LLM-Guided Evolution of Symbolic Acceleration Inequality Families for MILP

Anonymous authors
Paper under double-blind review

Abstract

Integer programming (IP) is central to many combinatorial optimization tasks but remains challenging due to its NP-hard nature. A practical way to improve IP solvers is to manually design inequality families that exploit problem structure. However, this creative process requires deep expertise and has therefore been difficult to automate. Our proposed framework, Symbolic Acceleration via Generative Evolution (SAGE), automates symbolic discovery of acceleration inequality families at the modeling level. It reasons over a symbolic IP formulation and a natural language description of the problem to discover a reusable set of acceleration inequality families that can be instantiated for each problem instance. These acceleration inequality families are selected through empirical admissibility and impact screening rather than mathematical proofs. SAGE (i) initializes a population of candidate acceleration inequality families via an initializer agent that uses an LLM; (ii) it screens candidates for admissibility on a small development admissibility set for solution feasibility and the separation of the relaxation solution; and (iii) it iteratively refines the population of inequalities through evolutionary crossover and mutation agents. Across seven MILP benchmarks, SAGE reduces optimality gaps by up to 62.1% relative to baseline MILP formulations solved with a fixed time budget and reaches target gaps up to 5.51× faster according to shifted geometric mean speedup. Comprehensive ablations show its robustness across different LLM backends and across solvers and internal cut settings. Code: anonymous repository.

1 Introduction

Operations Research (OR) workflows usually consist of two main phases: formulating an optimization model and solving it. The first phase involves understanding the problem, identifying decision variables, objectives, and constraints, and building a mathematical formulation. The second phase uses numerical algorithms and generic optimization solvers to find exact or approximate solutions. As problems become more complex in practice, using artificial intelligence (AI) to improve OR has become increasingly important. Recent advances have used large language models (LLMs) to automate mathematical modeling, converting natural language descriptions directly into mathematical formulations Ramamonjison et al. (2023); Jiang et al. (2025); Huang et al. (2025); Mostajabdaveh et al. (2025). While there have been promising results in generating correct models, improving the efficiency of those models remains challenging. Parallel efforts have applied AI to the solving phase, including AI for heuristic design Liu et al. (2024); Ye et al. (2024) and machine learning techniques for accelerating generic optimization solvers Alvarez et al. (2017); Li et al. (2023). However, the crucial feedback loop between how a problem is formulated and how efficiently it can be solved remains largely unexplored. Optimization problems can often be formulated in multiple ways, with solver performance varying substantially depending on the formulation and its interaction with the solver Klotz & Oberdieck (2024).

In this study, we focus on automated pre-solve model augmentation for Mixed Integer Linear Programming (MILP). Starting from a base MILP formulation, we generate additional linear inequality families, called acceleration inequality families, intended to improve solver efficiency. Many widely studied problem classes

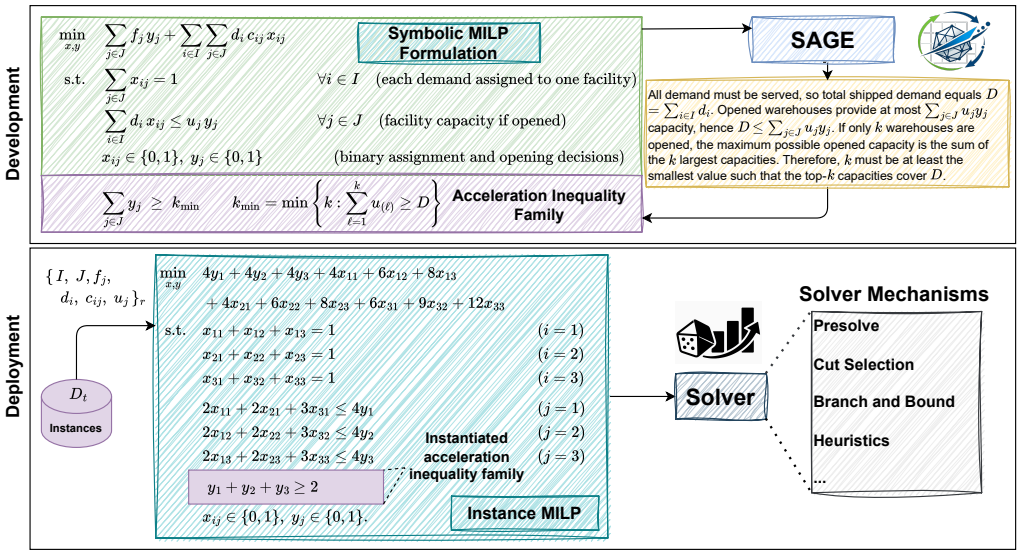

Figure 1: View of SAGE in two stages: during development, SAGE learns a reusable set of acceleration inequality families for a symbolic MILP formulation using a natural language description. During deployment, the families are instantiated for each problem instance and appended to its instance MILP before solving.

admit problem-specific inequalities derived in the OR literature. However, we aim for a general pipeline that operates directly on a symbolic MILP formulation shared across instances. We refer to this shared formulation as the symbolic MILP formulation and to its specialization with instance data as the instance MILP. Our proposed method, SAGE, uses LLM agents guided by evolutionary search to discover and refine such families by reasoning over a symbolic MILP formulation and a natural language description of the problem. In particular, SAGE learns a reusable set of acceleration inequality families during development and instantiates them for each problem instance at deployment time (Fig. 1). SAGE is a general framework for symbolic discovery at the modeling level and integrates into standard solver pipelines across different MILP models.

Together, this paper makes three contributions. First, we introduce SAGE, a general framework for *symbolic discovery of empirically screened acceleration inequality families at the modeling level.* Given a symbolic MILP formulation and a small admissibility set, SAGE discovers and selects a *reusable* set of acceleration inequality families that can be instantiated for each problem instance. Second, we evaluate SAGE on seven diverse MILP problems and show consistent solver speedups on unseen instances. The largest row-average checkpoint gap reduction is 62.1%, and the largest row-average target-gap speedup is 5.51×. Third, we provide an ablation and robustness analysis of SAGE: (a) data efficiency by varying the sizes of the impact-assessment and admissibility sets (Section 5.8), (b) LLM sensitivity across DeepSeek-R1, Gemini 3 Pro, and GPT-5.1 (Table 1), (c) solver robustness by transferring the same discovered acceleration inequality families across solvers (Appendix E.6), and (d) interaction with solver internal cut generation by comparing Gurobi with internal cuts disabled vs. default settings (Appendix E.7).

