DECOMPPILOT: LONG-HORIZON AGENTIC DISCOVERY OF DECOMPOSITION STRATEGIES FOR LARGE-SCALE MIXED-INTEGER LINEAR PROGRAMS

Anonymous authors
Paper under double-blind review

ABSTRACT

Large-scale mixed-integer linear programs (MILPs) are computationally intensive due to the combinatorial complexity introduced by discrete decisions. Decomposition-based optimization reduces this complexity by partitioning the original program into smaller subproblems and coordinating the subproblem solutions to satisfy coupling constraints and maintain consistency among linking variables. We propose DECOMPPILOT, an LLM-agent framework that uses the semantics of decision variables and constraints to automatically design and execute partitioning and coordination strategies tailored to different problem structures. We formulate decomposition design as a long-horizon experimental search problem rather than a one-shot partitioning task, and study whether LLM agents can discover decomposition operators that can be reused across problem instances. This setting introduces two challenges: preserving useful evidence from successful and failed solver trials over long search histories, and accumulating reusable design knowledge from instance-specific experience. To address these challenges, we propose two complementary mechanisms: a *Trial Memory Tree* that preserves instance-level trial evidence for refinement and backtracking, and *Decomposition Operator Induction* that identifies and validates reusable algorithmic components across instances and triggers targeted trials when further evidence is needed. Across four benchmark suites spanning unit commitment and bin packing, the reusable decomposition operators discovered by DecompPilot achieve a 94.5% feasibility rate on unseen instances, including 73 million-scale PEGASE instances. They reduce total solving wall-time by 52.7% on average relative to Gurobi while keeping the average relative objective gap to the Gurobi reference at 0.75%.

1 INTRODUCTION

Mixed-integer linear programming (MILP) provides a powerful optimization paradigm for a wide range of applications, including planning, scheduling, and resource allocation. Many of these problems are computationally challenging: they often contain a large number of discrete decisions, leading to exponential growth in the combinatorial search space (Bengio et al., 2021; Gasse et al., 2019). Modern solvers reduce this burden through presolve, cutting planes, primal heuristics, and branch-and-bound. Yet large practical instances can still contain millions of variables and constraints after presolve and remain difficult to solve (Nair et al., 2020). Decomposition-based optimization manages this scale by partitioning the original MILP into smaller subproblems and coordinating their decisions to ensure that coupling constraints and linking variables remain globally consistent. However, classical decomposition methods typically rely on a predefined, problem-specific partitioning scheme, while mixed-integer settings require additional coordination and recovery heuristics (Bergner et al., 2015; Watson & Woodruff, 2011).

Our goal is to leverage the semantic reasoning capabilities of LLM agents to automate these model-level decomposition decisions. Given variable and constraint descriptions, indexing patterns, and other model information, an agent can explore how to partition a MILP in a meaningful way and how to coordinate the resulting subproblems. Figure 1 illustrates one possible trial of decomposing a unit commitment: the agent groups generators into subproblems, assigns local variables and constraints to each block, identifies system-wide coupling constraints, and proposes coordination rules for global

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.

demand and reserve requirements.

Recent work has shown that LLMs can assist combinatorial optimization in modeling, heuristic discovery, and solution repair (Ahmaditeshnizi et al., 2024; Romera-Paredes et al., 2024; Chen et al., 2026). Unlike these tasks that mainly refine the latest candidate, model-level decomposition involves an interdependent chain of decisions: defining the subproblem blocks, identifying cross-subproblem couplings, and choosing how the resulting subproblems are coordinated. Solver feedback is delayed until this chain is executed, and a failed trial does not directly reveal which earlier decision should be revised. This makes linear refinement insufficient: the agent must retain the evidence and lineage of previous trials so that it can revisit earlier design decisions when necessary. We therefore formulate decomposition design as a long-horizon experimental search problem rather than a one-shot partitioning task. Considering that repeating such a costly search from scratch for every new instance would limit the practical value of this approach, we also seek to distill instance-level trial experience into reusable decomposition operators for unseen instances from the same family.

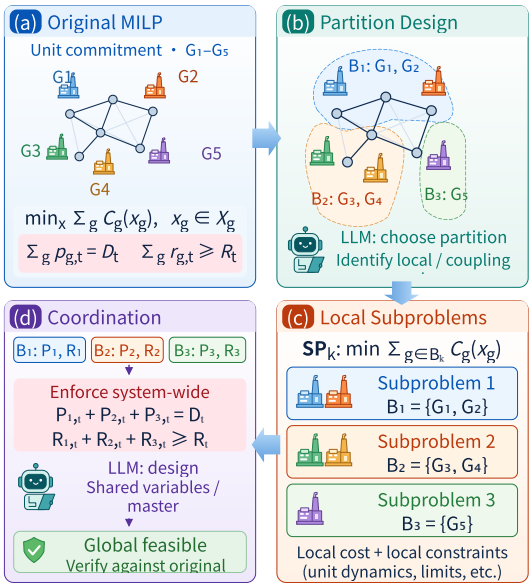

Figure 1: Illustration of model-level decomposition on a unit-commitment example. The LLM agent chooses a generator-axis partition, groups generator-indexed variables into local subproblems, and specifies how their solutions are coordinated to recover a globally feasible solution for the original MILP.

To this end, we propose DECOMPPILOT, an LLM-agent framework for long-horizon empirical discovery of reusable decomposition operators for families of large-scale MILPs. We study two central questions: *(1) Can an agentic framework conduct long-horizon trial-and-error search to discover effective decomposition strategies? (2) Based on instance-level discoveries, can the framework induce reusable decomposition operators that transfer to unseen instances?*

DECOMPPILOT addresses these questions with two complementary mechanisms. For instance-level discovery, a *Trial Memory Tree* preserves successful and failed evidence in a branching history of instance-level solver trials, enabling refinement, branching, and backtracking. In each trial, the agent uses variable and constraint semantics to propose a decomposition axis and classify constraint families as local or coupled, then generates code to materialize these decisions at scale, including block construction and the required master or boundary interfaces. For cross-instance reuse, *Decomposition Operator Induction* abstracts recurring design principles from successful trial histories into candidate operators and validates them on additional instances. An Orchestrator agent controls progression through this loop, deciding whether the current evidence is sufficient to advance, or whether additional trials and operator revision are required. We freeze the resulting operator before held-out evaluation, so test performance measures reuse rather than continued instance-specific search.

We evaluate DECOMPPILOT on four large-scale unit-commitment and bin-packing suites spanning the $10^5 - 10^6$ scale in variables and constraints, including million-scale PEGASE instances. In particular, PEGASE represents the size and complexity of the European high-voltage network, for which large-scale UC formulations have been reported to require prohibitive solution times (MATPOWER contributors, 2016; Colonetti, 2024). DECOMPPILOT discovers reusable operators adapted to different problem families, including an adapted time-axis rollout for CA and a generator-axis operator built around column generation for PEGASE (see Appendix G for detailed operator definitions). Since these operators prioritize efficient decomposition and recovery over exhaustive search, we evaluate both runtime and solution quality. Across the non-seed instances from these four suites, the reusable operators pass strict original-model feasibility checks on 121 out of 128 instances (94.5%). Under a 3600-second time limit, they reduce wall-clock time by 52.7% on average relative to Gurobi, while keeping the average relative objective gap to the reference below 1%.

2 Related Work

LLM agents for mathematical optimization. LLMs and LLM-based agents increasingly support multiple stages of combinatorial optimization, including model formulation, heuristic design, and solver configuration. For model formulation, OptiMUS and Chain-of-Experts automate natural-language-to-optimization modeling through iterative solver-code refinement and role-specialized agent collaboration, respectively (Ahmaditeshnizi et al., 2024; Xiao et al., 2024). For heuristic design, EoH and ReEvo evolve executable heuristics through LLM-generated ideas and reflective feedback (Liu et al., 2024; Ye et al., 2024). For solver configuration, recent works use LLMs for cold-start separator selection and instance-specific MIP solver tuning (Lawless et al., 2025; Luo et al., 2026). Most of these methods iteratively refine a candidate within a predefined search template. For DecompPilot, the candidate itself is a chain of interdependent decomposition decisions, and its action and revision spaces adapt to the problem structure and evolve with accumulated trial evidence.

LLMs for algorithm discovery and agentic search. Beyond task-specific optimization, recent agentic systems use execution feedback to support longer-horizon algorithmic and scientific discovery. ADAS searches over code-defined agent systems, while AFlow uses Monte Carlo Tree Search to optimize executable agent workflows (Hu et al., 2025; Zhang et al., 2025). The AI Scientist automates iterative hypothesis generation, code modification, experimentation, and result analysis (Lu et al., 2024), while OR-Agent structures automated research as a tree of candidate solutions with branching, backtracking, and hierarchical reflection (Liu et al., 2026). These systems motivate treating algorithm design as an experimental search process rather than one-shot generation. DecompPilot similarly organizes trial histories in a tree, but grounds the search in optimization-specific mathematical validity and strict verification. It also retains unsuccessful trials as evidence to avoid repeated failure modes and guide targeted revisions, and accumulated trial histories support cross-instance operator induction.

Classical and LLM-based decomposition for large-scale MILPs. Classical decomposition methods exploit predefined block structures and provide iterative schemes for solving and coordinating subproblems, as in Dantzig–Wolfe decomposition, rolling-horizon optimization, and progressive hedging (Dantzig & Wolfe, 1960; Costley et al., 2017; Watson & Woodruff, 2011). Automatic methods generate candidate block structures from algebraic problem features, while learning-based approaches use learned quality scores to select and refine promising decomposition candidates (Bergner et al., 2015; Basso & Ceselli, 2023). Recent LLM-based methods have also explored reducing large-scale optimization to localized subproblem search around an existing solution. LLM-LNS uses a self-evolving LLM agent to discover neighborhood-selection heuristics that restrict large-scale MILP search to smaller neighborhoods, while DRAGON decomposes an initial global solution into promising regions for local reconstruction and reintegration (Ye et al., 2025; Chen et al., 2026). In contrast, DecompPilot performs model-level decomposition directly from a MILP formulation, discovering the partition and coordination strategy without requiring an initial solution.

3 Decomposition Preliminaries for DecompPilot Agents

This section introduces the model-level decomposition task and the prior decomposition knowledge provided to the agents. We first formalize how a MILP is partitioned into subproblems, and then summarize common coupling patterns and classical decomposition primitives used as search priors in DecompPilot. Detailed constructions and validity rules are given in Appendix C.

3.1 Model-Level Decomposition

Consider the MILP

$$(\mathrm{P}) \qquad \min_{x} \; c^\top x \quad \text{s.t.} \quad Ax \bowtie b, \qquad \ell \le x \le u, \qquad x_j \in \mathbb{Z}\ (j \in \mathcal{I}), \tag{1}$$

where $\bowtie$ denotes the row-wise constraint senses $\{\le, =, \ge\}$ and $\mathcal{I}$ indexes the integer variables. Besides the algebraic formulation, DecompPilot receives semantic descriptions and naming or indexing information for variable and constraint families.

A decomposition partitions the variables into B disjoint blocks,

$$\mathcal{P} = \{\mathcal{V}_1, \ldots, \mathcal{V}_B\}, \qquad \mathcal{V} = \dot{\bigcup_{b=1}^{B}} \mathcal{V}_b, \tag{2}$$

with x_b denoting the variables owned by block b. For a constraint row i, let

$$\mathcal{B}(i) = \{b : \exists j \in \mathcal{V}_b,\ A_{ij} \neq 0\} \tag{3}$$

denote the blocks touched by that row. A constraint is *local* if $|\mathcal{B}(i)| = 1$ and *coupling* if $|\mathcal{B}(i)| > 1$.

Collecting the local constraints of each block into its feasible set X_b, the partitioned MILP can be written in the standard block-angular form (Dantzig & Wolfe, 1960; Bergner et al., 2015):

$$\begin{aligned} \min_{x_1,\dots,x_B} \quad & \sum_{b=1}^{B} c_b^\top x_b \\ \text{s.t.} \quad & x_b \in X_b, \qquad\qquad b = 1,\dots,B, \\ & \sum_{b=1}^{B} G_b x_b \ \bowtie\ h. \end{aligned} \tag{4}$$

Here, X_b contains the local constraints and variable domains of block b, while $\sum_b G_b x_b \bowtie h$ collects the coupling constraints that prevent the blocks from being solved independently.

The agent may use both algebraic structure and model semantics to propose $\mathcal{P}$. For example, unit-commitment variables may be grouped by time period or by generator even when these structures are not obvious from the raw constraint matrix. The proposed decomposition must nevertheless preserve every variable, constraint coefficient, constraint sense, and right-hand side of the original MILP. DECOMPPILOT assigns the Master agent to compile such proposals through a layered plan-and-check process: observed axis values and variable families must be covered, coupled rows must be assigned to a valid interface, and deterministic probes reject incomplete blocks or inconsistent row assignments before solver execution. The final assembled solution is additionally checked against the untouched original MILP. These mechanisms are detailed in Section 4.

