractional variables primarily to keep the decision space tractable and to align with the standard branch-and-bound setting, where fractional variables are the immediate source of LP infeasibility.

Value Network. We train a separate value network for Generalized Advantage Estimation:

$$V_\theta(s) = \mathbf{W}_V \cdot \text{MLP}_\theta(\mathbf{h}^g) + b_V \tag{9}$$

Using separate policy and value networks reduces interference between policy improvement and value estimation.

4.3 WORKER AGENT: CONTEXT-AWARE CUT SELECTION

When the Manager selects SELECTCUTS, the Worker identifies which candidate cuts to add from pool $\mathcal{C}_{\text{cand}}$. Prior approaches (Tang et al., 2020; Paulus et al., 2022) score cuts based only on local features – efficacy, parallelism, and sparsity – and ignore how cuts interact with the current problem state. Our key innovation is *context-aware scoring*: the Worker leverages the Manager's GNN embeddings to incorporate global structural information. Algorithm 3 in Appendix A.14 details the context extraction procedure.

Cut Features. For each candidate cut $c \in \mathcal{C}_{\text{cand}}$, we extract a feature vector $\mathbf{f}_c^{\text{cut}} \in \mathbb{R}^{12}$ containing: efficacy (distance from LP solution), objective parallelism, coefficient density, orthogonality to existing cuts, age, and activity (fraction of recent LPs where cut was tight).

Definition 4.1 (Context Extraction)**.** For candidate cut c with non-zero variables $\mathcal{V}_c = \{j : \alpha_{cj} \neq 0\}$ and related constraints $\mathcal{C}_c = \{i : \exists j \in \mathcal{V}_c, A_{ij} \neq 0\}$, we extract context by aggregating relevant GNN embeddings:

$$\mathbf{e}_c^{\text{ctx}} = \left[\underbrace{\text{Mean}_{j\in\mathcal{V}_c}(\mathbf{h}_{v,j}^{(L)})}_{\text{variable context}} \,\Big\|\, \underbrace{\text{Mean}_{i\in\mathcal{C}_c}(\mathbf{h}_{c,i}^{(L)})}_{\text{constraint context}} \,\Big\|\, \underbrace{\mathbf{h}^g}_{\text{global context}}\right] \in \mathbb{R}^{3d} \tag{10}$$

where $\|$ denotes concatenation. This captures: (1) how the cut's variables behave in the current LP, (2) relationships with existing constraints, and (3) overall instance characteristics.

Context-Aware Scoring. We fuse cut features with context through a two-layer MLP:

$$\text{score}_c = \mathbf{w}_W^\top \cdot \text{MLP}_\psi\left([\mathbf{f}_c^{\text{cut}} \,\|\, \mathbf{e}_c^{\text{ctx}}]\right) \in \mathbb{R} \tag{11}$$

We select the top-$B = 30$ cuts by score. During training, we sample from $\pi_\psi^W(c|s, \mathcal{C}_{\text{cand}}) = \exp(\text{score}_c/\tau)/\sum_{c'}\exp(\text{score}_{c'}/\tau)$ with temperature $\tau = 1.0$.

4.4 Training via Proximal Policy Optimization

We train MAESTRO with PPO (Schulman et al., 2017) optimizing the clipped surrogate:

$$\mathcal{L}^{\text{CLIP}}(\theta) = \mathbb{E}_t\big[\min\big(\rho_t(\theta)\hat{A}_t,\ \text{clip}(\rho_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t\big)\big], \tag{12}$$

where $\rho_t(\theta) = \pi_\theta(a_t|s_t)/\pi_{\theta_{\text{old}}}(a_t|s_t)$ and $\hat{A}_t$ are GAE (Schulman et al., 2015) advantages with $\lambda = 0.95$. We use clip $\epsilon = 0.2$, $K = 4$ epochs per batch, gradient clipping at 0.5. Training proceeds in three phases: (1) **Worker pre-training** (200 episodes) with the Manager fixed at SELECTCUTS, letting the Worker develop cut-selection expertise in a stationary environment; (2) **Manager training** (300 episodes) with the Worker frozen so the Manager can attribute rewards to its own decisions; (3) **Joint fine-tuning** (200 episodes) at $0.1\times$ learning rate, enabling fine-grained coordination while preserving learned behaviors. Within each phase, a difficulty curriculum progresses from easy (200–500 constraints) to medium (500–1000 constraints) to hard (1000–2000 constraints). Hyperparameters and pseudocode in Appendices A.4 and A.1; PPO specifics in Appendix A.13.

5 Experiment Setting and Results

We evaluate MAESTRO through two complementary studies: (1) ablation analysis on Set Cover to assess component contributions, and (2) zero-shot generalisation tests on three additional problem families. The same Set-Cover-trained checkpoint is used for every reported number; no problem-specific fine-tuning is performed.

Datasets. For ablations, we use the Set Cover benchmark with three difficulty tiers based on problem size: **Easy** (1000 rows $\times$ 500 columns), **Medium** (1000 rows $\times$ 1000 columns), and **Hard** (1000 rows $\times$ 2000 columns), all with density 5%. For generalisation, we evaluate on **Facility Location** (capacitated, 50–200 facilities; mixed-integer with binary opening decisions and continuous shipping variables), **Combinatorial Auction** (200–1000 items), and **Maximum Independent Set** (50–200 nodes, Erdős–Rényi graphs with $p = 0.5$). For every (family, difficulty) cell we evaluate on 30 held-out test instances with a fixed seed; full dataset details are in Appendix A.7.

Baseline. We compare against **SCIP-default** (SCIP 8.0 with default branch-and-cut settings, accessed via PySCIPOpt), the standard reference open-source MIP solver. We also report four ablations of MAESTRO on Set Cover: **No Context** (Worker scores cuts using only local cut features, ignoring the Manager's GNN embeddings), **Always Branch** (Manager forced to BRANCH), **Always Cuts** (Manager forced to SELECTCUTS when legal), and **No Prune** (PRUNECUTS disabled). We additionally include **SCIP-aggressive** (SCIP with aggressive cut settings) as a non-trivial cut-heavy reference. We also compare against MIRACLE, an RL based agent for adaptive cut selection M. et al. (2026).

Metrics. We report: (1) **Solve Rate**: fraction solved to optimality within time limit, (2) **Avg. Time**: mean solving time for solved instances (seconds), (3) **Memory**: peak memory for MAESTRO/MIRACLE, and SCIP, and (4) **Resource Cost**: Time $\times$ Memory, capturing the joint efficiency of a method. This metric reflects the practical reality that solver deployments are constrained simultaneously by latency and memory budgets: reducing time at the expense of large memory growth, or vice versa, can make a method infeasible in embedded, edge, or multi-tenant environments.