2 Related Work

Recent work has explored using LLMs to automate various stages of OR workflows, from problem understanding to algorithm generation Liu et al. (2026); Huang et al. (2024). A central line of research focuses on converting natural language descriptions into optimization formulations, initiated by the NL4Opt competition Ramamonjison et al. (2023) and advanced by systems such as OptiMUS AhmadiTeshnizi et al. (2024), ORLM Huang et al. (2025), and LM4Opt Ahmed & Choudhury (2024). Recent frameworks with multiple agents incorporate dedicated verification agents to ensure correctness before generating code that can be passed directly to a solver, including the chain of experts Xiao et al. (2023) and the staged agent

architecture of Mostajabdaveh et al. (2024). These efforts focus on modeling fidelity. In contrast, SAGE targets efficiency after a base formulation already exists: it proposes additional acceleration inequality families, screens them empirically on held-out instances, and ranks them by measured effect on time-limited solver progress. To our knowledge, prior research has not studied the automatic discovery of *reusable* pre-solve acceleration inequality families at the symbolic modeling level. Another emerging direction uses LLMs within evolutionary frameworks to generate heuristics for combinatorial optimization. These methods aim to boost heuristic algorithm performance by modifying parts of the algorithm code (e.g., heuristic rules) Liu et al. (2024); Ye et al. (2024). FunSearch Romera-Paredes et al. (2024) couples a frozen LLM with evolutionary search and an evaluator to discover heuristics, while the evolution of heuristics Liu et al. (2024) refines natural language "thoughts" and executable code using evolutionary strategies. ReEvo Ye et al. (2024) introduces reflective evolution with pairwise comparisons and reflections over longer horizons to improve code generated by LLMs iteratively. While these works evolve heuristics or code, SAGE evolves reusable acceleration inequality families. More broadly, machine learning for MILP solving has targeted decisions *inside* branch-and-bound: learned branching approximates strong branching or variable-selection policies Alvarez et al. (2017); Khalil et al. (2016); Gasse et al. (2019); learned cut selection chooses which generated cuts to apply Tang et al. (2020); Turner et al. (2023); Paulus et al. (2022); Wang et al. (2023); separator-configuration methods such as L2Sep, cold-start LLM configuration, and dynamic separator configuration decide which solver separators to activate during the solve Li et al. (2023); Lawless et al. (2025); Ye et al. (2026); learned or optimized cut-generation mechanisms select cut-generating functions Chételat & Lodi (2023); Cheng & Basu (2024); and learned primal heuristics aim to find incumbents early Shen et al. (2021). The central distinction is that these methods modify runtime policies within the solver's branch-and-bound process, whereas SAGE operates one level earlier: it augments the symbolic MILP formulation before the solver is called, producing reusable acceleration inequality families that can then be solved with any branching, cut-selection, separator-configuration, cut-generation, or primal-heuristic policy. Thus our contribution is not a replacement for ML solver policies but an orthogonal modeling-level layer; the same learned runtime policy can be run on the baseline formulation or on the SAGE-augmented formulation.

3 Preliminaries

To justify the practical relevance of an automatic yet reliable generator of acceleration inequality families, we briefly recall MILPs, their linear relaxations, and the notions of valid inequalities and optimality preservation. Our goal is to discover useful acceleration inequality families, including valid inequalities when possible; because our pipeline does not automatically prove global validity, we call the generated candidates acceleration inequality families and retain them through empirical screening and measured solver progress.

We consider an MILP of the form

$$\max\big\{\, c^\top x + h^\top y :\ Ax + Gy \le b,\ x \in \mathbb{Z}^n_{\ge 0},\ y \in \mathbb{R}^p_{\ge 0} \big\}, \tag{1}$$

with integer variables $x \in \mathbb{Z}^n_{\ge 0}$ and continuous variables $y \in \mathbb{R}^p_{\ge 0}$. Its feasible set is

$$S \;=\; \{\, (x, y) \in \mathbb{Z}^n_{\ge 0} \times \mathbb{R}^p_{\ge 0} \;:\; Ax + Gy \le b \,\}.$$

Relaxing integrality yields the linear programming (LP) relaxation

$$\max\big\{\, c^\top x + h^\top y \,:\, Ax + Gy \le b,\, x \in \mathbb{R}^n_{\ge 0},\, y \in \mathbb{R}^p_{\ge 0} \big\}, \tag{2}$$

whose feasible set (the linear relaxation set) is

$$P \;=\; \{\, (x, y) \in \mathbb{R}^n_{\ge 0} \times \mathbb{R}^p_{\ge 0} \,:\, Ax + Gy \le b \,\}.$$

Let

$$\mathrm{conv}(S)$$

denote the convex hull of S, i.e., the smallest convex set that contains S. Although Eq. (2) can be solved in polynomial time, its optimum is generally fractional (i.e., not in S). This motivates cutting-plane and branch-and-cut methods, in which a solver adds valid inequalities dynamically during the solve. By contrast,

SAGE produces acceleration inequality families at the user or model level that are appended to the MILP before calling the solver. Valid inequalities tighten P toward $\text{conv}(S)$, narrowing the gap between the LP optimum and the true MILP optimum (see Appendix Fig. 6). Optimality-preserving inequalities need not be valid for $\text{conv}(S)$ but can still speed up branch-and-bound by pruning provably suboptimal regions.

Formally, an inequality

$$w^\top(x, y) \;\leq\; \delta$$

is valid for a set Q if and only if $Q \subseteq \{(x, y) : w^\top(x, y) \leq \delta\}$. Adding inequalities that are valid for $\text{conv}(S)$ can separate some fractional extreme points of P and can reduce the integrality gap. Appendix D reviews valid inequalities and their role in describing $\text{conv}(S)$. Designing strong valid inequalities tailored to a specific problem class typically requires nontrivial proofs and insight into the combinatorial structure of the problem.

Separately, an inequality is optimality-preserving for Eq. (1) if it does not change the optimal value $Z^* = \max\{c^\top x + h^\top y : (x, y) \in S\}$, i.e.,

$$\max\{c^\top x + h^\top y : (x, y) \in S,\; w^\top(x, y) \leq \delta\} = Z^*,$$

even though it may remove other feasible solutions Laporte & Semet (1999); da Cunha et al. (2015). We reserve the term optimality-preserving for inequalities that are proved to satisfy this.

Throughout the paper we use the term additional inequality family to mean a set of linear inequalities appended to the MILP. We write such a family in the form $C := \left\{ A'_r x + G'_r y + H'_r z \leq \delta_r \right\}_{r \in R}$, where R is an index set and z denotes auxiliary variables introduced by the added inequalities. Including z allows the inequality family to encode additional logical relations. In practice, one may add inequality families only to speed up solving. We call any such added inequality family, tailored to a particular problem class, an acceleration inequality family.

