CoMILP: Evolving Collaborative Solver Components for Mixed-Integer Linear Programming

Anonymous authors
Paper under double-blind review

Abstract

Mixed-integer linear programming (MILP) is a fundamental framework for solving combinatorial optimization problems. Modern MILP solvers rely on multiple interacting components whose design and configuration strongly influence performance. During the solving process, these components interact repeatedly, with decisions made in one component reshaping the solver states encountered by others. Although existing work in automated algorithm design has improved MILP solving performance, such approaches typically focus on either parameter tuning or heuristic design for a single selected component rather than optimizing jointly across the solver, leaving improvements that require complementary changes across components unexplored. To address this limitation, we propose CoMILP, a large language model (LLM)-driven framework for cross-component algorithm design in MILP solvers. Specifically, CoMILP jointly optimizes initial parameter settings, dynamic configuration policies across components, and decision rules for cut selection and branching. CoMILP employs a multi-agent coordination framework in which LLM agents repeatedly select solver components and edit type, propose a local update, and reflect on solver feedback, thereby enabling scalable optimization across multiple components. Tree search retains alternative refinement paths and balances exploration and exploitation, enabling complementary updates to build on earlier component changes. Experiments on five problem families and MIPLIB 2017 show large improvements over default SCIP, including a 34.4% reduction in primal–dual integral on MIPLIB. Ablations further establish the benefits of cross-component evolution.

1 Introduction

Mixed-integer linear programming (MILP) is a fundamental framework for combinatorial optimization and is widely used in industrial applications such as supply-chain planning, transportation, and power-system operations (Floudas & Lin, 2005; Hoffman & Padberg, 1993; Knueven et al., 2020). Modern MILP solvers are based on branch-and-cut, which searches over a tree of LP relaxations, using branching to divide the solution space and cutting planes to remove fractional solutions without excluding feasible integer solutions (Achterberg, 2009). In practice, these solvers also rely on many algorithmic components, including presolving, branching, cut generation and selection, node selection, and primal heuristics. The settings and heuristic rules used by these components can have a large impact on solver performance, and their effects often depend on one another (Hutter et al., 2010). Choosing and tuning these settings and rules requires substantial expert knowledge and experimentation, which has motivated learning-based methods to automate solver configuration and heuristic decisions (Bengio et al., 2021).

Learning-based methods reduce the need for manually designed solver decisions by learning from previously solved instances. Following the taxonomy of Bengio et al. (2021), existing methods broadly learn solutions, algorithm configurations, or decisions within the optimization process. For solution prediction, Neural Diving predicts variable assignments and fixes a subset of them before solving the reduced MILP (Nair et al., 2020), while Predict-and-Search predicts a candidate solution and searches a neighborhood around it for better feasible solutions (Han et al., 2023). Other methods learn decisions inside the solver, such as branching (Gasse et al., 2019) and cut selection (Paulus

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.

et al., 2022), or adapt solver configurations during the solving process (Li et al., 2023; Ye et al., 2025b). These methods showed that learned policies can replace manually designed rules for important parts of MILP solving. More recently, large language models (LLMs) have made it possible to search over the heuristic rules themselves rather than learn a parameterized model. LLM-based algorithm evolution generates executable code, evaluates it on problem instances, and uses the observed performance to produce improved variants (Novikov et al., 2025). EoH evolves heuristic descriptions together with their implementations (Liu et al., 2024), ReEvo uses performance feedback and reflection to revise candidate heuristics (Ye et al., 2024), and MCTS-AHD organizes candidate revisions in a search tree to explore different algorithm designs (Zheng et al., 2025). This idea has recently been applied to MILP solvers: LLM-LNS evolves neighborhood-selection rules (Ye et al., 2025a), LLM4Branch evolves branching programs and their parameters (Hou et al., 2026), and MILP-Evo jointly evolves branching and cut-selection policies (Nie et al., 2026). However, these methods still evolve either a single solver component or a subset of components fixed in advance.

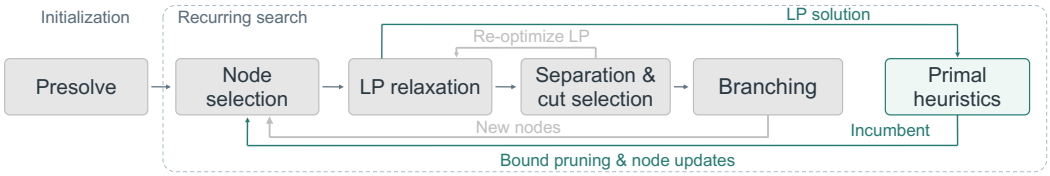

Figure 1: **Recurring interactions in branch-and-cut.** Cuts, branching decisions, and incumbent updates reshape the solver states encountered by other components throughout the solving process.

This fixed scope can be limiting because solver components interact repeatedly during solving (Figure 1). For example, a new cut-selection rule may offer little benefit under the current separation settings but become effective when an additional separator provides different candidate cuts. The selected cuts change the LP relaxation and influence branching decisions, which create new subproblems where separation and cut selection interact again. A search limited to selected components can therefore miss improvements that require changes elsewhere in the solver. Appendix D illustrates this dependence, including cases where changing separation settings reverses the relative performance of branching rules. However, jointly evolving multiple components creates a high-dimensional search space over their parameters and heuristics, with the number of possible combinations growing rapidly as more components are included.

To address these challenges, we propose **CoMILP**, an LLM-driven framework for **Co**ordinated algorithm evolution across the **MILP** solving process. CoMILP jointly optimizes initial parameter settings, dynamic configuration policies, and decision rules for cut selection and branching. It searches over complete solver strategies through component-level updates, inheriting the surrounding settings and policies so that each evaluation captures their interactions. Coordinated agents use a generate–evaluate–reflect loop to choose a component and edit type, implement a local change, and interpret its effect on solving. This feedback guides subsequent updates and allows components to be revisited as their context changes. Monte Carlo tree search is applied in our framework to retain alternative refinement paths, balancing exploitation and exploration (Zheng et al., 2025). For reuse on new instances, Hydra (Xu et al., 2010) assembles the evolved strategies into a portfolio with an instance feature-based selector. Experiments on five problem families and MIPLIB 2017 (Gleixner et al., 2021) show that our CoMILP improves primal–dual integral (PDI) over default SCIP in all eleven benchmarks, achieving the lowest PDI among the compared methods in nine and a 34.4% PDI reduction on MIPLIB. Equal-budget ablations establish the benefits of cross-component adaptation, joint configuration and heuristic evolution, and coordinated agent roles. Hydra-based portfolios further demonstrate the generalization ability on unseen instances.

2 Preliminaries

A mixed-integer linear program (MILP) has the form

$$\min_{x \in \mathbb{R}^n} \left\{ c^\top x : Ax \le b,\ l \le x \le u,\ x_j \in \mathbb{Z}\ (j \in \mathcal{J}) \right\}, \tag{1}$$

where $A \in \mathbb{R}^{m \times n}$, $b \in \mathbb{R}^m$, $c \in \mathbb{R}^n$, l and u are variable bounds, and $\mathcal{J}$ indexes integer variables.

Branch-and-Cut. Branch-and-cut explores a tree of subproblems. At each node, an LP relaxation provides a lower bound, while the best feasible solution found so far provides an upper bound. Cutting planes tighten the relaxation without excluding feasible mixed-integer solutions, and branching divides unresolved subproblems into smaller ones. Nodes are discarded when they are infeasible or cannot improve the best known solution. The search continues until optimality is proved, a gap tolerance is met, or a resource limit is reached (Achterberg, 2009).

Key Algorithmic Components. *Branching* typically selects an integer variable with a fractional LP value and splits its domain to create child subproblems. Candidates can be ranked using historical estimates or additional LP evaluations. *Cut selection* chooses which inequalities generated by separators enter the relaxation, balancing bound improvement against the cost of solving larger LPs. These decisions interact: cuts change branching candidates, while branching creates the subproblems where further cuts are generated (Achterberg, 2009; Hojny et al., 2025). Other algorithmic components in MILP solvers can simplify the model, tighten variable bounds, find feasible solutions, and select the next node.

Important Hyperparameters. Parameters in MILP solvers control how often components run, how much work they do, and how candidates are ranked. Examples include presolve and separation round limits, cut budgets, heuristic frequencies, strong-branching budgets, and cut-scoring weights (Hojny et al., 2025). Some of these, such as presolve limits, only matter during preprocessing and are fixed before the search starts. Others, such as separator frequencies, cut budgets, and cut-scoring weights, can be changed while the solver is running. Adjusting these parameters online forms a *dynamic configuration problem* (Adriaensen et al., 2022; Xue et al., 2022).

3 METHOD

CoMILP searches over complete solver strategies through a sequence of component-level updates, based on the idea that changing one component also changes the conditions under which the others operate. For this reason, every candidate is evaluated as a whole strategy that keeps the parent's remaining settings and policies. As shown in Figure 2, the search runs as a loop in which tree search picks a strategy to refine, three LLM agents choose a component and edit type, propose a bounded update, and reflect on the solver's feedback, and the evaluated child then goes back into the tree so that later edits to other components can build on it. In the end, the evolved strategies are reused through portfolio-based selection.

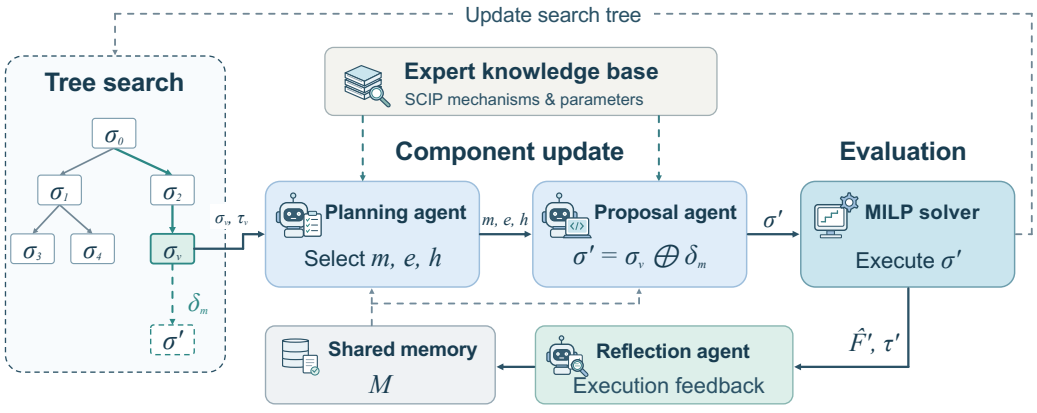

Figure 2: **CoMILP's coordinated algorithm evolution.** Tree search selects a complete strategy. Planning chooses a component block m, edit type, and hypothesis; proposal implements the update. Solver costs guide tree search, while reflection stores trajectory-based observations in shared memory to inform subsequent updates.

3.1 CROSS-COMPONENT STRATEGY SPACE

A complete solver strategy combines an initial configuration with policies executed during solving:

$$\sigma = (\theta, \pi), \qquad \pi = (\pi_{\text{param}}, \pi_{\text{cut}}, \pi_{\text{branch}}). \tag{2}$$

The three parts act at different times. The initial configurations θ are fixed before solving; they span 136 SCIP parameters in ten component groups, from presolve and separation through branching and node selection. The dynamic policy π_{param} changes configurations *while* solving: each rule is an if–then statement over the solver state s_t (gap, incumbent stagnation, LP effort per node, elapsed time) that sets or scales a parameter for a bounded number of nodes and then restores it. The two scoring policies act at individual decisions, ranking candidate cuts and branching variables.

We represent these decision rules as feature-based corrections to reference heuristic scores. For $m \in \{\text{cut}, \text{branch}\}$ and candidate a in the current pool $\mathcal{C}_t^m$,

$$f_m(a, s_t) = f_m^{\text{ref}}(a, s_t) + \sum_{j=1}^{d_m} \alpha_{mj} \widehat{z}_{mj}(a, s_t; \mathcal{C}_t^m). \tag{3}$$

Here, $\widehat{z}_{mj}$ is a candidate feature normalized by its rank within the current pool, and α_{mj} sets the direction and strength of the correction. The reference f_m^{ref} is SCIP's fractionality score for branching and a fixed efficacy–sparsity–age combination for cuts. In other words, the LLM never writes a scoring function from scratch: it nudges a known-good rule with at most three weighted features under $\|\alpha_m\|_1 \leq 0.75$, and zero coefficients recover the reference exactly. Ranks are recomputed at every decision, so the same coefficients can reorder candidates as the relaxation changes. The used feature lists and rule semantics are provided in Appendix B.2.

Let Σ denote the resulting strategy space and $\mathcal{D}_{\text{tr}}$ a set of training instances. We evaluate a strategy by its normalized primal–dual integral (PDI), the area under the primal–dual gap curve over a common horizon T. For an instance I, the expected cost is

$$J_I(\sigma) = \mathbb{E}_\omega \left[\frac{1}{T} \int_0^T g_{\text{PD}}(t; I, \sigma, \omega)\, dt \right], \tag{4}$$

where ω is the solver seed and $g_{\text{PD}} \in [0, 1]$ is the bounded primal–dual gap (Berthold, 2013). PDI is small only when a good incumbent and a tight dual bound both arrive early, and it still separates methods on instances that nothing solves within T. We minimize average training cost:

$$\min_{\sigma \in \Sigma} F(\sigma), \qquad F(\sigma) = \frac{1}{|\mathcal{D}_{\text{tr}}|} \sum_{I \in \mathcal{D}_{\text{tr}}} J_I(\sigma). \tag{5}$$

For instance-specific design, $\mathcal{D}_{\text{tr}} = \{I\}$; problem-level portfolio learning uses the complementary objective in Section 3.4. Evaluations return estimated costs $\widehat{J}_I$, their aggregate $\widehat{F}$, and trajectories τ_σ recording bound progress and component activity. Costs guide tree search, while trajectories inform component updates.

3.2 Multi-Agent Coordination

Three LLM agents carry out each component update in three stages, namely planning, proposal, and reflection, and they share a memory $\mathcal{M}$ that records prior interventions, their outcomes, and the strategy contexts in which they were made. An intervention mechanism ties the roles together: planning identifies a change, proposal implements it, and reflection assesses its effect.

Planning. Given a parent σ_v, objective F, trajectories τ_v, and relevant experience from $\mathcal{M}$, the planner selects a component block m, a supported edit type $e \in \{\text{initial}, \text{dynamic}, \text{scorer}\}$, and a hypothesis h linking a bottleneck to an intended change. For example, high separation effort with little dual-bound progress may motivate revised separation timing or cut selection. Here $m \subseteq \mathcal{G}$ is a nonempty subset of the ten component groups. An initial-configuration proposal can coordinate a small set of parameters across groups; a scoring-rule proposal targets either cut selection or branching. Reconsidering the block and edit type allows search to follow shifting bottlenecks.

Proposal. The proposal agent combines the hypothesis, parent strategy, solver documentation, and retrieved experience to produce a bounded edit δ_m. Applying it yields

$$\sigma' = \sigma_v \oplus \delta_m, \qquad \sigma'_{-m} = (\sigma_v)_{-m}, \tag{6}$$

where $\oplus$ applies the edit, and σ_{-m} denotes the configurations and heuristics outside the selected component block. Within the block, only the proposed assignments or heuristics change; all other entries are inherited. One trial evaluates one complete candidate, including a joint configuration edit, on the common search seeds. The solver executes the resulting complete strategy, so the evaluation captures how the update interacts with the inherited components.

Reflection. The reflection agent compares parent and child executions against h, examining component effort, bound progress, and per-instance outcomes. It stores these observations with the edited component and surrounding strategy in $\mathcal{M}$. Later updates can thus revisit an intervention when changes elsewhere alter its effect. Agent inputs and memory design are in Appendix B.3.

3.3 Monte Carlo Tree Search

A component update can become useful through subsequent changes elsewhere: revised separation may enable better cut selection and then branching. CoMILP explores these sequences with Monte Carlo tree search (Zheng et al., 2025). Each node stores a complete strategy, its estimated cost, and trajectories; each edge records one candidate edit. All components and edit types share the tree.

Selection and expansion. Progressive widening limits the number of children of a node u that has been visited N_u times to $\lfloor k(N_u+1)^\alpha \rfloor$ (we use $k = 1.5$ and $\alpha = 0.5$), so that a strategy accrues additional alternative refinements only as it is revisited (Couëtoux et al., 2011). Starting from the root, the search expands a node whenever this cap permits. Otherwise, it descends to the child v with the largest upper-confidence score:

$$U(v) = Q_v + c\sqrt{\frac{\log(N_{p(v)}+1)}{N_v}}, \qquad c > 0, \tag{7}$$

where $p(v)$ is the parent and N_v counts the evaluations whose rewards have been backed up through v (Kocsis & Szepesvári, 2006). The value Q_v is the mean of these rewards, with $r(\sigma) = 1 - \widehat{F}(\sigma)$. Thus, selection considers both the observed quality of a refinement subtree and the opportunity for further exploration. At the node selected for expansion, the agents plan an update, propose an edit, and evaluate the resulting strategy.

Reward backup and strategy refinement. Each evaluated child enters the tree and its reward is backed up to the root; reflection updates $\mathcal{M}$ at the same time. Children that do not beat their parent stay in the tree. Discarding them would close the complementary paths the search is meant to find. The best-so-far strategy $\sigma^\star$ is the node with the lowest estimated cost. Rewards decide *where* to search; reflection decides *how* to revise. Algorithm 1 in Appendix B.4 describes the procedure.