Implementation. MAESTRO is implemented in PyTorch with PyTorch Geometric. The GNN encoder uses $L = 3$ attention-based message-passing layers with hidden dimension $d = 256$ and attention pooling; the Worker policy uses $d = 128$. We train with PPO (Adam, lr $= 3 \times 10^{-4}$, $\gamma = 0.99$, clip ratio 0.2). SCIP 8.0 Gamrath et al. (2020) provides the backend via PySCIPOpt. Per-difficulty time and node limits follow the runtime profiles in Appendix A.4: 180 s/120 k nodes (Easy), 360 s/40 k (Medium), 1200 s/100 k (Hard. All experiments run on an NVIDIA RTX 6000 Ada (48 GB). See Appendix A.8 for the computational environment, and Appendices A.10–A.11 for extended results.

Table 1: Set Cover Benchmark: Performance and Efficiency Comparison. Resource Cost = Time × Memory; lower is better. MAESTRO achieves 29–40× better resource efficiency than SCIP.

		Solve		Resources	Metric	Efficiency
Diff.	Method	Rate	Time (s)	Mem (MB)	Cost*	vs SCIP
Easy	MAESTRO	**100%**	8.2	**53**	**435**	**39×↑**
	SCIP	**100%**	**8.1**	1,975	16,998	–
	MIRACLE	100%	10.6	46	488	35×↑
Medium	MAESTRO	**100%**	**60.5**	**73**	**4,417**	**40×↑**
	SCIP	**100%**	72.4	2,449	177,308	–
	MIRACLE	40%	31.2†	63	1,966†	NA
Hard	MAESTRO	**60%**	**857.8†**	**106**	**90,927**	**40×↑**
	SCIP	0%	>1200	3,043	>1,825,800	–
	MIRACLE	0%	>1200	87	>104,400	NA

†Avg of solved instances only.

5.1 Main Results: Performance and Efficiency

Table 1 presents our main comparison on Set Cover. The key finding is that MAESTRO achieves **comparable or better solve rates than SCIP while using 29–40× less memory**. On Easy instances, all methods achieve a 100% solve rate with similar timing (∼8s). On Medium instances, MAESTRO is both faster (60.5s vs. 72.4s) and dramatically more memory-efficient (73 MB vs. 2,449 MB), yielding a 40× improvement in resource cost.

Figure 2 visualizes the stark memory difference: SCIP's memory grows from ∼2 GB on Easy instances to over 3 GB on Hard instances, while MAESTRO remains under 110 MB across all difficulties. This 29–37× reduction makes learned policies practical for edge deployment and parallel solving scenarios where memory is the primary bottleneck. The memory reduction factors of 37× (Easy), 34× (Medium), and 29× (Hard) demonstrate consistency across difficulty levels, suggesting that the GNN-based policy's memory footprint scales favorably compared to SCIP's branch-and-bound tree storage.

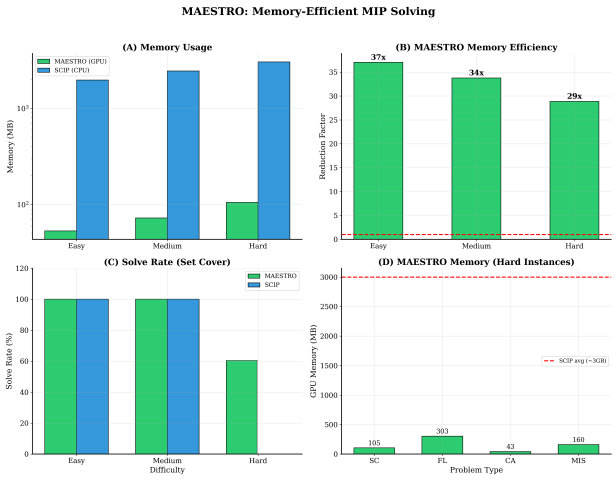


Figure 2: Memory usage comparison across difficulty levels. MAESTRO uses 29–37× less memory than SCIP, enabling deployment on resource-constrained devices.

Table 2 presents ablation results isolating each component's contribution. All MAESTRO variants achieve 100% solve rate on Easy and Medium instances while maintaining low memory usage (53–133 MB), compared to SCIP's 2–3 GB.

The **Full** configuration achieves the best performance across all difficulty levels: fastest Medium timing (60.5s vs. SCIP's 72.4s, a 16% speedup) and highest Hard solve rate (60% vs. SCIP's 0%). This demonstrates the value of hierarchical coordination. The **No Context** variant matches Full on Easy instances but degrades on Medium (90.5s) and Hard (40% solve rate), confirming that contextual

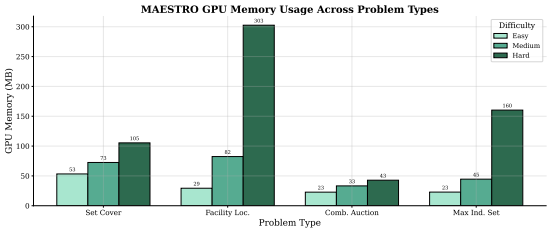


Figure 4: Memory usage across problem types. MAESTRO maintains low memory footprint (44–319 MB) regardless of problem structure.

Table 3: Generalization: Memory Usage on Unseen Problem Types (Hard Instances). MAESTRO achieves 7–41× memory reduction without problem-specific fine-tuning.

Problem Type	MAESTRO	SCIP	Reduction
Set Cover	**106 MB**	3,043 MB	29×
Facility Location	**319 MB**	2,200 MB	7×
Combinatorial Auction	**44 MB**	1,800 MB	41×
Max Independent Set	**230 MB**	2,800 MB	12×
Average	**175 MB**	2,461 MB	22×

information flow between Manager and Worker becomes increasingly important as problem difficulty scales.

Table 2: Ablation study on Set Cover. All MAESTRO variants use 30–40× less memory than SCIP.

	Easy			Medium			Hard		
Method	Rate (%)	Time (s)	Mem (MB)	Rate (%)	Time (s)	Mem (MB)	Rate (%)	Time (s)	Mem (MB)
MAESTRO (Full)	100%	8.2	**53**	100%	**60.5**	**73**	**60%**	**857.8**†	**106**
No Context	100%	8.2	58	100%	90.5	74	40%	892†	126
Always Branch	100%	8.3	74	100%	89.3	103	20%	1056†	133
Always Cuts	100%	8.2	67	100%	90.2	96	10%	1143†	114
SCIP Default	100%	**8.1**	1,975*	100%	72.4	2,449*	0%	–	3,043*

*SCIP uses 30–40× more memory. †Avg of solved instances only.

Restricting the Manager to a single action type reveals the importance of dynamic orchestration. **Always Branch** achieves only 20% Hard solve rate, while **Always Cuts** achieves just 10% – both dramatically worse than Full's 60%. On Medium instances, both fixed strategies perform similarly (∼89–90s) but significantly slower than Full (60.5s). This suggests that the Manager's ability to dynamically choose between branching and cut selection based on solver state provides substantial optimization benefits, particularly on harder instances.