Definition 3.1 (Acceleration inequality family)**.** An additional linear inequality family added to an MILP with the explicit aim of reducing solution time. Such a family need not be composed only of proven valid inequalities or proven optimality-preserving inequalities; in this work, it is retained only after empirical admissibility checks that (i) stored reference solutions remain feasible on a small admissibility set and (ii) at least one stored LP-relaxation solution is separated, with quality assessed by solver progress on an impact-assessment set.

We emphasize that SAGE's acceleration inequality families differ from several classical objects. A valid inequality is proved to contain all feasible integer points in S. A formulation strengthening is a reformulation or augmentation with established validity for the model class. Benders cuts are derived from a decomposition subproblem and added through a master–subproblem scheme. By contrast, a SAGE acceleration inequality family is proposed before solving and kept because it passes empirical OSS/LP-separation checks and improves solver progress. No global proof of validity or optimality preservation is assumed.

Modern MILP solvers routinely inject cuts that are general purpose during solving and that apply to many formulations (Chvátal, 1973; Balas, 1979). Commercial solvers such as Gurobi enable these cut-generation routines by default or through solver parameters (Gurobi Optimization LLC, 2026). Separately, problem-specific structure often gives rise to inequalities that improve solver speed in vehicle routing, network design, scheduling, and graph models (Toth & Vigo, 2002; Lysgaard et al., 2004; Costa et al., 2009; Queyranne & Wang, 1991; Aref et al., 2020). Given a symbolic MILP formulation, SAGE automatically proposes, empirically screens, and evolves acceleration inequality families to improve solve time.

4 Proposed Method

We propose SAGE, an evolutionary algorithm powered by multiple LLM agents to iteratively generate and refine acceleration inequality families that speed up solving across instances of a given MILP model. Candidate acceptance is determined by empirical admissibility and impact screens, not by a proof of global validity. Fig. 2 illustrates the flow diagram of SAGE at a high level. In addition, Appendix A (Algorithm 1) provides the pseudocode and additional details for the overall procedure. Our proposed method can be summarized in three main phases: (1) data preprocessing, (2) population initialization (driven by the initializer agent), and

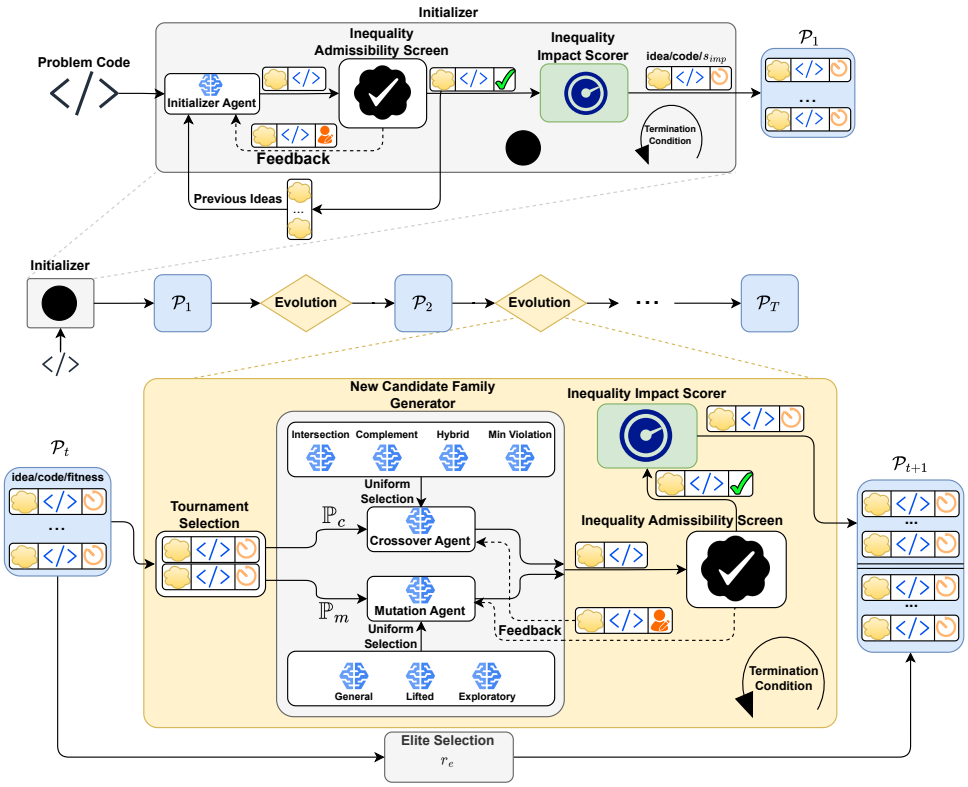

Figure 2: SAGE workflow: an initializer agent proposes a population of candidate acceleration inequality families. Each candidate passes through the Inequality Admissibility Screen for code validity, OSS on the admissibility set D_{adm}, and an LP-separation check, then through the Inequality Impact Scorer on D_{imp} to obtain an inequality impact score based on the optimality gap. Tournament/elite selection and crossover/mutation driven by LLM agents generate the next population.

(3) evolution (crossover and mutation driven by LLM agents). Next, we describe these phases, the Inequality Admissibility Screen, and the Inequality Impact Scorer.

We begin by splitting available development instances for a given MILP into an impact-assessment set D_{imp} and an admissibility set D_{adm}. We use D_{adm} only as a development screen for optimal solution screening (OSS) and LP-separation checks, and D_{imp} to score candidate family quality. For every instance $i \in D_{imp}$, we run the baseline MILP under a fixed computational budget. We use Gurobi's `WorkLimit` parameter, set to 10 (Gurobi Optimization LLC, 2026), and record its terminal optimality gap, $\text{gap}_{\text{ref}}(i)$. We construct D_{imp} so that $\text{gap}_{\text{ref}}(i) > 0$ for all $i \in D_{imp}$. This *reference gap* serves as the baseline for assessing the effect of candidate acceleration inequality families on solver performance within SAGE. For the smaller subset D_{adm}, we also store (i) a solver solution $(\hat{x}_i^*, \hat{y}_i^*)$ from solving the MILP to an optimality gap tolerance of 10^{-5} (for the OSS check) and (ii) the LP relaxation optimum $(x_i^{\text{LP}}, y_i^{\text{LP}})$ (for the LP-separation check). Each $i \in D_{adm}$ therefore carries its reference gap ($\text{gap}_{\text{ref}}(i)$) together with both solutions.