3.2 COUPLING PRIORS

After partitioning, subproblems are independent except for constraints or decisions that involve more than one block. We provide the Master agent with two common coupling patterns as structural priors.

Resource sharing. Several blocks may jointly satisfy the same global requirement. For example, generators assigned to different blocks must together meet the system demand. If $R_b x_b$ denotes the part of such a requirement contributed by block b, the original coupling row takes the form $\sum_b R_b x_b \bowtie r$. This representation applies to demand balance, reserve requirements, shared capacity, and, more generally, any linear row whose variables span multiple blocks.

Consensus coupling. Different blocks may instead contain decisions that represent the same shared or boundary state and therefore must agree. For neighboring blocks, this can be written as $E_b^+ x_b = E_{b+1}^- x_{b+1}$, where $E_b^+ x_b$ and $E_{b+1}^- x_{b+1}$ extract the two representations of the shared boundary. More generally, if several blocks share the same decision, their local values can be tied to a common variable z through $E_b x_b = z$ (Boyd et al., 2011).

These priors give the Master agent standard ways to represent cross-block dependencies when constructing a decomposition plan. For resource-sharing constraints, the plan records the part contributed by each block and preserves the original global relation; for consensus coupling, it records which shared or boundary values must agree across blocks. Problem-specific structures may be identified when they are apparent, while these generic forms provide fallback interfaces for ordinary cross-block linear coupling. The corresponding global variables and executable interfaces are constructed in the layered planning process described in Section 4 and detailed in Appendix C.

3.3 DECOMPOSITION OPERATORS AS SEARCH PRIORS

A candidate decomposition operator is represented as $o = (a, \mathcal{P}, \mathcal{C}, \kappa, r, \eta)$, where a is the partition axis, $\mathcal{P}$ specifies the blocks, $\mathcal{C}$ describes coupling treatment, κ selects a coordination method, r specifies solution assembly or recovery, and η contains its parameters. The operator library provides three main starting points: *rolling horizon* for ordered decompositions with local temporal dependencies, *DW/CG* for independently solvable blocks linked through a compact master problem, and *ADMM-style coordination* for consensus or shared-resource interfaces.

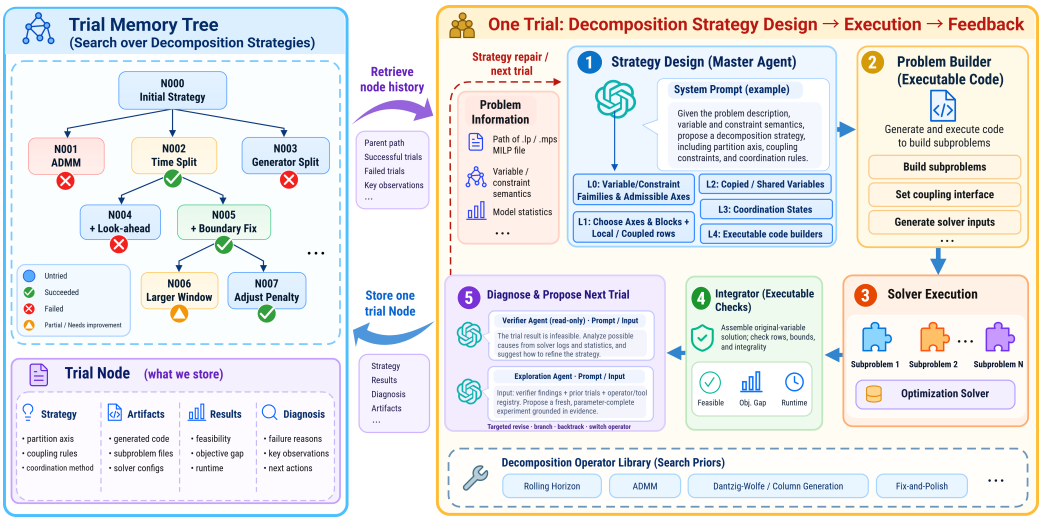

Figure 2: Single-instance search in the *Trial Memory Tree*. The Master agent designs a strategy, the builder and solver execute it, and the Integrator checks the assembled solution. The Verifier and Exploration agents use the resulting evidence to propose the next trial.

These primitives are exposed through a registered operator/tool interface rather than used as fixed solvers. The agents decide how to instantiate and revise them—for example, the block granularity and history/look-ahead rules of a rollout, coordination budgets and recovery choices for DW/CG, or initialization and penalty choices for ADMM—and may switch or augment the mechanism as new trial evidence becomes available. Their executions also expose structured signals such as boundary violations, consensus residuals and variable agreement, and master/column statistics, which help guide subsequent decomposition and recovery decisions. This is particularly important for MILPs, where these classical primitives do not necessarily produce an integer-feasible original-model solution directly (Boyd et al., 2011; Bergner et al., 2015). Detailed operator definitions and optimization formulations are given in Appendices C and F.

4 DECOMPPILOT

DECOMPPILOT searches for decomposition strategies at two levels. The *Trial Memory Tree* organizes trials on a single MILP instance, while *Decomposition Operator Induction* uses evidence from these searches to develop an operator for other instances of the same problem family.

4.1 TRIAL MEMORY TREE FOR SINGLE-INSTANCE DISCOVERY

As shown in Figure 2, each trial starts from the original MILP, descriptions and indices of its variables and constraints, relevant previous trials, and any diagnosis of the parent trial. It produces a decomposition strategy, an executable solver experiment, and a record of the outcome. Trials form a tree because a failed or partially successful strategy may support several different revisions.

Strategy design and execution. The Master agent develops the strategy in five dependent layers. **L0** identifies variable and constraint families and possible partition axes; **L1** assigns variables and local constraints to blocks and identifies coupling constraints; **L2** classifies the identified coupling as resource sharing or consensus and specifies the corresponding interface: each block's contribution to a shared constraint or equality links between copies of a shared variable; **L3** chooses the coordination rules; and **L4** generates code to build the subproblems and any global model. Before execution, fixed checks ensure that the blocks cover the model and their coupling interfaces preserve the original constraints. The Problem Builder then constructs the models, the solver executes them, and the Integrator assembles an assignment in the original variables. Fixed checks assess bounds, integrality, all original constraints, and the behavior of iterative coordination to determine whether the trial succeeds.

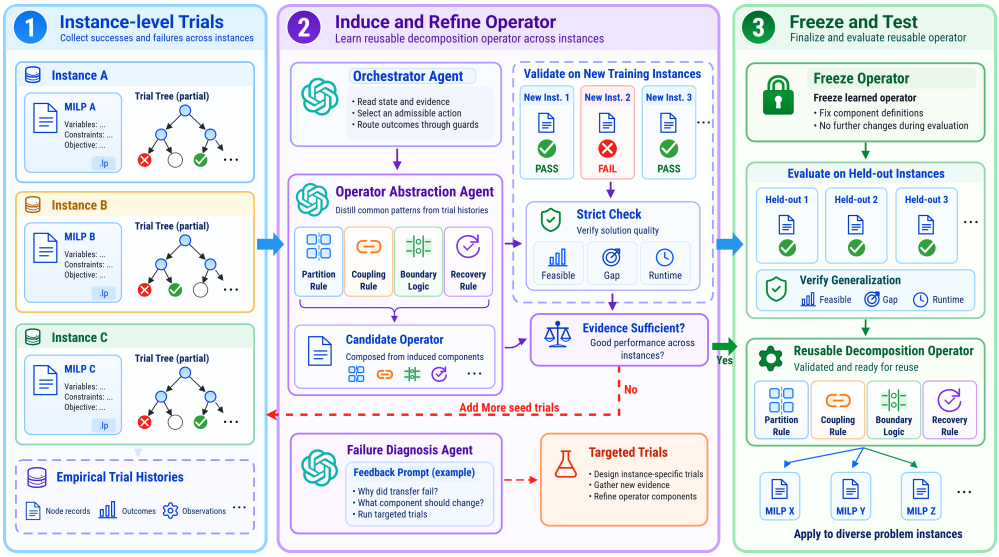

Figure 3: Cross-instance *Decomposition Operator Induction*. Trial histories from seed instances support operator abstraction; new training instances test the candidate and expose failures that guide revision or additional trials. The operator is frozen before evaluation on held-out instances.

Diagnosis and follow-up trials. When a trial fails, the Verifier agent examines solver outputs, violated constraints, and the decisions made in earlier layers. Its report links the observed failure to evidence, identifies the layer that needs revision, and suggests a concrete check for the proposed repair. The Exploration agent uses this report and previous outcomes to propose a follow-up experiment, such as changing an operator parameter, trying another coordination rule, or modifying a subproblem builder. The experiment is executed and evaluated by the same fixed procedure, and its outcome becomes a child of the trial from which it arose.

Trial memory and reuse. Each node retains its parent, strategy, generated code, solver outputs, original-model checks, and diagnosis in an isolated directory. The tree provides short summaries for later retrieval while preserving links to the complete evidence. Operators that pass the original-model checks can inform later searches; on a new instance, the agent rebuilds the blocks and interfaces from that instance's model and checks the assembled solution against its original MILP. Checks for repeated strategies and equivalent outcomes avoid unnecessary solver trials. Further details appear in Appendix D.

4.2 Decomposition Operator Induction Across Instances

A strategy that succeeds on one instance may depend on details that do not hold elsewhere. We call the training instances selected for a complete *Trial Memory Tree* search *seed instances*. Their successful and failed trials provide the evidence from which a reusable operator is proposed (Figure 3).

Operator abstraction. The Operator Abstraction agent compares seed trial histories to identify recurring partitioning, coupling, boundary, and recovery rules. It separates these rules from instance-specific values and specifies how they should be instantiated for another MILP of the same family. The resulting candidate must still produce executable subproblems and satisfy the model-level checks of Section 3 before it can be evaluated.

Orchestration and validation. The Orchestrator agent uses the current candidate and accumulated trial evidence to select the next step: continue seed search, form or revise an operator, validate it, or request more evidence. Its choices are limited to actions allowed at the current stage; fixed checks control execution and determine whether a stage can advance. The candidate is applied to new training instances, with feasibility checked against each original MILP and the outcome assessed for

objective quality and runtime. Successful checks on one instance do not suffice: the candidate must also pass the required training instances and a full training re-evaluation before it is frozen.

Failure-guided refinement. When validation fails, the Failure Diagnosis agent identifies the failing mechanism and links it to solver and feasibility evidence. The Fix Adviser agent suggests a change to the operator or an experiment that could distinguish between possible causes. The Orchestrator may then revise the candidate, run a targeted trial, or add another seed instance if the current seeds do not cover the relevant structure. Each revision is tested again on training instances under the same fixed checks.

Freeze and held-out evaluation. Once the training checks pass, the partitioning, coordination, and recovery rules are fixed and applied to held-out instances without further strategy search. Held-out outcomes therefore measure whether the discovered operator transfers to new instances. If an implementation error arises, it may be corrected while keeping these rules fixed and repeating the training checks. Held-out results are not used to redesign the operator. Appendix E describes the validation and data-access rules.

5 Experiments

5.1 Setup

Benchmarks. We study three security-constrained unit-commitment (SCUC) regimes and one packing family. CA and FERC come from the PGLib-UC benchmark collection (PGLib-UC, 2024) through UnitCommitment.jl (UC.jl) (Xavier et al., 2024). CA combines generator histories with demand and reserve rows, whereas FERC adds longer temporal logic and shared resource coupling. PEGASE uses the `case13659pegase`-based SCUC construction in UC.jl, derived from the PEGASE European transmission-grid case (UnitCommitment.jl contributors, 2024; MATPOWER contributors, 2016); its 13,659-bus network yields a million-scale test with transmission and reserve constraints. BinPacking uses 20 Falkenauer-uniform `u500` instances (Delorme et al., 2016), whose item-coverage constraints couple feasible packing patterns. Table 1 summarizes model scale; PEGASE's reported protocol uses 80 instances (five seeds, two validation cases, and 73 final tests) from a larger 365-model source directory. Exact splits and per-instance results are in Appendices B and H.

Table 1: Mean original-MILP size; n is the number of instances in each reported protocol.