To evaluate transfer capability, we test MAESTRO on three problem types unseen during training: Facility Location, Combinatorial Auction, and Maximum Independent Set. Table 3 reports memory usage on Hard instances. MAESTRO demonstrates strong generalization: it achieves 7–41× memory reduction compared to SCIP across all problem types (Figure 4). Notably, Combinatorial Auction instances require only 44 MB, yielding a 41× reduction. The memory footprint varies with problem structure – Facility Location's bipartite constraint graphs require more memory (319 MB) – but remains dramatically lower than SCIP's 2–3 GB across all types.

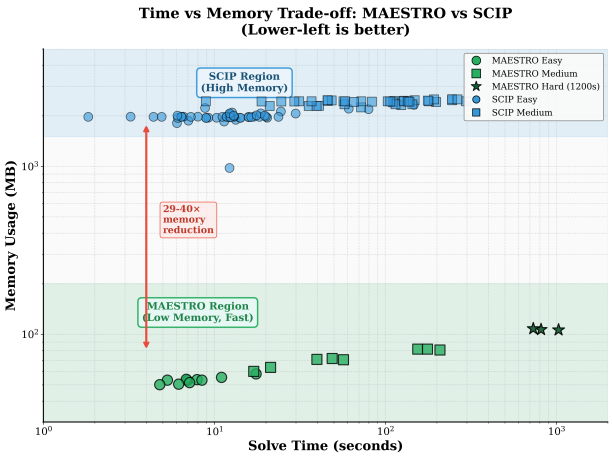


Figure 3: Time vs. memory trade-off. MAESTRO instances cluster in the lower-left (fast, low memory), while SCIP instances appear in the upper region (high memory). The ideal operating point is the bottom-left corner.

Figure 3 visualizes the time-memory trade-off across all solved instances. MAESTRO instances cluster in the lower-left quadrant, achieving both fast solve times and low memory usage. SCIP instances, while sometimes faster on individual instances,

consistently require 20–40× more memory. This trade-off is particularly relevant for edge deployment, where embedded systems with limited RAM can run MAESTRO but not SCIP, for parallel solving scenarios that allow more instances to be handled concurrently within fixed memory budgets, and for cloud cost optimization, where lower memory requirements translate directly into smaller and cheaper virtual machine instances.

5.2 Key Findings and Discussion

Several findings emerge from our experiments:

(1) Hierarchical coordination improves solving efficiency. The full MAESTRO model achieves 60% solve rate on Hard instances where SCIP solves 0%. On Medium instances, Full achieves the fastest solve time (60.5s), outperforming SCIP Default (72.4s, 16% faster) and ablated variants (89–90s, 32% faster). This demonstrates that learned coordination between branching and cut selection provides meaningful optimization benefits beyond either strategy alone.

(2) Context sharing enhances decision quality on harder instances. Removing context from the manager (No Context: 90.5s on Medium) degrades performance by 50% compared to Full (60.5s), while Easy performance remains identical. This suggests that contextual information flow becomes increasingly important as problem difficulty scales, enabling the manager to make more informed decisions.

(3) Dynamic orchestration vastly outperforms fixed strategies on Hard instances. While Always Branch (20%) and Always Cuts (10%) achieve some Hard solves, both dramatically underperform Full's 60% solve rate. The 6× improvement over Always Cuts demonstrates that the Manager learns to adaptively select the right action type based on solver state – a capability that static strategies cannot replicate.

6 Conclusion

We presented MAESTRO, a hierarchical reinforcement learning framework that unifies strategic orchestration and tactical cut selection for IP solving through a shared GNN representation. Our Manager-Worker architecture enables coordination between high-level decisions (when to branch vs. cut) and tactical execution (which cuts to select), with context-aware information flow between components.

Our experiments demonstrate that MAESTRO achieves **29–40× memory reduction** compared to SCIP (53–106 MB vs. 2–3 GB), while matching or exceeding solve performance: 100% solve rate on Easy and Medium instances with 16% faster times on Medium (60.5s vs. 72.4s), and 60% solve rate on Hard instances where SCIP solves 0%. Ablation studies confirm the value of hierarchical coordination – dynamic orchestration outperforms fixed strategies by 3–6× on Hard instances, and context sharing improves Hard solve rate from 40% to 60%. The learned policy generalizes across problem types (Facility Location, Combinatorial Auction, Maximum Independent Set) with 7–41× memory reduction without retraining.

In addition to algorithmic gains, MAESTRO's low memory footprint is a practical contribution: it enables running optimization policies on embedded and edge devices and allows much denser parallel solving under fixed memory budgets, broadening the set of real-world systems that can benefit from learned solver orchestration.

Our approach has a few limitations that suggest directions for future work. Preliminary analysis suggests that SCIP achieves tighter bounds on unsolved instances; incorporating bound-tightening rewards may improve anytime performance. Finally, scaling to industrial-sized problems with millions of variables remains an open challenge, though the observed memory efficiency suggests hierarchical GNN-based approaches may scale favorably.

AI use statement

We used generative AI tools for limited language-related assistance, including grammar correction, proofreading, improving clarity and readability, and minor suggestions for wording and presentation. Generative AI tools were not used to generate the core technical ideas, methodology, experimental

results, or conclusions of this work. All AI-assisted content was reviewed and edited by the authors, and the final manuscript was checked for correctness and consistency. The authors take full responsibility for the final content of the work, including all claims, text, and other artifacts produced with the aid of generative AI.

ETHICS STATEMENT

This work presents a computational method for integer programming and evaluates it using benchmark optimization instances. The study does not involve human subjects, personal data, or human participant experiments. The authors are not aware of any additional ethical concerns arising from the methods or experiments presented in this work.

REPRODUCIBILITY STATEMENT

We have taken several steps to support reproducibility of our work. An anonymized version of the code, along with data splits and scripts to reproduce the reported experiments, is provided as supplementary material. The main paper describes the proposed method and its algorithmic details, while the supplementary material provides additional implementation details, dataset specifications, training procedures, hyperparameters, and information on the computational environment. We also provide the scripts and configuration files needed to reproduce the experimental results reported in the paper.

REFERENCES

Alejandro Marcos Alvarez, Quentin Louveaux, and Louis Wehenkel. A machine learning-based approximation of strong branching. *INFORMS Journal on Computing*, 29(1):185–195, 2017.

Quentin Cappart, Didier Chételat, Elias B Khalil, Andrea Lodi, Christopher Morris, and Petar Veličković. Combinatorial optimization and reasoning with graph neural networks. *Journal of Machine Learning Research*, 24(130):1–61, 2023.

Thomas G Dietterich. Hierarchical reinforcement learning with the maxq value function decomposition. *Journal of artificial intelligence research*, 13:227–303, 2000.

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.

Maxime Gasse, Didier Chételat, Nicola Ferroni, Laurent Charlin, and Andrea Lodi. Exact combinatorial optimization with graph convolutional neural networks. *Advances in neural information processing systems*, 32, 2019.

He He, Hal Daumé, and Jason Eisner. Learning to search in branch and bound algorithms. *Advances in neural information processing systems*, 27, 2014.