Thus, D_{imp} stores $\text{gap}_{\text{ref}}(i)$, while D_{adm} stores $(\hat{x}_i^*, \hat{y}_i^*)$, $(x_i^{\text{LP}}, y_i^{\text{LP}})$, and $\text{gap}_{\text{ref}}(i)$ for each instance i. With these datasets in place, SAGE seeds an initial population of candidate acceleration inequality families using an initializer agent. At each iteration, the LLM agent receives three inputs: (1) the complete MILP formulation code with a natural language description of model components (Fig. 1), (2) the set of acceleration inequality families generated so far, and (3) instructions to propose novel acceleration inequality families intended to separate poor LP-relaxation behavior in Eq. (2). The complete prompt templates are provided in Appendix B. The LLM response includes a brief idea for the proposed family together with executable code (e.g., Pyomo Hart et al. (2011)). Each LLM-proposed candidate acceleration inequality family is first passed through the Inequality Admissibility Screen, which checks whether the candidate can be implemented,

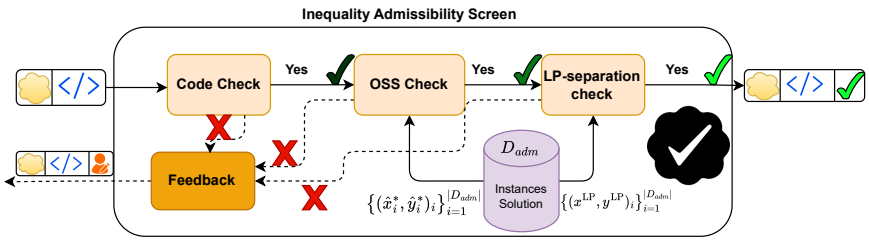

Figure 3: SAGE Inequality Admissibility Screen: candidate family code must compile (code check), pass OSS check on the admissibility set D_{adm} (i.e., not exclude the recorded reference solution), and separate at least one stored fractional LP optimum (LP-separation check). Failures trigger feedback and regeneration.

preserves the stored reference solutions in the admissibility set D_{adm}, and separates at least one LP-relaxation solution. Failures trigger a diagnostic prompt and allow the agent to retry up to a preset limit. We log each family idea to prevent duplication. This loop continues until the population reaches its predefined size, at which point the admissible and distinct candidates enter the evolutionary search phase (Fig. 2). To enter the population, each newly proposed candidate acceleration inequality family C must pass three admissibility checks (Fig. 3): (i) a code check, (ii) an OSS check, and (iii) an LP-separation check. First, we run the code check by parsing and compiling the code snippet that implements C. If a syntax or runtime error occurs, we return the error message to the generator agent for revision (up to a predefined retry limit). Next, we perform the OSS check to ensure that adding C does not exclude the recorded reference solutions on D_{adm}. Concretely, for each $i \in D_{adm}$, we ensure that the recorded solution $(\hat{x}_i^*, \hat{y}_i^*)$ remains feasible after appending C to the baseline MILP (Eq. (1)). We fix $(x, y) = (\hat{x}_i^*, \hat{y}_i^*)$ and solve the resulting reduced model to test feasibility. Because C may introduce auxiliary variables, we cannot simply evaluate the inequalities on the fixed solution. Instead, we ask whether there exists z that makes the added inequalities feasible. This solve is efficient because all original variables are fixed. If the reduced model is infeasible, C fails the OSS check and we provide feedback for revision (up to the retry limit). Formally, for instance i, C passes the check iff there exists z such that

$$A'_r\, \hat{x}_i^* + G'_r\, \hat{y}_i^* + H'_r\, z \;\leq\; \delta_r \quad \forall r \in R$$

is feasible. Otherwise, C fails the OSS check for instance i. If C passes the OSS check, we finally run the LP-separation check by discarding any candidate acceleration inequality family that does not separate any stored LP optimum in D_{adm}. For each $i \in D_{adm}$, we relax all integrality constraints, fix $(x, y) = \left(x_i^{\mathrm{LP}}, y_i^{\mathrm{LP}}\right)$, and test feasibility with C appended. We retain C iff there exists $i \in D_{adm}$ such that $\nexists z$ with

$$A'_r\, x_i^{\mathrm{LP}} + G'_r\, y_i^{\mathrm{LP}} + H'_r\, z \;\leq\; \delta_r \quad \forall r \in R$$

(i.e., the reduced LP becomes infeasible). Otherwise, C is discarded and the agent receives feedback (up to the retry limit). This LP-separation check is a minimal check: passing it is sufficient to show that C removes at least one sampled point of the LP relaxation. It does not imply that all alternative LP optima are removed, that the root bound improves, or that branch-and-bound search improves. These effects are assessed by the impact scorer on D_{imp} and by the test-set solve metrics.

Candidate families that pass this screen are then processed by the Inequality Impact Scorer. We append C to the baseline MILP and solve every impact-assessment instance $i \in D_{imp}$ under the same budget, record the resulting optimality gap $\mathrm{gap}_{\mathrm{aug}}(i)$, and compare it to $\mathrm{gap}_{\mathrm{ref}}(i)$. The key intuition is that a candidate family should receive a higher inequality impact score when it reduces the gap on average, and a lower score when it increases it. We implement this by computing the signed relative gap change per instance, averaging over D_{imp}, and then applying a monotone mapping so that more effective candidate families receive higher inequality impact scores $s_{imp}(C)$. The score definition is provided in Appendix H. With the initial population of admissible and scored candidate families ready, the SAGE algorithm evolves the population for T generations. The main steps of the evolution are as follows: **(i)** Elitism: The top r_e fraction of families with the highest inequality impact scores carry over to the next generation. **(ii)** Selection: Parent families are picked via tournament selection based on inequality impact score. **(iii)** Crossover: With probability $\mathbb{P}_c$,

two parents are passed to a Crossover LLM, which attempts to produce a new child family C_o by merging parent features. **(iv)** Mutation: With probability $\mathbb{P}_m$, a single parent undergoes a Mutation LLM, which alters coefficients or terms to produce a variant C_m. **(v)** Admissibility Screening & Impact Scoring: Each newly created candidate (C_o or C_m) is checked according to the admissibility checks in Section 4. Failed candidates are discarded, and a feedback prompt is provided to the LLM (up to a predefined maximum retry limit). Admissible candidates are then forwarded to the impact-scoring stage (Section 4). Their inequality impact scores are computed and then they are added to the next generation. At each iteration, reproduction (i.e., the generation of a new candidate family) continues until the next population reaches its required size. This evolutionary process then repeats for T generations. In each reproduction step, SAGE chooses whether to perform crossover (with probability $\mathbb{P}_c$) or mutation (with probability $\mathbb{P}_m$). Once the operation type is chosen, one of the corresponding agents is selected uniformly at random. The full list of mutation and crossover agents, along with their prompt details, is provided in Appendix B.