3.4 Portfolio Construction and Strategy Selection

A single strategy tuned on training instances may not transfer to unseen ones. To generalize across a problem family, we therefore learn a *portfolio* of complementary strategies together with an instance feature-based selector, using Hydra (Xu et al., 2010). The portfolio is initialized with SCIP-default. In each round, we run Algorithm 1 with an objective that rewards improvement over the best strategy currently assigned to each training instance, so that each new strategy targets instances the current portfolio handles poorly. The resulting strategy is added to the portfolio, and the selector is retrained on the observed costs. At test time, given an unseen instance, we run a five-second SCIP-default probe and extract structural features. The selector maps these instance features to a strategy from the portfolio to enable generalization *across* instances. Appendix B.5 details the portfolio objective, the feature set, and the selector training procedure.

4 Experiments

We evaluate cross-component evolution through instance-specific performance, equal-budget ablations, search behavior, and reuse on unseen instances. Together, these studies test the value of adapting multiple solver components within complete strategies.

4.1 Experimental Setup

Benchmarks and evaluation. We use set covering (SC), maximum independent set (MIS), combinatorial auctions (CA), and capacitated facility location (CFL) following Gasse et al. (2019), and multiple knapsack (MK) following Wang et al. (2023). Each family has two scales with 25 instances per scale; MIPLIB 2017 adds 240 heterogeneous instances (Gleixner et al., 2021). Each adaptation method receives 100 trials per target, scored on two common search seeds. The selected strategy is frozen and evaluated on four held-out seeds of the same instance. All solves use SCIP 10.0.1, one thread, a 300-second horizon, and a 10^6-node limit. We report mean PDI in percentage-seconds and capped solving time, averaging over seeds and then instances; lower is better. PDI uses the same gap integral as Equation (4), scaled by $100T$ into percentage-seconds. Appendix A.2 specifies the generators and scales; Appendix A.5 defines gap integration, node-limit handling, and aggregation.

Baselines and budgets. SCIP-default (Achterberg, 2009) is the reference. We compare learned solver policies (HEM Wang et al. (2023), Collab Li et al. (2025), DynSep (Ye et al., 2025b)), parameter configuration tuning (SMAC-I (Hutter et al., 2011; Lindauer et al., 2022), GP-BO (Snoek et al., 2012), GRIMIP (Luo et al., 2026), ColdStart (Lawless et al., 2025)), and LLM policy evolution (LLM4Branch (Hou et al., 2026), MILP-Evo (Nie et al., 2026)). HEM, Collab, and DynSep are pretrained per family and fine-tuned on each target; we omit them from MIPLIB because this protocol requires family-level training data. CoMILP uses `gpt-5.6-terra` at temperature zero. The comparison matches target-adaptation trials while retaining each method's search space and training procedure. Appendix A.6 describes the implementations and adaptations, and Appendix A.4 specifies search spaces and budget accounting. Reported solving times exclude strategy design.

4.2 Evolved Solver Performance

Performance across benchmarks. Table 1 reports results on one scale per family plus MIPLIB. CoMILP improves mean PDI over SCIP-default on all eleven benchmarks and attains the lowest mean PDI on nine. On MIPLIB, it reduces PDI by 34.4%, compared with 23.1% for the strongest baseline GRIMIP, and reduces capped time by 14.3%. On MK, PDI drops by 71.9% and capped time falls from 291.0 to 139.0 seconds. Appendix Tables 18 and 19 report complete results across all scales, where CoMILP achieves the lowest PDI in four additional settings.

Bound progress and proof time. The two metrics capture distinct benefits and can diverge. On CA, every method hits the 300-second cap, yet CoMILP reduces PDI by 27.6%, indicating faster progress in closing the primal–dual gap. On CFL, the pattern reverses: CoMILP attains the lowest capped time (32.4 seconds), while SMAC-I achieves lower PDI (56.5 versus 97.6), as does GRIMIP on the other CFL scale. Taken together, cross-component search improves both bound progress and time to proof across the benchmark collection.

4.3 Ablation Studies

We isolate component coverage, edit types, and agent support under matched candidate-evaluation and LLM token budgets. Each variant uses 25 targets per family and the same search and held-out seeds. Table 2 shows three families; Appendix Table 20 includes MIS and MK. Appendix A.7 specifies which configurations, policies, and agent inputs each control retains.

Component coverage. Both *Branching only* and *No branching* have higher PDI than full CoMILP in every family. On SC, their reductions over SCIP-default are 38.4% and 52.7%, respectively, versus 76.0% for the full method. On CFL, the full method exceeds the stronger restricted variant by 12.2 percentage points. These comparisons support adapting branching together with other components rather than fixing either side throughout search.

Initial configurations and online policies. *Initial configuration only* and *Policies only* both underperform joint evolution across all five families. On CA, their PDI reductions are 4.4% and 4.3%, compared with 23.6% for CoMILP. Allowing both edit types therefore yields better strategies under the same trial budget. The gain from joint evolution is particularly pronounced here: initial configurations and online policies provide complementary ways to adapt the same solving process.

Table 1: Solver performance on five generated benchmarks and MIPLIB 2017. Time is limited to 300s; PDI is measured in percentage-seconds; Impr. is the relative reduction in mean PDI compared to SCIP-default. Bold marks the lowest time, the lowest PDI and the highest improvement.

	Set covering 1,000 columns, 1,700 rows			**Maximum independent set** 1,100 vertices, 4,384 edges			**Multiple knapsack** 600 items, 16 knapsacks		
Method	Time (s)	PDI	Impr. (%)	Time (s)	PDI	Impr. (%)	Time (s)	PDI	Impr. (%)
SCIP-default	294.5	3453.6	0.0	235.7	883.9	0.0	291.0	275.3	0.0
HEM	298.7	3223.6	6.7	202.8	630.4	28.7	272.4	185.8	32.5
Collab	268.6	3286.7	4.8	162.3	464.3	47.5	277.9	223.9	18.7
DynSep	288.4	3645.2	−5.5	142.1	484.5	45.2	247.5	145.1	47.3
SMAC-I	259.3	2323.6	32.7	204.8	528.6	40.2	291.1	154.7	43.8
GP-BO	257.3	2365.7	31.5	225.5	608.5	31.2	275.7	173.2	37.1
GRIMIP	277.0	2744.5	20.5	228.1	525.2	40.6	272.6	155.7	43.4
ColdStart	271.4	2821.6	18.3	288.1	747.5	15.4	300.0	135.6	50.7
LLM4Branch	272.3	3307.8	4.2	227.8	609.3	31.1	288.2	207.0	24.8
MILP-Evo	286.1	3533.0	−2.3	184.6	592.0	33.0	290.1	198.8	27.8
CoMILP	**238.2**	**2044.2**	**40.8**	**124.9**	**426.2**	**51.8**	**139.0**	**77.4**	**71.9**

	Combinatorial auctions 600 items, 4,000 bids			**Capacitated facility location** 80 facilities, 240 customers			**MIPLIB 2017** 240 instances		
Method	Time (s)	PDI	Impr. (%)	Time (s)	PDI	Impr. (%)	Time (s)	PDI	Impr. (%)
SCIP-default	300.0	4943.4	0.0	98.7	194.2	0.0	246.6	15603.5	0.0
HEM	300.0	5054.7	−2.3	101.5	181.0	6.8	–	–	–
Collab	300.0	4598.3	7.0	68.2	137.2	29.4	–	–	–
DynSep	300.0	4799.4	2.9	82.9	182.4	6.1	–	–	–
SMAC-I	300.0	4282.9	13.4	42.8	**56.5**	**70.9**	230.2	14909.5	4.4
GP-BO	300.0	4402.6	10.9	53.6	75.8	61.0	231.2	13703.6	12.2
GRIMIP	300.0	4346.6	12.1	51.0	71.6	63.1	223.8	12006.0	23.1
ColdStart	300.0	4935.1	0.2	58.4	78.8	59.4	245.2	15041.8	3.6
LLM4Branch	300.0	5139.7	−4.0	49.9	91.4	52.9	244.7	16218.2	−3.9
MILP-Evo	300.0	4982.5	−0.8	108.0	112.6	42.0	238.4	15625.0	−0.1
CoMILP	300.0	**3578.3**	**27.6**	**32.4**	97.6	49.7	**211.3**	**10235.9**	**34.4**

Retrieval, memory, and role separation. Removing documentation retrieval or experience memory, or replacing the three roles with one agent, raises PDI in every family. On SC, the respective reductions are 72.0%, 66.7%, and 72.1%, versus 76.0% for CoMILP. The largest losses occur on MK, where the full method reduces PDI by 89.8%, while these variants achieve -12.4%, -5.2%, and 12.5%. With representation and tree search fixed, these results support informed proposals and contextual feedback for cross-component evolution.

4.4 Search Behavior and Component Interactions

Progress within the search budget. Figure 3 tracks best-so-far PDI on six MIPLIB instances. CoMILP finishes lowest on four and continues improving late in several searches; on `comp21-2idx` and `opm2-z10-s4`, substantial gains occur around trial 80. This sustained progress is consistent with the framework's ability to revisit component choices and refine complete strategies throughout the budget. The curves use search seeds, whereas Table 1 evaluates the frozen strategies on held-out seeds. Appendix E.4 defines the evaluation axis and common reference.

Table 2: Ablation results on 25 instances per problem family at the indicated instance sizes. All variants use 100 candidate evaluations with matched LLM token and solver-evaluation budgets. Appendix A.7 defines the configuration of the ablation study. Impr. denotes the percentage of relative improvements; metric definitions and boldface follow Table 1.

	Set covering 1,000 columns, 900 rows			**Combinatorial auctions** 400 items, 2,000 bids			**Capacitated facility location** 80 facilities, 240 customers		
Method	Time (s)	PDI	Impr.	Time (s)	PDI	Impr.	Time (s)	PDI	Impr.
SCIP-default	183.7	1610.8	0.0	300.0	2222.8	0.0	98.7	194.2	0.0
Branching only	132.0	991.5	38.4	300.0	2071.5	6.8	62.3	132.6	31.7
No branching	90.5	762.5	52.7	300.0	1962.7	11.7	52.1	121.4	37.5
Initial config. only	75.6	612.2	62.0	300.0	2125.0	4.4	45.8	154.5	20.4
Policies only	106.1	722.3	55.2	300.0	2126.7	4.3	42.7	130.1	33.0
No retrieval	60.4	450.5	72.0	300.0	1992.2	10.4	40.8	143.3	26.2
No memory	78.1	536.4	66.7	300.0	1973.8	11.2	67.2	154.0	20.7
Single agent	54.8	449.0	72.1	300.0	1834.1	17.5	41.3	143.9	25.9
CoMILP	**52.4**	**387.0**	**76.0**	300.0	**1698.4**	**23.6**	**32.4**	**97.6**	**49.7**

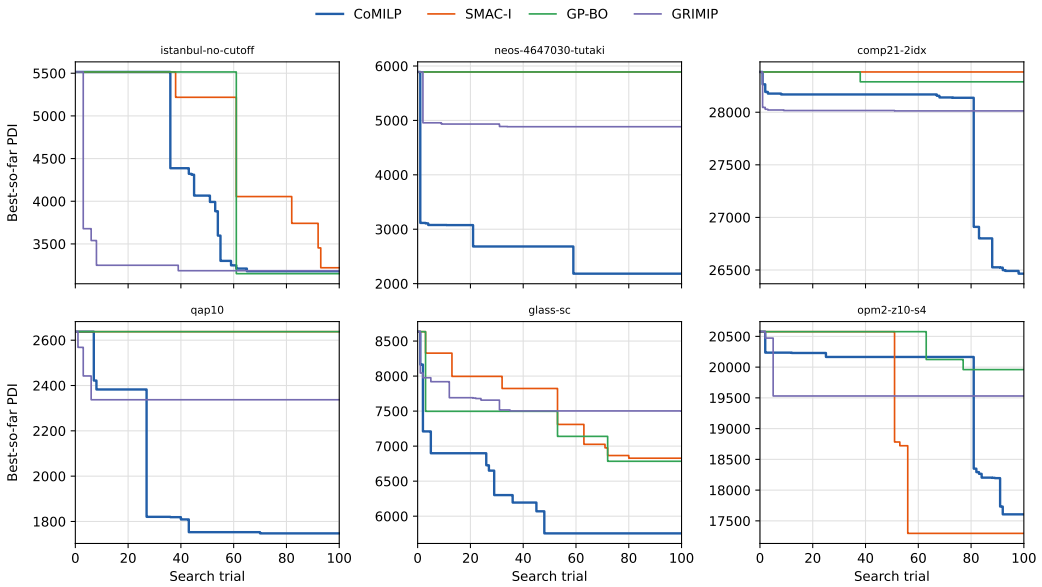

Figure 3: Best-so-far PDI over 100 candidate evaluations on six MIPLIB instances.

Allocation across components. Figure 4 shows that proposal allocation varies by family and instance. Branching receives the largest share on MIS and MK, while primal heuristics receive the largest share on SC and CFL. On MIPLIB, the primal-heuristic share ranges from zero to roughly two thirds. This variation shows how the cross-component search space is used in practice: the planner directs proposal effort toward different mechanisms as the instance and inherited strategy change. Appendix E.4 explains how joint proposals are fractionally attributed and how instance shares are aggregated.

Jointly optimizing branching and cuts. One candidate on MK jointly revises branching and cut separation, inheriting all other configurations and policies. Mean PDI falls from 85.68 to 24.56, capped time from 89.04 to 43.69 seconds, and strong-branching LP iterations by 65.9%. Cut management shapes the relaxation on which branching spends its LP work. CoMILP adjusts both controls in one trial within the evolving strategy. Appendix E.5 documents all eight assignments, the inherited policy, and evolution details.

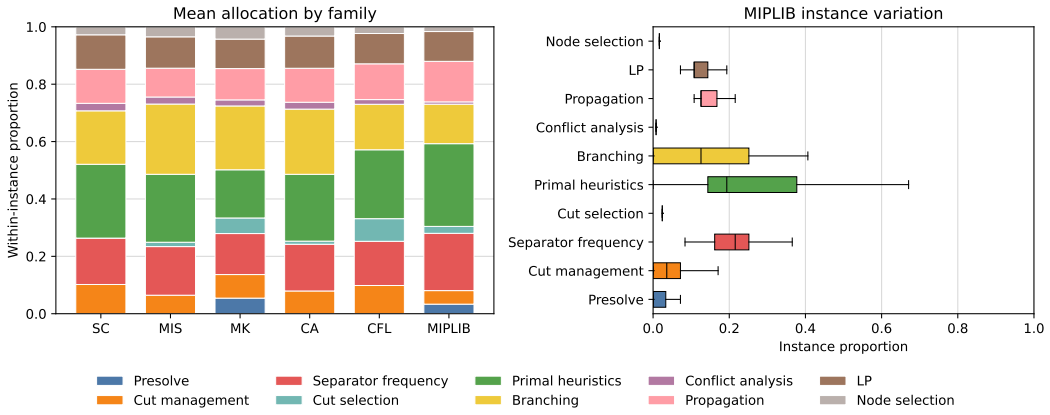

Figure 4: Frequency of proposals on each component. Left: mean frequency per problem family and MIPLIB. Right: box plot across MIPLIB instances with median, IQR, and 1.5-IQR whiskers.

4.5 Strategy Reuse on Unseen Instances

To test reuse across multiple instances, we apply Hydra (Xu et al., 2010) to build a portfolio from 200 training instances per family in five rounds of 60 updates. A separate set of 100 validation instances selects the classifier, and 100 unseen instances form the test set. The five-second probe and selected solve share the 300-second budget. Appendix A.3 specifies the disjoint splits and sizes. Appendix B.5 details the complementary objective, selector training, and probe-inclusive evaluation.

Table 3: Problem-level strategy reuse on 100 unseen instances per problem family. Mean PDI includes the probe and selected solve. Reduction is relative to SCIP-default. Table 21 in Appendix gives full results.

	SC	MIS	MK	CA	CFL
SCIP-default	1008.1	687.6	57.9	2182.1	119.4
CoMILP+Hydra	636.1	486.8	42.9	2025.0	97.5
PDI reduction	36.9%	29.2%	25.9%	7.2%	18.3%

CoMILP+Hydra reduces PDI over SCIP-default by 7.2%–36.9% across the five families (Table 3), without target-specific evolution at test time. The same cross-component representation supports all five families, with selection across instances and policy adaptation within each solve. Appendix Table 21 reports PDI and capped time for all family-trained comparators (HEM, Collab, DynSep, SMAC-P, and GP-BO). These results show that the evolved strategies retain practical value beyond the instances used to construct them.

5 Conclusion

CoMILP enables cross-component algorithm design for MILP by jointly evolving initial settings, dynamic configuration policies, and decision rules for cut selection and branching. Coordinated agents refine complete strategies using solver feedback, while tree search preserves paths for complementary component updates. Experiments yield the lowest mean PDI on nine of eleven benchmarks, including a 34.4% reduction over default SCIP on MIPLIB. Equal-budget ablations support the benefits of component coverage and joint configuration–policy evolution, and family-trained portfolios extend the gains over default SCIP to unseen instances. The central design principle is to optimize each component in the context of its partners, enabling coordinated solver adaptation through a unified strategy space.