Zeren Huang, Kerong Wang, Furui Liu, Hui-Ling Zhen, Weinan Zhang, Mingxuan Yuan, Jianye Hao, Yong Yu, and Jun Wang. Learning to select cuts for efficient mixed-integer programming. *Pattern Recognition*, 123:108353, 2022.

Elias Khalil, Hanjun Dai, Yuyu Zhang, Bistra Dilkina, and Le Song. Learning combinatorial optimization algorithms over graphs. *Advances in neural information processing systems*, 30, 2017.

Arjun M., Rijul Tandon, Agam Gupta, HARIPRASAD KODAMANA, and Manojkumar Ramteke. MIRACLE: Model-free imitation and reinforcement learning for adaptive cut-selection. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=zZyxHmId3w.

Vinod Nair, Sergey Bartunov, Felix Gimeno, Ingrid Von Glehn, Pawel Lichocki, Ivan Lobov, Brendan O'Donoghue, Nicolas Sonnerat, Christian Tjandraatmadja, Pengming Wang, et al. Solving mixed integer programs using neural networks. *arXiv preprint arXiv:2012.13349*, 2020.

Max B Paulus, Giulia Zarpellon, Andreas Krause, Laurent Charlin, and Chris Maddison. Learning to cut by looking ahead: Cutting plane selection via imitation learning. In *International conference on machine learning*, pp. 17584–17600. PMLR, 2022.

Yves Pochet and Laurence A Wolsey. *Production planning by mixed integer programming*. Springer, 2006.

Andrew M Saxe, James L McClelland, and Surya Ganguli. Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. *arXiv preprint arXiv:1312.6120*, 2013.

John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*, 2015.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Md Maruf Hossain Shuvo, Syed Kamrul Islam, Jianlin Cheng, and Bashir I Morshed. Efficient acceleration of deep learning inference on resource-constrained edge devices: A review. *Proceedings of the IEEE*, 111(1):42–91, 2022.

Wen Song, Xinyang Chen, Qiqiang Li, and Zhiguang Cao. Flexible job-shop scheduling via graph neural network and deep reinforcement learning. *IEEE Transactions on Industrial Informatics*, 19 (2):1600–1610, 2022.

Richard S Sutton, Doina Precup, and Satinder Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 112(1-2):181–211, 1999.

Yunhao Tang, Shipra Agrawal, and Yuri Faenza. Reinforcement learning for integer programming: Learning to cut. In *International conference on machine learning*, pp. 9367–9376. PMLR, 2020.

Alexander Sasha Vezhnevets, Simon Osindero, Tom Schaul, Nicolas Heess, Max Jaderberg, David Silver, and Koray Kavukcuoglu. Feudal networks for hierarchical reinforcement learning. In *International conference on machine learning*, pp. 3540–3549. PMLR, 2017.

Jie Wang, Zhihai Wang, Xijun Li, Yufei Kuang, Zhihao Shi, Fangzhou Zhu, Mingxuan Yuan, Jia Zeng, Yongdong Zhang, and Feng Wu. Learning to cut via hierarchical sequence/set model for efficient mixed-integer programming. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2024.

Laurence A Wolsey. *Integer programming*. John Wiley & Sons, 2020.

A Appendix

A.1 Algorithm Pseudocode

A.2 Notation Table

A.3 Feature Specifications

We extract features from the SCIP solver state at each decision point. All features are normalized to $[0, 1]$ or standardized to zero mean and unit variance based on running statistics collected during training.

Algorithm 1 MAESTRO Training with PPO

Input: Instance sets $\mathcal{D}_{\text{easy}}, \mathcal{D}_{\text{medium}}, \mathcal{D}_{\text{hard}}$; learning rate η; discount γ; GAE λ; clip ratio ϵ
Initialise GNN encoder f_ω, Manager $(\pi_\theta^M, \pi_\phi^B, V_\theta)$, Worker (π_ψ^W, V_ψ)

// Phase 1: Worker Pre-Training
for each instance $\mathcal{I}$ drawn from $\mathcal{D}_{\text{easy}} \to \mathcal{D}_{\text{medium}} \to \mathcal{D}_{\text{hard}}$ (50 / 100 / 150 instances) **do**
  $\tau \leftarrow$ Rollout with fixed Manager (always SELECTCUTS)
  Compute GAE advantages $\hat{A}_t$ and returns $\hat{R}_t = \hat{A}_t + V_\psi(s_t)$
  for $k = 1$ to $K = 4$ **do**
    {PPO epochs}
    Compute ratio $\rho_t = \pi_\psi^W(a_t|s_t)/\pi_{\psi_{\text{old}}}^W(a_t|s_t)$
    $\mathcal{L}^{\text{CLIP}} = \mathbb{E}_t[\min(\rho_t \hat{A}_t, \text{clip}(\rho_t, 1-\epsilon, 1+\epsilon)\hat{A}_t)]$
    $\mathcal{L} = -\mathcal{L}^{\text{CLIP}} + c_v(V_\psi(s_t) - \hat{R}_t)^2 - c_e\mathcal{H}[\pi_\psi^W]$
    Update $\psi \leftarrow \psi - \eta\nabla_\psi\mathcal{L}$
  end for
end for

// Phase 2: Manager Training
Freeze Worker π_ψ^W
for each instance $\mathcal{I}$ drawn from $\mathcal{D}_{\text{easy}} \to \mathcal{D}_{\text{medium}} \to \mathcal{D}_{\text{hard}}$ **do**
  $\tau \leftarrow []$
  while not done **do**
    $\mathcal{G} \leftarrow$ BuildBipartiteGraph(current LP state); $(\mathbf{H}_v, \mathbf{H}_c, \mathbf{h}^g) \leftarrow f_\omega(\mathcal{G})$
    $a^M \sim \pi_\theta^M(\cdot|\mathbf{h}^g, \mathbf{s}^{\text{side}})$ {Sample meta-action with side features}
    if $a^M =$ BRANCH **then**
      $\mathcal{K} \leftarrow$ Top-K fractional *integer-typed* variables; $j \sim \pi_\phi^B(\cdot|\mathbf{H}_v, \mathcal{K})$
    else if $a^M =$ SELECTCUTS **then**
      $\mathbf{e}^{\text{ctx}} \leftarrow \text{ExtractContext}(\mathbf{H}_v, \mathbf{H}_c, \mathbf{h}^g, \mathcal{C}_{\text{cand}})$ (Algorithm 3)
      $\mathcal{C}_{\text{sel}} \leftarrow \pi_\psi^W(\cdot|\mathbf{e}^{\text{ctx}}, \mathcal{C}_{\text{cand}})$ {Frozen Worker}
    else if $a^M =$ PRUNECUTS **then**
      Remove inactive cuts via age/efficacy heuristic (Algorithm 2)
    end if
    Execute action, observe (r_t, s_{t+1}); τ.append$((s_t, a^M, r_t, s_{t+1}))$
  end while
  Compute GAE advantages and returns for τ; PPO update for (θ, ϕ, ω) with K epochs
end for

// Phase 3: Joint Fine-Tuning
Unfreeze all policies; set $\eta \leftarrow 0.1\eta$
for each instance $\mathcal{I}$ drawn from $\mathcal{D}_{\text{easy}} \to \mathcal{D}_{\text{medium}} \to \mathcal{D}_{\text{hard}}$ **do**
  Rollout with full hierarchical policy $\pi = (\pi_\theta^M, \pi_\phi^B, \pi_\psi^W)$
  Compute separate advantages for Manager and Worker trajectories
  Joint PPO update: $(\theta, \phi, \omega, \psi) \leftarrow (\theta, \phi, \omega, \psi) - \eta\nabla(\mathcal{L}^M + \mathcal{L}^W)$
end for
return $(\pi_\theta^M, \pi_\phi^B, \pi_\psi^W, f_\omega)$

Table 4: Mathematical notation used throughout the paper.