Dataset	n	Integer	Continuous	Rows
CA	20	146400	159275	379620
FERC	24	195792	282601	544325
PEGASE	80	1178496	699780	2463408
BinPacking	20	250500	0	251499

Baselines for reusable operators. We compare four approaches under the same instance populations. (1) **Gurobi-3600s**: solves each original MILP without decomposition and provides a finite-time objective reference; its incumbent need not be optimal. (2) **SCIP**: solves the same MILP directly, providing a reference for the SCIP backend used by GCG. (3) **GCG-Auto**: uses GCG's automatic matrix-based block detection and decomposition with its SCIP backend, without a supplied partition axis or model semantics; comparison with SCIP probes the value of this automatic pipeline. (4) **Canonical CG**: applies a fixed, family-specific Dantzig–Wolfe partition and Coluna-style column generation, testing whether a partition specified once is sufficient. BinPacking uses a validated pattern master, knapsack pricing, and integer recovery. Fixed generator-schedule formulations were also attempted for CA, FERC, and PEGASE, but their mapping back to the original MILP has not passed the equivalence check; their outcomes are reported only as diagnostic evidence.

Implementation and evaluation protocol. Agent decisions use an API-hosted proprietary LLM; the harness executes the candidate with mathematical-programming solvers and independently evaluates its output. The monolithic baselines have a 3600-s limit; final frozen-operator runs account for construction, solving, and verification under the declared budget. A frozen operator shares partition, coupling, coordination, and recovery rules but transfers no incumbent or variable assignment; final-test instances are withheld from discovery. Solver-process and discovery accounting, historical timing coverage, and the dataset-specific splits are detailed in Appendices A and B.

Table 2: FERC strategies discovered on `2015-01-01_hw`; "Ref. diff." is the signed difference from the Gurobi incumbent.

Operator	Method obj.	Ref. obj.	Time (s)	Ref. diff.	Strict
Time: stable-pool + resource-aware polish	41,697,829.21	41,482,588.83	182.286	+0.519%	✓
Gen.: DW/CG k=2 + consensus polish	41,694,001.10	41,482,588.83	180.069	+0.510%	✓
Gen.: DW/CG k=4 + consensus polish	41,764,328.29	41,482,588.83	178.879	+0.679%	✓
Gen.: DW/CG k=6 + consensus polish	43,340,955.64	41,482,588.83	176.595	+4.480%	✓

Feasibility check. The evaluator reloads the untouched original MILP and checks assignment coverage, variable bounds, integrality, and every original row to tolerance 10^{-6} before recomputing the objective. Failed solves and near-feasible reconstructions remain failures even when an agent labels them successful.

Metrics. We report strict-feasible counts over all non-seeds and $\Delta_{\text{ref}} = 100(f(x) - f_{\text{ref}})/|f_{\text{ref}}|$ for feasible solutions paired with a Gurobi incumbent. This signed objective difference is not a solver MIP gap. Key solve time measures sequential subproblem solves or the critical path of parallel CG plus recovery; recorded wall time also covers model loading and construction, initial columns, solution assembly, and verification. Wall-time components and missing-value rules appear in Appendices A and H.4.

5.2 Performance Evaluation

5.2.1 Single-Instance Strategy Discovery

Table 2 reports FERC discoveries on `2015-01-01_hw`. Direct time-block stitching fails under long-range resource and commitment constraints, prompting stable-trajectory recovery. On the generator axis, the search finds that two DW/CG blocks with limited consensus fixing give a smaller reference-objective difference than finer partitions at similar solve times. The tree retains the trials that explain these revisions.

Trial-tree evidence in two CA searches. Figure 4 follows the decision-relevant branches of 29 recorded trials. Screening four block counts under ADMM-based diagnostics and sequential rollout identifies `N0007` as a low-residual, fast candidate. Refinement reaches `N0019`, an eight-block rollout with eight startup violations. Repeating its one-period history at `N0027` leaves the same violations; the verifier's three-period offset evidence instead sets the history depth at `N0028`, which passes the strict original-model check in 5.58 s of subproblem solve time.

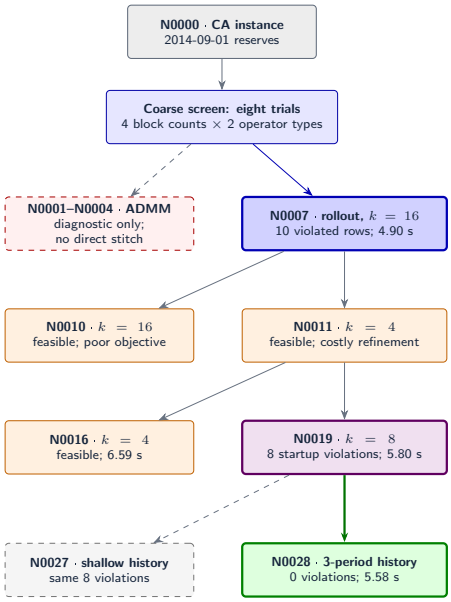

Figure 4: Abridged CA search tree. Node colors distinguish diagnostic, inferior feasible, near-feasible, and strictly feasible trials.

The second search, on `2015-03-01_reserves_1`, uses prior successful trials as structural hypotheses, not transferred assignments. After incomplete proposals fail validation, the current instance is reconstructed and solved; its near-feasible eight-block rollout branches into two strictly feasible variants, exchanging a lower reference-objective difference for higher solver and recorded wall times. Its abridged tree and the node-level evidence for both searches appear in Appendix D.4 (Figure 5).

5.2.2 Cross-Instance Reuse of Discovered Operators

Table 3 compares frozen operators on matched non-seed instances.

Table 3: Non-seed comparison: incumbent/strict-feasible counts, solve time, recorded wall time, and signed objective difference from the Gurobi reference. PEGASE includes two validation and 73 held-out instances.

	CA (16 non-seed)				FERC (21 non-seed)				PEGASE (75 non-seed)				BinPacking (16 non-seed)			
Method	Inc./strict	Time (s)	Wall (s)	Δ_{ref} (%)	Inc./strict	Time (s)	Wall (s)	Δ_{ref} (%)	Inc./strict	Time (s)	Wall (s)	Δ_{ref} (%)	Inc./strict	Time (s)	Wall (s)	Δ_{ref} (%)
Gurobi-3600s	16/16	61.5	**61.5**	0.000	21/21	1500.5	1502.7	0.000	75/75	3601.0	3601.0	0.000	16/16	3600.1	3600.6	0.000
SCIP	16/16	3378.8	3381.8	−0.193	1/21	3600.5	3603.1	+15.722	33/75	3602.6	3618.8	+126.360	16/16	3600.1	3602.3	+1.967
GCG-Auto	0/16	–	3600.3	–	0/21	–	3603.5	–	0/75	–	3603.1	–	0/16	–	3601.1	–
Canonical CG[§]	0/16	–	3605.2	–	0/21	–	3609.8	–	0/75	–	3621.6	–	16/16	1714.9	1779.7	−0.302
DECOMPPILOT (ours)	15/16	**30.4**	85.4	+0.150	15/21	**178.3**	**301.4**	+3.423	75/75	**630.5**	**1009.1**	+0.210	16/16	**77.8**	**87.1**	−0.769

"Inc./strict" counts solver incumbents for Gurobi/SCIP and original-MILP-verified solutions for the decomposition methods. "Time" is Gurobi/SCIP solver runtime or DECOMPPILOT/CG key solve time; "Wall" is recorded elapsed time, with different coverage across suites. – denotes an unavailable value, not zero. See Appendices A and H.4 for timing coverage and reference provenance.

CA solves four time blocks in sequence, passing instance-derived history between blocks. FERC uses two generator blocks with DW/CG and limited consensus fixing before original-MILP polish; PEGASE uses eight generator blocks, a MIP start from the current instance, and no hard fixing. BinPacking prices capacity-feasible patterns and solves an integer master. Appendix G specifies operator's detailed rules.

Aggregate performance and baseline interpretation. Across the four non-seed suites, DECOMPPILOT passes the original-MILP check on 121/128 instances (94.5%). The suite-level signed Δ_{ref} values are $+0.150\%$ (CA), $+3.423\%$ (FERC), $+0.210\%$ (PEGASE), and -0.769% (BinPacking), averaging $+0.754\%$. Key solve time is shorter than Gurobi's in all four suites. Recorded wall time improves on FERC (301.4 versus 1502.7 s), PEGASE (1009.1 versus 3601.0 s), and BinPacking (87.1 versus 3600.6 s), but not CA (85.4 versus 61.5 s). The mean of these four suite-level wall-time reductions is 52.7%, with the coverage differences noted in Table 3.

SCIP finds 66/128 incumbents, whereas SCIP-backed GCG-Auto finds none within 3600 s on any suite. Automatic matrix-based detection and decomposition therefore yield no feasible solution here; the available aggregates do not isolate detection time or report usable block counts. Fixed canonical CG gives 0/16, 0/21, and 0/75 strictly verified solutions on CA, FERC, and PEGASE, respectively, but their original-model mappings remain unvalidated; these runs are diagnostic rather than evidence against CG itself. On BinPacking, where the formulation is validated, DECOMPPILOT reduces key solve time from 1714.9 to 77.8 s. The results motivate solver-guided revision of partition and recovery rules, although FERC transfer remains partial (15/21). Appendix H reports the separate FERC time-axis stress test.

Transfer limitations and system overhead. The discovered operators do not guarantee successful transfer to every instance: DECOMPPILOT is strict-feasible on 121/128 non-seed instances overall, with FERC remaining the most challenging suite at 15/21. Nevertheless, the transferred FERC operator reduces recorded wall time from 1502.7 s for Gurobi to 301.4 s on average, showing that partial transfer can still provide substantial computational benefit. CA exposes a different limitation: although its decomposition reduces solver work, its recorded wall time (85.4 s) exceeds Gurobi's 61.5 s because the end-to-end operator also incurs model construction, subproblem generation, solver invocation, solution integration, and verification overhead. These results indicate that both transfer robustness and harness overhead remain important targets for further improvement.

6 CONCLUSION

We presented DECOMPPILOT, an agentic framework for discovering reusable decomposition operators for large-scale MILPs. A *Trial Memory Tree* supports solver-grounded trial-and-error search, while *Decomposition Operator Induction* abstracts validated instance-level experience into reusable operators. Experiments across CA, FERC, PEGASE, and BinPacking show that these operators can transfer across instances.

Limitations. The current harness still has limitations in long-horizon execution stability. Agents may occasionally repeat similar refinements without recognizing when the current decomposition direction should be abandoned and replanned from an earlier decision point.

REPRODUCIBILITY STATEMENT

All reported successes are backed by saved solver artifacts and strict validation on the untouched original MILP. Appendix A specifies evaluation and timing semantics, Appendix G records the frozen operator schemas and their applicability conditions, and Appendix H reports the per-instance results underlying the main aggregates. An anonymized implementation of DECOMPPILOT is available at `https://anonymous.4open.science/r/DecompPilot-4D9D`.

AI USE STATEMENT

Large language models are part of the proposed research method and are used to propose and revise decomposition experiments inside the harness. AI tools were also used during research for code assistance, debugging, and language editing. All experimental claims in the paper are grounded in executable solver artifacts and were checked by the authors; no unverified model-generated statement or solution is treated as experimental evidence.

REFERENCES

Ali Ahmaditeshnizi, Wenzhi Gao, and Madeleine Udell. Optimus: Scalable optimization modeling with (mi)lp solvers and large language models. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235, pp. 577–596. PMLR, 2024.

Saverio Basso and Alberto Ceselli. A data driven dantzig–wolfe decomposition framework. *Mathematical Programming Computation*, 15:153–194, 2023. doi: 10.1007/s12532-022-00230-4.

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. doi: 10.1016/j.ejor.2020.07.063.

Martin Bergner, Alberto Caprara, Alberto Ceselli, Fabio Furini, Marco E. Lübbecke, Enrico Malaguti, and Emiliano Traversi. Automatic dantzig–wolfe reformulation of mixed integer programs. *Mathematical Programming*, 149(1–2):391–424, 2015. doi: 10.1007/s10107-014-0761-5.

Stephen Boyd, Neal Parikh, Eric Chu, Borja Peleato, and Jonathan Eckstein. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends in Machine Learning*, 3(1):1–122, 2011.

Shengkai Chen, Zhiguang Cao, Jianan Zhou, Yaoxin Wu, Senthilnath Jayavelu, Zhuoyi Lin, Xiaoli Li, and Shili Xiang. Dragon: Llm-driven decomposition and reconstruction agents for large-scale combinatorial optimization. In *Proceedings of the 25th International Conference on Autonomous Agents and Multiagent Systems*, 2026.

Bruno Rocha Colonetti. *Day-ahead Unit Commitment: Models and Algorithms*. Ph.d. thesis, Federal University of Santa Catarina, 2024.

Mitch Costley, Mohammad Javad Feizollahi, Shabbir Ahmed, and Santiago Grijalva. A rolling-horizon unit commitment framework with flexible periodicity. *International Journal of Electrical Power & Energy Systems*, 90:280–291, 2017. doi: 10.1016/j.ijepes.2017.01.026.

George B. Dantzig and Philip Wolfe. Decomposition principle for linear programs. *Operations Research*, 8(1):101–111, 1960. doi: 10.1287/opre.8.1.101.