Reproducibility Statement

Section 3 specifies the strategy space, search objective, agent roles, and tree-search procedure. Appendix B defines policy execution, reward backup, and portfolio selection; Appendix A.5 defines metric aggregation. Appendix C provides role-specific prompt templates and an executed MK configuration-update example. The solver environment, instance-generation settings, data splits, baseline adaptations, and ablation definitions are given in Appendix A. Appendix D specifies the constructed instances and search rules used to illustrate component interactions. Appendix E gives the additional results, search-history analysis, and the MK component assignments and paired parent–child outcomes.

AI Use Statement

LLM-guided planning, candidate generation, and reflection are integral components of the proposed algorithm, as described in Section 3. Generative AI was also used to assist with manuscript revision, literature organization and the organization and structuring of supplementary material. The authors take full responsibility for the final content of the paper, including all claims, experimental materials, and AI-assisted text.

References

Tobias Achterberg. SCIP: Solving constraint integer programs. *Mathematical Programming Computation*, 1(1):1–41, 2009.

Steven Adriaensen, André Biedenkapp, Gresa Shala, Noor Awad, Theresa Eimer, Marius Lindauer, and Frank Hutter. Automated dynamic algorithm configuration. *Journal of Artificial Intelligence Research*, 75:1633–1699, 2022.

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

Timo Berthold. Measuring the impact of primal heuristics. *Operations Research Letters*, 41(6): 611–614, 2013.

Leo Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.

Adrien Couëtoux, Jean-Baptiste Hoock, Nataliya Sokolovska, Olivier Teytaud, and Nicolas Bonnard. Continuous upper confidence trees. In *Proceedings of the 5th International Conference on Learning and Intelligent Optimization (LION)*, pp. 433–445, Rome, Italy, 2011.

David Eriksson, Michael Pearce, Jacob Gardner, Ryan D. Turner, and Matthias Poloczek. Scalable global optimization via local Bayesian optimization. In *Proceedings of the 33rd Conference on Neural Information Processing Systems (NeurIPS)*, pp. 5496–5507, Vancouver, Canada, 2019.

Christodoulos A. Floudas and Xiaoxia Lin. Mixed integer linear programming in process scheduling: Modeling, algorithms, and applications. *Annals of Operations Research*, 139(1):131–162, 2005.

Maxime Gasse, Didier Chételat, Nicola Ferroni, Laurent Charlin, and Andrea Lodi. Exact combinatorial optimization with graph convolutional neural networks. In *Proceedings of the 33rd Conference on Neural Information Processing Systems (NeurIPS)*, pp. 15554–15566, Vancouver, BC, Canada, 2019.

Ambros Gleixner, Gregor Hendel, Gerald Gamrath, Tobias Achterberg, Michael Bastubbe, Timo Berthold, Philipp M. Christophel, Kati Jarck, Thorsten Koch, Jeff Linderoth, Marco Lübbecke, Hans D. Mittelmann, Derya Ozyurt, Ted K. Ralphs, Domenico Salvagnin, and Yuji Shinano. MIPLIB 2017: Data-driven compilation of the 6th mixed-integer programming library. *Mathematical Programming Computation*, 13(3):443–490, 2021.

Qingyu Han, Linxin Yang, Qian Chen, Xiang Zhou, Dong Zhang, Akang Wang, Ruoyu Sun, and Xiaodong Luo. A GNN-guided predict-and-search framework for mixed-integer linear programming. In *Proceedings of the 11th International Conference on Learning Representations (ICLR)*, pp. 25380–25394, Kigali, Rwanda, 2023.

Karla L. Hoffman and Manfred Padberg. Solving airline crew scheduling problems by branch-and-cut. *Management Science*, 39(6):657–682, 1993.

Christopher Hojny, Mathieu Besançon, Ksenia Bestuzheva, Sander Borst, João Dionísio, Johannes Ehls, Leon Eifler, Mohammed Ghannam, Ambros Gleixner, Adrian Göß, Alexander Hoen, Jacob von Holly-Ponientzietz, Rolf van der Hulst, Dominik Kamp, Thorsten Koch, Kevin Kofler, Jurgen Lentz, Marco Lübbecke, Stephen J. Maher, Paul Matti Meinhold, Gioni Mexi, Til Mohr, Erik Mühmer, Krunal Kishor Patel, Marc E. Pfetsch, Sebastian Pokutta, Chantal Reinartz Groba, Felipe Serrano, Yuji Shinano, Mark Turner, Stefan Vigerske, Matthias Walter, Dieter Weninger, and Liding Xu. The SCIP optimization suite 10.0. *arXiv:2511.18580*, 2025.

Zhinan Hou, Xingchen Li, Yankai Zhang, Tianxun Li, and Keyou You. LLM4Branch: Large language model for discovering efficient branching policies of integer programs. *arXiv:2605.10401*, 2026.

Qi Huangfu and Julian A. J. Hall. Parallelizing the dual revised simplex method. *Mathematical Programming Computation*, 10(1):119–142, 2018.

Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. Automated configuration of mixed integer programming solvers. In *Proceedings of the 7th International Conference on Integration of Artificial Intelligence and Operations Research Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR)*, pp. 186–202, Bologna, Italy, 2010.

Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In *Proceedings of the 5th International Conference on Learning and Intelligent Optimization (LION)*, pp. 507–523, Rome, Italy, 2011.

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

Bernard Knueven, James Ostrowski, and Jean-Paul Watson. On mixed-integer programming formulations for the unit commitment problem. *INFORMS Journal on Computing*, 32(4):857–876, 2020.

Levente Kocsis and Csaba Szepesvári. Bandit based Monte-Carlo planning. In *Proceedings of the 17th European Conference on Machine Learning (ECML)*, pp. 282–293, Berlin, Germany, 2006.

Connor Lawless, Yingxi Li, Anders Wikum, Madeleine Udell, and Ellen Vitercik. LLMs for cold-start cutting plane separator configuration. In *Proceedings of the 22nd International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research (CPAIOR), Part II*, pp. 51–69, Melbourne, VIC, Australia, 2025.

Sirui Li, Wenbin Ouyang, Max B. Paulus, and Cathy Wu. Learning to configure separators in branch-and-cut. In *Proceedings of the 37th Conference on Neural Information Processing Systems (NeurIPS)*, pp. 60021–60034, New Orleans, LA, 2023.

Siyuan Li, Yifan Yu, Zhihao Zhang, Mengjing Chen, Fangzhou Zhu, Tao Zhong, Peng Liu, and Jianye Hao. Collab-Solver: Collaborative solving policy learning for mixed-integer linear programming. *arXiv:2508.03030*, 2025.

Marius Lindauer, Katharina Eggensperger, Matthias Feurer, André Biedenkapp, Difan Deng, Carolin Benjamins, Tim Ruhkopf, René Sass, and Frank Hutter. SMAC3: A versatile Bayesian optimization package for hyperparameter optimization. *Journal of Machine Learning Research*, 23(54):1–9, 2022.

Fei Liu, Xialiang Tong, Mingxuan Yuan, Xi Lin, Fu Luo, Zhenkun Wang, Zhichao Lu, and Qingfu Zhang. Evolution of heuristics: Towards efficient automatic algorithm design using large language model. In *Proceedings of the 41st International Conference on Machine Learning (ICML)*, pp. 32201–32223, Vienna, Austria, 2024.

Yidong Luo, Xuemin Chen, Chenguang Wang, Fangzhou Zhu, Tao Zhong, and Tianshu Yu. GRIMIP: A general framework for instance-specific configuration of MIP solvers using LLMs. *arXiv:2606.23299*, 2026.

Stephen Maher, Matthias Miltenberger, João Pedro Pedroso, Daniel Rehfeldt, Robert Schwarz, and Felipe Serrano. PySCIPOpt: Mathematical programming in Python with the SCIP optimization suite. In *Proceedings of the 5th International Congress on Mathematical Software (ICMS)*, pp. 301–307, Berlin, Germany, 2016.

Vinod Nair, Sergey Bartunov, Felix Gimeno, Ingrid von Glehn, Pawel Lichocki, Ivan Lobov, Brendan O'Donoghue, Nicolas Sonnerat, Christian Tjandraatmadja, Pengming Wang, Ravichandra Addanki, Tharindi Hapuarachchi, Thomas Keck, James Keeling, Pushmeet Kohli, Ira Ktena, Yujia Li, Oriol Vinyals, and Yori Zwols. Solving mixed integer programs using neural networks. *arXiv:2012.13349*, 2020.

Jinbiao Nie, Kewei Feng, Xiaoyuan Zhang, Shan Yin, Zizhuo Wang, and Bin Dong. MILP-Evo: Closed-loop fully automatic design of MILP solvers. *arXiv:2607.18252*, 2026.

Alexander Novikov, Ngân Vũ, Marvin Eisenberger, Emilien Dupont, Po-Sen Huang, Adam Zsolt Wagner, Sergey Shirobokov, Borislav Kozlovskii, Francisco J. R. Ruiz, Abbas Mehrabian, M. Pawan Kumar, Abigail See, Swarat Chaudhuri, George Holland, Alex Davies, Sebastian Nowozin, Pushmeet Kohli, and Matej Balog. AlphaEvolve: A coding agent for scientific and algorithmic discovery. *arXiv:2506.13131*, 2025.

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 *Proceedings of the 39th International Conference on Machine Learning (ICML)*, pp. 17584–17600, Baltimore, MD, 2022.

Stephen Robertson and Hugo Zaragoza. The probabilistic relevance framework: BM25 and beyond. *Foundations and Trends in Information Retrieval*, 3(4):333–389, 2009.

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

Jasper Snoek, Hugo Larochelle, and Ryan P. Adams. Practical Bayesian optimization of machine learning algorithms. In *Proceedings of the 26th Conference on Neural Information Processing Systems (NIPS)*, pp. 2951–2959, Lake Tahoe, NV, 2012.

Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0: Fundamental algorithms for scientific computing in Python. *Nature Methods*, 17(3):261–272, 2020.

Zhihai Wang, Xijun Li, Jie Wang, Yufei Kuang, Mingxuan Yuan, Jia Zeng, Yongdong Zhang, and Feng Wu. Learning cut selection for mixed-integer linear programming via hierarchical sequence model. In *Proceedings of the 11th International Conference on Learning Representations (ICLR)*, pp. 16274–16305, Kigali, Rwanda, 2023.

Lin Xu, Holger H. Hoos, and Kevin Leyton-Brown. Hydra: Automatically configuring algorithms for portfolio-based selection. In *Proceedings of the 24th AAAI Conference on Artificial Intelligence (AAAI)*, pp. 210–216, Atlanta, GA, 2010.

Lin Xu, Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. Evaluating component solver contributions to portfolio-based algorithm selectors. In *Proceedings of the 15th International Conference on Theory and Applications of Satisfiability Testing (SAT)*, pp. 228–241, Trento, Italy, 2012.

Ke Xue, Jiacheng Xu, Lei Yuan, Miqing Li, Chao Qian, Zongzhang Zhang, and Yang Yu. Multi-agent dynamic algorithm configuration. In *Proceedings of the 36th Conference on Neural Information Processing Systems (NeurIPS)*, pp. 20147–20161, New Orleans, LA, 2022.

Haoran Ye, Jiarui Wang, Zhiguang Cao, Federico Berto, Chuanbo Hua, Haeyeon Kim, Jinkyoo Park, and Guojie Song. ReEvo: Large language models as hyper-heuristics with reflective evolution. In *Proceedings of the 38th Conference on Neural Information Processing Systems (NeurIPS)*, pp. 43571–43608, Vancouver, BC, Canada, 2024.

Huigen Ye, Hua Xu, An Yan, and Yaoyang Cheng. Large language model-driven large neighborhood search for large-scale MILP problems. In *Proceedings of the 42nd International Conference on Machine Learning (ICML)*, pp. 72131–72180, Vancouver, BC, Canada, 2025a.

Mingxuan Ye, Jie Wang, Fangzhou Zhu, Zhihai Wang, Yufei Kuang, Xijun Li, Weilin Luo, Jianye Hao, and Feng Wu. Dynamic configuration for cutting plane separators via reinforcement learning on incremental graph. In *Proceedings of the 39th Conference on Neural Information Processing Systems (NeurIPS)*, pp. 152298–152338, San Diego, CA, 2025b.

Zhi Zheng, Zhuoliang Xie, Zhenkun Wang, and Bryan Hooi. Monte Carlo tree search for comprehensive exploration in LLM-based automatic heuristic design. In *Proceedings of the 42nd International Conference on Machine Learning (ICML)*, pp. 78338–78373, Vancouver, BC, Canada, 2025.

A Experimental Details

The experiments share the solver limits and metric definitions below. Table 4 distinguishes their design targets, search budgets, and evaluation sets; the following subsections provide generation settings, splits, baseline procedures, and ablation controls.

Table 4: Overview of experimental protocols. Final performance entries average four held-out solver seeds within each instance. Search curves use the two search seeds; component allocation summarizes proposal histories.

Study	Design targets	Design budget	Evaluation
Solver performance	One target at a time; 250 generated instances and 240 MIPLIB instances	100 trials per target; neural fine-tuning uses 10 epochs per trial	Selected strategy on each target
Search efficiency	Six displayed MIPLIB instances	100 candidate evaluations for CoMILP, SMAC-I, GP-BO, and GRIMIP	Best-so-far PDI on the two search seeds
Ablations	25 targets per family at the ablation sizes below	100 candidate evaluations; matched LLM token budgets	Selected strategy on each target
Portfolio reuse	200 training and 100 validation instances per family at the sizes in Appendix A.3	Five Hydra rounds; 60 update attempts per round	100 unseen instances per family; probe and solve share 300 s
Component allocation	Main instance-specific experiments	Component updates during search	Within-instance proportions, then family means

A.1 Hardware and Solver Settings

Table 5 gives the solver, model, and search settings (Maher et al., 2016; Achterberg, 2009). Time, node, randomization, and numerical-tolerance controls are fixed by the evaluator. The 330-second process timeout allows result collection and cleanup after the 300-second solving limit. Strategy evaluations use fresh solver models and common evaluation seeds.

A.2 Generated Instances and MIPLIB

SC, MIS, CA, and CFL use the generator families of Gasse et al. (2019): random set-cover matrices, preferential-attachment graphs, combinatorial auctions, and capacitated facility-location models, respectively. MK follows the multiple-knapsack benchmark used by HEM (Wang et al., 2023), with binary item–knapsack assignments, at most one assignment per item, and a capacity constraint for each knapsack. We set the instance sizes and generation parameters as follows.

Table 6 specifies two instance scales for each family, with 25 instances at each scale and 250 instances overall. Dimensions refer to the formulations before presolve; result-panel headings instead name the generating quantities explicitly. Thus, 600 items and 16 knapsacks give 9,600 assignment variables and 616 constraints, while 600 items and 4,000 bids give 4,000 auction variables and 600 item constraints. For multiple knapsack, variables represent item–knapsack assignments. Capacitated facility location has $n_{\text{fac}}n_{\text{cust}} + n_{\text{fac}}$ assignment and opening variables, with one demand constraint per customer and one capacity constraint per facility. MIPLIB 2017 contributes its 240-instance benchmark set to the instance-specific comparison (Gleixner et al., 2021).

A.3 Training, Validation, and Test Splits

Training supplies the objectives for family-level configuration and policy learning, and validation selects model settings and the portfolio selector. All five-family methods share the same 100 unseen

Table 5: Environment and search settings.

Setting	Value
SCIP / PySCIPOpt / Python	10.0.1 / 6.1.0 / 3.13.13
CPU	Intel Xeon Gold 6530 (64 physical cores)
Host memory / operating system	503 GiB / Ubuntu 24.04.4 LTS
Training GPUs	NVIDIA GeForce RTX 4090
LP backend	SoPlex (SCIP default)
Solver threads / node limit	1 / 10^6
Solving limit / process timeout	300 s / 330 s
LLM	`gpt-5.6-terra`
Temperature	0
Trials per target (main comparison)	100
Search seeds	2
Held-out evaluation seeds	4
Neural fine-tuning epochs per trial	10
UCT coefficient c	1.2
Widening coefficient k / exponent α	1.5 / 0.5
Parent–child evaluation	paired comparison on two common search seeds
Retrieval index and settings	SCIP 10.0.1 knowledge base; at most 4500 characters
Portfolio rounds / updates per round	5 / 60
Portfolio evaluation budget	5 s default probe + 295 s selected solve
Selector candidates	Pairwise cost-sensitive forests; multiclass RF
Selector trees / minimum leaf size	300 / 4
Selector feature subsampling / random seed	Square root / fixed

Table 6: Generated instance specifications. SC, MIS, CA, and CFL follow Gasse et al. (2019); MK follows Wang et al. (2023). Each row contributes 25 instances to the instance-specific comparison.

Family	Variables	Constraints	Generation parameters
SC	1,000	900	1,000 columns; 900 rows; density 0.05
SC	1,000	1,700	1,000 columns; 1,700 rows; density 0.05
MIS	1,100	4,384	1,100 vertices; attachment parameter 4
MIS	1,500	5,984	1,500 vertices; attachment parameter 4
MK	1,600	208	200 items; 8 knapsacks; capacity ratio 0.12
MK	9,600	616	600 items; 16 knapsacks; capacity ratio 0.06
CA	2,000	400	400 items; 2,000 bids
CA	4,000	600	600 items; 4,000 bids
CFL	19,280	320	80 facilities; 240 customers
CFL	40,100	500	100 facilities; 400 customers

test instances per family (Table 7). All three splits use SC with 1,000 columns and 900 rows, MIS with 1,100 vertices, MK with 200 items and 8 knapsacks, CA with 400 items and 2,000 bids, and CFL with 80 facilities and 240 customers. The remaining generation parameters are given in Table 6. Training and validation instances have distinct identities and generation seeds from the shared test set.