5 Experiments and Results

We structure our evaluation around three key questions. RQ1 asks whether SAGE can improve solver efficiency and to what extent. We measure reductions in optimality gaps and wall clock time across diverse unseen instances. RQ2 asks whether evolutionary search improves acceleration inequality families generated by LLMs beyond a few shot. We ablate the evolutionary component by comparing against families produced solely by the initializer agent. RQ3 asks how sensitive SAGE is to the sizes of the impact-assessment (D_{imp}) and admissibility (D_{adm}) sets. We vary $|D_{imp}|$ and $|D_{adm}|$ and report test set gap metrics for the selected acceleration inequality family.

5.1 Experimental Setup

We evaluate SAGE by comparing the baseline MILP to the baseline augmented by the selected SAGE acceleration inequality family under fixed solver budgets, across seven NP-hard MILP benchmarks (Table 1). As the MILP solver, we use the Gurobi Optimizer 10.0.0 (build v10.0.0rc2, Linux 64-bit), which is widely regarded as one of the fastest commercial MIP solvers Rehfeldt et al. (2023). We access Gurobi via the Pyomo interface Hart et al. (2011) using eight threads. Unless otherwise stated, solver parameters are defaults except for `MIPGap=1e-5` and the parameters stated here. We fix `Method`=4 (deterministic concurrent algorithm for continuous models and the initial root relaxation). We set Gurobi's `WorkLimit` parameter to 10 (Gurobi Optimization LLC, 2026), for SAGE's internal impact-assessment runs (Section 4). We set `TimeLimit`=2000 s for test set benchmarking. In SAGE, we run $T = 20$ generations with a population size of $|\mathcal{P}| = 8$, using a mutation probability of $\mathbb{P}_m = 0.3$, an elitism ratio of $r_e = 0.2$, and a crossover probability of $\mathbb{P}_c = 0.7$. Agents may retry candidate family generation up to three times if admissibility checks fail. This setup required roughly 20 hours of wall clock time on our hardware. We use Gemini 3 Pro (Google DeepMind, 2025) as the main LLM in SAGE (full API settings and approximate costs are given in Appendix G). Unless otherwise stated, the main results use the family selected from the Gemini 3 Pro SAGE run for each benchmark. To assess sensitivity to the LLM, we also run SAGE with DeepSeek-R1 Guo et al. (2025) and GPT-5.1 OpenAI (2025) (Table 1).

Our four core benchmarks include: Traveling Salesman Problem (TSP), Multi-Commodity Network Design (MCND), Capacitated Warehouse Location Problem (CWLP), and Job Shop Scheduling Problem (JSSP). Additionally, we report results on the Pickup and Delivery Problem with Time Windows (PDPTW), International Mathematical Olympiad 2025 Problem 6 (IMO6), and a unit commitment benchmark with a horizon under one hour (SHUC) (Table 1). IMO6 is a newly formulated MILP benchmark of a 2025 IMO problem, so LLMs are less likely to have been exposed to optimization literature specific to it. PDPTW and SHUC are practically motivated models with many constraints, and some of our instances are included in MIPLIB Koch et al. (2011). The test set sizes are 54 (TSP), 81 (MCND), 36 (CWLP), 80 (JSSP), 165 (PDPTW), 34 (IMO6), and 56 (SHUC). Appendix F provides a dedicated subsection for each benchmark problem, including its formulation, instance source, and dataset details. Appendix Tables 12 and 13 summarize the held-out test set scales and the synthetic development source scales, respectively. During evolution, SAGE relies on relatively small development datasets D_{imp} (default size 10) and D_{adm} (default size 2), as

detailed in Section 4. The default $|D_{adm}| = 2$ is used only for candidate screening during development. We do not use benchmark test instances during development. For all problems except IMO6, both D_{imp} and D_{adm} are drawn from synthetic generators released alongside our code. The test set D_t is separate and consists of public benchmark instances. For IMO6, all instances are generated and split into D_{imp}, D_{adm}, and D_t.

5.2 Generalization to Unseen Instances

We compare the baseline MILP to the baseline MILP augmented with the selected SAGE acceleration inequality family on unseen test instances in D_t. Each instance is solved twice under a 2000 s wall clock limit: once as the baseline MILP (reference) and once with the selected acceleration inequality family. At each of the seven checkpoints (5 s, 10 s, 50 s, 150 s, 300 s, 1000 s, 2000 s), we record the optimality gap g reported by the solver. We then compute the stabilized relative gap improvement:

$$\Delta_g(i,t) = \frac{g_{\text{ref}}(i,t) - g_{\text{aug}}(i,t)}{\max\{g_{\text{ref}}(i,t), \epsilon\}}, \quad \epsilon = 10^{-5}.$$

This avoids division by very small reference gaps. We report $\bar{\Delta}_g$, the average of Δ_g over all instances in D_t. At a checkpoint, $\Delta_g > 0$ iff $g_{\text{aug}} < g_{\text{ref}}$. We summarize checkpoint performance in Table 1, reporting $\bar{\Delta}_g \pm\sigma$ (%) across the test set at each checkpoint for each benchmark and LLM. In Table 1, the mean relative

Table 1: Test set mean stabilized relative gap improvement $\bar{\Delta}_g$ (%) from adding the selected SAGE acceleration inequality family (Gurobi, three LLMs). Mean $\pm\sigma$ is reported at each checkpoint. Positive values correspond to $g_{\text{aug}} < g_{\text{ref}}$ at that checkpoint. Average reports the row average of checkpoint means.