Maxence Delorme, Manuel Iori, and Silvano Martello. BPPLIB: Falkenauer uniform bin-packing instances. `https://github.com/mdelorme2/BPPLIB`, 2016.

Maxime Gasse, Didier Chételat, Nicola Ferroni, Laurent Charlin, and Andrea Lodi. Exact combinatorial optimization with graph convolutional neural networks. In *Advances in Neural Information Processing Systems*, volume 32, 2019.

Shengran Hu, Cong Lu, and Jeff Clune. Automated design of agentic systems. In *International Conference on Learning Representations*, 2025.

IEEE PES Task Force on Benchmarks for Validation of Emerging Power System Algorithms. Power Grid Lib: Unit Commitment benchmark collection. https://github.com/power-grid-lib/pglib-uc, 2024. California and FERC instance provenance.

Connor Lawless, Yingxi Li, Anders Wikum, Madeleine Udell, and Ellen Vitercik. LLMs for cold-start cutting plane separator configuration. In *Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, volume 15763 of *Lecture Notes in Computer Science*, pp. 51–69. Springer Nature Switzerland, 2025. doi: 10.1007/978-3-031-95976-9_4.

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*, volume 235, pp. 32201–32223. PMLR, 2024.

Qi Liu, Ruochen Hao, Can Li, and Wanjing Ma. OR-Agent: Bridging evolutionary search and structured research for automated algorithm discovery. *arXiv preprint arXiv:2602.13769*, 2026.

Chris Lu, Cong Lu, Robert Tjarko Lange, Jakob Foerster, Jeff Clune, and David Ha. The ai scientist: Towards fully automated open-ended scientific discovery. *arXiv preprint arXiv:2408.06292*, 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 preprint arXiv:2606.23299*, 2026.

MATPOWER contributors. MATPOWER case13659pegase: European transmission-system test case. https://matpower.org/docs/ref/matpower6.0/case13659pegase.html, 2016.

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

Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Matej Balog, M. Pawan Kumar, Emilien Dupont, Francisco J. R. Ruiz, Jordan S. Ellenberg, Pengming Wang, Omar Fawzi, Pushmeet Kohli, and Alhussein Fawzi. Mathematical discoveries from program search with large language models. *Nature*, 625(7995):468–475, 2024. doi: 10.1038/s41586-023-06924-6.

UnitCommitment.jl contributors. UnitCommitment.jl benchmark collection: MATPOWER/PEGASE cases. https://anl-ceeesa.github.io/UnitCommitment.jl/0.3/instances/, 2024.

Jean-Paul Watson and David L. Woodruff. Progressive hedging innovations for a class of stochastic mixed-integer resource allocation problems. *Computational Management Science*, 8(4):355–370, 2011. doi: 10.1007/s10287-010-0125-4.

Alinson S. Xavier, Aleksandr M. Kazachkov, Ogün Yurdakul, Jun He, and Feng Qiu. Unitcommitment.jl: A julia/jump optimization package for security-constrained unit commitment (version 0.4), 2024.

Ziyang Xiao, Dongxiang Zhang, Yangjun Wu, Lilin Xu, Yuan Jessica Wang, Xiongwei Han, Xiaojin Fu, Tao Zhong, Jia Zeng, Mingli Song, and Gang Chen. Chain-of-experts: When LLMs meet complex operations research problems. In *International Conference on Learning Representations*, 2024.

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 *Advances in Neural Information Processing Systems*, volume 37, 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*, volume 267 of *Proceedings of Machine Learning Research*, pp. 72131–72180. PMLR, 2025.

Jiayi Zhang, Jinyu Xiang, Zhaoyang Yu, Fengwei Teng, Xiong-Hui Chen, Jiaqi Chen, Mingchen Zhuge, Xin Cheng, Sirui Hong, Jinlin Wang, Bingnan Zheng, Bang Liu, Yuyu Luo, and Chenglin Wu. Aflow: Automating agentic workflow generation. In *International Conference on Learning Representations*, 2025.

A Experimental Protocol and Evaluation Semantics

Strict feasibility. A candidate is reported as an original-MILP solution only if the saved assignment passes the untouched original model's row, bound, and integrality checks within numerical tolerances. Fractional RMP solutions, local block incumbents, stitched but violating assignments, and ADMM trajectories remain diagnostic artifacts.

Finite-time Gurobi reference. The Gurobi baseline is a finite-time incumbent produced under the recorded reference budget; it is not assumed to be a proven optimum. For a strict-feasible candidate with objective $f(x)$ and reference incumbent f_{ref}, we report

$$\Delta_{\mathrm{ref}}(\%) = 100\,\frac{f(x) - f_{\mathrm{ref}}}{|f_{\mathrm{ref}}|}. \tag{5}$$

A negative value only means that the method's finite-time incumbent is lower than the recorded reference incumbent. It is not a proof of optimality. We never compute Δ_{ref} for an infeasible method assignment. This quantity must not be confused with Gurobi's solver MIP gap, which measures the primal–dual gap within a single solve.

Runtime accounting. We report solver-centric and wall-clock quantities separately. *Key solve time* captures the expensive optimization stages that are comparable across runs: CA uses the sum of subproblem solver runtimes; FERC generator-axis runs use CG critical path plus the final original-MILP polish time; PEGASE uses CG critical path plus original-MIP polish solver runtime; and BinPacking uses CG critical path plus integer-RMP recovery. *Recorded wall time* is used only when the raw artifact preserves it. CA has end-to-end wall time for 11/16 non-seed runs, FERC generator-axis has end-to-end wall time for successful recovery runs, all 75 PEGASE non-seed recovery artifacts retain full end-to-end wall time, and BinPacking retains guarded per-instance operator wall. Missing wall values remain undefined rather than being reconstructed from solver counters. Appendix H.4 decomposes the recorded solver and wall quantities.

Reference-objective provenance. For the 11 CA validation rows, the original reference scalar was not stored directly in the historical ledger. The appendix therefore marks reference objectives reconstructed from the saved final objective and signed reference gap. The five CA held-out rows retain the saved reference scalar. The failed `Scenario400_reserves_5` run has no strict method objective and no recoverable reference-objective scalar in the supplied result package; both cells are left undefined. FERC generator-axis rows use the frozen 24-instance 3600-second/$\texttt{MIPGap} = 0$ Gurobi ledger, with exact instance-ID joins and an incumbent for every instance. The FERC time-axis stress report stores gaps and times but not the paired objective scalars, so it remains excluded from the per-instance objective tables below. The BinPacking instance-detail table retains the reference objective and runtime saved with the original method runs. For the main baseline comparison, all 16 non-seed BinPacking instances are instead paired by instance ID to the same independent 20260923 Gurobi 3600-second rerun; that rerun supplies both the denominator for every method's signed gap and the Gurobi wall-time cell. Its finite-time incumbents differ from the saved method-run references, so the two objective sources are reported separately.

B Dataset Details

Table 4 records the split protocol used by each reported result. The instance-level tables in Appendix H include the corresponding split role for every evaluated row.

CA. Seed discovery uses `2014-09-01_reserves_0`, `2014-12-01_reserves_1`, `2015-06-01_reserves_1`, and `2015-03-01_reserves_1`. The remaining 16 instances

Table 4: Dataset splits used by the reported experiments.

Dataset / protocol	Seed	Validation / non-seed	Gate / pool	Final test	Notes
CA historical frozen protocol	4	11	–	5	20 instances total
FERC `first_feasible` matrix	3	13	–	8	24 instances total
PEGASE reported frozen protocol	5	2	–	73	evidence protocol for reported result
BinPacking frozen pattern protocol	4	6	5	5	20 instances; main comparison uses 16 non-seed

form the 11-instance validation and 5-instance held-out populations listed in Table 10. The frozen operator uses four contiguous time blocks, one-period look-ahead, no carry of future values, and an instance-derived history depth. `Scenario400_reserves_5` is the single non-seed failure because a resource-type long-chain coupling makes a rollout block infeasible or unbounded under the frozen short-history treatment.

FERC. The 24-instance generator-axis matrix uses three seeds (`2015-01-01_hw`, `2015-04-01_lw`, and `2015-07-01_hw`), 13 non-seed instances, and eight held-out instances. The main cross-instance result uses the `first_feasible` stopping rule: stop immediately after the first coupling-feasible Phase-I iterate, with zero extra Phase-II pricing rounds and no `phase2_finalize` RMP. The `post_phase2_1` and `post_phase2_3` variants are retained only as an appendix stopping-policy ablation.

PEGASE (case13659pegase SCUC). The reported frozen-template evidence protocol uses five discovery seeds (`2017-12-30`, `2017-01-01`, `2017-01-03`, `2017-05-04`, and `2017-08-31`), two validation instances (`2017-03-15`, `2017-07-15`), and 73 chronological final-test instances listed in Table 13. The 75 non-seed rows (validation plus final test) are used consistently in Table 3; these 80 historical evidence instances do not exhaust the 365 daily source models. Other registry splits of the source pool are not used to recompute this 75-row aggregate.

BinPacking. The 20 `u500` instances are split into four seed-regression rows (`u500_07`, `u500_09`, `u500_12`, and `u500_18`), six dev rows, five gate rows, and five frozen final-test rows. The headline aggregate and all matched baselines use only the 16 dev/gate/final-test rows listed in Table 16; seed-regression rows are excluded from every method's average. The frozen operator starts the DW master from singleton feasible patterns, prices capacity-feasible knapsack patterns, certifies the no-negative-reduced-cost stopping condition, and obtains an integral packing through an integer-RMP recovery before strict original-model verification.

C Detailed Decomposition Priors and Validity Rules

This appendix expands the decomposition knowledge summarized in Section 3. The coupling patterns and solver primitives below are provided as priors to the agents; they restrict the search to meaningful decomposition mechanisms without fixing a single decomposition strategy in advance.

C.1 Coupling Priors

Given a variable partition $\mathcal{P}$, the block support $\mathcal{B}(i)$ in Eq. (3) determines whether a constraint is local or couples multiple blocks. For coupled constraints, DecompPilot exposes two recurring patterns to the agents.

Resource-sharing coupling. A resource-sharing constraint aggregates contributions from several blocks:

$$\sum_{b \in \mathcal{B}(i)} g_{ib}(x_b) \;\bowtie_i\; b_i. \tag{6}$$

To expose the block-wise contributions, we may introduce $y_{ib} = g_{ib}(x_b)$ and coordinate them through

$$\sum_{b \in \mathcal{B}(i)} y_{ib} \;\bowtie_i\; b_i. \tag{7}$$

Eliminating the auxiliary variables recovers the original constraint. This pattern captures, for example, system-wide demand and reserve constraints in unit commitment, as well as shared capacity or budget constraints in other MILPs.

Consensus coupling. Some decompositions require multiple blocks to maintain consistent representations of a shared decision. Introducing a global variable z, the corresponding relation can be written as

$$E_b x_b = z, \qquad b \in \mathcal{B}_z, \tag{8}$$

where $\mathcal{B}_z$ contains the blocks that share this information. For ordered decompositions, the same structure may appear as continuity between neighboring blocks, e.g., $E_b^+ x_b = E_{b+1}^- x_{b+1}$. This form naturally supports consensus-based coordination such as ADMM (Boyd et al., 2011).

The resource-sharing and consensus patterns are priors rather than an exhaustive classification. The harness may use row support, coefficient patterns, indexing structure, and semantic descriptions as evidence when identifying them. Regardless of the classification, the resulting representation must preserve the coefficients, constraint senses, and right-hand sides of the corresponding rows in the original MILP.

C.2 Decomposition Operator Priors

The operator $o = (a, \mathcal{P}, \mathcal{C}, \kappa, r, \eta)$ combines a partition with a coordination and recovery strategy. The current library provides three classical primitives that cover common structures encountered during search.

Rolling horizon. For an ordered partition, blocks are solved sequentially. Previously committed decisions provide history or boundary information to later blocks, while a look-ahead window can reduce boundary effects. This primitive is most natural when cross-block dependencies are mainly local along an ordered axis.

Dantzig–Wolfe / column generation. When the local regions X_b can be optimized independently and only a compact set of linking constraints remains, DW/CG coordinates the blocks through a restricted master problem. Block subproblems generate improving columns using the current master dual information. For MILPs, the resulting master solution can be fractional, so the operator may require an additional integer recovery or polishing stage. The full master and pricing formulations are given in Appendix F.

ADMM-style coordination. For consensus-type interfaces, each block solves its local problem while agreement is encouraged through shared variables and multiplier updates (Boyd et al., 2011). Because the subproblems in our setting may contain integer variables, we use ADMM as a coordination and trajectory-generation primitive rather than relying on its standard convex convergence guarantees. Its update equations and residual definitions are given in Appendix F.

These primitives are starting points rather than complete decomposition algorithms. During discovery, an operator may change the partition, coupling treatment, coordination primitive, recovery rule, or their parameters based on the problem structure and accumulated trial evidence.