Table 7: Instance counts for family-level learning and evaluation. All methods share the test instances within each column.

Split	SC	MIS	MK	CA	CFL
Training	200	200	200	200	200
Validation	100	100	100	100	100
Test	100	100	100	100	100

Generalization uses two common search seeds for training and validation, and four held-out evaluation seeds for testing, with the solver limits in Section 4.1.

A.4 Search Budgets and Search Spaces

Common evaluation settings. All adaptation methods receive 100 trials per target in the main comparison. Candidate performance is measured by mean PDI on two common search seeds, and the selected strategy is evaluated on four held-out seeds. Each solve uses one CPU thread, a 300-second horizon, and a 10^6-node limit. The comparison therefore measures the quality of the strategies obtained with the same trial budget and solver resources. Ablations additionally match the LLM token budget and solver evaluations.

Search spaces. The configuration baselines optimize initial SCIP settings using their published search procedures. SMAC-I, SMAC-P, and GP-BO share the seven-dimensional SCIP configuration space in Table 8, which maps the solver controls used by GRIMIP (Luo et al., 2026) to SCIP. This compact space concentrates the fixed evaluation budget on broad solver controls: global surrogate-based search becomes more difficult as dimension grows, particularly for GP-BO (Eriksson et al., 2019). The shared compact space gives both optimizers a focused search under the 100-trial budget. GRIMIP selects a subspace from the same 136 initial SCIP parameters available to CoMILP (Table 13), then searches that selected subspace; ColdStart proposes separator configurations. HEM, Collab, DynSep, LLM4Branch, and MILP-Evo optimize their respective component policies. Table 10 summarizes these design choices and their references. CoMILP jointly evolves its 136 initial parameters, dynamic rules acting on 38 of those parameters, and cut-selection and branching scores, as specified in Appendix B.

The round limits set `separating/maxrounds`, `separating/maxroundsroot`, and `presolving/maxrounds`. The LP control sets `lp/initalgorithm` and `lp/resolvealgorithm`; barrier initialization uses dual simplex for reoptimization. Selecting a branching rule sets its priority to 10^6. The heuristic control sets invocation frequency to -1 (off) or 1 (aggressive) for `coefdiving`, `crossover`, `feaspump`, `fracdiving`, `guideddiving`, `linesearchdiving`, `objpscostdiving`, `pscostdiving`, `rens`, `rins`, `rootsoldiving`, `rounding`, `shifting`, `undercover`, `veclendiving`, `alns`, `mutation`, `reoptsols`, and `trivialnegation`. The search space is the Cartesian product of these choices.

Training and strategy evaluation. Neural baselines first learn family models using the training settings in Table 9. Instance-specific adaptation then uses solver trajectories from the target, with 10 fine-tuning epochs per trial. CoMILP generates and refines strategies between solver evaluations. Its initial reference and 100 candidates require 202 solver runs over the two search seeds, followed by four runs for final evaluation. Reported solving time and PDI measure execution of the selected strategy. The family-training and target-adaptation budgets are specified separately from this solving horizon.

Table 8: SCIP configuration space for SMAC and GP-BO. Each dimension includes a default option, which leaves the corresponding setting unchanged. The emphasis preset is applied before the explicit parameter assignments.

Control	Candidate values (in addition to default)
Solver emphasis	Counter, feasibility, optimality
Primal heuristics	Off, aggressive
Tree separation rounds	0, 5, 20
Root separation rounds	0, 50, 200
Presolving rounds	0, 10, 500
LP algorithm	Automatic, primal simplex, dual simplex, barrier
Branching rule	`relpscost`, `pscost`, `fullstrong`, `mostinf`, `inference`, `random`

Portfolio construction and evaluation. Hydra constructs each family portfolio using 200 training instances and five rounds of 60 update attempts; 100 validation instances select the classifier. Training, validation, and testing use the portfolio evaluation procedure in Appendix B.5, with a five-second default probe followed by the selected strategy within the common 300-second horizon.

A.5 Evaluation Metrics

Primal–dual integral. The primal–dual integral measures bound progress throughout solving, as in HEM and DynSep (Wang et al., 2023; Ye et al., 2025b). We use the bounded percentage-gap convention implemented in SCIP. Let $p(t)$ and $d(t)$ denote the global primal and dual bounds, respectively, expressed in the original objective space (Berthold, 2013). Define

$$g_{\mathrm{PD}}(t) = \begin{cases} 0, & \text{if optimality has been proved,} \\ \dfrac{|p(t) - d(t)|}{\max\{|p(t)|, |d(t)|\}}, & \text{if both bounds are finite and } p(t)d(t) > 0, \\ 1, & \text{otherwise.} \end{cases} \tag{8}$$

Thus, $g_{\mathrm{PD}}(t) \in [0, 1]$, including before a feasible solution has been found. The reported integral is

$$\mathrm{PDI}(I, \sigma, \omega) = 100 \int_0^T g_{\mathrm{PD}}(t; I, \sigma, \omega)\, \mathrm{d}t. \tag{9}$$

PDI is measured in percentage-seconds. The gap is zero after optimality is proved; if a run stops at the node limit before T, its final gap is carried forward to T. Integrating the piecewise-constant bound trajectory therefore gives a common-horizon measure for every strategy. The search cost in Equation (4) is $J_I(\sigma) = \mathbb{E}_\omega[\mathrm{PDI}(I, \sigma, \omega)/(100T)]$. This normalization preserves the ordering of strategies while expressing the search objective in $[0, 1]$.

Search objective and selection. The search estimates the normalized PDI objective in Section 3.1 using the common search seeds:

$$\begin{aligned} \widehat{J}_I(\sigma) &= \frac{1}{|\Omega_{\mathrm{search}}|} \sum_{\omega \in \Omega_{\mathrm{search}}} \left[\frac{1}{T} \int_0^T g_{\mathrm{PD}}(t; I, \sigma, \omega)\, \mathrm{d}t \right], \\ \widehat{F}(\sigma) &= \frac{1}{|D_{\mathrm{tr}}|} \sum_{I \in D_{\mathrm{tr}}} \widehat{J}_I(\sigma). \end{aligned} \tag{10}$$

Parent and child use the same instances and seeds; an identical cached parent evaluation can be reused. Tree search backs up $1 - \widehat{F}$ and returns the strategy with the lowest estimated objective. The selected strategy is fixed before evaluation on four held-out solver seeds.

Solving time and aggregation. Capped solving time is the proof time for a completed solve and T otherwise. For metric $y_{i\omega}$ on instance i and evaluation seed ω, define $\bar{y}_i = |\Omega_{\mathrm{eval}}|^{-1} \sum_{\omega \in \Omega_{\mathrm{eval}}} y_{i\omega}$.

A table entry reports

$$\mu_y = \frac{1}{n} \sum_{i=1}^{n} \bar{y}_i. \tag{11}$$

Relative PDI improvement over SCIP-default is

$$\text{Improvement}(m) = 100 \left(1 - \frac{\mu_{\text{PDI},m}}{\mu_{\text{PDI,SCIP}}} \right), \tag{12}$$

using the same instances for both means.

A.6 Baseline Methods

On the generated families, the instance-specific study compares against HEM, Collab, DynSep, SMAC-I, GP-BO, GRIMIP, ColdStart, LLM4Branch, and MILP-Evo. The MIPLIB comparison uses SMAC-I, GP-BO, GRIMIP, ColdStart, LLM4Branch, and MILP-Evo. The generalization study compares against HEM, Collab, DynSep, SMAC-P, and GP-BO. SCIP-default provides the reference solver in both settings.

HEM learns cut-selection decisions, DynSep learns dynamic separator configuration, and Collab jointly learns collaborating cut-selection and branching policies (Wang et al., 2023; Ye et al., 2025b; Li et al., 2025).

Neural baseline training and fine-tuning. We train a separate HEM, Collab, and DynSep model for each generated problem family, using the training and validation splits in Table 7. HEM uses an LSTM pointer network to order cuts and a higher-level policy to determine the selection ratio. Collab combines a hierarchical cut-selection policy with a bipartite graph convolutional branching policy; after component pretraining, solver feedback jointly improves the two policies. DynSep combines a graph encoder with a decoder-only Transformer and learns separator activation decisions with proximal policy optimization (PPO) (Schulman et al., 2017). Table 9 gives their training settings.

For the instance-specific comparison, each target starts from a fresh copy of its family checkpoint. We then fine-tune the model on solver trajectories from that target using the baseline's policy-learning objective: hierarchical cut selection for HEM, joint cut-selection and branching updates for Collab, and PPO updates for DynSep. Each target receives 100 trials, with 10 fine-tuning epochs per trial. Adaptation and checkpoint selection use the two search seeds; the selected model is evaluated with the four held-out seeds. For generalization, the family checkpoint selected on validation instances is applied directly to the shared unseen test instances.

Table 9: Neural baseline settings for family-level pretraining. All three methods use Adam (Kingma & Ba, 2014). Instance-specific fine-tuning starts from the resulting family checkpoints.

Method	Network	Training configuration
HEM	128-dimensional LSTM hidden state; 13 cut features	100 epochs; batch size 16; learning rates 10^{-4} (pointer) and 5×10^{-4} (selection ratio)
Collab	128-dimensional cut and graph embeddings; hierarchical cut policy and graph branching policy	100 cut-policy and 100 branching-policy pretraining epochs; 35 joint-training epochs; learning rates 10^{-4} (cut policy), 10^{-3} (branching), and 10^{-5} (joint training)
DynSep	64-dimensional embeddings; four Transformer layers and four attention heads	200 PPO epochs; 16 sampled instances per epoch; learning rate 10^{-4}; PPO clipping parameter 0.2

SMAC and Gaussian-process Bayesian optimization. SMAC uses a random-forest surrogate to guide sequential model-based algorithm configuration (Hutter et al., 2011; Lindauer et al., 2022). We use SMAC-I to denote optimization of PDI on one target instance and SMAC-P to denote optimization of mean PDI over a family's training instances, following the distinction in Luo et al.

(2026). SMAC3 is the Python framework for SMAC described by Lindauer et al. (2022). GP-BO denotes Gaussian-process Bayesian optimization (Snoek et al., 2012): a Gaussian-process surrogate predicts configuration performance, and an acquisition function guides the next evaluation. Its instance-specific and family-trained variants use the corresponding PDI objectives.

Both optimizers use SMAC3 version 2.4. SMAC represents the choices in Table 8 as categorical variables and uses the algorithm-configuration facade. GP-BO encodes each choice by its integer index and uses a Gaussian process with a Matérn-$5/2$ kernel and a fixed white-noise term of 10^{-8}. It uses the hyperparameter-optimization facade with Sobol initialization and expected improvement. The initial design and acquisition settings follow the respective facade defaults.

LLM-based configuration and policy evolution. GRIMIP uses LLM reasoning to select a subspace of the 136 initial parameters and search configurations within it; ColdStart uses LLM-generated separator configurations (Luo et al., 2026; Lawless et al., 2025). LLM4Branch evolves branching programs, and MILP-Evo jointly evolves cut-selection and branching policies (Hou et al., 2026; Nie et al., 2026).

The configuration and evolution baselines follow the common trial budget and evaluation settings in Appendix A.4. Instance-specific search uses the target instance; family-level learning uses the training and validation splits. Table 10 summarizes the methods used in each comparison.

Table 10: Baseline mechanisms and comparison settings. I denotes the instance-specific study; G denotes the family-trained comparison on unseen instances. HEM, Collab, and DynSep are evaluated on the generated problem families.

Method	Setting	Mechanism
SCIP-default (Achterberg, 2009)	I, G	SCIP's default algorithmic settings under the shared resource limits
HEM (Wang et al., 2023)	I, G	Hierarchical cut-selection policy for the number and ordering of cuts
Collab (Li et al., 2025)	I, G	Collaborative learning of cut-selection and branching policies through component pretraining and joint fine-tuning
DynSep (Ye et al., 2025b)	I, G	Reinforcement-learning policy for state-dependent separator configuration
SMAC-I (Lindauer et al., 2022)	I	Random-forest-based configuration search minimizing PDI on the target instance
SMAC-P (Lindauer et al., 2022)	G	Random-forest-based configuration search minimizing mean PDI over family training instances
GP-BO (Snoek et al., 2012)	I, G	Gaussian-process surrogate for initial-configuration search
GRIMIP (Luo et al., 2026)	I	LLM-guided parameter-space selection, candidate generation, and performance-based configuration search
ColdStart (Lawless et al., 2025)	I	LLM generation of separator configurations followed by clustering and evaluation of representative configurations
LLM4Branch (Hou et al., 2026)	I	LLM evolution of branching programs and optimization of their coefficients
MILP-Evo (Nie et al., 2026)	I	Joint evolution of cut-selection and branching rules

A.7 Ablation Settings

The ablations use the instance sizes specified in Appendix A.3, and share the objective, parameter catalog, LLM, solver limits, and evaluation seeds with CoMILP. Each variant receives 100 candidate evaluations with the same LLM token and solver-evaluation budgets, and its selected strategy is fixed before held-out seed evaluation.

Branching only. All updates target branching: its initial parameter settings, dynamic configuration rules, or candidate-scoring rule. Every other component retains its default configuration and behavior. The agent chooses the branching edit type from solver feedback throughout evolution.

No branching. Branching retains its default parameter settings and decision rule. All other components and their supported edit types remain available, and the planning agent selects which component to refine next using solver feedback. Both component ablations retain the full generate–evaluate–reflect loop, tree search, retrieval, and experience memory under the shared budget in Appendix B.1.

Initial configuration only. All updates revise initial parameter settings. SCIP uses its built-in policies under the resulting configuration. Planning, proposal, retrieval, reflection, and tree search remain active.

Policies only. All updates revise dynamic configuration rules or candidate-scoring rules for cut selection and branching. The initial configuration remains SCIP-default, and the policy representation, features, and execution limits match full CoMILP.

No retrieval. Solver-documentation retrieval is disabled. Instance features, solver trajectories, experience memory, and parameter introspection remain available, and the proposal, reflection, and search procedures are unchanged.

No memory. The shared experience memory $\mathcal{M}$ is removed: intervention records and reflection summaries from earlier updates are neither stored nor supplied to subsequent agent calls. Each update uses the selected parent strategy and its solver feedback, and reflection analyzes the current parent–child comparison. Documentation retrieval, tree search, the three agent roles, and the continuous generate–evaluate–reflect loop remain active under the same candidate-evaluation and LLM token budgets.

Single-agent evolution. One LLM agent performs component and edit-type selection, candidate generation, and feedback analysis within a persistent conversation. At each iteration, it uses the selected parent strategy and accumulated experience to propose a component update. The solver evaluates the candidate on the common search seeds and returns its objective value and execution trajectory. The same agent analyzes this feedback, updates the experience memory, and uses the resulting information in subsequent proposals. The strategy representation, feedback-driven choice of edit type, tree search, retrieval access, and final strategy-selection rule match CoMILP. Both methods receive 100 candidate evaluations, the same solver evaluations per candidate, and the same total LLM token budget. The single agent shares this token budget across proposal and feedback analysis, replacing the separate planning, proposal, and reflection roles.

B Strategy Search and Policy Execution

This appendix specifies how the component-level updates in Section 3 refine and execute a complete strategy. Each child inherits all settings and policies outside the selected component block, so its evaluation captures the effect of the update in the configuration in which it will operate.

B.1 Configuration and Policy Updates

An initial-configuration update uses a selected space of six to eight parameters, which may span several components, from the 136-parameter catalog summarized in Table 11. A policy update revises either the selected dynamic configuration rules or a candidate-scoring rule for cut selection or branching. Dynamic configuration rules act on a subset of 38 parameters. All other settings and policies are inherited from the parent. The ten groups in Table 11 also define the proposal categories in Figure 4. Cut management and separator frequency form two parts of separation control: the former sets global effort and cut budgets, and the latter sets invocation frequencies of individual separators. Cut selection acts on the resulting candidate pool. Conflict analysis derives constraints from infeasibility, while propagation tightens variable bounds. Branching chooses a variable; node

selection chooses an open subproblem. LP controls govern relaxation solving. Each group therefore identifies a distinct set of solver decisions.

Each search in the main comparison uses 100 candidate evaluations. At every iteration, the planner selects a component block and an edit type using the complete parent strategy, solver trajectories, and accumulated experience. The proposal is applied to the parent, the solver evaluates the resulting strategy, and reflection records the feedback for subsequent updates. Initial-configuration, dynamic-configuration, and scoring-rule edits share the same tree and memory. Their sequence is determined by the agents as the strategy evolves.

Table 11: Parameter space for configuration updates. The 38 parameters used by dynamic configuration policies are a subset of the 136 initial-configuration parameters. Cut-selection and branching scores are additional policy updates defined in Appendix B.2.