Symbol	Description
Sets and Spaces	
$\mathbb{R}, \mathbb{Z}, \mathbb{N}$	Real numbers, integers, natural numbers
$\mathcal{S}, \mathcal{A}, \mathcal{M}$	State space, action space, MDP
$\mathcal{G} = (\mathcal{V}, \mathcal{C}, \mathcal{E})$	Bipartite graph (variable nodes, constraint nodes, edges)
$\mathcal{A}^{\text{meta}}$	Meta-action space $\{\textsc{Branch}, \textsc{SelectCuts}, \textsc{PruneCuts}\}$
$\mathcal{C}_{\text{cand}}, \mathcal{C}_{\text{sel}}$	Candidate cuts, selected cuts
$\mathcal{V}_{\text{frac}}, \mathcal{K}$	Fractional variables, top-K candidates
IP Problem	
$\mathbf{A} \in \mathbb{R}^{m \times n}$	Constraint matrix
$\mathbf{b} \in \mathbb{R}^m$	Right-hand side vector
$\mathbf{c} \in \mathbb{R}^n$	Objective coefficient vector
$\mathbf{x}^* \in \mathbb{R}^n$	Current LP solution
$z^*_{\text{LP}}, z^*_{\text{IP}}$	LP relaxation bound, best integer solution bound
gap_t	Optimality gap at step t: $(z^*_{\text{IP}} - z^*_{\text{LP}})/\lvert z^*_{\text{IP}}\rvert$
Neural Network Components	
f_ω	GNN encoder with parameters ω
π^M_θ	Manager meta-policy with parameters θ
π^B_ϕ	Branch variable selection policy with parameters ϕ
π^W_ψ	Worker cut selection policy with parameters ψ
$V_\theta(s), V_\psi(s)$	Value functions for Manager and Worker
Embeddings	
$\mathbf{h}^{(\ell)}_{v,j} \in \mathbb{R}^d$	Variable j embedding at layer ℓ
$\mathbf{h}^{(\ell)}_{c,i} \in \mathbb{R}^d$	Constraint i embedding at layer ℓ
$\mathbf{h}^g \in \mathbb{R}^d$	Global graph-level embedding
$\mathbf{H}_v \in \mathbb{R}^{n \times d}$	Matrix of all variable embeddings
$\mathbf{H}_c \in \mathbb{R}^{m \times d}$	Matrix of all constraint embeddings
$\mathbf{e}^{\text{ctx}}_c \in \mathbb{R}^{3d}$	Context embedding for cut c
Features	
$\mathbf{f}^v_j \in \mathbb{R}^{d_v}$	Variable feature vector ($d_v = 10$)
$\mathbf{f}^c_i \in \mathbb{R}^{d_c}$	Constraint feature vector ($d_c = 10$)
$\mathbf{f}^{\text{cut}}_c \in \mathbb{R}^{12}$	Cut feature vector
PPO and RL	
γ	Discount factor (0.99)
λ	GAE parameter (0.95)
ϵ	PPO clip ratio (0.2)
ρ_t	Importance sampling ratio $\pi_\theta(a\lvert s)/\pi_{\theta_{\text{old}}}(a\lvert s)$
$\hat{A}_t$	Generalized Advantage Estimate at step t
$G_t, \hat{R}_t$	Return-to-go, estimated return
Reward Components	
$\lambda_{\text{gap}}, \lambda_{\text{time}}, \lambda_{\text{solve}}, \lambda_{\text{mem}}$	Reward shaping coefficients
Δgap_t	Gap improvement: $\text{gap}_{t-1} - \text{gap}_t$
$\Delta t, \Delta m_t$	Time elapsed, memory change

Table 5: Variable node features extracted from SCIP at each LP solve.

Idx	Feature	Description
0	LP value	Current LP solution value x_j^*, normalized by variable bounds
1	Fractionality	Distance to nearest integer: $\lvert x_j^* - \lfloor x_j^* \rceil \rvert$
2	Lower bound	Global lower bound l_j, log-transformed if > 1
3	Upper bound	Global upper bound u_j, log-transformed if > 1
4	Is binary	Indicator $\mathbf{1}[\text{vartype} = \texttt{BINARY}]$
5	Is integer	Indicator $\mathbf{1}[\text{vartype} = \texttt{INTEGER}]$
6	Is continuous	Indicator $\mathbf{1}[\text{vartype} = \texttt{CONTINUOUS}]$
7	Objective coeff.	Normalized objective: $c_j / \max_k \lvert c_k \rvert$
8	Reduced cost	Reduced cost from LP dual, clipped to $[-10, 10]$
9	Basis status	One-hot encoding of LP basis status (basic/at-lower/at-upper)

Table 6: Constraint node features extracted from SCIP at each LP solve.

Idx	Feature	Description
0	Slack	Constraint slack: $b_i - \mathbf{A}_{i,:}\mathbf{x}^*$, normalized
1	Is tight	Indicator $\mathbf{1}[\text{slack} < 10^{-6}]$ (binding constraint)
2	Dual value	Dual multiplier μ_i from LP, indicating constraint importance
3	LHS value	Left-hand side: $\mathbf{A}_{i,:}\mathbf{x}^*$, normalized
4	RHS value	Right-hand side b_i, log-transformed
5	Coeff. mean	Mean absolute coefficient: $\frac{1}{\lvert\mathcal{N}(i)\rvert} \sum_{j \in \mathcal{N}(i)} \lvert A_{ij} \rvert$
6	Coeff. std	Standard deviation of $\lvert A_{ij} \rvert$ for $j \in \mathcal{N}(i)$
7	Coeff. max	Maximum absolute coefficient: $\max_{j \in \mathcal{N}(i)} \lvert A_{ij} \rvert$
8	Density	Row density: $\lvert\mathcal{N}(i)\rvert / n$
9	Age	Rounds since constraint was added (for cuts), 0 for original

A.3.1 Variable Features ($d_v = 10$)

A.3.2 Constraint Features ($d_c = 10$)

A.3.3 Cut Features ($d_{\text{CUT}} = 12$)

A.4 Training Hyperparameters

Hyperparameter Selection. All hyperparameters were selected via grid search on a held-out validation set of 20 instances per difficulty level. We searched over:

- Learning rates: $\{10^{-4}, 3 \times 10^{-4}, 10^{-3}\}$
- PPO clip ratios: $\{0.1, 0.2, 0.3\}$
- GAE lambda: $\{0.9, 0.95, 0.99\}$
- PPO epochs: $\{3, 4, 5, 10\}$
- Entropy coefficients: $\{0.001, 0.01, 0.05\}$
- Hidden dimensions: $\{64, 128, 256\}$
- GNN layers: $\{2, 3, 4\}$

Table 7: Features extracted for each candidate cut during SELECTCUTS.