Problem	LLM	5 s	10 s	50 s	150 s	300 s	1000 s	2000 s	Average
TSP	DeepSeek-R1	8.2 (± 26.0)	9.1 (± 26.9)	11.9 (± 32.2)	18.1 (± 30.9)	18.5 (± 33.8)	12.9 (± 31.3)	20.7 (± 30.7)	14.2
	Gemini 3 Pro	13.1 (± 27.5)	15.0 (± 27.9)	22.6 (± 31.3)	34.2 (± 30.3)	27.6 (± 33.1)	26.4 (± 33.3)	37.0 (± 31.1)	25.1
	GPT-5.1	12.1 (± 28.4)	8.8 (± 29.7)	15.6 (± 33.2)	17.4 (± 34.1)	11.3 (± 35.0)	16.7 (± 33.8)	18.2 (± 31.6)	14.3
MCND	DeepSeek-R1	3.3 (± 17.8)	4.3 (± 19.9)	11.7 (± 27.0)	13.0 (± 28.2)	14.7 (± 28.9)	18.9 (± 32.4)	16.7 (± 29.7)	11.8
	Gemini 3 Pro	6.8 (± 17.7)	8.2 (± 19.9)	15.1 (± 25.8)	17.4 (± 26.9)	17.8 (± 25.5)	23.2 (± 27.9)	15.8 (± 28.6)	14.9
	GPT-5.1	25.5 (± 30.1)	22.3 (± 29.6)	16.5 (± 27.7)	6.9 (± 30.0)	15.6 (± 28.9)	18.1 (± 29.8)	20.4 (± 25.2)	17.9
CWLP	DeepSeek-R1	17.8 (± 21.4)	18.6 (± 22.4)	25.2 (± 26.7)	36.3 (± 26.6)	34.1 (± 34.7)	33.8 (± 29.9)	32.5 (± 30.5)	28.3
	Gemini 3 Pro	43.8 (± 20.7)	49.4 (± 22.3)	68.8 (± 28.6)	76.3 (± 23.8)	64.8 (± 23.0)	35.2 (± 39.3)	30.1 (± 38.9)	52.6
	GPT-5.1	35.9 (± 20.6)	41.3 (± 22.2)	64.7 (± 28.6)	73.7 (± 23.7)	57.5 (± 23.0)	31.9 (± 39.2)	18.3 (± 38.8)	46.2
JSSP	DeepSeek-R1	41.7 (± 18.5)	38.2 (± 19.6)	27.5 (± 22.8)	37.5 (± 25.6)	28.9 (± 24.1)	39.8 (± 22.7)	41.5 (± 20.7)	36.4
	Gemini 3 Pro	64.1 (± 9.9)	63.7 (± 10.7)	63.3 (± 14.4)	61.8 (± 16.6)	63.2 (± 16.5)	59.4 (± 21.2)	59.2 (± 20.6)	62.1
	GPT-5.1	49.0 (± 14.8)	46.5 (± 16.0)	39.2 (± 19.5)	38.0 (± 23.0)	37.4 (± 24.6)	42.8 (± 23.8)	56.7 (± 20.3)	44.2
PDPTW	DeepSeek-R1	0.2 (± 4.8)	-0.6 (± 5.3)	5.4 (± 9.2)	12.8 (± 14.9)	17.1 (± 19.6)	18.3 (± 21.5)	24.0 (± 19.9)	11.0
	Gemini 3 Pro	2.7 (± 3.4)	3.8 (± 4.1)	16.4 (± 12.3)	31.4 (± 23.4)	19.1 (± 30.5)	34.4 (± 30.2)	46.4 (± 31.7)	22.0
	GPT-5.1	7.5 (± 5.2)	8.2 (± 5.6)	9.7 (± 15.7)	12.9 (± 16.8)	14.3 (± 21.9)	18.4 (± 17.2)	27.8 (± 25.5)	14.1
IMO6	DeepSeek-R1	12.2 (± 13.0)	14.3 (± 14.9)	23.7 (± 20.7)	26.4 (± 19.6)	31.0 (± 18.7)	42.0 (± 14.8)	53.6 (± 11.1)	29.0
	Gemini 3 Pro	13.6 (± 11.2)	15.7 (± 13.5)	24.6 (± 18.4)	25.0 (± 21.4)	28.9 (± 23.0)	24.3 (± 26.8)	19.7 (± 27.8)	21.7
	GPT-5.1	14.0 (± 14.1)	16.6 (± 16.2)	26.9 (± 22.0)	29.5 (± 25.2)	34.1 (± 25.0)	43.6 (± 20.8)	53.3 (± 17.8)	31.1
SHUC	DeepSeek-R1	44.8 (± 37.9)	43.9 (± 38.3)	27.3 (± 37.0)	18.3 (± 31.8)	9.7 (± 29.4)	7.6 (± 28.7)	6.2 (± 27.6)	22.5
	Gemini 3 Pro	51.2 (± 35.1)	50.6 (± 34.8)	32.2 (± 37.7)	6.3 (± 29.4)	3.7 (± 24.9)	2.6 (± 24.2)	4.3 (± 25.0)	21.6
	GPT-5.1	34.2 (± 36.8)	34.8 (± 37.6)	25.2 (± 35.7)	6.9 (± 29.6)	4.0 (± 26.4)	6.0 (± 24.4)	7.0 (± 24.4)	16.9

gap improvement is positive at 2000 s for all 21 benchmark/LLM pairs. At 2000 s, the mean ranges from 4.3% on SHUC with Gemini 3 Pro to 59.2% on JSSP with Gemini 3 Pro. The row-average checkpoint improvement is positive for all 21 rows. Gemini 3 Pro has the largest row average on TSP (25.1%), CWLP (52.6%), JSSP (62.1%), and PDPTW (22.0%). The largest single checkpoint mean is 76.3% on CWLP with Gemini 3 Pro at 150 s. GPT-5.1 has the largest row average on MCND (17.9%) and IMO6 (31.1%). DeepSeek-R1 has the largest row average on SHUC (22.5%). Averaging the 7 row results by LLM gives aggregate improvements of 31.4% for Gemini 3 Pro, 26.4% for GPT-5.1, and 21.9% for DeepSeek-R1. Appendix Fig. 7 illustrates the full 2000 s time profiles. For SHUC with Gemini 3 Pro, the checkpoint means move from 51.2% at 5 s to 3.7% at 300 s and 4.3% at 2000 s.

The appendix also reports trajectory checks for representative instances. Appendix E.8 compares four TSPLIB instances under the same 2000 s limit, and the augmented gap curve is below the baseline curve for most of

the horizon in all four panels. Appendix E.9 uses the same four instances and shows higher best-bound curves for the augmented model. Appendix E.10 reports roughly 75−95% lower PDI for the augmented model on those instances. Appendix E.11 shows four MCND instances where the selected acceleration inequality family slightly increases the gap, illustrating that the added inequalities can negatively impact some individual cases.

5.3 Post-hoc Screening Against Proven Optimal Solutions

SAGE screens candidate acceleration inequality families using the small admissibility set D_{adm} during development (Section 4). In the default setting, this admissibility set has only two instances. After selecting an acceleration inequality family, we run an additional offline check on the held-out test sets to answer the following question: Does the selected SAGE acceleration inequality family ever exclude a proven optimal solution that we can otherwise compute? This post-hoc check is not an extra step required by SAGE at deployment time. For each test instance in every benchmark, we run the baseline MILP for 48 h and record the instances where the solver proves optimality. Across the full test sets, proven optima are obtained for CWLP at the rate of 36/36. Similarly, the rates are 58/80 for JSSP, 77/81 for MCND, 54/54 for TSP, 9/34 for IMO6, and 53/56 for SHUC. For each such instance, we fix the original decision variables to the proven optimal solution, append the selected SAGE acceleration inequality family, and solve only for any auxiliary variables introduced by the family to test feasibility. In all cases, the feasibility check succeeds. We find no case in which the selected SAGE acceleration inequality family excludes a proven optimal solution in our test set. For PDPTW, none of the 48 h baseline runs proves optimality (0/165), so the above check was not possible. We therefore report the following bound consistency results: using the 2000 s augmented run from Table 1, (i) the 2000 s lower bound does not exceed the 48 h upper bound and (ii) the 48 h lower bound does not exceed the 2000 s upper bound. This condition holds for PDPTW at the rate of 165/165. Across the instances for which this check is possible, the number of detected cases of optimal being excluded is 0. This test-set check serves solely as an empirical conjecture and does not constitute a formal guarantee of global validity or optimality preservation.