Component	Initial	Dynamic	Controls
Presolve	14	0	Reductions and round limits
Cut management	12	5	Separation rounds, cut limits, and thresholds
Separator frequency	17	6	Separator activation frequencies
Cut selection	10	0	Selection weights, priorities, and thresholds
Primal heuristics	26	8	Invocation frequencies and subproblem budgets
Branching	17	4	Candidate evaluation and branching-rule controls
Conflict analysis	15	5	Activation and conflict limits
Propagation	13	4	Round limits and propagation frequencies
LP	6	2	Solve frequency and reoptimization controls
Node selection	6	4	Node-selection and plunging controls
Total	**136**	**38**	

Table 12: Representative SCIP parameters grouped by component. Each entry in the right column is appended to the prefix in the middle column. Counts for the full configuration space are given in Table 11.

Component	Prefix	Parameter examples
Presolve	`presolving/`	`maxrounds`, `maxrestarts`
Cut management	`separating/`	`maxrounds`, `maxcuts`, `maxcutsroot`
Separator frequency	`separating/`	`gomory/freq`, `cmir/freq`, `flowcover/freq`
Cut selection	`cutselection/hybrid/`	`efficacyweight`, `objparalweight`
Primal heuristics	`heuristics/`	`rins/freq`, `rens/freq`, `feaspump/freq`
Branching	`branching/relpscost/`	`maxdepth`, `maxlookahead`, `inititer`
Conflict analysis	`conflict/`	`enable`, `useprop`, `maxstoresize`
Propagation	`propagating/`	`maxrounds`, `obbt/freq`, `redcost/freq`
LP	`lp/`	`solvefreq`, `resolveiterfac`
Node selection	`nodeselection/estimate/`	`bestnodefreq`, `minplungedepth`

Complete admissible domains. Tables 13 and 14 enumerate the complete 136/38 parameter space. Initial assignments use the listed finite sets. Dynamic assignments must satisfy the listed type, interval, and execution scope; integer scaling is rounded and checked against the same domain. Frequency -1 disables a plugin, 0 invokes it only at the root, and positive values set periodic invocation. Eligibility checks within each plugin still apply. The catalog specifies admissible interventions.

Table 13: Complete initial-configuration catalog: all 136 parameters and their admissible search values. T/F denote Boolean true/false. Numeric braces denote a finite set, not a continuous interval.

Parameter	Admissible values
Presolve	
`presolving/maxrounds`	$\{-1, 0, 1, 2, 3, 5, 10, 20, 50, 100\}$
`presolving/abortfac`	$\{0.0, 0.0002, 0.0008, 0.002, 0.01\}$
`presolving/donotaggr`	$\{\mathrm{F}, \mathrm{T}\}$
`presolving/donotmultaggr`	$\{\mathrm{F}, \mathrm{T}\}$
`presolving/domcol/maxrounds`	$\{-1, 0, 1, 2, 3, 5, 10, 20, 50, 100\}$
`presolving/domcol/predbndstr`	$\{\mathrm{F}, \mathrm{T}\}$
`presolving/dualcomp/maxrounds`	$\{-1, 0, 1, 2, 3, 5, 10, 20, 50, 100\}$
`presolving/dualinfer/maxrounds`	$\{-1, 0, 1, 2, 3, 5, 10, 20, 50, 100\}$
`presolving/dualsparsify/maxrounds`	$\{-1, 0, 1, 2, 3, 5, 10, 20, 50, 100\}$
`presolving/gateextraction/maxrounds`	$\{-1, 0, 1, 2, 3, 5, 10, 20, 50, 100\}$
`presolving/implics/maxrounds`	$\{-1, 0, 1, 2, 3, 5, 10, 20, 50, 100\}$
`presolving/redvub/maxrounds`	$\{-1, 0, 1, 2, 3, 5, 10, 20, 50, 100\}$
`presolving/sparsify/maxrounds`	$\{-1, 0, 1, 2, 3, 5, 10, 20, 50, 100\}$
`presolving/tworowbnd/maxrounds`	$\{-1, 0, 1, 2, 3, 5, 10, 20, 50, 100\}$
Cut management	
`separating/maxrounds`	$\{-1, 0, 1, 2, 3, 5, 10, 20, 50, 100\}$
`separating/maxroundsroot`	$\{-1, 0, 1, 2, 3, 5, 10, 20, 50, 100\}$
`separating/maxcuts`	$\{0, 32, 50, 100, 200, 500\}$
`separating/maxcutsroot`	$\{0, 50, 100, 300, 500, 1000, 2000, 4000\}$
`separating/maxcutsgenfactor`	$\{0.5, 1.0, 2.0, 4.0\}$
`separating/maxcutsrootgenfactor`	$\{0.5, 1.0, 2.0, 4.0\}$
`separating/maxstallrounds`	$\{-1, 0, 1, 2, 3, 5, 10, 20, 50, 100\}$
`separating/maxstallroundsroot`	$\{-1, 0, 1, 2, 3, 5, 10, 20, 50, 100\}$
`separating/maxbounddist`	$\{0.0, 0.25, 0.5, 0.75, 1.0\}$
`separating/minefficacy`	$\{10^{-5}, 5 \times 10^{-5}, 0.0001, 0.0005, 0.001\}$
`separating/minefficacyroot`	$\{10^{-5}, 5 \times 10^{-5}, 0.0001, 0.0005, 0.001\}$
`separating/poolfreq`	$\{-1, 0, 1, 2, 5, 10, 20, 50\}$
Separator frequency	
`separating/gomory/freq`	$\{-1, 0, 1, 2, 5, 10, 20, 50\}$
`separating/gomorymi/freq`	$\{-1, 0, 1, 2, 5, 10, 20, 50\}$
`separating/cmir/freq`	$\{-1, 0, 1, 2, 5, 10, 20, 50\}$
`separating/flowcover/freq`	$\{-1, 0, 1, 2, 5, 10, 20, 50\}$
`separating/knapsackcover/freq`	$\{-1, 0, 1, 2, 5, 10, 20, 50\}$
`separating/zerohalf/freq`	$\{-1, 0, 1, 2, 5, 10, 20, 50\}$
`separating/aggregation/freq`	$\{-1, 0, 1, 2, 5, 10, 20, 50\}$
`separating/clique/freq`	$\{-1, 0, 1, 2, 5, 10, 20, 50\}$
`separating/impliedbounds/freq`	$\{-1, 0, 1, 2, 5, 10, 20, 50\}$
`separating/strongcg/freq`	$\{-1, 0, 1, 2, 5, 10, 20, 50\}$
`separating/mixing/freq`	$\{-1, 0, 1, 2, 5, 10, 20, 50\}$
`separating/mcf/freq`	$\{-1, 0, 1, 2, 5, 10, 20, 50\}$
`separating/oddcycle/freq`	$\{-1, 0, 1, 2, 5, 10, 20, 50\}$
`separating/disjunctive/freq`	$\{-1, 0, 1, 2, 5, 10, 20, 50\}$
`separating/rapidlearning/freq`	$\{-1, 0, 1, 2, 5, 10, 20, 50\}$
`separating/rlt/freq`	$\{-1, 0, 1, 2, 5, 10, 20, 50\}$
`separating/flower/freq`	$\{-1, 0, 1, 2, 5, 10, 20, 50\}$
Cut selection	
`cutselection/hybrid/efficacyweight`	$\{0.5, 0.75, 1.0, 1.5, 2.0\}$
`cutselection/hybrid/intsupportweight`	$\{0.0, 0.05, 0.1, 0.25, 0.5\}$
`cutselection/hybrid/objparalweight`	$\{0.0, 0.05, 0.1, 0.25, 0.5\}$
`cutselection/hybrid/minortho`	$\{0.5, 0.7, 0.8, 0.9, 0.95\}$

Parameter	Admissible values
`cutselection/hybrid/minorthoroot`	$\{0.5, 0.7, 0.8, 0.9, 0.95\}$
`cutselection/ensemble/priority`	$\{7000, 8500\}$
`cutselection/ensemble/maxcuts`	$\{50, 100, 200, 500\}$
`cutselection/dynamic/priority`	$\{7000, 8500\}$
`cutselection/dynamic/mingain`	$\{0.0, 0.005, 0.01, 0.025, 0.05\}$
`cutselection/dynamic/minortho`	$\{0.5, 0.7, 0.8, 0.9, 0.95\}$
Primal heuristics	
`heuristics/rins/freq`	$\{-1, 0, 1, 2, 5, 10, 15, 20, 50\}$
`heuristics/rens/freq`	$\{-1, 0, 1, 2, 5, 10, 20, 50\}$
`heuristics/alns/freq`	$\{-1, 0, 1, 2, 5, 10, 20, 50\}$
`heuristics/feaspump/freq`	$\{-1, 0, 1, 2, 5, 10, 20, 50\}$
`heuristics/rounding/freq`	$\{-1, 0, 1, 2, 5, 10, 20, 50\}$
`heuristics/oneopt/freq`	$\{-1, 0, 1, 2, 5, 10, 20, 50\}$
`heuristics/localbranching/freq`	$\{-1, 0, 1, 2, 5, 10, 20, 50\}$
`heuristics/crossover/freq`	$\{-1, 0, 1, 2, 5, 10, 15, 20, 50\}$
`heuristics/dins/freq`	$\{-1, 0, 1, 2, 5, 10, 20, 50\}$
`heuristics/gins/freq`	$\{-1, 0, 1, 2, 5, 10, 20, 50\}$
`heuristics/mutation/freq`	$\{-1, 0, 1, 2, 5, 10, 20, 50\}$
`heuristics/proximity/freq`	$\{-1, 0, 1, 2, 5, 10, 20, 50\}$
`heuristics/trustregion/freq`	$\{-1, 0, 1, 2, 5, 10, 20, 50\}$
`heuristics/adaptivediving/freq`	$\{-1, 0, 1, 2, 5, 10, 20, 50\}$
`heuristics/conflictdiving/freq`	$\{-1, 0, 1, 2, 5, 10, 20, 50\}$
`heuristics/farkasdiving/freq`	$\{-1, 0, 1, 2, 5, 10, 20, 50\}$
`heuristics/fracdiving/freq`	$\{-1, 0, 1, 2, 5, 10, 20, 50\}$
`heuristics/guideddiving/freq`	$\{-1, 0, 1, 2, 5, 10, 20, 50\}$
`heuristics/objpscostdiving/freq`	$\{-1, 0, 1, 2, 5, 10, 20, 50\}$
`heuristics/pscostdiving/freq`	$\{-1, 0, 1, 2, 5, 10, 20, 50\}$
`heuristics/rootsoldiving/freq`	$\{-1, 0, 1, 2, 5, 10, 20, 50\}$
`heuristics/lpface/freq`	$\{-1, 0, 1, 2, 5, 10, 15, 20, 50\}$
`heuristics/intshifting/freq`	$\{-1, 0, 1, 2, 5, 10, 20, 50\}$
`heuristics/shifting/freq`	$\{-1, 0, 1, 2, 5, 10, 20, 50\}$
`heuristics/simplerounding/freq`	$\{-1, 0, 1, 2, 5, 10, 20, 50\}$
`heuristics/rens/maxnodes`	$\{50, 100, 250, 500, 1000, 5000\}$
Branching	
`branching/scorefac`	$\{0.05, 0.1, 0.167, 0.25, 0.4\}$
`branching/preferbinary`	$\{\mathrm{F}, \mathrm{T}\}$
`branching/relpscost/maxdepth`	$\{-1, 0, 5, 10, 20, 50, 100\}$
`branching/relpscost/maxlookahead`	$\{1, 3, 5, 9, 15, 30\}$
`branching/relpscost/inititer`	$\{0, 10, 50, 100, 500, 1000\}$
`branching/relpscost/initcand`	$\{10, 25, 50, 100, 200\}$
`branching/relpscost/maxreliable`	$\{2.0, 3.0, 5.0, 8.0, 12.0\}$
`branching/relpscost/minreliable`	$\{0.5, 1.0, 2.0, 3.0\}$
`branching/relpscost/usehyptestforrelia bility`	$\{\mathrm{F}, \mathrm{T}\}$
`branching/relpscost/dynamiclookahead`	$\{\mathrm{F}, \mathrm{T}\}$
`branching/relpscost/dynamicweights`	$\{\mathrm{F}, \mathrm{T}\}$
`branching/relpscost/degeneracyaware`	$\{0, 1, 2\}$
`branching/relpscost/discountfactor`	$\{0.0, 0.1, 0.2, 0.4, 0.6\}$
`branching/relpscost/filtercandssym`	$\{\mathrm{F}, \mathrm{T}\}$
`branching/relpscost/maxbdchgs`	$\{-1, 1, 3, 5, 10, 20\}$
`branching/relpscost/maxbounddist`	$\{0.25, 0.5, 0.75, 1.0\}$
`branching/relpscost/sbiterquot`	$\{0.1, 0.25, 0.5, 0.75, 1.0\}$
Conflict analysis	
`conflict/enable`	$\{\mathrm{F}, \mathrm{T}\}$
`conflict/useprop`	$\{\mathrm{F}, \mathrm{T}\}$
`conflict/usesb`	$\{\mathrm{F}, \mathrm{T}\}$
`conflict/usepseudo`	$\{\mathrm{F}, \mathrm{T}\}$
`conflict/usegenres`	$\{\mathrm{F}, \mathrm{T}\}$
`conflict/allowlocal`	$\{\mathrm{F}, \mathrm{T}\}$
`conflict/separate`	$\{\mathrm{F}, \mathrm{T}\}$
`conflict/repropagate`	$\{\mathrm{F}, \mathrm{T}\}$
`conflict/maxstoresize`	$\{0, 1000, 5000, 10000, 20000, 50000\}$
`conflict/reconvlevels`	$\{-1, 0, 1, 2, 5, 10\}$

Parameter	Admissible values
conflict/fuiplevels	$\{-1, 0, 1, 2, 5, 10\}$
conflict/maxconss	$\{-1, 0, 5, 10, 20, 50\}$
conflict/maxlploops	$\{-1, 0, 1, 2, 4, 8\}$
conflict/lpiterations	$\{-1, 5, 10, 20, 50, 100\}$
conflict/restartnum	$\{0, 100, 500, 1000, 5000\}$
Propagation	
propagating/maxrounds	$\{-1, 0, 1, 2, 3, 5, 10, 20, 50, 100\}$
propagating/maxroundsroot	$\{-1, 0, 1, 2, 3, 5, 10, 20, 50, 100, 1000\}$
propagating/abortoncutoff	$\{\mathrm{F}, \mathrm{T}\}$
propagating/obbt/freq	$\{-1, 0, 1, 2, 5, 10, 20, 50\}$
propagating/pseudoobj/freq	$\{-1, 0, 1, 2, 5, 10, 20, 50\}$
propagating/redcost/freq	$\{-1, 0, 1, 2, 5, 10, 20, 50\}$
propagating/rootredcost/freq	$\{-1, 0, 1, 2, 5, 10, 20, 50\}$
propagating/probing/freq	$\{-1, 0, 1, 2, 5, 10, 20, 50\}$
propagating/genvbounds/freq	$\{-1, 0, 1, 2, 5, 10, 20, 50\}$
propagating/vbounds/freq	$\{-1, 0, 1, 2, 5, 10, 20, 50\}$
propagating/dualfix/freq	$\{-1, 0, 1, 2, 5, 10, 20, 50\}$
propagating/symmetry/freq	$\{-1, 0, 1, 2, 5, 10, 20, 50\}$
propagating/probing/maxdepth	$\{-1, 0, 5, 10, 20, 50, 100\}$
LP	
lp/solvefreq	$\{-1, 0, 1, 2, 5, 10, 20, 50\}$
lp/solvedepth	$\{-1, 0, 5, 10, 50, 100\}$
lp/resolveiterfac	$\{-1.0, 0.25, 0.5, 1.0, 2.0\}$
lp/resolveitermin	$\{100, 250, 500, 1000, 2000, 5000\}$
lp/fastmip	$\{0, 1\}$
lp/presolving	$\{\mathrm{F}, \mathrm{T}\}$
Node selection	
nodeselection/estimate/bestnodefreq	$\{0, 5, 10, 50, 100, 500, 1000\}$
nodeselection/estimate/minplungedepth	$\{-1, 0, 5, 10, 20, 50, 100\}$
nodeselection/estimate/maxplungedepth	$\{-1, 0, 5, 10, 20, 50, 100\}$
nodeselection/estimate/maxplungequot	$\{0.05, 0.1, 0.25, 0.5, 1.0\}$
nodeselection/hybridestim/bestnodefreq	$\{0, 10, 100, 500, 1000, 2000\}$
nodeselection/hybridestim/estimweight	$\{0.0, 0.05, 0.1, 0.25, 0.5\}$

Table 14: Complete dynamic-configuration catalog: all 38 parameters, admissible assigned values, and execution scopes. Integer and real intervals include their endpoints. These names are a subset of the initial catalog, but dynamic assignments use the domains listed here.