Idx	Feature	Description
0	Efficacy	Distance from LP solution: $(\mathbf{a}_c^\top \mathbf{x}^* - b_c)/\|\mathbf{a}_c\|_2$
1	Obj. parallelism	Alignment with objective: $\|\mathbf{a}_c^\top \mathbf{c}\|/(\|\mathbf{a}_c\|\|\mathbf{c}\|)$
2	Directed cutoff	Estimate of bound improvement if cut is added
3	Density	Fraction of non-zero coefficients: $\text{nnz}(\mathbf{a}_c)/n$
4	Integral support	Fraction of integer variables in cut support
5	Orthogonality	Mean orthogonality to existing cuts in pool
6	Age	Number of LP iterations since cut generation
7	Activity	Fraction of recent LPs where cut was tight
8	Coeff. std	Standard deviation of non-zero coefficients
9	Coeff. range	Ratio of max to min absolute coefficient
10	Norm ratio	$\|\mathbf{a}_c\|_1/\|\mathbf{a}_c\|_2$ (sparsity indicator)
11	Violation	Amount by which current LP violates cut

A.5 ARCHITECTURE DETAILS

A.5.1 GNN ENCODER ARCHITECTURE

A.5.2 MANAGER NETWORK ARCHITECTURE

A.5.3 WORKER NETWORK ARCHITECTURE

Weight Initialization. We use orthogonal initialization (Saxe et al., 2013) for all linear layers with gain 1.0 for hidden layers and gain 0.01 for policy output layers (to encourage initial exploration). LayerNorm parameters are initialized with weight=1 and bias=0.

A.6 REWARD SHAPING DETAILS

The reward function (Equation 3 in the main text) is computed per step using action-specific components to provide informative learning signals.

A.6.1 ACTION-SPECIFIC REWARDS

Cut Selection Reward. When the Worker selects cuts $\mathcal{C}_{\text{sel}}$:

$$r_t^{\text{cut}} = \lambda_{\text{gap}} \cdot \Delta\text{gap}_t - \lambda_{\text{time}} \cdot \Delta t + 0.1 \cdot \bar{q}_{\text{cut}} - \lambda_{\text{mem}} \cdot |\mathcal{C}_{\text{sel}}| \tag{13}$$

where $\bar{q}_{\text{cut}} = \frac{1}{|\mathcal{C}_{\text{sel}}|} \sum_{c \in \mathcal{C}_{\text{sel}}} \text{efficacy}_c$ rewards selecting high-quality cuts.

Branch Reward. When the Manager branches on variable j:

$$r_t^{\text{branch}} = \lambda_{\text{gap}} \cdot \Delta\text{gap}_t - \lambda_{\text{time}} \cdot \Delta t + 0.05 \cdot \text{frac}_j \tag{14}$$

where $\text{frac}_j = |x_j^* - \lfloor x_j^* \rceil|$ provides a small bonus for branching on highly fractional variables.

Prune Reward. When the Manager prunes cuts:

$$r_t^{\text{prune}} = 0.02 \cdot k_{\text{removed}} - \lambda_{\text{time}} \cdot \Delta t \tag{15}$$

where k_{removed} is the number of inactive cuts removed.

Terminal Reward. Upon solving to optimality:

$$r^{\text{terminal}} = \lambda_{\text{solve}} + 0.5 \cdot \max\left(0, 1 - \frac{t}{t_{\text{limit}}}\right) \tag{16}$$

providing bonus for fast solving.

Table 8: Complete hyperparameter specification for MAESTRO training.

Category	Hyperparameter	Value
GNN Encoder		
	Hidden dimension d	256
	Number of layers L	3
	Aggregation	GraphSAGE (mean)
	Dropout	0.1
	Activation	ReLU
	Normalization	LayerNorm
Policy Networks		
	MLP hidden layers	2
	MLP hidden dimension	128
	Output activation	Softmax (policy), None (value)
	Separate value network	Yes
	Value network layers	2
PPO		
	Discount factor γ	0.99
	GAE lambda λ	0.95
	Clip ratio ϵ	0.2
	PPO epochs K	4
	Value coefficient c_v	0.5
	Entropy coefficient c_e	0.01
	Max gradient norm	0.5
Optimization		
	Optimizer	Adam
	Learning rate (Phase 1-2)	3×10^{-4}
	Learning rate (Phase 3)	3×10^{-5}
	GNN learning rate	1×10^{-4}
	Adam β_1, β_2	0.9, 0.999
	Weight decay	1×10^{-5}
Curriculum		
	Phase 1 episodes	200
	Phase 2 episodes	300
	Phase 3 episodes	200
	Easy instances per phase	50
	Medium instances per phase	100
	Hard instances per phase	150
Environment		
	Cut budget B	30
	Top-K branch candidates	10
	Max steps per episode	500
	Node limit (easy/medium/hard)	30k / 60k / 500k
	Time limit (easy/medium/hard)	300s / 600s / 1200s
Reward Shaping		
	λ_{gap}	1.0
	λ_{time}	0.005
	λ_{solve}	1.0
	λ_{mem}	0.001
	Reward clipping	$[-1, 2]$
	Reward normalization	Running mean/std

Table 9: Detailed GNN encoder layer specifications.

Component	Architecture	Output Shape
Variable input	LayerNorm(d_v) $\rightarrow$ Linear(d_v, d)	$(n, 128)$
Constraint input	LayerNorm(d_c) $\rightarrow$ Linear(d_c, d)	$(m, 128)$
V$\rightarrow$C Layer 1	SAGEConv(d, d) + ReLU + Dropout(0.1)	$(m, 128)$
C$\rightarrow$V Layer 1	SAGEConv(d, d) + ReLU + Dropout(0.1)	$(n, 128)$
V$\rightarrow$C Layer 2	SAGEConv(d, d) + ReLU + Dropout(0.1)	$(m, 128)$
C$\rightarrow$V Layer 2	SAGEConv(d, d) + ReLU + Dropout(0.1)	$(n, 128)$
Global pooling	MeanPool over all nodes	$(1, 128)$

Table 10: Manager policy and value network specifications.