C.3 Validity and Use in Search

Any decomposition proposed by an agent must remain consistent with the original MILP. In particular, every original variable belongs to exactly one owner block, every constraint is represented either locally or through a coupling interface, and cross-block representations must preserve all original terms. These checks are performed before expensive solver experiments whenever possible.

After subproblem execution and recovery, the assembled solution $\hat{x}$ is validated on the untouched original MILP:

$$\mathrm{Valid}(\hat{x}) = 1 \iff A\hat{x} \bowtie b, \qquad \ell \le \hat{x} \le u, \qquad \hat{x}_j \in \mathbb{Z}\ (j \in \mathcal{I}), \tag{9}$$

up to the numerical tolerances used by the verifier. Local incumbents, fractional master solutions, or stitched assignments that fail this check remain useful trial evidence, but are not treated as feasible solutions of the original MILP.

Candidate operators are therefore assessed first by construction validity and strict feasibility, and then by solution quality, runtime, residual behavior, and cross-instance stability. Reference solutions and objectives are reserved for evaluation and are not used to construct partitions, choose recovery values, or retrieve trial memory.

D Instance-Level Trial Protocol

D.1 One Trial from Input to Repair

Table 5 expands the execution summarized in Section 4.1. A trial is complete only when its terminal status and retained evidence can be interpreted without conversational context.

Table 5: One *Trial Memory Tree* trial. "Agent" stages propose typed scientific content; harness stages own side effects and acceptance decisions.

Stage	Consumed evidence	Produced evidence	Control condition
Plan proposal	MILP, semantic/index metadata, retrieved memory, parent diagnosis	L0–L4 plan, probes, operator hypothesis	all layer schemas and structural invariants pass
Materialization	authorized plan and selected operator parameters	isolated builders, model files, sidecars, command manifest	parameter completeness and pre-audit pass before a solver is launched
Numerical execution	local/global/master models and iteration state	solver statuses, solutions, bounds, residuals, timing, recovery provenance	only harness code invokes solvers and writes execution artifacts
Integration and evaluation	subproblem/master outputs	original-coordinate assignment, row/bound/integrality checks, normalized metrics	a solution is successful only if the untouched original model passes strict checks
Verifier loop	registered artifact index, strategy context, verifier memory	evidence-chain findings, target layers, concrete edits, validation tests	bounded read-only queries; unsupported claims and arbitrary paths are rejected
Experiment proposal	verifier findings, prior node outcomes, operator registry	executable candidate with parameters, hypothesis, expected effect, evidence refs	duplicate, unsupported, or incomplete candidates are rejected before tree insertion

D.2 Layer Contracts of the Master Agent

Table 6 records the abstraction carried by each Master layer. The "invariant" column is evaluated by deterministic code or a materialization probe rather than by self-report from the agent.

Table 6: Layered structural compilation. Later layers can revise an earlier decision only by returning to the indicated layer and regenerating all dependent artifacts.

Layer	Principal input	Structured output	Audited invariant
L0: model vocabulary	model statistics, semantic description, authoritative family inventories	candidate axes; variable/row families; anchored index extractors	selectors originate from the supplied inventory; every extractor has one axis token and matches observed names
L1: ownership	L0 map; observed axis domains	axis, block count, owner ranges, local and coupled row families	observed owner tokens are covered; blocks are nonempty; a local row contains only variables available in its block
L2: coupling lift	L1 blocks and row families; sampled row support	resource/consensus rule, stable term identities, row allocation, copied-variable set	every coupled row is allocated; resource contributions and consensus copies cover all original nonzeros; no invented family selector
L3: coordination	L2 interface and copies	global variables, multipliers, penalties, update equations, primal/dual residuals	identifiers agree across equalities, state tables, and updates; all interface equalities have initialized state and a recorded penalty mode
L4: executable builders	L0–L3 artifacts and aggregated state tables	block builder, global/master builder, sidecar manifests	builders compile and materialize; required columns are checked by name; written models and sidecars are traceable to the input plan

The dependency relation also defines repair scope. A wrong row-family classification restarts at L1 or L2; an incorrect auxiliary-variable identity restarts at L2 or L3; and a builder defect with unchanged mathematical intent may be repaired at L4. Verifier reports identify both the target layers and the earliest restart layer, preventing a local code patch from concealing an upstream mathematical inconsistency.

D.3 Evidence Isolation and Compact Memory

The controller is the sole writer of the tree index. Node v is assigned a stable identifier and a private namespace containing: (i) its parent and hypothesis; (ii) the modification and behavior fingerprints; (iii) the materialized plan and generated modules; (iv) run, solver, integrator, and verifier directories; (v) normalized metrics and score; and (vi) terminal failure classification or success type. Workers may create outputs only in the assigned node namespace and return their manifests to the controller. This preserves causal lineage under parallel execution and makes a resumed search observationally equivalent to one that did not lose its conversational context.

The tree index is a compressed retrieval view, not the evidentiary record itself. It retains parent edges, status, short hypothesis, operator identity, fingerprints, score, and a metrics summary. Verifier memory similarly stores de-duplicated recommendations, contraindications, prior outcomes, and short experience entries. Full CSV, JSON, LP, solution, and trace files remain referenced by path. Consequently, later agents receive bounded context while every summarized claim remains traceable to immutable trial artifacts.

D.4 Reading the Trial-Tree Case Studies

Each rectangle in Figure 4 and Figure 5 represents one retained experiment or a labeled group of experiments. The screening and retrieval boxes denote compressed context rather than additional trials. Solid arrows preserve the decision-relevant parent–child relation; dashed arrows lead to diagnostic, rejected, or repeated branches. Green nodes pass the strict original-model check, purple nodes are near-feasible, orange nodes mark inferior feasible alternatives (except the explicitly infeasible `N0015`), and gray nodes carry negative or non-executable evidence. The colors summarize recorded status rather than replace the verification artifacts.

The two compact labels in Table 7 denote `2014-09-01_reserves_0` and `2015-03-01_reserves_1`, respectively.

D.5 Promotion, Retrieval, and Repetition Control

An operator outcome is appended once per instance–node–operator key and may be reconciled if a stricter audit changes its status. A reusable L4 template is registered only when execution is not an infrastructure/solver failure, the directly assembled assignment is feasible on the original model, and the outcome is classified as a direct-template success. Its card contains structural applicability information and aggregate metrics, not solution values.

For a new instance, retrieval ranks prior trials using overlap between variable/constraint-family fingerprints and a success bonus. The current instance is excluded by identifier. Retrieved parameters are hypotheses rather than executable defaults: the current model must reconstruct L0–L3, and the candidate becomes invalid if current-instance evidence cannot supply required fields. Before inserting a child, the tree checks the exact modification fingerprint and a normalized behavior fingerprint under the same parent. At the induction level, the candidate semantic hash ignores evidence provenance such as seed IDs but includes executable behavior and applicability conditions. These two checks prevent repeated exploration at both the local and cross-instance levels.

E Cross-Instance Induction Protocol

E.1 State-Controlled Lifecycle

The persistent lifecycle is

$$\begin{aligned}\textsc{DataPrep} \rightarrow \textsc{SeedDiscovery} \rightleftarrows \textsc{OperatorAbstraction} \\ \rightleftarrows \textsc{NonSeedValidation} \rightarrow \textsc{CandidateFrozen} \rightarrow \textsc{FinalTest}.\end{aligned} \tag{10}$$

with a return from validation to seed discovery when structural coverage is insufficient. Final-test implementation defects may enter an implementation-repair/full-regression loop, but a semantic change requires a new candidate and protocol. All nonterminal phases may pause; resumption returns only to the recorded source phase.

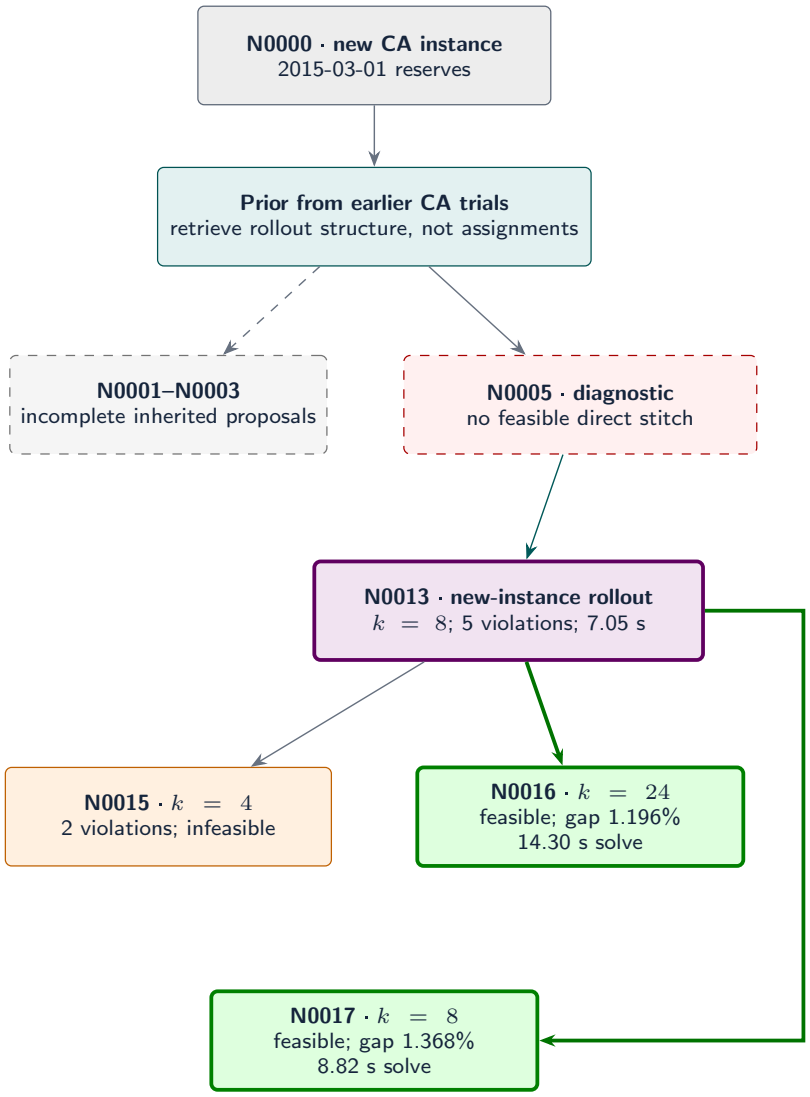

Figure 5: Abridged memory-guided search on CA instance `2015-03-01_reserves_1`. Historical rollout evidence supplies a hypothesis, but incomplete inherited candidates are rejected before execution. The verified current-instance refinements `N0016` and `N0017` trade objective quality against solver time (14.30 versus 8.82 s) and recorded wall time (844.7 versus 230.2 s).

Table 8 groups the legal actions used in the production state machine. The complete implementation contains finer-grained wait/join and inspection actions, but all belong to one of these scientific transitions.

E.2 Validation and Data Visibility

The data roles are disjoint. `seed_initial` supports full trial-tree search; `validation_dev` supports repeated refinement; `validation_gate` is fresh until its failed evidence is explicitly consumed; `adaptive_train_pool` is revealed in batches; and `final_test` is unavailable until the operator is frozen. Training agents and retrieval tools can access only the first four roles. A gate failure used for revision is marked consumed, and the revised candidate must establish new validation evidence. Freeze requires an executable artifact, complete training coverage, a successful full regression for the current candidate and implementation revisions, and consistent semantic, implementation, and split hashes.

Table 7: Expanded reading guide for the two simplified trial-memory trees. Node identifiers are stable artifact identifiers; a range denotes a set of individual persisted trials that is visually collapsed in the main paper.

Case and nodes	Role in the search	Retained evidence	Transition to the next group
CA-2014-09-01: N0000–N0008	Root and eight coarse-screening trials	Four partition granularities are evaluated with a diagnostic coordination style and a constructive rollout style. N0007 has only 10 startup violations with 4.90 s key solve time.	Keep the low-residual rollout direction; retain diagnostics as negative evidence rather than treating them as feasible solutions.
CA-2014-09-01: N0009–N0019	Targeted refinement and comparison	The search tests partition granularity, retained state, and variable-family choices. Some alternatives are feasible but have poor objective quality; N0019 leaves only eight startup violations with a -0.074% reference difference.	Inspect the remaining violation offsets instead of merely repeating the same shallow retained history.
CA-2014-09-01: N0020–N0028	Rejected proposals, repetition check, and final repair	Incomplete proposals and redundant coverage checks are preserved but not executed as valid strategies. N0027 repeats the shallow-history outcome; N0028 uses the observed three-period offset span.	N0028 is directly feasible with zero violations, objective 48,740.804, and 5.58 s key solve time.
CA-2015-03-01: N0000–N0012	Retrieval, schema checks, and initial screening	Historical rollout experience is retrieved as a candidate prior. Proposals missing required current-instance fields are rejected before numerical execution; diagnostic branches remain available as negative evidence.	Rebuild the plan on the new instance and execute the first parameter-complete rollout.
CA-2015-03-01: N0013–N0015	Current-instance near-feasible evidence	N0013 (eight blocks) has five remaining violations in 7.05 s. Other granularity trials reduce but do not eliminate violations.	Branch on the concrete trade-off: finer partitioning versus deeper retained history.
CA-2015-03-01: N0016–N0017	Independently verified candidates	N0016 uses 24 blocks and is directly feasible (1.196% gap, 14.30 s key time, 844.7 s recorded wall). N0017 uses eight blocks with deeper history and is directly feasible (1.368% gap, 8.82 s key time, 230.2 s recorded wall).	Both are retained as locally verified outcomes; the later abstraction step can select according to an explicit quality–time criterion.