Parameter	Domain	Scope
Cut management		
separating/maxcuts	$[0, 10000] \cap \mathbb{Z}$	root/tree
separating/maxcutsroot	$[0, 10000] \cap \mathbb{Z}$	root
separating/maxrounds	$[-1, 100] \cap \mathbb{Z}$	root/tree
separating/maxroundsroot	$[-1, 100] \cap \mathbb{Z}$	root
separating/maxstallrounds	$[-1, 100] \cap \mathbb{Z}$	root/tree
Separator frequency		
separating/aggregation/freq	$[-1, 10000] \cap \mathbb{Z}$	root/tree
separating/cmir/freq	$[-1, 10000] \cap \mathbb{Z}$	root/tree
separating/flowcover/freq	$[-1, 10000] \cap \mathbb{Z}$	root/tree
separating/gomory/freq	$[-1, 10000] \cap \mathbb{Z}$	root/tree
separating/knapsackcover/freq	$[-1, 10000] \cap \mathbb{Z}$	root/tree
separating/zerohalf/freq	$[-1, 10000] \cap \mathbb{Z}$	root/tree
Primal heuristics		
heuristics/alns/freq	$[-1, 10000] \cap \mathbb{Z}$	root/tree
heuristics/feaspump/freq	$[-1, 10000] \cap \mathbb{Z}$	root/tree
heuristics/localbranching/freq	$[-1, 10000] \cap \mathbb{Z}$	root/tree
heuristics/oneopt/freq	$[-1, 10000] \cap \mathbb{Z}$	root/tree
heuristics/rens/freq	$[-1, 10000] \cap \mathbb{Z}$	root/tree
heuristics/rens/maxnodes	$[1, 1000000000] \cap \mathbb{Z}$	root

Parameter	Domain	Scope
`heuristics/rins/freq`	$[-1, 10000] \cap \mathbb{Z}$	root/tree
`heuristics/rounding/freq`	$[-1, 10000] \cap \mathbb{Z}$	root/tree
Branching		
`branching/relpscost/inititer`	$[0, 1000000] \cap \mathbb{Z}$	root/tree
`branching/relpscost/maxdepth`	$[-1, 100000] \cap \mathbb{Z}$	root/tree
`branching/relpscost/maxlookahead`	$[1, 100] \cap \mathbb{Z}$	root/tree
`branching/relpscost/usehyptestforreliability`	$\{\mathrm{T}, \mathrm{F}\}$	root/tree
Conflict analysis		
`conflict/enable`	$\{\mathrm{T}, \mathrm{F}\}$	root/tree
`conflict/maxstoresize`	$[100, 10000000] \cap \mathbb{Z}$	root/tree
`conflict/reconvlevels`	$[-1, 100] \cap \mathbb{Z}$	root/tree
`conflict/useprop`	$\{\mathrm{T}, \mathrm{F}\}$	root/tree
`conflict/usesb`	$\{\mathrm{T}, \mathrm{F}\}$	root/tree
Propagation		
`propagating/maxrounds`	$[-1, 10000] \cap \mathbb{Z}$	root/tree
`propagating/obbt/freq`	$[-1, 10000] \cap \mathbb{Z}$	root/tree
`propagating/pseudoobj/freq`	$[-1, 10000] \cap \mathbb{Z}$	root/tree
`propagating/redcost/freq`	$[-1, 10000] \cap \mathbb{Z}$	root/tree
LP		
`lp/resolveiterfac`	$[-1.0, 100.0] \cap \mathbb{R}$	root/tree
`lp/solvefreq`	$[-1, 10000] \cap \mathbb{Z}$	root/tree
Node selection		
`nodeselection/estimate/bestnodefreq`	$[0, 2147483647] \cap \mathbb{Z}$	root/tree
`nodeselection/estimate/maxplungedepth`	$[-1, 2147483647] \cap \mathbb{Z}$	root/tree
`nodeselection/estimate/maxplungequot`	$[0.0, 10^{99}] \cap \mathbb{R}$	root/tree
`nodeselection/estimate/minplungedepth`	$[-1, 2147483647] \cap \mathbb{Z}$	root/tree

For each evaluation, a fresh SCIP model reads the instance and applies the initial configuration before `optimize()`. Event handlers and scoring callbacks execute the policies during solving. The LLM agents are invoked between solver evaluations, as specified in Section 3.2.

B.2 Dynamic Configuration Policies and Decision Rules

Dynamic configuration rules. A dynamic rule is evaluated when SCIP focuses a node, improves the incumbent, or performs a separation round. Its conditions use the root or tree phase, processed-node count, depth, relative gap, incumbent availability and stagnation, LP iterations per node, elapsed and remaining time, and primal and dual progress. An action assigns a parameter value, scales a numerical setting by a factor in $[0.25, 4]$, or enables or disables a mechanism through its configuration controls. Each rule remains active for 1–64 processed nodes and can be applied at most three times. Incumbent-dependent heuristics require a feasible solution; rule priorities and cooldown conditions govern repeated activation.

At each event, expired changes are restored before eligible rules are applied. Each parameter has at most one temporary override, and its saved entry value is restored when the override expires. Root-scoped changes are restored when root processing ends. A rule therefore acts relative to the initial configuration of the strategy currently being evaluated.

Candidate features and normalization. For a numerical feature z and candidate pool C of size n, let $r_z(a)$ be the ascending average rank of candidate a. The normalized feature is

$$\widehat{z}(a; C) = \begin{cases} \dfrac{r_z(a) - 1}{n - 1}, & n > 1, \\ \dfrac{1}{2}, & n = 1. \end{cases} \tag{13}$$

Tied values receive average ranks. Feature orientation determines the quantity being ranked: sparsity, for example, ranks the negative number of nonzeros. Features and ranks are recomputed at every decision event, so the same coefficients can change the candidate ordering as the LP relaxation and candidate pool evolve.

The branching reference score ranks fractionality, $\min\{x - \lfloor x \rfloor, \lceil x \rceil - x\}$. Additional features are LP value, objective-coefficient magnitude, and upward and downward locks. The rule selects the highest-scoring candidate among SCIP's eligible, highest-priority LP branching candidates, preserving their incoming order for ties. It is applied at depth at most eight and for at most 32 branching decisions per solve; SCIP's other branching rules handle the remaining decisions.

We define the cut-selection reference score:

$$f_{\text{cut}}^{\text{ref}}(a) = 0.60\widehat{z}_{\text{efficacy}}(a) + 0.25\widehat{z}_{\text{sparsity}}(a) + 0.15\widehat{z}_{\text{age}}(a). \tag{14}$$

The feature pool also includes cut norm and a binary locality indicator; all numerical features use Equation (13). Optional cuts are ordered by decreasing score, with forced cuts and the solver's selection limit preserved. For either decision rule, the residual in Equation (3) has at most three additional terms, with $\|\alpha_m\|_\infty \leq 0.5$ and $\|\alpha_m\|_1 \leq 0.75$.

B.3 Agent Inputs and Knowledge Retrieval

Planning, proposal, and reflection use the same LLM with role-specific instructions. The planner receives the complete parent strategy, current objective, structural instance features, solver trajectories, and relevant experience. Features describe problem size, variable types, matrix sparsity, constraint structures, and finite-bound fractions. Trajectories summarize bound progress, incumbent discovery, node processing, and component effort.

The planner selects a component block, an edit type, and an intervention hypothesis. Up to four read-only tool calls retrieve the parent state, experience records, admissible parameter settings, or entries from the SCIP knowledge base. The proposal agent uses this information to generate one bounded candidate edit. Validation checks its structure, parameter values, selected parameter support, policy size, and duplication; an invalid proposal can receive one repair attempt.

The knowledge base indexes SCIP 10.0.1 mechanisms: initial and dynamic parameters, branching and cut-selection rules, and guidance for finding a feasible solution, improving it, and proving optimality. Entries distinguish parameters available for evolution from those excluded from candidate generation. An entry states the mechanism, legal stage, value domain, interactions, applicable problem structure, required solver observations, restoration behavior, and version-specific documentation, and it carries a stable citation. Retrieval ranks whole entries with a BM25-style lexical score inside the requested component categories and returns at most six hits (Robertson & Zaragoza, 2009). The query contains at most 512 characters. The 4500-character limit in Table 5 bounds the assembled return: hits are added in rank order, and an entry that does not fit is omitted. An agent tool call further limits that return to 1300 characters under the default tool budget.

After solver evaluation, reflection compares the child with its parent and relates changes in cost and trajectories to the intervention hypothesis. Experience memory records these observations together with the edited component and surrounding strategy. Memory is local to the target instance for instance-specific search and to the training data for portfolio learning. Consolidation keeps reusable lessons, excluded actions, and discouraged actions. Lessons are deduplicated and ordered from newest to oldest, with at most 16 stored and at most 12 returned on an agent call. The search brief retains at most six lessons, and the brief itself is limited to 6000 characters; when it exceeds that budget, the oldest lessons are dropped first. The *No retrieval* ablation removes knowledge-base retrieval while retaining this experience memory. Appendix C specifies the role prompts, context fields, runtime operations, and worked updates.

B.4 Monte Carlo Tree Search

Selection and expansion. For node v with visit count N_v and children $C(v)$, progressive widening sets the expansion limit to

$$w(N_v) = \max\{1, \lfloor k(N_v + 1)^\alpha \rfloor\}, \qquad k > 0, \quad 0 < \alpha < 1. \tag{15}$$

A node is expanded when $|C(v)| < w(N_v)$; the planner then selects a component and edit type for that strategy. Otherwise, selection descends using the UCT score in Equation (7).

Reward backup. For an evaluated child with reward $r = 1 - \widehat{F}(\sigma')$, each ancestor u, including the child itself, is updated by

$$N_u \leftarrow N_u + 1, \qquad S_u \leftarrow S_u + r, \qquad Q_u \leftarrow S_u / N_u. \tag{16}$$

Thus, Q_u estimates the mean reward of strategies explored through its refinement subtree. The best-so-far strategy separately records the lowest individual estimated cost. Valid evaluated children remain in the tree even when they do not immediately improve on their parent, allowing subsequent updates to other components to develop their potential. Algorithm 1 is the full procedure.

Algorithm 1 Coordinated evolution of a solver strategy

Require: Training instances $\mathcal{D}_{\mathrm{tr}}$, initial strategy σ_0, objective F, budget B, memory $\mathcal{M}$
Ensure: Best evaluated strategy $\sigma^\star$ and updated memory $\mathcal{M}$

$(\widehat{F}_0, \tau_0) \leftarrow \textsc{Evaluate}(\mathcal{D}_{\mathrm{tr}}, \sigma_0, F)$; initialize tree $\mathcal{T}$ at $(\sigma_0, \widehat{F}_0, \tau_0)$; $\sigma^\star \leftarrow \sigma_0$, $\widehat{F}^\star \leftarrow \widehat{F}_0$
for $b = 1, \ldots, B$ **do**
  $v \leftarrow \textsc{SelectForExpansion}(\mathcal{T})$ ▷ *UCT descent with progressive widening*
  $(m, e, h) \leftarrow \textsc{Plan}(\sigma_v, \tau_v, F, \mathcal{M})$ ▷ *Component block, edit type, hypothesis*
  $R \leftarrow \textsc{Retrieve}(m, e, h, \sigma_v, \mathcal{M})$; $\delta_m \leftarrow \textsc{Propose}(\sigma_v, m, e, h, R)$ ▷ *Bounded edit*
  $\sigma' \leftarrow \sigma_v \oplus \delta_m$; $(\widehat{F}', \tau') \leftarrow \textsc{Evaluate}(\mathcal{D}_{\mathrm{tr}}, \sigma', F)$
  $v' \leftarrow \textsc{AddChild}(\mathcal{T}, v, \sigma', \widehat{F}', \tau')$; $\textsc{Backup}(v', 1 - \widehat{F}')$ ▷ *Keep the child even if it is worse*
  $\mathcal{M} \leftarrow \textsc{Reflect}(\mathcal{M}, v, \delta_m, h, v')$ ▷ *Test the hypothesis; store the lesson*
  if $\widehat{F}' < \widehat{F}^\star$ **then** $\sigma^\star \leftarrow \sigma'$, $\widehat{F}^\star \leftarrow \widehat{F}'$
  end if
end for
return $(\sigma^\star, \mathcal{M})$

The tree is retained across configuration and policy updates because the objective and evaluation data remain fixed. Each new Hydra round starts a new tree for its complementary objective and retains the previously learned portfolio strategies.

B.5 Portfolio Construction and Strategy Selection

Hydra initializes the portfolio with SCIP-default and adds one strategy in each of five rounds (Xu et al., 2010). Each round invokes Algorithm 1 for 60 continuous update attempts, with the agent choosing both component and edit type from feedback. In round k, let P_{k-1} be the current portfolio and $s_{k-1}(\phi(I))$ the strategy chosen for training instance I. The complementary objective is

$$\begin{aligned} b_I^{(k-1)} &= \widehat{J}_I\big(s_{k-1}(\phi(I))\big), \\ \widehat{F}_k(\sigma) &= \frac{1}{|D_{\mathrm{tr}}|} \sum_{I \in D_{\mathrm{tr}}} \min\{b_I^{(k-1)}, \widehat{J}_I(\sigma)\}. \end{aligned} \tag{17}$$

The baseline $b_I^{(k-1)}$ is fixed throughout the round. A candidate gains credit on instances where it improves the current selection; elsewhere, the minimum retains the baseline cost. The reward $1 - \widehat{F}_k$ therefore guides the search toward a complementary strategy.

After each round, the evolved strategy is added to the portfolio and cross-evaluated on the training instances. All six strategies, including SCIP-default, remain available to the final selector. Training and validation costs are averaged over two common search seeds; the final portfolio and selector are then evaluated on four held-out seeds.

Instance features and strategy selection. The routing features combine structural descriptors with a five-second SCIP-default probe. Structural features describe problem size, variable types, matrix sparsity, finite bounds, and constraint structure. Probe features summarize bound progress, node processing, and LP work. Feature preprocessing is fitted on the training set. For each strategy pair (i, j), a random forest predicts the lower-cost strategy, weighting each training instance by $|\widehat{J}_I(\sigma_i) - \widehat{J}_I(\sigma_j)|$ (Breiman, 2001; Xu et al., 2012). Pairwise predictions vote for a strategy:

$$s(\phi(I)) = \sigma_{j^\star}, \qquad j^\star \in \arg\max_j \sum_{i \neq j} \mathbf{1}\{h_{ij}(\phi(I)) = j\}. \tag{18}$$

Ties are resolved by lower mean training cost. An ordinary multiclass random forest, trained to predict the lowest-cost strategy, provides the second selector candidate. Both use 300 trees, a minimum leaf size of four, and the square root of the feature count at each split, with a fixed random seed. Validation selects the classifier with lower mean normalized PDI.

Portfolio evaluation. Portfolio construction, selector training, validation, and testing use the same 300-second evaluation protocol for each instance–seed pair. The default probe runs for up to five seconds. If it proves optimality, its result is returned; otherwise, the selected strategy runs on a fresh model for up to 295 seconds. Reported time includes the probe and selected solve. PDI sums their percentage-gap integrals over their respective time intervals, using the common-horizon convention in Appendix A.5. The probe and selected solve use the same solver seed. Final table entries first average the four held-out evaluations within each instance, then report the mean across instances.

C Agent Prompts and Policy Implementation

This appendix expands the planning, proposal, and reflection steps in Algorithm 1. The prompts expose the complete parent strategy so that a local edit can account for the components with which it will interact. Structural instance features, execution trajectories, retrieved solver knowledge, and accumulated experience supply complementary context: they describe the problem, the current bottleneck, the available mechanisms, and the effects of previous interventions.

C.1 Agent Inputs and Workflow

Each update starts from a parent selected by tree search. The context contains its initial configuration, dynamic rules, cut-selection and branching scores, and measured trajectories on the current training instances. The context exposes the initial-parameter catalog, dynamic controls, and scoring features. The planner chooses both the component block and the edit type from this information and the accumulated feedback. Its hypothesis determines the documentation and experience retrieved for the proposal. The proposal produces one edit, which is applied to a copy of the parent strategy. The resulting complete strategy is evaluated on the same instances and two search seeds as the parent. Final evaluation uses the four held-out seeds specified in Table 5.

Reflection receives the paired results and the original hypothesis. It records how the intervention affected bound progress and component work, together with the surrounding configuration and policies. Numerical rewards update the search tree, while these contextual records inform later proposals. The two channels implement the feedback described in Section 3.3.

Table 15: Context supplied to the LLM agents. Double-braced tokens in the prompt listings are replaced by the corresponding records.

Field	Contents
`INSTANCE_FEATURES`	Variable and constraint counts, nonzeros, variable-type fractions, matrix density, constraint structures, and finite-bound fractions
`PARENT_STRATEGY`	Complete initial configuration, dynamic rules, and candidate-scoring rules
`PARENT_TRAJECTORIES`	Primal and dual progress, incumbent discovery, node processing, LP work, and component activity on each evaluation instance and seed
`SEARCH_OBJECTIVE`	The current objective F, its aggregation over instances, and the parent estimate $\widehat{F}(\sigma_v)$
`COMPONENT_CATALOG`	Solver components and the initial-configuration, dynamic-rule, or scoring-rule edits available for each component
`PARAMETER_CATALOG`	Component, parameter name, type, default, admissible values, and initial or dynamic execution scope
`EXPERIENCE`	Previous component edits, their parent contexts, observed outcomes, and reflection summaries
`RETRIEVED_KNOWLEDGE`	SCIP mechanism descriptions and parameter guidance relevant to the component and intervention hypothesis

The planner can request up to four reads through `inspect_parent_state`, `inspect_experience_memory`, `inspect_parameter_surface`, and `search_scip_knowledge`. The controller executes these requests and attaches their returned records to the proposal context. Parameter introspection supplies types and execution scope; documentation retrieval explains how a setting affects the solving mechanism. This lets the LLM use domain knowledge to formulate an intervention and then assess its observed effect through solver feedback.

C.2 Role-Specific Prompt Templates

The following templates specify the role instructions and structured outputs for the algorithm described in Section 3. Double-braced fields are populated at each update. Initial-configuration, dynamic-configuration, and scoring-rule proposal templates implement the edit types in Appendix B.1; the parent always supplies the inherited parts of the strategy.