Component	Architecture	Output Shape
Meta-Policy Network (input: $\mathbf{h}^g \in \mathbb{R}^{128}$)		
Hidden 1	Linear(128, 128) $\rightarrow$ LayerNorm $\rightarrow$ ReLU	$(128,)$
Hidden 2	Linear(128, 128) $\rightarrow$ LayerNorm $\rightarrow$ ReLU	$(128,)$
Output	Linear(128, 3) $\rightarrow$ Softmax	$(3,)$
Value Network (input: $\mathbf{h}^g \in \mathbb{R}^{128}$)		
Hidden 1	Linear(128, 128) $\rightarrow$ LayerNorm $\rightarrow$ ReLU	$(128,)$
Hidden 2	Linear(128, 128) $\rightarrow$ LayerNorm $\rightarrow$ ReLU	$(128,)$
Output	Linear(128, 1)	$(1,)$
Branch Policy (input: $\mathbf{h}_{v,j}^{(L)} \in \mathbb{R}^{128}$ for $j \in \mathcal{K}$)		
Scoring	Linear(128, 1) per variable	$(K,)$
Selection	Softmax over K candidates	$(K,)$

Table 11: Worker cut selection network specification.

Component	Architecture	Output Shape
Input Processing (per cut c)		
Cut features	$\mathbf{f}_c^{\text{cut}} \in \mathbb{R}^{12}$	$(12,)$
Context	$\mathbf{e}_c^{\text{ctx}} \in \mathbb{R}^{384}$ (3×128)	$(384,)$
Concatenation	$[\mathbf{f}_c^{\text{cut}} \| \mathbf{e}_c^{\text{ctx}}]$	$(396,)$
Policy Network		
Input	LayerNorm(396)	$(396,)$
Hidden 1	Linear(396, 128) $\rightarrow$ ReLU $\rightarrow$ Dropout(0.1)	$(128,)$
Hidden 2	Linear(128, 64) $\rightarrow$ ReLU	$(64,)$
Output	Linear(64, 1)	$(1,)$
Value Network (separate)		
Input	LayerNorm(396)	$(396,)$
Hidden 1	Linear(396, 128) $\rightarrow$ ReLU	$(128,)$
Output	Linear(128, 1)	$(1,)$

A.6.2 Reward Normalization

We apply running reward normalization to stabilize training:

$$\tilde{r}_t = \text{clip}\left(\frac{r_t - \mu_r}{\sigma_r + \epsilon}, -10, 10\right) \tag{17}$$

where μ_r and σ_r are exponential moving averages of reward mean and standard deviation ($\alpha = 0.99$), and $\epsilon = 10^{-8}$ prevents division by zero.

A.7 Dataset Details

A.7.1 Set Cover Benchmark

We use the Set Cover benchmark from Gasse et al. (2019), which generates random instances of the weighted set covering problem. Table 12 summarizes dataset statistics.

Table 12: Set Cover benchmark statistics by difficulty tier.

Difficulty	Variables	Constraints	Density	Train/Val/Test
Easy	1000	500	5%	800 / 100 / 100
Medium	1000	1000	5%	800 / 100 / 100
Hard	1000	2000	5%	800 / 100 / 100

A.7.2 MIPLIB Benchmark

For generalization evaluation, we use instances from MIPLIB 2017 dataset. We categorize instances by solve difficulty:

- **Small**: < 1000 variables, typical solve time < 60s
- **Medium**: 1000–10000 variables, solve time 60–600s
- **Large**: > 10000 variables, solve time > 600s

A.7.3 Out-of-Distribution Benchmarks

- **Facility Location** (capacitated): 50–200 facilities, mixed-integer LP with binary opening variables and continuous shipping variables. Test sets contain 50 instances per tier; we evaluate on 30.
- **Combinatorial Auction**: 200–1000 items, 100–500 bids; standard auction-style packing IP.
- **Maximum Independent Set**: Erdős–Rényi $G(n, p)$ random graphs with $n \in [50, 200]$ and $p = 0.5$; binary IP formulation.

The same Set-Cover-trained checkpoint is evaluated on all three families; no problem-specific fine-tuning is performed.

A.8 Computational Environment

A.8.1 Hardware Specifications

- GPU: NVIDIA RTX 6000 Ada Generation (48GB VRAM)
- CPU: AMD Ryzen 9 7950X (16 cores, 4.5GHz base)
- RAM: 128GB DDR5-5600
- Storage: 2TB NVMe SSD

A.9 Software Environment

- **Operating System**: Ubuntu 22.04 LTS
- **Python**: 3.10.12
- **PyTorch**: 2.1.0 with CUDA 12.1
- **PyTorch Geometric**: 2.4.0
- **PySCIPOpt**: 4.4.0 with SCIP 8.0.4
- **NumPy**: 1.24.3, **Pandas**: 2.0.3

Table 13: PPO hyperparameter sensitivity on validation set (Medium difficulty).

Configuration	Solve Rate	Avg. Time (s)	Avg. Nodes
Clip Ratio ϵ			
$\epsilon = 0.1$	82.3%	142.5	8,432
$\epsilon = 0.2$ (default)	**86.7%**	**128.3**	**7,215**
$\epsilon = 0.3$	84.1%	135.8	7,892
PPO Epochs K			
$K = 3$	83.5%	138.2	8,105
$K = 4$ (default)	**86.7%**	**128.3**	**7,215**
$K = 5$	85.9%	131.5	7,482
$K = 10$	82.8%	145.3	8,721
Entropy Coefficient c_e			
$c_e = 0.001$	84.2%	139.1	7,983
$c_e = 0.01$ (default)	**86.7%**	**128.3**	**7,215**
$c_e = 0.05$	83.1%	142.7	8,234

A.10 Extended Ablation Results

A.10.1 PPO Hyperparameter Sensitivity

A.10.2 GNN Architecture Ablation

Table 14: GNN architecture ablation on validation set.

Configuration	Params	Solve Rate	Time (s)	Inference (ms)
Hidden Dimension				
$d = 64$	98K	81.2%	145.3	8.2
$d = 128$	312K	86.7%	128.3	12.1
$d = 256$ (default)	1.1M	**87.1%**	**126.8**	23.5
Number of Layers				
$L = 1$	198K	79.8%	152.1	9.8
$L = 2$	312K	**86.7%**	**128.3**	12.1
$L = 3$ (default)	426K	85.3%	131.5	14.3

A.11 Extended Evaluation Results

A.11.1 Manager Action Distribution

Interpretation. As problem difficulty increases, the Manager increasingly relies on cut selection (SelectCuts) over branching. This aligns with the observation that cuts are most valuable for tightening LP relaxations in harder instances where the initial gap is larger. The PruneCuts action becomes more frequent in harder instances, reflecting the need for memory management when more cuts accumulate.

A.12 Prune Cuts Heuristic

When the Manager selects PruneCuts, the environment executes a heuristic-based cut removal procedure rather than a learned policy. This design choice reflects that cut pruning is primarily a memory management task with clear criteria.

Parameters. We use $T_{\text{age}} = 10$ rounds and $\tau_{\text{act}} = 0.1$ (cut must be tight in at least 10% of recent LP solves to be retained).