Table 8: Phase-conditioned action families and their evidence requirements.

Phase	Legal action families	Required evidence	Possible scientific transition
Data preparation	scan, fingerprint, split, validate, freeze manifest	mutually exclusive instance roles and a stable split hash	seed discovery
Seed discovery	retrieve memory, dispatch/wait/join, expand tree, diagnose, add seed, retry infrastructure	terminal trial manifests, success/failure registry, coverage summary	abstraction or further seed search
Operator abstraction	abstract, pre-audit, revise, branch, request evidence, reject	whitelisted seed records and current diagnosis/advice references	validation or return to seed
Non-seed validation	run batch, evaluate gate, full regression, diagnose, targeted experiment, revise/branch, expand seed, freeze	revision-matched dev/gate/pool results and complete training coverage	abstraction, seed discovery, or freeze
Frozen candidate / final test	verify freeze, run/wait test, classify failure, finalize	semantic/implementation/split hashes and isolated test manifest	done or implementation repair
Implementation repair	patch, contract check, verify unchanged semantic hash, full training regression	failure classified as implementation/environmental and regression evidence	new final-test round or continued repair

E.3 Diagnosis, Advice, and Repair Authority

Validation failures are diagnosed per instance and stored as immutable reports. The diagnosis includes failure class, direct evidence references, and the affected mechanism. A separate advice module combines reports from the same candidate revision into possible repairs or evidence-gathering actions. Advice has no execution authority. The Orchestrator observes a bounded view containing the current candidate, recent failures, diagnoses, advice, targeted-experiment results, and the phase action mask;

it selects one legal action. The router checks its typed payload and data permissions, and the reducer is the only component that commits the resulting event to persistent state.

A semantic revision changes the candidate version and semantic hash and makes earlier gate/regression evidence stale. An implementation-only repair increments the implementation revision while preserving the semantic hash. The latter is permitted after final-test triage only if a contract check and a full training regression succeed before testing resumes. Test outcomes are never returned to the abstraction observation as tuning evidence.

F Classical Decomposition Primitives

This appendix specifies the optimization primitives exposed through the operator library and fixes the notation needed to reproduce their use within the harness.

F.1 History-Aware Rolling Horizon

Let ordered owner blocks be $B_1, \dots, B_K$. For block $B_s = [t_s, t_e]$, the local solve window is expanded to

$$W_s = [t_s - h_s,\ t_e + \ell_s], \tag{11}$$

where h_s is a history depth and ℓ_s is lookahead. Values from committed history are fixed inside W_s; only variables whose owner time lies in B_s are committed to the global stitched assignment. Under *no carry*, values in the lookahead portion are discarded and re-optimized in the next block. The CA operator uses $K = 4$, $\ell = 1$, and an instance-derived history depth.

F.2 Consensus ADMM

For the lifted formulation $F_b x_b = y_b$ with $y \in \mathcal{Y}$, a generic scaled ADMM iteration is

$$x_b^{k+1} \in \arg\min_{x_b \in X_b} f_b(x_b) + \frac{\rho_b}{2}\|F_b x_b - y_b^k + u_b^k\|_2^2, \tag{12}$$

$$y^{k+1} \in \arg\min_{y \in \mathcal{Y}} \sum_b \frac{\rho_b}{2}\|F_b x_b^{k+1} - y_b + u_b^k\|_2^2, \tag{13}$$

$$u_b^{k+1} = u_b^k + F_b x_b^{k+1} - y_b^{k+1}. \tag{14}$$

For resource rows, $\mathcal{Y}$ contains Eq. (7); for consensus rows, it enforces the common z values in Eq. (8). The implementation records $r_b^k = F_b x_b^k - y_b^k$ and the change in global variables as primal and dual diagnostics. With mixed-integer X_b, these trajectories are evaluated empirically and may provide stable-variable evidence for a subsequent original-MILP recovery.

F.3 Dantzig–Wolfe / Column Generation

Let block b have feasible extreme points or integer trajectories x_{bk}. The restricted master problem (RMP) uses convex-combination variables θ_{bk}:

$$\min_{\theta} \quad \sum_b \sum_k c_b^\top x_{bk}\, \theta_{bk} \tag{15}$$

$$\text{s.t.} \quad \sum_b \sum_k D_b x_{bk}\, \theta_{bk} \le d, \tag{16}$$

$$\sum_k \theta_{bk} = 1 \quad \forall b, \qquad \theta_{bk} \ge 0. \tag{17}$$

Given coupling duals π and convexity dual σ_b, the pricing problem for block b minimizes reduced cost

$$(c_b - D_b^\top \pi)^\top x_b - \sigma_b, \qquad x_b \in X_b. \tag{18}$$

The harness checks consistency between native column cost, dual contribution, and the RMP reduced cost before accepting new columns. Fractional RMP solutions are not original-MILP solutions. Recovery uses same-instance trajectory agreement or theta-weighted voting as a MIP start, optionally with a capped fixing rule selected by the search.

G Discovered Operator Templates

Table 9: Frozen operator structures; all numerical recovery inputs are derived from the current instance.

Dataset	Partition / representation	Coordination and recovery
CA	Four contiguous time blocks	Sequential rollout with one-period look-ahead and instance-derived fixed history; stitch owned periods.
FERC	Two generator-owned blocks	DW/CG on shared system rows; cap consensus-based binary fixing and polish the original MILP.
PEGASE	Eight generator-owned blocks	DW/CG with system-row master; current-instance RMP votes form a MIP start, followed by original-MIP polish without hard fixing.
BinPacking	Capacity-feasible item patterns	Set-covering master, knapsack pricing, and integer-master recovery from newly generated patterns.

The following code-like descriptions show the frozen high-level operators. They intentionally omit instance-specific variable values and reference objectives.

G.1 CA: General Horizon-Rollout Operator

```
operator: ca_history_aware_horizon
axis: ordered_time
partition: 4_contiguous_blocks
lookahead: 1
carry_future_values: false
fixed_history_families: [dg, pg, rg, ug, vg, wg]
history_depth(instance):
    max(offset_policy(instance),
        startup_allowed_history_span(instance))
commit: owner_block_only
verify: strict_original_milp
```

This template is derived from four seed instances. The computed history depth is 13 for the reported batch, but 13 is not hard-coded as a universal constant.

G.2 FERC Time Axis: Stable-Trajectory Recovery Operator

```
operator: ferc_resource_aware_stable_polish
axis: ordered_time
trajectory_source: binary_ADMM + resource_aware_ADMM
hard_fix:
    old_stable_binaries: top_stable_subset
    resource_stable_zero: top_1_percent
mip_start: all_available_stable_binaries
recovery_model: untouched_original_milp
polish: {method: 2, mip_focus: 2, budget_s: 65}
verify: strict_original_milp
```

On the seed instance this gives 182.286 seconds total time and 0.519% signed objective difference from the finite-time reference in Table 2. Increasing stable-zero fixing to 3% produces a dramatically worse incumbent, providing negative evidence against aggressive fixing.

G.3 FERC Generator Axis: DW/CG + Capped Consensus Polish

```
operator: ferc_generator_dw_cg_consensus
axis: generator
partition: 2_contiguous_generator_blocks
cg:
```

```
    cold_start: [high, low]
    phase_I_rounds: 3
    pricing_workers: 1
    pricing_threads: 4
recovery:
    source: fractional_RMP_integer_consensus
    confidence: 0.99
    hard_fix: agreed_zero
    hard_fix_cap: 5000
    mip_start: all_consensus_integers
    original_milp_polish_s: 57
verify: strict_original_milp
```

The corresponding strict-feasible solution has 0.510% signed reference difference and 180.069 seconds total time in Table 2. Under a matched recovery budget, $k = 4$ remains competitive in time but has 0.679% reference difference, whereas $k = 6$ degrades to 4.480%.

G.4 case13659pegase: Frozen Generator-Axis Template

```
operator: pegase_generator_dw_cg_template
axis: generator_ownership
partition: 8_contiguous_generator_blocks
master: LP_RMP_with_cross_block_system_rows
pricing: pure_local_MILP
cg_rounds: 3
recovery:
    source: final_RMP_theta_vote
    hard_fix: none
    mip_start: same_instance_integer_vote
    original_mip_polish_s: 600
verify: strict_original_mip
cross_instance_memory:
    transfer: [structure, operator_schema, gates, aggregate_metrics]
    never_transfer: [solution_values, mip_start_values, incumbents]
```

This conservative zero-hard-fixing rule is itself a discovered outcome: full high-confidence hard fixing over 1.1M variables is strict-feasible but worsens the seed objective by more than 53%.

G.5 BinPacking: Frozen Pattern DW/CG Template

```
operator: binpacking_adaptive_pattern_dw_cg
axis: item_coverage_by_feasible_patterns
initial_columns: one singleton pattern per item
master: LP_set_covering_RMP
pricing: capacity_feasible_knapsack
column_policy: add_negative_reduced_cost_patterns
stop: no_negative_reduced_cost
recovery:
    source: final_generated_pattern_pool
    method: integer_RMP
verify: strict_original_mip
cross_instance_memory:
    transfer: [operator_schema, pricing_rule, stopping_rule]
    never_transfer: [instance_patterns, incumbent_values]
```

The reported operator is frozen before the 16 non-seed evaluations. Generated pattern columns and integer-RMP incumbents remain instance-specific and are not transferred across instances.

H Detailed Results

The following tables report every instance used in the four headline cross-instance evaluations: 16 CA non-seed instances, 24 FERC generator-axis matrix instances, 75 PEGASE non-seed evidence instances, and 16 BinPacking non-seed instances (131 rows total). "Ref. gap" is Δ_{ref} from Appendix A. Dashes indicate values that are either not applicable or not preserved by the source artifacts; they are never imputed as zero.

H.1 CA: All 16 Non-Seed Instances

The CA reference-objective provenance is shown explicitly because the historical validation ledger did not store the scalar reference objective. "recon." denotes a value reconstructed from the saved method objective and signed reference gap; "saved" denotes a directly retained reference scalar. End-to-end method wall time exists only for the 11 validation runs.

Table 10: CA frozen time-axis operator on all 16 non-seed instances. Solver work is the sum of subproblem Gurobi runtimes; wall is recorded end-to-end method wall time when available.

Instance	Split	Strict	Method obj.	Ref. obj.	Ref. gap (%)	Solver work (s)	Wall (s)	Gurobi ref. (s)	Ref. source
2014-09-01_reserves_1	Val.	✓	48,414.72	48,289.83	+0.2586%	41.65	103.91	63.15	recon.
2014-09-01_reserves_3	Val.	✓	48,510.25	48,412.55	+0.2018%	59.05	120.29	82.66	recon.
2014-12-01_reserves_0	Val.	✓	39,359.33	39,234.26	+0.3188%	11.65	73.72	50.20	recon.
2014-12-01_reserves_3	Val.	✓	39,490.84	39,389.35	+0.2577%	30.64	92.10	42.78	recon.
2014-12-01_reserves_5	Val.	✓	39,594.77	39,614.28	-0.0492%	55.06	116.01	74.98	recon.
2015-03-01_reserves_0	Val.	✓	31,982.46	31,785.49	+0.6197%	11.86	75.84	51.68	recon.
2015-03-01_reserves_5	Val.	✓	32,061.58	31,987.63	+0.2312%	27.46	85.46	74.19	recon.
2015-06-01_reserves_0	Val.	✓	41,818.02	41,712.53	+0.2529%	11.16	63.97	24.42	recon.
2015-06-01_reserves_3	Val.	✓	41,963.72	41,886.79	+0.1837%	38.46	95.75	70.26	recon.
2015-06-01_reserves_5	Val.	✓	42,069.39	42,208.47	-0.3295%	28.35	82.34	66.53	recon.
Scenario400_reserves_5	Val.	✗	–	–	–	10.14	30.26	81.36	missing
2014-09-01_reserves_5	Held-out	✓	48,649.55	48,549.46	+0.2062%	40.03	–	67.59	saved
2015-03-01_reserves_3	Held-out	✓	32,026.90	31,935.11	+0.2874%	31.78	–	67.43	saved
Scenario400_reserves_0	Held-out	✓	33,648.93	33,577.78	+0.2119%	17.02	–	46.87	saved
Scenario400_reserves_1	Held-out	✓	33,686.38	33,606.50	+0.2377%	30.84	–	42.31	saved
Scenario400_reserves_3	Held-out	✓	33,842.70	34,060.26	-0.6388%	41.48	–	77.21	saved

The single failed row, `Scenario400_reserves_5`, terminates with an infeasible-or-unbounded rollout block. No strict method objective exists, and the supplied historical artifact does not retain the paired reference-objective scalar for that failed row; the objective cells therefore remain undefined rather than being reconstructed without evidence.