System Prompt

```
You are a SCIP 10.0.1 expert designing a complete MILP strategy.
A strategy consists of an initial parameter configuration,
state-dependent parameter rules, a cut-selection score,
and a branching score.

Use the instance structure, solver trajectories, and prior
interventions to reason about the current solving bottleneck.
Explain how the selected intervention interacts with the
settings and policies inherited from the parent.

Optimize the supplied SEARCH_OBJECTIVE: lower normalized
primal-dual integral is better. Relate intended changes to
incumbent discovery, dual-bound progress, or component work.
The controller evaluates proposals and updates the search tree.

Use the supplied parameter names, values, and policy features.
Return the structured output requested for your current role.

a) Available components and edit types: {{COMPONENT_CATALOG}}
b) Search objective: {{SEARCH_OBJECTIVE}}
c) Instance features: {{INSTANCE_FEATURES}}
d) Complete parent strategy: {{PARENT_STRATEGY}}
e) Parent trajectories: {{PARENT_TRAJECTORIES}}
f) Relevant experience: {{EXPERIENCE}}
```

Planning Agent

```
Task: choose the next component block and edit type.

Read the complete parent strategy and its solving trajectories.
Identify a bottleneck and select a component block and a supported
edit type: initial settings, dynamic rules, or a scoring rule.
Use the latest feedback and previous edits to decide which
change is most useful in the current strategy.

Formulate an intervention hypothesis: the observed behavior,
the mechanism to change, and the expected effect on solving.
Request the parent, experience, parameter, or documentation
records needed to turn this hypothesis into a concrete edit.

Available components and edit types:
{{COMPONENT_CATALOG}}
```

```
Available read-only tools:
{{TOOL_DESCRIPTIONS}}

Return JSON:
{
  "components": "one or more groups from COMPONENT_CATALOG",
  "edit_type": "initial | dynamic | scorer",
  "hypothesis": {
    "observation": "bottleneck in the parent trajectory",
    "mechanism": "how the proposed intervention addresses it",
    "interaction": "relation to inherited settings or rules",
    "expected_effect": "predicted change in solving behavior"
  },
  "tool_calls": [
    {"name": "tool name", "arguments": {}}
  ]
}
```

INITIAL CONFIGURATION PROPOSAL

```
Task: realize the plan as one initial-configuration edit.

Plan: {{PLAN}}
Component parameter catalog: {{PARAMETER_CATALOG}}
Retrieved solver knowledge: {{RETRIEVED_KNOWLEDGE}}
Tool results: {{TOOL_RESULTS}}
Relevant experience: {{EXPERIENCE}}

Select the parameter assignments that implement the hypothesis
within the chosen block of six to eight parameters. Use admissible
values and consider their joint effect under the parent.
Retain all unselected parameter assignments and runtime policies.
The controller applies this edit before optimize().

Return JSON:
{
  "components": "the component block selected in PLAN",
  "kind": "initial",
  "static_parameters": {
    "catalog parameter name": "admissible value"
  },
  "hypothesis": "mechanism and expected solving effect"
}
```

DYNAMIC CONFIGURATION PROPOSAL

```
Task: realize the plan as one dynamic-policy edit.

Plan: {{PLAN}}
Online parameter catalog: {{PARAMETER_CATALOG}}
Inherited rules: {{PARENT_DYNAMIC_RULES}}
Retrieved solver knowledge: {{RETRIEVED_KNOWLEDGE}}
Tool results and experience: {{TOOL_RESULTS_AND_EXPERIENCE}}

Specify when the intervention is useful and how long it acts.
Use an event (node_focused, best_solution, separation_round),
a phase (root or tree), and conditions on the supplied signals.
Available signals include processed nodes, depth, gap,
incumbent availability and stagnation, LP work, elapsed time,
```

```
and primal and dual progress.

Actions set or scale a setting, activate or deactivate a
mechanism, or open a temporary neighborhood-search window.
Use the catalog's operation for the chosen parameter.
Use a lifetime of 1-64 nodes and at most three applications;
root-scoped settings are restored on leaving the root.
Express cooldown and hysteresis in processed nodes.
Retain all initial settings and all unselected rules.

Return JSON:
{
  "components": "the component block selected in PLAN",
  "kind": "dynamic",
  "rules": [{
    "rule_id": "identifier for this component rule",
    "trigger": {
      "event": "event name", "phases": ["root or tree"],
      "conditions": [{"condition": "signal test", "value": 0}]
    },
    "action": {
      "parameter": "online catalog name", "operator": "set",
      "value": 0, "lifetime_nodes": 4, "revert": "original"
    },
    "max_applications": 1,
    "cooldown_nodes": 4, "hysteresis_nodes": 0
  }],
  "hypothesis": "trigger, intervention, and intended effect"
}
```

CUT-SELECTION AND BRANCHING RULES

```
Task: revise the selected component's candidate preferences.

Plan: {{PLAN}}
Reference score and candidate features: {{SCORER_CATALOG}}
Parent scoring rule: {{PARENT_SCORER}}
Retrieved knowledge and experience: {{RETRIEVED_CONTEXT}}

Return a score of the form
  reference_score + sum(coefficient * candidate_rank_feature).
Use at most three residual terms, each with absolute coefficient
at most 0.5 and total absolute coefficient at most 0.75.
Ranks are recomputed within the current candidate pool.
For branching, the highest score selects an eligible variable.
For cut selection, scores order the current optional cuts.

Explain how the changed preferences complement the parent's
configuration and other policies. Replace the selected scoring
rule and retain the other parts of the parent strategy.

Return JSON:
{
  "component": "cut_selection or branching",
  "kind": "scorer",
  "terms": [{"feature": "catalog feature", "coefficient": 0.2}],
  "hypothesis": "intended change in candidate preference"
}
```

REFLECTION AGENT

```
Task: turn the paired evaluation into reusable experience.

Parent strategy: {{PARENT_STRATEGY}}
Component edit and hypothesis: {{EDIT_AND_HYPOTHESIS}}
Parent costs and trajectories: {{PARENT_EVALUATION}}
Child costs and trajectories: {{CHILD_EVALUATION}}
Current search objective: {{SEARCH_OBJECTIVE}}

Compare the child and parent on the same instances and seeds.
Describe whether the intended behavioral change occurred,
how primal and dual progress changed, and how work shifted
across solver components. Identify the inherited settings or
rules relevant to this effect and the next useful intervention.
Attach the observations to the parent context and edited
component so future proposals can retrieve this experience.

Return JSON:
{
  "component": "edited component",
  "observed_effect": "paired cost and trajectory comparison",
  "mechanism_evidence": "behavior relevant to the hypothesis",
  "interaction_context": "inherited settings and policies",
  "next_intervention": "component and mechanism to investigate",
  "experience_summary": "contextual lesson for later updates"
}
```

C.3 STRATEGY UPDATES AND POLICY EXECUTION

Strategy updates. The controller copies the parent strategy and applies the selected component block's edit according to Equation (6). Initial updates replace the named parameter assignments. Dynamic updates insert or revise the selected rules by identifier, and scoring updates replace that component's residual coefficients. The resulting strategy contains the inherited settings and policies as well as the new edit. The agent can select any supported edit type at the next iteration. Each candidate inherits all assignments and rules outside its edit, so successive evaluations measure complete strategies as they evolve.

For each evaluation, the controller loads a fresh SCIP model and writes θ with `setParam` before calling `optimize()`. Reading each value back establishes the starting configuration for the dynamic runtime. Policy handlers and scoring callbacks are registered on this same model. The solver then executes the entire strategy.

Event-driven parameter changes. A PySCIPOpt `Eventhdlr` handles focused-node and new-incumbent events; a separation-round observer supplies the corresponding cut-loop events. The runtime computes the state signals, restores expired changes, and checks the rules for the current event and phase. When a rule fires, it saves the current parameter value, applies the operation through `setParam`, and records when the change expires. A temporary override therefore resumes the inherited setting at expiry. Root-scoped changes expire when root processing finishes. Priorities order eligible rules, while cooldown and application counts control repeated activation.

The following runtime pseudocode shows the ordering of event processing. The matching step includes event, phase, state conditions, cooldown, and remaining application count.

```
on_solver_event(event, state):
    restore_expired_overrides(state)
    for rule in priority_order(policy.rules):
        if matches(rule, event, state):
            name = rule.action.parameter
```

Table 16: Dynamic operations and examples of their role in the solver. The condition and duration are part of the evolved rule.

Operation	Runtime action	Example intervention
`set`	Assign a catalog value for the active interval	Reduce `separating/maxcuts` when LP work per node is high
`scale`	Multiply a numerical control by a factor in $[0.25, 4]$ using its catalog type and bounds	Adjust a component's effort relative to the inherited setting
`enable`/`disable`	Apply the parameter's activation or deactivation setting	Activate a separator or suspend a heuristic during a selected search phase
`lns_window`	Temporarily activate a catalog-supported neighborhood-search control	Increase primal-search effort when an incumbent exists but has stagnated

```
if name not in active_overrides:
    saved = model.getParam(name)
    value = apply_operation(rule.action, saved)
    model.setParam(name, value)
    register_override(rule, saved, state)
    record_activation(rule, state)
```

Candidate-scoring callbacks. A `Branchrule` callback extracts the eligible LP branching candidates and computes their numerical feature ranks with Equation (13). It evaluates Equation (3) and branches on the highest-scoring eligible variable. A `Cutsel` callback computes ranks for the optional cuts in its current pool, scores them, and returns their order to SCIP's selection procedure. The policy coefficients are part of the strategy; the candidate features and ranks are computed afresh at each callback. The reference rules, feature pools, and execution scopes are given in Appendix B.2.

C.4 Executed Cross-Component Refinement on Multiple Knapsack

The multiple-knapsack case in Appendix E.5 is one initial-configuration candidate with eight assignments: four in Branching and four in Cut management. It inherits the parent's dynamic policy unchanged. The joint edit is evaluated once as a complete strategy on two search seeds, consuming one candidate trial.

Root separation and branching settings. The candidate enables dynamic lookahead for reliability branching, sets reliability thresholds to 1 and 3, and sets the strong-branching LP-iteration quotient to 0.25. It assigns a root-cut cap of 500, three root separation rounds, one stalling round, and minimum efficacy 5×10^{-4}. The lower reliability threshold and stalling-round value explicitly retain their SCIP defaults. The edit coordinates the relaxation controls with the branching decisions that use that relaxation.

D Examples of Component Interactions

The following examples illustrate why component updates must be considered within a complete strategy. They contrast useful intervention points, show how an initial primal heuristic changes the effect of a dynamic configuration policy, and demonstrate a reversal in the ranking of branching rules under different separation settings.

D.1 Separation and Presolve

For the triangle-cover instance

$$\min_{x \in \{0,1\}^3} x_1 + x_2 + x_3 \quad \text{subject to} \quad x_1 + x_2 \geq 1,\ x_1 + x_3 \geq 1,\ x_2 + x_3 \geq 1, \tag{19}$$

the LP optimum is $(1/2, 1/2, 1/2)$ with value $3/2$, while the integer optimum is two. Starting from the all-ones incumbent, branching on any variable processes the root and two integral children. The valid cut $x_1 + x_2 + x_3 \geq 2$ solves the problem at the root, reducing three nodes to one.

In contrast, consider

$$\min_{x \in \{0,1\}^n} \sum_{i=1}^{n} x_i \quad \text{subject to} \quad x_1 = 1, \qquad x_{i-1} - x_i = 0 \quad (i = 2, \ldots, n). \tag{20}$$

Its LP relaxation already has the unique integral solution $x = \mathbf{1}$, so cut selection and branching make no decisions. Presolve propagates $x_1 = 1$ through the equalities and removes all variables and rows. Solving these relaxations with the HiGHS dual-simplex method through SciPy's `linprog` (Virtanen et al., 2020; Huangfu & Hall, 2018) takes 10, 100, and 1,000 simplex iterations for $n = 10, 100, 1000$ with presolve disabled, and zero in all three cases with presolve enabled. These examples distinguish stronger relaxations from simplification before the LP solve.

D.2 Primal Heuristics and Dynamic Separation

Table 17(a) shows a complementary configuration–policy change: enabling an initial primal heuristic or changing the separation trigger alone leaves the node count at five, while combining them reduces it to three. The instance is

$$\min_{x \in \{0,1\}^9} c^\top x \quad \text{subject to} \quad x_i + x_j \geq 1 \quad \text{for all } (i,j) \in E_p, \tag{21}$$

where $c = (11, 3, 13, 8, 6, 10, 13, 15, 14)$ and

$$\begin{aligned} E_p = \{&(1,2), (1,4), (1,5), (1,6), (1,7), (2,8), \\ &(3,5), (3,8), (3,9), (4,6), (5,7), (8,9)\}. \end{aligned} \tag{22}$$

The separator adds $x_i + x_j + x_k \geq 2$ for triangles $(1,4,6)$, $(1,5,7)$, and $(3,8,9)$. These cuts follow by summing the covering constraints and using integrality. Binary enumeration gives 61 feasible covers and optimal cost 53.

Configuration and policy. All strategies start from the all-ones cover of cost 93. The initial configuration either disables a greedy heuristic or enables it to delete variables in decreasing cost order, retaining each deletion when feasibility is preserved and breaking ties by index. The heuristic produces an incumbent of cost 55. The separation-round limit initially equals zero. One policy enables separation at the first node after a strict incumbent improvement; the other enables it after two consecutive processed nodes without improvement. Initialization is excluded from these improvement events. At the triggering node, the limit is raised to one and then restored to zero, while the added cuts remain. Each strategy makes one separator call.

All strategies use depth-first search, explore the zero child first, and choose the lexicographically smallest LP optimum. Branching selects a fractional variable of highest degree in the original constraints, with index-based tie breaking. Every processed node, including the root and bound-pruned nodes, is counted.

Effect of component interactions. The incumbent of 55 prunes $x_1 = 0$, whose LP bound is 61. After the root and this child, the stagnation trigger enables separation at $x_1 = 1$, raising its bound from 47.5 to an integral optimum of 53. Only three nodes are processed (Figure 5). Without the heuristic, the left subtree must be explored; with the heuristic but the improvement trigger, separation occurs too late to avoid branching at the right child. The five-node intermediate strategies therefore become useful starting points for a complementary update to the other component.

D.3 Separation Settings and Branching Rules

The second construction shows that changing separation can reverse the relative performance of branching rules. Let

$$\begin{aligned} E = \{&(1,2), (2,6), (2,8), (3,5), (3,6), (3,7), (3,8), (3,9), \\ &(4,5), (4,6), (4,9), (5,6), (5,7), (6,7), (6,8), (7,8), (8,9)\}, \end{aligned} \tag{23}$$

Table 17: Configuration–policy interactions in the constructed set-cover instances. Entries count all processed nodes; lower is better. In (a), all strategies use one separator call and the same branching rule. In (b), separation changes the relative performance of two branching rules.

(a) Initial heuristic and separation trigger

Initial heuristic	Improvement-triggered	Stagnation-triggered
Disabled	5	5
Enabled	5	3

(b) Separation setting and branching rule

Triangle cuts	High-degree branching	Low-degree branching
Disabled	7	11
Enabled	5	3

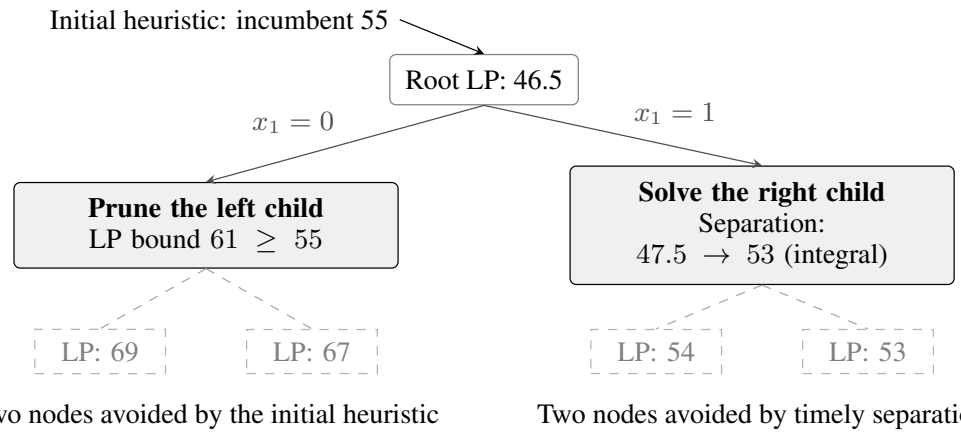

Figure 5: The initial heuristic and the stagnation-triggered separation policy remove different subtrees. Solid boxes are the three processed nodes; dashed boxes are the avoided descendants.

and consider the weighted set-cover instance

$$\min_{x\in\{0,1\}^9} \sum_{j=1}^{9} (101-j)x_j \quad \text{subject to} \quad x_i + x_j \geq 1 \quad \text{for all } (i,j) \in E. \tag{24}$$

Binary enumeration gives 40 feasible covers and optimal cost 569. The initial separation setting either adds no cuts or adds all ten triangle inequalities $x_i + x_j + x_k \geq 2$ before the root relaxation and retains them throughout the search.

The high- and low-degree rules choose a fractional variable with the largest and smallest degree in the original constraints, respectively, breaking ties by index. All strategies use the search procedure and LP tie breaking of Appendix D.2, with the all-ones incumbent of cost 864 and no additional presolve, heuristics, or cuts.