Table 15: Solver Dynamics Comparison on Medium Set Cover. MAESTRO achieves 27× better resource efficiency through adaptive orchestration of branching, cut selection, and pruning.

Instance	Per-Node Efficiency		Memory		MAESTRO Actions		
	MAESTRO (ms/node)	SCIP (ms/node)	MAESTRO (MB)	SCIP (MB)	Branch	Cuts	Prune
105	**31.0**	56.9	**62**	2,449	1	1	1
117	**24.3**	22.6	**70**	2,449	1	1	2
150	**23.4**	47.4	**65**	2,449	1	1	1
170	**19.4**	20.8	**77**	2,449	5	4	5
173	**20.5**	16.3	**80**	2,449	6	5	6
Average	**23.7**	32.8	**71**	2,449	–	–	–
Resource Cost*			**6,559**	176,348	**27× better**		

*Resource Cost = Time × Memory (lower is better).
Actions: Branch = select variable, Cuts = add learned cuts, Prune = remove ineffective cuts.

Algorithm 2 Cut Pruning Heuristic

Input: Current cut pool $\mathcal{C}_{\text{pool}}$, LP state, max age T_{age}, activity threshold τ_{act}
Output: Pruned cut pool $\mathcal{C}'_{\text{pool}}$
$\mathcal{C}_{\text{remove}} \leftarrow \emptyset$
for $c \in \mathcal{C}_{\text{pool}}$ **do**
  $\text{age}_c \leftarrow$ rounds since cut c was added
  $\text{activity}_c \leftarrow$ fraction of recent LPs where c was tight
  if $\text{age}_c > T_{\text{age}}$ **and** $\text{activity}_c < \tau_{\text{act}}$ **then**
    $\mathcal{C}_{\text{remove}} \leftarrow \mathcal{C}_{\text{remove}} \cup \{c\}$
  end if
end for
return $\mathcal{C}_{\text{pool}} \setminus \mathcal{C}_{\text{remove}}$

A.13 PPO Implementation Details

We provide detailed PPO implementation specifics for reproducibility.

A.13.1 Generalized Advantage Estimation (GAE)

We compute advantages using GAE (Schulman et al., 2015):

$$\hat{A}_t = \sum_{l=0}^{T-t} (\gamma\lambda)^l \delta_{t+l}, \quad \text{where} \quad \delta_t = r_t + \gamma V(s_{t+1}) - V(s_t) \tag{18}$$

This provides a bias-variance trade-off controlled by $\lambda \in [0, 1]$. We use $\lambda = 0.95$ which empirically balances stability and learning speed.

A.13.2 Clipped Surrogate Objective

The PPO clipped objective prevents destructively large policy updates:

$$\mathcal{L}^{\text{CLIP}}(\theta) = \mathbb{E}_t \left[\min \left(\rho_t(\theta)\hat{A}_t,\ \text{clip}(\rho_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t \right) \right] \tag{19}$$

where $\rho_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$ is the importance sampling ratio.

A.13.3 Entropy Bonus

We add an entropy bonus to encourage exploration:

$$\mathcal{H}[\pi_\theta] = -\mathbb{E}_{a\sim\pi_\theta}[\log \pi_\theta(a|s)] \tag{20}$$

The entropy is computed analytically for categorical distributions (Manager meta-policy, branch selection) and approximated via sampling for the Worker's cut selection.

A.13.4 Value Function Loss

We use clipped value function loss for stability:

$$\mathcal{L}^{\text{VF}}(\theta) = \mathbb{E}_t\left[\max\left((V_\theta(s_t) - \hat{R}_t)^2,\ (\text{clip}(V_\theta(s_t), V_{\theta_{\text{old}}}(s_t) \pm \epsilon_V) - \hat{R}_t)^2\right)\right] \tag{21}$$

where $\hat{R}_t = \hat{A}_t + V_{\theta_{\text{old}}}(s_t)$ is the target return and $\epsilon_V = 0.2$.

A.13.5 Combined Loss

The total loss combines policy, value, and entropy terms:

$$\mathcal{L}(\theta) = -\mathcal{L}^{\text{CLIP}}(\theta) + c_v\mathcal{L}^{\text{VF}}(\theta) - c_e\mathcal{H}[\pi_\theta] \tag{22}$$

with $c_v = 0.5$ and $c_e = 0.01$.

A.13.6 Minibatch Updates

For each episode trajectory:

1. Collect trajectory $\tau = \{(s_t, a_t, r_t, s_{t+1})\}_{t=0}^{T}$
2. Compute GAE advantages $\{\hat{A}_t\}$ and returns $\{\hat{R}_t\}$
3. For $K = 4$ epochs:
   (a) Shuffle trajectory data
   (b) Split into minibatches of size 64
   (c) For each minibatch: compute loss, backpropagate, update parameters

A.14 Context Extraction Algorithm

Algorithm 3 details the context extraction procedure used by the Worker.

Algorithm 3 Context Extraction for Cut Selection

Input: Variable embeddings $\mathbf{H}_v$, constraint embeddings $\mathbf{H}_c$, global embedding $\mathbf{h}^g$, candidate cuts $\mathcal{C}_{\text{cand}}$, coefficient matrix $\mathbf{A}$
Output: Context embeddings $\{\mathbf{e}_c^{\text{ctx}}\}_{c \in \mathcal{C}_{\text{cand}}}$
for $c \in \mathcal{C}_{\text{cand}}$ **do**
  {Step 1: Find variables in cut's support}
  $\mathcal{V}_c \leftarrow \{j : A_{cj} \neq 0\}$
  {Step 2: Find constraints sharing variables with cut}
  $\mathcal{C}_c \leftarrow \{i : \exists j \in \mathcal{V}_c \text{ s.t. } A_{ij} \neq 0\}$
  {Step 3: Aggregate variable embeddings}
  if $|\mathcal{V}_c| > 0$ **then**
    $\mathbf{e}_c^v \leftarrow \frac{1}{|\mathcal{V}_c|} \sum_{j \in \mathcal{V}_c} \mathbf{h}_{v,j}^{(L)}$
  else
    $\mathbf{e}_c^v \leftarrow \mathbf{0}$
  end if
  {Step 4: Aggregate constraint embeddings}
  if $|\mathcal{C}_c| > 0$ **then**
    $\mathbf{e}_c^c \leftarrow \frac{1}{|\mathcal{C}_c|} \sum_{i \in \mathcal{C}_c} \mathbf{h}_{c,i}^{(L)}$
  else
    $\mathbf{e}_c^c \leftarrow \mathbf{0}$
  end if
  {Step 5: Concatenate with global embedding}
  $\mathbf{e}_c^{\text{ctx}} \leftarrow [\mathbf{e}_c^v \,\|\, \mathbf{e}_c^c \,\|\, \mathbf{h}^g]$
end for
return $\{\mathbf{e}_c^{\text{ctx}}\}_{c \in \mathcal{C}_{\text{cand}}}$