H.2 FERC Generator Axis: All 24 `first_feasible` Instances

For strict-feasible rows, FERC key solve time is CG critical path plus final polish wall. If Phase I never becomes coupling feasible, recovery is not executed; the table then reports the executed CG critical path as the available key-solver work and leaves end-to-end recovery wall undefined. All six failures in this matrix are `phase_I_not_coupling_feasible`. Reference objective and runtime come from the common frozen 3600-second, one-thread, zero-relative-gap Gurobi ledger; all 24 reference rows have an incumbent.

Table 11: FERC generator-axis `first_feasible` matrix. Key solve is CG critical path plus polish time when the recovery stage is executed.

Instance	Split	Strict	Method obj.	Ref. obj.	Ref. gap (%)	CG crit. (s)	Polish (s)	Key solve (s)	E2E wall (s)	Gurobi ref. (s)
2015-01-01_hw	Seed	✓	41,665,182.40	41,482,588.83	+0.4402%	25.86	129.09	154.95	242.80	3600.15
2015-01-01_lw	Non-seed	✓	85,007,374.02	84,786,481.31	+0.2605%	140.75	104.56	245.32	443.96	534.18
2015-02-01_hw	Held-out	✓	47,545,924.48	46,144,730.13	+3.0365%	12.07	127.86	139.94	213.16	1338.79
2015-02-01_lw	Held-out	✓	97,515,479.75	97,232,799.05	+0.2907%	148.15	132.22	280.37	488.33	1108.46
2015-03-01_hw	Non-seed	✓	69,993,788.30	68,486,602.78	+2.2007%	24.05	128.05	152.09	237.31	3600.36
2015-03-01_lw	Non-seed	✗	–	103,712,980.89	–	172.51	–	172.51	–	1026.53
2015-04-01_hw	Non-seed	✓	22,123,937.70	20,954,818.48	+5.5792%	11.51	128.25	139.76	212.09	915.94
2015-04-01_lw	Seed	✓	57,095,251.88	57,000,440.33	+0.1663%	145.67	97.92	243.59	451.83	3190.95
2015-05-01_hw	Non-seed	✓	39,482,986.39	36,647,066.84	+7.7385%	34.86	111.64	146.50	241.63	407.05
2015-05-01_lw	Non-seed	✗	–	46,639,937.17	–	171.30	–	171.30	–	663.57
2015-06-01_hw	Held-out	✓	50,586,282.63	50,575,965.58	+0.0204%	35.54	78.55	114.09	211.96	514.46
2015-06-01_lw	Held-out	✗	–	61,336,177.35	–	169.77	–	169.77	–	1992.28
2015-07-01_hw	Seed	✓	55,167,547.58	55,085,628.60	+0.1487%	35.35	124.66	160.01	258.09	3600.40
2015-07-01_lw	Non-seed	✗	–	70,696,645.00	–	168.73	–	168.73	–	2061.18
2015-08-01_hw	Non-seed	✓	51,963,143.53	49,787,045.61	+4.3708%	23.33	128.69	152.02	237.21	3600.43
2015-08-01_lw	Non-seed	✗	–	81,583,261.43	–	168.56	–	168.56	–	1288.97
2015-09-01_hw	Non-seed	✓	92,096,550.27	91,985,328.28	+0.1209%	87.62	130.09	217.71	367.20	3600.41
2015-09-01_lw	Non-seed	✗	–	115,437,877.34	–	175.87	–	175.87	–	3600.17
2015-10-01_hw	Held-out	✓	14,406,655.89	13,726,274.42	+4.9568%	11.06	128.22	139.29	212.36	2174.48
2015-10-01_lw	Held-out	✓	61,160,855.16	61,149,681.93	+0.0183%	109.18	102.43	211.61	381.38	510.28
2015-11-02_hw	Non-seed	✓	55,732,162.80	49,854,013.13	+11.7907%	36.15	128.28	164.43	260.70	958.64
2015-11-02_lw	Non-seed	✓	66,714,144.54	66,710,708.22	+0.0052%	126.34	103.81	230.14	415.79	400.73
2015-12-01_hw	Held-out	✓	40,856,934.76	36,846,314.18	+10.8847%	12.21	127.75	139.96	213.45	632.89
2015-12-01_lw	Held-out	✓	76,400,299.09	76,340,397.34	+0.0785%	122.21	79.55	201.76	384.50	579.67

Stopping-policy ablation. The main paper uses `first_feasible`. The two alternatives continue for one or three Phase-II pricing rounds after first coupling feasibility and include one no-pricing `phase2_finalize` RMP. They do not increase the number of strict-feasible instances in this 24-instance matrix.

Table 12: Aggregate FERC stopping-policy ablation over the same 24 instances. Gap and timing means are over strict-feasible runs.

Stop rule	Strict	Mean gap	Median gap	CG crit. (s)	Key solve (s)	E2E wall (s)
`first_feasible`	18/24	2.895%	0.365%	63.44	179.64	304.10
`post_phase2_1`	18/24	2.538%	0.379%	77.45	194.62	332.77
`post_phase2_3`	18/24	11.532%	0.070%	105.79	221.76	387.07

H.3 CASE13659PEGASE: ALL 75 NON-SEED EVIDENCE INSTANCES

All 75 rows (two validation and 73 frozen final-test instances) are strict feasible. The table reports both solver-centric key solve time and the recovered full end-to-end wall time. Key solve is CG critical path plus original-MIP polish solver runtime; end-to-end wall additionally includes loading/building, initial-column generation, trajectory preparation, verification, and other measured runner overhead. The mean terminal Gurobi solver MIP gap after polish is 1.090045%, which is separate from the reference-objective gap shown per row.

Table 13: Frozen case13659pegase generator-axis operator on all 75 non-seed evidence instances. CG work wall and full end-to-end wall are separately retained source measurements.

Instance	Split	Strict	Method obj.	Ref. obj.	Ref. gap (%)	CG crit. (s)	CG work wall (s)	Polish (s)	Key solve (s)	E2E wall (s)	Gurobi ref. (s)
2017-01-02	Final test	✓	190,147,992.14	189,896,070.28	+0.1327%	27.05	282.94	600.70	627.75	1009.40	3601.46
2017-01-07	Final test	✓	240,732,789.43	240,366,608.98	+0.1523%	30.36	287.24	600.87	631.23	1014.39	3601.22
2017-01-12	Final test	✓	224,511,583.91	224,406,638.53	+0.0468%	30.79	300.72	602.67	633.46	1034.28	3601.22
2017-01-17	Final test	✓	235,234,978.67	234,687,768.08	+0.2332%	26.60	288.57	600.72	627.32	1008.97	3601.58
2017-01-22	Final test	✓	186,698,931.01	186,809,647.42	-0.0593%	29.97	282.27	600.78	630.75	1005.37	3602.23
2017-01-27	Final test	✓	206,139,266.28	205,798,549.77	+0.1656%	26.66	277.59	601.09	627.75	998.73	3601.60
2017-02-01	Final test	✓	204,993,838.85	204,667,431.13	+0.1595%	27.81	286.92	600.67	628.48	1008.43	3601.00
2017-02-06	Final test	✓	202,420,037.86	202,183,345.39	+0.1171%	31.57	300.06	601.11	632.68	1024.67	3600.79
2017-02-11	Final test	✓	211,664,510.45	211,403,539.22	+0.1234%	28.99	287.71	600.68	629.67	1012.09	3601.13
2017-02-16	Final test	✓	227,486,272.31	227,290,434.71	+0.0862%	29.81	292.38	601.18	631.00	1014.57	3600.89
2017-02-21	Final test	✓	177,442,235.68	177,106,451.19	+0.1896%	28.83	286.69	600.96	629.79	1009.99	3601.21
2017-02-26	Final test	✓	164,462,153.55	163,699,431.50	+0.4659%	28.84	286.85	600.82	629.66	1010.08	3600.82
2017-03-03	Final test	✓	189,887,947.77	189,221,980.86	+0.3520%	28.70	293.25	600.68	629.39	1018.29	3600.79
2017-03-09	Final test	✓	144,432,239.54	144,359,537.02	+0.0504%	28.58	287.18	600.72	629.30	1006.99	3600.62
2017-03-14	Final test	✓	186,848,470.63	185,698,833.30	+0.6191%	28.89	285.91	600.67	629.56	1005.84	3600.81
2017-03-15	Validation	✓	197,545,645.41	197,016,738.26	+0.2685%	33.17	297.43	600.87	634.04	1019.16	3600.63
2017-03-19	Final test	✓	151,153,957.30	150,678,067.72	+0.3158%	29.98	284.75	601.61	631.58	1005.63	3600.74
2017-03-24	Final test	✓	156,052,335.03	155,911,222.28	+0.0905%	28.23	283.59	601.14	629.38	1004.00	3600.83
2017-03-29	Final test	✓	149,391,831.51	149,327,367.94	+0.0432%	28.39	285.31	600.76	629.14	1006.56	3600.66
2017-04-03	Final test	✓	154,090,025.70	153,867,658.62	+0.1445%	29.30	280.17	600.61	629.91	1003.93	3600.71
2017-04-08	Final test	✓	149,834,441.69	149,837,314.49	-0.0019%	28.82	285.76	600.69	629.51	1005.98	3600.69
2017-04-13	Final test	✓	153,235,502.86	153,119,918.95	+0.0755%	29.29	287.10	600.74	630.03	1007.25	3601.14
2017-04-18	Final test	✓	143,124,603.11	142,962,429.05	+0.1134%	40.46	275.58	601.11	641.57	998.64	3600.93
2017-04-23	Final test	✓	130,902,002.76	130,850,202.75	+0.0396%	40.00	271.91	600.73	640.74	991.85	3600.88
2017-04-28	Final test	✓	166,159,130.38	165,577,479.45	+0.3513%	34.38	266.16	600.83	635.20	991.66	3600.72
2017-05-03	Final test	✓	154,356,518.87	154,265,447.79	+0.0590%	32.36	265.37	600.70	633.07	992.11	3600.66
2017-05-08	Final test	✓	158,460,581.99	158,444,197.44	+0.0103%	29.50	280.87	600.69	630.19	1001.98	3600.62
2017-05-13	Final test	✓	134,319,065.22	134,309,102.19	+0.0074%	27.23	294.38	600.80	628.02	1017.40	3601.08
2017-05-18	Final test	✓	236,054,424.14	234,329,949.65	+0.7359%	28.94	279.11	601.22	630.16	1001.38	3600.71
2017-05-23	Final test	✓	141,484,375.97	141,477,509.78	+0.0049%	27.55	283.19	600.68	628.24	1003.85	3601.11
2017-05-28	Final test	✓	123,767,641.58	123,743,213.17	+0.0197%	28.82	282.55	600.63	629.45	1003.70	3601.20
2017-06-02	Final test	✓	168,866,074.13	168,607,774.16	+0.1532%	29.90	285.98	600.70	630.61	1006.52	3600.65
2017-06-07	Final test	✓	142,321,973.94	142,300,154.42	+0.0153%	31.14	280.02	601.50	632.64	1006.73	3601.12
2017-06-12	Final test	✓	274,869,394.64	274,304,367.90	+0.2060%	30.05	288.28	600.95	631.00	1014.19	3600.63
2017-06-17	Final test	✓	188,352,268.58	187,836,182.85	+0.2748%	28.84	285.31	600.87	629.71	1017.68	3600.78
2017-06-22	Final test	✓	243,562,727.69	243,032,486.83	+0.2182%	27.79	287.47	600.68	628.47	1014.04	3600.71
2017-06-27	Final test	✓	182,273,583.54	181,243,259.56	+0.5685%	30.65	291.22	600.80	631.46	1016.13	3601.29
2017-07-03	Final test	✓	212,302,088.25	211,112,659.34	+0.5634%	28.73	292.90	600.93	629.66	1020.67	3601.03
2017-07-08	Final test	✓	223,351,210.01	222,379,955.05	+0.4368%	29.67	286.24	600.81	630.48	1008.92	3601.06
2017-07-13	Final test	✓	281,389,625.86	279,571,218.13	+0.6504%	28.64	281.11	603.43	632.08	1005.87	3601.01
2017-07-15	Validation	✓	225,940,085.50	224,881,829.43	+0.4706%	31.28	287.01	601.12	632.39	1012.20	3601.16
2017-07-18	Final test	✓	238,698,665.03	237,296,705.62	+0.5908%	28.60	283.65	600.88	629.48	1005.54	3601.23