Without cuts, high-degree branching processes 7 nodes and low-degree branching processes 11. With cuts, their counts are 5 and 3, respectively (Table 17(b)). In the latter three-node tree, low-degree branching selects x_4 at the root. The zero child has integral optimum 569, and the one child is pruned at bound $1724/3 > 569$. The preferred branching rule thus changes with the separation setting, illustrating why decision rules must be evaluated and refined within their surrounding configuration.

Exhaustive binary enumeration verifies the integer optima and cut validity in both set-cover constructions. Replaying the specified LP tie breaking and search rules reproduces all node counts in Table 17.

E Additional Experimental Results

E.1 Additional Benchmark Results

Table 18 reports all metrics for the main benchmarks. Table 19 complements these with SC (1,000 columns, 900 rows), MIS (1,500 vertices), MK (200 items, 8 knapsacks), CA (400 items, 2,000 bids), and CFL (100 facilities, 400 customers). All use the same instance-specific search and held-out-seed evaluation protocol. CoMILP has the lowest PDI on four of these five benchmarks.

Table 18: Full results for the main benchmarks. Time is capped at 300 seconds; PDI is in percentage-seconds. Impr. is the percentage reduction in mean PDI over SCIP-default. Bold marks the best value; all-timeout columns are not highlighted.

Method	Time (s)	PDI	Impr. (%)
Set covering			
1,000 columns, 1,700 rows			
SCIP-default	294.5	3453.6	0.0
HEM	298.7	3223.6	6.7
Collab	268.6	3286.7	4.8
DynSep	288.4	3645.2	−5.5
SMAC-I	259.3	2323.6	32.7
GP-BO	257.3	2365.7	31.5
GRIMIP	277.0	2744.5	20.5
ColdStart	271.4	2821.6	18.3
LLM4Branch	272.3	3307.8	4.2
MILP-Evo	286.1	3533.0	−2.3
CoMILP	**238.2**	**2044.2**	**40.8**
Maximum independent set			
1,100 vertices, 4,384 edges			
SCIP-default	235.7	883.9	0.0
HEM	202.8	630.4	28.7
Collab	162.3	464.3	47.5
DynSep	142.1	484.5	45.2
SMAC-I	204.8	528.6	40.2
GP-BO	225.5	608.5	31.2
GRIMIP	228.1	525.2	40.6
ColdStart	288.1	747.5	15.4
LLM4Branch	227.8	609.3	31.1
MILP-Evo	184.6	592.0	33.0
CoMILP	**124.9**	**426.2**	**51.8**
Multiple knapsack			
600 items, 16 knapsacks			
SCIP-default	291.0	275.3	0.0
HEM	272.4	185.8	32.5
Collab	277.9	223.9	18.7
DynSep	247.5	145.1	47.3
SMAC-I	291.1	154.7	43.8
GP-BO	275.7	173.2	37.1
GRIMIP	272.6	155.7	43.4
ColdStart	300.0	135.6	50.7
LLM4Branch	288.2	207.0	24.8
MILP-Evo	290.1	198.8	27.8
CoMILP	**139.0**	**77.4**	**71.9**

Table 18 (continued)

Method	Time (s)	PDI	Impr. (%)
Combinatorial auctions			
600 items, 4,000 bids			
SCIP-default	300.0	4943.4	0.0
HEM	300.0	5054.7	−2.3
Collab	300.0	4598.3	7.0
DynSep	300.0	4799.4	2.9
SMAC-I	300.0	4282.9	13.4
GP-BO	300.0	4402.6	10.9
GRIMIP	300.0	4346.6	12.1
ColdStart	300.0	4935.1	0.2
LLM4Branch	300.0	5139.7	−4.0
MILP-Evo	300.0	4982.5	−0.8
CoMILP	300.0	**3578.3**	**27.6**
Capacitated facility location			
80 facilities, 240 customers			
SCIP-default	98.7	194.2	0.0
HEM	101.5	181.0	6.8
Collab	68.2	137.2	29.4
DynSep	82.9	182.4	6.1
SMAC-I	42.8	**56.5**	**70.9**
GP-BO	53.6	75.8	61.0
GRIMIP	51.0	71.6	63.1
ColdStart	58.4	78.8	59.4
LLM4Branch	49.9	91.4	52.9
MILP-Evo	108.0	112.6	42.0
CoMILP	**32.4**	97.6	49.7
MIPLIB 2017			
240 instances			
SCIP-default	246.6	15603.5	0.0
SMAC-I	230.2	14909.5	4.4
GP-BO	231.2	13703.6	12.2
GRIMIP	223.8	12006.0	23.1
ColdStart	245.2	15041.8	3.6
LLM4Branch	244.7	16218.2	−3.9
MILP-Evo	238.4	15625.0	−0.1
CoMILP	**211.3**	**10235.9**	**34.4**

Table 19: Additional instance sizes (25 instances each). Metrics and boldface follow Table 18.

Method	Time (s)	PDI	Impr. (%)
Set covering			
1,000 columns, 900 rows			
SCIP-default	183.7	1610.8	0.0
HEM	128.6	1016.2	36.9
Collab	86.0	726.3	54.9
DynSep	79.0	630.4	60.9
SMAC-I	69.2	521.2	67.6
GP-BO	69.9	475.0	70.5
GRIMIP	89.7	747.4	53.6
ColdStart	113.6	1011.6	37.2
LLM4Branch	185.5	1486.9	7.7
MILP-Evo	174.1	1632.9	−1.4
CoMILP	**52.4**	**387.0**	**76.0**
Maximum independent set			
1,500 vertices, 5,984 edges			
SCIP-default	291.0	1004.9	0.0
HEM	275.8	972.7	3.2
Collab	**235.4**	829.5	**17.5**
DynSep	255.9	845.8	15.8
SMAC-I	275.1	897.2	10.7
GP-BO	278.0	940.1	6.4
GRIMIP	265.0	916.9	8.8
ColdStart	296.4	898.9	10.5
LLM4Branch	277.1	960.8	4.4
MILP-Evo	290.6	1112.3	−10.7
CoMILP	264.0	**829.1**	**17.5**
Multiple knapsack			
200 items, 8 knapsacks			
SCIP-default	193.4	80.6	0.0
HEM	120.8	49.2	39.0
Collab	112.1	53.5	33.6
DynSep	63.5	21.0	73.9
SMAC-I	88.1	26.1	67.6
GP-BO	89.0	34.2	57.6
GRIMIP	82.5	29.8	63.0
ColdStart	170.0	52.4	35.0
LLM4Branch	82.3	68.2	15.4
MILP-Evo	100.5	67.6	16.1
CoMILP	**18.4**	**8.24**	**89.8**
Combinatorial auctions			
400 items, 2,000 bids			
SCIP-default	300.0	2222.8	0.0
HEM	300.0	2135.5	3.9
Collab	300.0	1984.4	10.7
DynSep	300.0	2067.1	7.0
SMAC-I	300.0	1949.0	12.3
GP-BO	300.0	1982.9	10.8
GRIMIP	300.0	1944.0	12.5
ColdStart	300.0	2498.4	−12.4
LLM4Branch	300.0	2269.5	−2.1
MILP-Evo	300.0	1911.6	14.0
CoMILP	300.0	**1698.4**	**23.6**

Table 19 (continued)

Method	Time (s)	PDI	Impr. (%)
Capacitated facility location			
100 facilities, 400 customers			
SCIP-default	169.1	471.5	0.0
HEM	169.4	527.5	−11.9
Collab	**123.0**	341.9	27.5
DynSep	168.8	427.4	9.4
SMAC-I	148.0	221.2	53.1
GP-BO	151.7	267.9	43.2
GRIMIP	144.9	**213.1**	**54.8**
ColdStart	174.3	223.6	52.6
LLM4Branch	172.9	313.1	33.6
MILP-Evo	169.4	308.7	34.5
CoMILP	137.4	219.4	53.5

E.2 Additional Ablation Results

Table 20 reports all metrics for the five-family ablation study, using 25 targets per family at the sizes shown in the table and the controls in Appendix A.7. Full CoMILP achieves the lowest PDI in all five families. Thus, the benefit of coordinated evolution extends to bound progress even where proof times are similar.

Table 20: Full five-family ablation results. Metrics and boldface follow Table 18; all variants use the common budgets in Table 2.

Method	Time (s)	PDI	Impr. (%)
Set covering			
1,000 columns, 900 rows			
SCIP-default	183.7	1610.8	0.0
Branching only	132.0	991.5	38.4
No branching	90.5	762.5	52.7
Initial config. only	75.6	612.2	62.0
Policies only	106.1	722.3	55.2
No retrieval	60.4	450.5	72.0
No memory	78.1	536.4	66.7
Single agent	54.8	449.0	72.1
CoMILP	**52.4**	**387.0**	**76.0**
Maximum independent set			
1,100 vertices, 4,384 edges			
SCIP-default	235.7	883.9	0.0
Branching only	152.0	484.4	45.2
No branching	147.3	477.6	46.0
Initial config. only	159.8	525.3	40.6
Policies only	155.2	583.9	33.9
No retrieval	148.1	509.2	42.4
No memory	151.6	520.6	41.1
Single agent	**122.7**	495.7	43.9
CoMILP	124.9	**426.2**	**51.8**

Table 20 (continued)

Method	Time (s)	PDI	Impr. (%)
Multiple knapsack			
200 items, 8 knapsacks			
SCIP-default	193.4	80.6	0.0
Branching only	127.4	87.8	−8.9
No branching	87.4	41.1	49.0
Initial config. only	65.8	35.4	56.1
Policies only	138.5	79.7	1.1
No retrieval	115.9	90.6	−12.4
No memory	133.2	84.8	−5.2
Single agent	94.2	70.5	12.5
CoMILP	**18.4**	**8.24**	**89.8**
Combinatorial auctions			
400 items, 2,000 bids			
SCIP-default	300.0	2222.8	0.0
Branching only	300.0	2071.5	6.8
No branching	300.0	1962.7	11.7
Initial config. only	300.0	2125.0	4.4
Policies only	300.0	2126.7	4.3
No retrieval	300.0	1992.2	10.4
No memory	300.0	1973.8	11.2
Single agent	300.0	1834.1	17.5
CoMILP	300.0	**1698.4**	**23.6**
Capacitated facility location			
80 facilities, 240 customers			
SCIP-default	98.7	194.2	0.0
Branching only	62.3	132.6	31.7
No branching	52.1	121.4	37.5
Initial config. only	45.8	154.5	20.4
Policies only	42.7	130.1	33.0
No retrieval	40.8	143.3	26.2
No memory	67.2	154.0	20.7
Single agent	41.3	143.9	25.9
CoMILP	**32.4**	**97.6**	**49.7**

E.3 Full Results on Unseen Instances

Table 21 reports all five families, all seven methods, and all three metrics for the portfolio-reuse experiment. All methods use the family training and validation splits in Appendix A.3 and the same 100 unseen test instances per family. The table retains both PDI and capped time alongside relative PDI improvement, complementing the reuse summary in Table 3. CoMILP+Hydra improves mean PDI over SCIP-default in every family.

Table 21: Full family-trained comparison on 100 unseen instances per family at the sizes shown below. Entries average four held-out seeds within each instance and then average across instances. Time is capped at 300 s; PDI is in percentage-seconds; Impr. is the percentage reduction in mean PDI relative to SCIP-default. CoMILP+Hydra includes its probe and selected solve within the same budget. Boldface follows Table 1.

Method	Time (s)	PDI	Impr. (%)
Set covering: 1,000 columns, 900 rows			
SCIP-default	120.9	1008.1	0.0
HEM	101.3	818.6	18.8
Collab	67.8	559.7	44.5
DynSep	83.7	791.3	21.5
SMAC-P	97.9	692.3	31.3
GP-BO	**57.0**	**405.3**	**59.8**
CoMILP+Hydra	75.8	636.1	36.9
Maximum independent set: 1,100 vertices, 4,384 edges			
SCIP-default	219.3	687.6	0.0
HEM	179.2	521.9	24.1
Collab	167.7	549.7	20.1
DynSep	**162.8**	**478.2**	**30.5**
SMAC-P	220.9	683.7	0.6
GP-BO	237.3	709.7	−3.2
CoMILP+Hydra	196.1	486.8	29.2
Multiple knapsack: 200 items, 8 knapsacks			
SCIP-default	121.5	57.9	0.0
HEM	92.6	44.0	24.0
Collab	81.0	51.3	11.4
DynSep	**54.8**	**25.9**	**55.3**
SMAC-P	83.3	30.4	47.5
GP-BO	86.1	31.7	45.3
CoMILP+Hydra	85.8	42.9	25.9
Combinatorial auctions: 400 items, 2,000 bids			
SCIP-default	300.0	2182.1	0.0
HEM	300.0	2377.8	−9.0
Collab	300.0	**1930.1**	**11.5**
DynSep	300.0	2275.5	−4.3
SMAC-P	300.0	1979.6	9.3
GP-BO	300.0	2053.6	5.9
CoMILP+Hydra	300.0	2025.0	7.2
Capacitated facility location: 80 facilities, 240 customers			
SCIP-default	80.9	119.4	0.0
HEM	86.6	121.0	−1.3
Collab	69.7	92.9	22.2
DynSep	82.3	119.4	0.0
SMAC-P	84.6	86.6	27.5
GP-BO	**69.1**	**83.4**	**30.2**
CoMILP+Hydra	77.6	97.5	18.3

E.4 Search Efficiency and Component Allocation

Equal-budget search comparison. Figure 3 compares CoMILP with reproduced SMAC-I, GP-BO, and GRIMIP searches on six instances from the MIPLIB dataset. Each method follows its original search procedure and receives 100 candidate evaluations per instance. In particular, GRIMIP

retains the parameter-space selection and configuration-search procedure of the original algorithm. All methods use the solver limits and two common search seeds specified in Section 4.1.

One evaluation scores a candidate strategy by its mean PDI over the two search seeds. At evaluation b, the curve reports the lowest mean PDI obtained through evaluations $1, \ldots, b$, including SCIP-default as the common reference at evaluation zero. The horizontal axis therefore measures the same number of candidate evaluations for every method. After search, the selected strategy is fixed and assessed on four held-out evaluation seeds for the final performance comparison. The curves and result tables thus measure search progress and final strategy performance, respectively, under the shared two-search-seed, four-held-out-seed protocol.

Component allocation. Figure 4 summarizes the search histories from the main instance-specific experiments in Tables 1 and 19, covering both instance scales in each generated family and MIPLIB. Figure 4 counts proposal attribution, not solver calls or independent single-component trials. A proposal's attributed set A_k contains its declared mechanism and parameter/rule categories; each member receives weight $1/|A_k|$. Thus $n_{im} = \sum_{k \in \mathcal{P}_i} \mathbf{1}\{m \in A_k\}/|A_k|$ and $p_{im} = n_{im} / \sum_{m'} n_{im'}$. A joint Branching–Cut management proposal contributes one half to each group while remaining one candidate trial. Family means give each instance equal weight.

E.5 Cross-Component Refinement on Multiple Knapsack

The multiple-knapsack case introduced in Section 4.2 demonstrates a joint **Branching–Cut management** configuration update. Its mean PDI improves by 71.3% and capped time by 50.9% relative to the evolved parent.

One candidate, two component groups. The recorded parent-to-child transition is one initial-configuration candidate with eight assignments, four per group (Table 22). The parameter categories follow Table 12. The candidate consumes one trial, evaluated on search seeds 11 and 13. All other settings, including the parent's OBBT frequency rule, are inherited. Branching adjusts the work spent gathering and using reliability information; Cut management controls the relaxation on which branching operates.

Table 22: The single joint configuration candidate in the multiple-knapsack case study. Eight assignments span two component groups; unlisted settings and the dynamic policy are inherited. The lower reliability threshold and stalling-round value retain their defaults.

Solver component	Refinement within the complete strategy	Role
Branching	Enable dynamic lookahead; set reliability thresholds to 1 and 3 and strong-branching LP-iteration quotient to 0.25.	Revised: variable-selection effort.
Cut management	Set root-cut cap to 500, root rounds to 3, stalling rounds to 1, and minimum cut efficacy to 5×10^{-4}.	Revised: relaxation used by Branching.

Table 23: Paired outcomes in the multiple-knapsack case study. The final block averages both search seeds. Time is capped at 300 seconds, and PDI is in percentage-seconds.

Search seed	Strategy	Time (s)	PDI	Nodes
11	SCIP-default	300.00	156.33	106,992
11	Evolved parent	101.23	135.68	16,608
11	Refined strategy	6.84	7.95	1,045
13	SCIP-default	49.82	61.35	6,814
13	Evolved parent	76.85	35.68	52,432
13	Refined strategy	80.55	41.17	53,093
Mean	SCIP-default	174.91	108.84	56,903
Mean	Evolved parent	89.04	85.68	34,520
Mean	Refined strategy	**43.69**	**24.56**	27,069

Measured gains along the refinement path. Across search seeds 11 and 13, mean PDI decreases from 108.84 to 85.68 to 24.56, and capped time from 174.91 to 89.04 to 43.69 seconds (Table 23). The refined strategy improves PDI by 77.4% relative to default.

Solver work and inherited policy. Across the two seeds, strong-branching LP iterations fall from 24,464 and 24,426 in the parent to 5,350 and 11,322 in the candidate, a 65.9% reduction in their mean. This supplies a solver-work readout alongside the complete-strategy improvement.