Appendix E.2 reports a related held-out LP separation check. For each test instance, we fix the original variables to the baseline LP optimum, append the selected acceleration inequality family, and solve only for auxiliary variables. Appendix Table 6 shows 100.0% LP optimum separation on TSP, CWLP, JSSP, PDPTW, and IMO6, 88.9% on MCND, and 85.7% on SHUC. Since this is a single-LP-point test, it does not prove validity.

5.4 Time to Target Optimality Gap

To complement the checkpoint gap analysis (Section 5.2), we measure how much SAGE shortens the time to reach target optimality gaps $g \in \{10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}\}$ on the unseen test sets. The Gurobi runs used for this table use `MIPGap=1e-5`, as stated in Section 5.1. For each instance and method (baseline vs. augmented), we compute the time to reach gap $t(g)$ by linearly interpolating the solver trace. If a run never reaches the target gap, we assign a penalized time $t(g) = 2T_{\max}$. We aggregate using the shifted geometric mean (SGM, with shift s=1):

$$\mathrm{SGM}(t) = \Big(\prod_{i \in \mathcal{I}_g} (t_i + s)\Big)^{1/|\mathcal{I}_g|} - s,$$

where $\mathcal{I}_g$ is the set of instances for which at least one method reaches g. Table 2 reports $\mathrm{SGM}(t_{\mathrm{ref}})/\mathrm{SGM}(t_{\mathrm{aug}})$ and solved counts (B/A). In Table 2, $\mathrm{SGM}(t_{\mathrm{ref}})/\mathrm{SGM}(t_{\mathrm{aug}}) > 1$ for all problems at $g \in \{10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}\}$, and for six of seven problems at $g = 10^{-1}$. The row-average SGM speedup is largest on PDPTW (5.51×), followed by CWLP (4.02×), JSSP (3.33×), and TSP (2.87×). The largest single target-gap SGM speedup is 7.20× on JSSP at $g = 10^{-1}$. MCND is nearly neutral overall (1.01×) and is the only benchmark below 1 at $g = 10^{-1}$ (0.92×). For PDPTW, the baseline reaches none of the target gaps within 2000 s (0 instances at each target), while the augmented model reaches targets on a subset of instances (5 instances for $g \leq 10^{-2}$ and 20 at $g = 10^{-1}$).

Table 2: Time to reach target gaps g: baseline vs. SAGE. B/A: instances reaching g within 2000 s (Baseline/SAGE). SGM: shifted geometric mean speedup (> 1 better). Average reports the row average over all target gaps.

	$g = 10^{-5}$		$g = 10^{-4}$		$g = 10^{-3}$		$g = 10^{-2}$		$g = 10^{-1}$		Average	
Problem	B/A	SGM	B/A	SGM	B/A	SGM	B/A	SGM	B/A	SGM	B/A	SGM
TSP	26/35	3.21	27/37	3.13	27/37	3.16	31/44	3.00	44/47	1.83	31.0/40.0	2.87
MCND	56/57	1.04	62/62	1.04	62/62	1.03	69/69	1.04	76/74	0.92	65.0/64.8	1.01
CWLP	26/23	2.44	35/36	4.24	35/36	4.42	35/36	5.19	35/36	3.83	33.2/33.4	4.02
JSSP	7/8	2.18	7/8	2.31	7/8	2.42	7/8	2.55	10/19	7.20	7.6/10.2	3.33
PDPTW	0/5	5.18	0/5	5.18	0/5	5.19	0/5	5.25	0/20	6.77	0.0/8.0	5.51
IMO6	7/8	1.23	7/8	1.23	7/8	1.23	7/8	1.24	7/8	1.39	7.0/8.0	1.26
SHUC	2/2	1.47	6/6	1.28	17/17	1.26	39/39	1.25	53/54	1.34	23.4/23.6	1.32

5.5 Comparison with Static Literature Augmentation

For the four benchmarks with well-known published valid inequalities, we also compare against adding these inequalities up front, using the same time-to-target protocol as in Table 2. The detailed formulas and sources are given in Appendix E.3. Averaged over the five target gaps $g \in \{10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}\}$, the static literature valid inequalities achieve mean SGM speedups of 3.01×, 1.91×, 1.01×, and 2.57× on CWLP, JSSP, MCND, and TSP, respectively. On the same benchmarks, the SAGE families achieve 4.02×, 3.33×, 1.01×, and 2.87×. Thus, relative to these instantiated static literature valid inequalities, SAGE is higher by about 34% on CWLP, 74% on JSSP, and 12% on TSP, while MCND is unchanged. The main gains come from CWLP and JSSP. TSP shows a smaller gain, and MCND shows no measurable difference.

5.6 Complementarity with Learned Solver-Layer Policies

Appendix E.4 reports a CWLP factorial experiment that crosses SAGE with two learned solver-layer methods: the Turner et al. Turner et al. (2023) adaptive cut selector and the Lawless et al. Lawless et al. (2025) cold-start LLM separator configuration. All cells are evaluated on the held-out 36-instance OR-Library set under SCIP 10.0 with the same 2000 s time limit and the same five target gaps used in Table 2. The reference cell is plain SCIP defaults without SAGE.

Appendix Table 8 shows that each learned solver-layer method has standalone benefit: Turner reaches 2.12× row-average SGM speedup, Lawless-LLM(0) reaches 2.58×, and Lawless-LLM(5) reaches 2.60×. SAGE alone reaches 3.13×. Combining SAGE with learned solver-layer policies gives higher row-average SGM speedups than SAGE alone in this CWLP experiment. The strongest combined cell is SAGE plus Lawless-LLM(0) at 3.51×.

5.7 Solver Robustness Ablations

Appendix E.6 compares the same selected `Gemini 3 Pro` acceleration inequality families under Gurobi and HiGHS. Each solver uses its default settings unless specified in Section 5.1, and each run uses the same 2000 s time limit and seven checkpoints as Table 1. Appendix Table 10 shows that the HiGHS 2000 s means are positive for 7/7 benchmarks, ranging from 5.7% on MCND to 64.2% on CWLP. At 2000 s, HiGHS gives larger means than Gurobi on CWLP (64.2% vs. 30.1%) and SHUC (20.4% vs. 4.3%), and smaller means on the other 5 benchmarks.