Continued on next page

Table 13 – continued

Instance	Split	Strict	Method obj.	Ref. obj.	Ref. gap (%)	CG crit. (s)	CG work wall (s)	Polish (s)	Key solve (s)	E2E wall (s)	Gurobi ref. (s)
2017-07-23	Final test	✓	233,536,538.00	232,519,491.32	+0.4374%	29.51	295.84	600.75	630.26	1018.87	3601.17
2017-07-28	Final test	✓	221,214,431.33	220,242,728.74	+0.4412%	31.81	298.34	600.93	632.75	1035.39	3601.19
2017-08-02	Final test	✓	249,782,077.12	248,378,059.19	+0.5653%	28.50	281.54	600.69	629.20	1003.61	3601.09
2017-08-07	Final test	✓	209,251,767.07	208,351,431.66	+0.4321%	30.04	285.61	600.72	630.76	1007.16	3601.11
2017-08-12	Final test	✓	203,000,574.89	202,053,092.69	+0.4689%	29.62	292.76	600.69	630.31	1018.50	3600.77
2017-08-17	Final test	✓	241,945,432.69	240,855,932.44	+0.4523%	29.17	287.87	600.82	629.99	1013.30	3601.10
2017-08-22	Final test	✓	268,750,952.14	267,161,510.65	+0.5949%	29.61	288.63	600.89	630.50	1015.39	3601.13
2017-08-27	Final test	✓	162,939,156.02	162,586,249.64	+0.2171%	28.40	288.79	600.74	629.14	1014.24	3601.18
2017-09-01	Final test	✓	171,838,770.78	171,357,277.41	+0.2810%	28.04	280.50	600.76	628.80	1013.42	3600.61
2017-09-06	Final test	✓	175,500,450.29	175,482,317.68	+0.0103%	28.79	283.26	600.81	629.60	1010.73	3601.07
2017-09-11	Final test	✓	163,493,614.41	163,474,919.25	+0.0114%	29.71	287.91	600.64	630.35	1010.03	3601.15
2017-09-16	Final test	✓	185,581,839.16	185,365,637.65	+0.1166%	31.08	286.53	600.70	631.78	1012.52	3601.23
2017-09-21	Final test	✓	242,024,814.94	240,695,911.38	+0.5521%	29.75	288.49	600.79	630.54	1011.01	3601.22
2017-09-26	Final test	✓	231,465,759.51	230,502,126.43	+0.4181%	29.86	292.73	600.77	630.64	1017.46	3600.98
2017-10-01	Final test	✓	126,486,807.34	126,377,505.26	+0.0865%	29.89	290.13	600.74	630.63	1016.69	3601.11
2017-10-06	Final test	✓	178,263,326.72	178,054,115.60	+0.1175%	29.94	284.01	600.89	630.83	1009.78	3600.62
2017-10-11	Final test	✓	169,650,089.21	169,578,618.43	+0.0421%	29.55	283.32	600.69	630.24	1009.03	3600.66
2017-10-16	Final test	✓	162,900,437.28	162,869,340.86	+0.0191%	29.05	285.08	600.67	629.72	1009.15	3601.18
2017-10-21	Final test	✓	130,858,688.57	130,799,423.73	+0.0453%	30.30	283.96	600.68	630.98	1032.55	3601.12
2017-10-26	Final test	✓	165,723,070.81	165,682,892.85	+0.0242%	28.28	275.95	600.72	629.00	1022.40	3600.63
2017-10-31	Final test	✓	176,362,238.76	176,239,797.64	+0.0695%	29.19	285.10	600.64	629.83	1009.64	3600.64
2017-11-06	Final test	✓	169,007,041.38	168,895,512.16	+0.0660%	26.92	284.17	600.69	627.61	1004.88	3601.17
2017-11-11	Final test	✓	208,279,582.47	208,255,901.73	+0.0114%	30.15	281.04	600.76	630.91	1001.47	3600.66
2017-11-16	Final test	✓	198,498,562.14	198,372,042.19	+0.0638%	28.09	283.19	600.66	628.74	1004.01	3600.83
2017-11-21	Final test	✓	192,428,697.62	192,429,402.86	-0.0004%	30.19	284.79	600.71	630.90	1006.06	3601.14
2017-11-26	Final test	✓	171,029,472.16	171,024,200.16	+0.0031%	28.55	273.72	600.74	629.29	993.40	3601.15
2017-12-01	Final test	✓	184,315,744.47	184,224,745.08	+0.0494%	29.97	265.33	600.97	630.93	985.29	3601.27
2017-12-06	Final test	✓	202,358,964.92	202,222,735.68	+0.0674%	28.39	261.30	600.73	629.12	981.91	3600.93
2017-12-11	Final test	✓	274,494,591.22	273,812,597.65	+0.2491%	26.51	282.83	600.72	627.23	1002.91	3601.20
2017-12-16	Final test	✓	356,237,293.56	355,353,725.43	+0.2486%	26.68	285.63	601.15	627.83	1006.57	3601.25
2017-12-21	Final test	✓	271,760,653.71	271,726,172.63	+0.0127%	31.27	288.86	600.67	631.93	1009.10	3601.10
2017-12-26	Final test	✓	310,696,067.43	310,412,821.13	+0.0912%	29.29	284.58	600.71	630.00	1006.78	3601.11
2017-12-31	Final test	✓	347,917,455.47	347,059,341.90	+0.2473%	29.24	287.60	601.00	630.23	1007.16	3600.71

H.4 Runtime Audit and CG Critical Paths

Table 14 makes explicit why "Time" in Table 3 can be substantially smaller than the recorded outer wall. The table reports solver counters and construction timers separately. For a parallel DW/CG iteration, the critical-path contribution is $t_{\mathrm{RMP}} + \max_b t_{\mathrm{price},b}$, whereas the sum over all pricing blocks is a resource-use quantity rather than elapsed parallel solve time. Each cell averages only the rows for which that timer was saved; consequently, a row is not necessarily additive across columns when a historical runner did not preserve every timer.

Table 14: Recovered timing composition. "Build/load" is non-solver wall time. "Init. price" is the parallel initial-column pricing critical path. "Iter. price" is the sum over CG iterations of the per-iteration maximum pricing-block Gurobi Runtime. CA blocks are sequential and have no RMP/pricing split. FERC is shown for its raw 24-instance `first_feasible` archive; the Table 3 FERC row remains the separately matched 21-row population.

Dataset / source population	n	Build/load (s)	Init. price (s)	Iter. RMP (s)	Iter. price (s)	Recovery (s)	Key solve (s)	E2E wall (s)
CA non-seed rollout	16	not saved	–	–	30.414[a]	–	30.414	85.423 (11)
FERC raw `first_feasible` archive	24	5.120+3.147	14.143	0.009	65.843	107.549 (19)	165.138	316.303 (19)
PEGASE non-seed evidence	75	23.442+59.574	32.377	0.0015	29.605	600.881	630.488	1009.148
BinPacking non-seed DW/CG	16	1.178	–	2.387	61.199	14.252	77.838	87.137

[a] CA's 30.414 s is the sum of sequential rollout-subproblem Gurobi runtimes. The 75 PEGASE key-solve value is $0.0015 + 29.605 + 600.881$ s (rounding explains the last digit); its E2E wall additionally includes initial-column generation, model construction, preparation, verification, and other runner overhead. The FERC raw-archive columns have different saved-timer coverages and must not be algebraically recombined.

To keep the appendix readable, Table 15 shows compact representative traces rather than thousands of repeated rows. The released timing ledger retains every recorded RMP and pricing call, including each individual block; this table displays pricing-block count, the sum over block runtimes, and the maximum that enters the parallel elapsed critical path. The FERC final row is an RMP-only Phase-I check and therefore has no pricing block.

Table 15: Representative recorded CG iterations. All values are Gurobi Runtime in seconds, except that "pricing sum" is total solver work across blocks. The iteration critical path is $\mathrm{RMP} + \max(\mathrm{pricing})$.

Dataset / instance	Iteration	RMP	Blocks	Pricing sum	Pricing max	RMP+max	Phase
PEGASE `2017-03-15`	1	0.00027	8	2.580	0.388	0.389	I
	2	0.00055	8	114.821	16.740	16.740	II
	3	0.00063	8	119.668	16.041	16.041	II
FERC `2015-01-01_lw`	1	0.00073	2	16.283	9.122	9.123	I
	6	0.00105	2	16.448	9.251	9.252	I
	11	0.00140	2	16.492	9.279	9.280	I
	12	0.00135	0	–	–	0.001	I
BinPacking `u500_13`	1	0.00079	1	0.025	0.025	0.026	CG
	10	0.00575	1	0.466	0.466	0.472	CG
	12	0.00376	1	0.492	0.492	0.496	CG

H.5 BinPacking: All 16 Non-Seed Instances

All rows use the same frozen pattern-generation operator. "Integer RMP" is the original integer recovery over the generated pattern pool, not a reconstruction of CG time. The saved reference runtime is retained from the original method-run evidence; the separately matched 20260923 Gurobi run is summarized in Table 3.

Table 16: Frozen BinPacking pattern DW/CG operator on all 16 non-seed instances. Key solve is CG critical path plus integer-RMP recovery; wall is guarded per-instance operator wall.

Instance	Split	Strict	Method obj.	Saved ref.	Ref. gap (%)	CG crit. (s)	Integer RMP (s)	Key solve (s)	Wall (s)	Saved ref. time (s)
u500_03	Dev	✓	204	204	+0.0000%	75.381	3.358	78.739	88.220	3600.144
u500_06	Dev	✓	207	207	+0.0000%	81.113	2.068	83.182	92.764	3600.077
u500_11	Dev	✓	200	200	+0.0000%	63.776	2.176	65.951	75.191	3600.076
u500_15	Dev	✓	201	201	+0.0000%	59.799	2.976	62.775	71.709	3600.117
u500_17	Dev	✓	198	198	+0.0000%	74.327	2.548	76.875	87.064	3600.093
u500_19	Dev	✓	196	196	+0.0000%	55.476	3.392	58.868	68.073	3600.104
u500_00	Gate	✓	198	198	+0.0000%	70.360	5.191	75.551	84.862	3600.080
u500_02	Gate	✓	202	202	+0.0000%	55.365	26.884	82.248	91.811	3600.201
u500_04	Gate	✓	206	206	+0.0000%	55.597	12.788	68.384	77.159	3602.302
u500_08	Gate	✓	196	196	+0.0000%	51.781	3.226	55.008	63.839	3601.255
u500_10	Gate	✓	200	200	+0.0000%	63.418	8.701	72.119	81.230	3600.138
u500_01	Final	✓	201	201	+0.0000%	59.208	24.544	83.752	92.657	3600.086
u500_05	Final	✓	206	206	+0.0000%	57.660	5.350	63.011	72.562	3600.069
u500_13	Final	✓	197	196	+0.5102%	58.818	120.001	178.820	188.219	3600.057
u500_14	Final	✓	204	204	+0.0000%	69.180	0.544	69.724	79.177	3600.100
u500_16	Final	✓	202	202	+0.0000%	66.125	4.285	70.410	79.656	3600.155

H.6 FERC Time-Axis Stress Test: Aggregate Negative Evidence

The time-axis stress test is intentionally not expanded into an objective-scalar per-instance table. Its source report retains strict-feasibility outcomes, signed reference gaps, solver MIP gaps, and most wall times, but it does not retain the paired final method and reference objective scalars required by the reporting standard used above. The aggregate result is 8/8 strict-feasible after recovery (not direct stitch), with mean / median signed reference gaps of 21.946% / 0.001%. Among the 7 runs with comparable saved wall time, method wall averages 359.322 s versus 214.659 s for the Gurobi reference. We therefore use this experiment only to characterize the structural transfer boundary until the missing objective ledgers are recovered.

I Additional Negative Evidence

Negative trials are retained because they constrain the operator search space. Representative examples include: (i) FERC direct resource-envelope and hard-y horizon rollouts that become locally infeasible; (ii) FERC solution-pool residual-aware stitching that produces incumbents in every block but still violates 87 original rows; (iii) overly aggressive stable-zero fixing, where moving from top 1% to top 3% yields an extremely poor incumbent; (iv) generator-axis FERC pricing parallelism that slows the run through resource contention; and (v) million-scale PEGASE hard fixing of 1.1M variables, which remains feasible but worsens objective quality by over 53%. These failures motivate the harness's explicit distinction between execution success, original-model validity, and optimization quality.