Appendix E.7 tests the same selected `Gemini 3 Pro` acceleration inequality families under Gurobi with internal cuts disabled versus default internal cuts. Appendix Table 11 shows that disabling internal cuts gives larger checkpoint means on TSP and CWLP at 7/7 checkpoints, and smaller checkpoint means on MCND and PDPTW at 7/7 checkpoints. At 2000 s, the no-internal-cuts and default values are 62.0% vs. 37.0% for TSP, 93.9% vs. 30.1% for CWLP, 5.1% vs. 15.8% for MCND, and 34.2% vs. 46.4% for PDPTW. The row averages remain positive for all seven benchmarks in both settings.

Table 3: JSSP: test stabilized $\bar{\Delta}_g \pm\sigma$ (%) at the 2000 s checkpoint for the selected SAGE acceleration inequality family vs. $(|D_{adm}|, |D_{imp}|)$.

$(\lvert D_{adm}\rvert, \lvert D_{imp}\rvert)$	$(2, 10)$	$(5, 10)$	$(10, 10)$	$(10, 5)$	$(10, 2)$
$\bar{\Delta}_g \pm\sigma$ (%)	37.3 ± 22.0	37.3 ± 22.0	40.4 ± 22.3	37.3 ± 22.0	7.1 ± 16.4

5.8 Sensitivity to $|D_{imp}|$ and $|D_{adm}|$

We vary $|D_{imp}|$ and $|D_{adm}|$ on the JSSP benchmark using DeepSeek-R1. We run two experiments. In the first, we fix $|D_{imp}|$=10 and vary $|D_{adm}|$ over $\{2, 5, 10\}$. In the second, we hold $|D_{adm}|$=10 constant and vary $|D_{imp}|$ over the same three values. These settings are independent SAGE reruns. In every setting, the selected acceleration inequality family found by SAGE is evaluated on the held-out test set D_t. Following the post-hoc screening protocol in Section 5.3, we also check whether any generated acceleration inequality family excludes a proven-optimal JSSP solution. Table 3 reports test-set $\bar{\Delta}_g \pm\sigma$ at the 2000 s checkpoint for the selected acceleration inequality family found under each setting. In Table 3, the $|D_{adm}| = 2$ and $|D_{adm}| = 5$ configurations both report 37.3%, while $|D_{adm}| = 10$ reports 40.4%. With $|D_{adm}| = 10$, the reported means for $|D_{imp}| = 2, 5, 10$ are 7.1%, 37.3%, and 40.4%, respectively. None of the generated families under these configurations excludes a proven-optimal JSSP solution in the post-hoc check. All setups pass on 58/58 test instances.

5.9 Evolutionary Diagnostics

We report three diagnostics of the evolutionary process: (i) agent outcome statistics (Appendix Table 5), (ii) a traced ancestry graph for a different JSSP evolutionary trajectory (Appendix Fig. 4), and (iii) population score trajectories over generations (Appendix Fig. 5). Full details are in Appendix C. In Appendix Table 5, the mutation and crossover agents have higher success rates than the initializer agent across the three LLMs. The table reports mean inequality-impact-score gains $\bar{\Delta}_{imp}$ that include both negative and positive values. The initializer success rates are 25.4%, 36.5%, and 29.6% for DeepSeek-R1, Gemini 3 Pro, and GPT-5.1, respectively. Across mutation agents, success rates range from 63.3% to 85.2%. Across crossover agents, they range from 67.9% to 84.4%. The reported $\bar{\Delta}_{imp}$ values range from -0.4% to 19.1%. Second, we trace one JSSP evolutionary trajectory. This trajectory is included to illustrate how SAGE mutates and recombines candidates. Appendix Fig. 4 shows the traced graph for this trajectory. In Appendix F, we also include the selected acceleration inequality family discovered by SAGE for each problem. Third, we plot score curves at the population level. For each of the seven problems, we tracked the maximum and mean inequality impact scores of the population over 20 generations. Appendix Fig. 5 shows the trajectories of these values across generations. For each benchmark, the best score curve increases over the 20 generations.

5.10 Case Study: Post-Hoc Interpretation of the SHUC Acceleration Inequality

We next examine the selected acceleration inequality family for the sub-hour unit commitment problem (SHUC). This is a post-hoc case study for one acceleration inequality family found by SAGE, not a general validity guarantee for the method. SAGE selected this family through the empirical screens in Section 4. The proof below was written after selection to interpret this particular output. The selected family combines several familiar unit-commitment strengthening motifs into one aggregate reachability inequality. The SHUC benchmark follows a reference unit commitment formulation with piecewise production costs, startup categories, ramping limits, minimum up/down times, demand balance, and spinning reserve requirements (Knueven et al., 2020; Power Grid Lib, 2019). For the SHUC formulation in Appendix F.7, this selected acceleration inequality family can be proved valid as a valid inequality family.

To avoid overloading notation, we denote the auxiliary capacity variable from Appendix F.7 by $\bar{P}_{g,t}$. Define

$$D_t^{\text{net}} := L_t - \sum_{w \in W} p_{w,t}^w + R_t, \qquad a_g^{\text{SU}} := \max\{P_g^{\max} - SU_g, 0\}, \qquad a_g^{\text{SD}} := \max\{P_g^{\max} - SD_g, 0\}.$$

The selected SHUC acceleration inequality family is:

$$
\begin{aligned}
& b_{g,t} \le v_{g,t}, && \forall g \in G,\ t < T, && (3)\\
& b_{g,t} \le w_{g,t+1}, && \forall g \in G,\ t < T, && \\
& b_{g,t} \ge v_{g,t} + w_{g,t+1} - 1, && \forall g \in G,\ t < T, && \\
& \bar{P}_{g,t} \le P_g^{\max} u_{g,t} - a_g^{\mathrm{SU}} v_{g,t}, && \forall g \in G,\ t \in \mathcal{T}, && (4)\\
& \bar{P}_{g,t} \le P_g^{\max} u_{g,t} - a_g^{\mathrm{SD}} w_{g,t+1}, && \forall g \in G,\ t < T, && (5)\\
& \bar{P}_{g,t} \le P_g^{\max} u_{g,t} - a_g^{\mathrm{SU}} v_{g,t} - a_g^{\mathrm{SD}} w_{g,t+1} + \min\{a_g^{\mathrm{SU}}, a_g^{\mathrm{SD}}\} b_{g,t}, && \forall g \in G,\ t < T, && (6)\\
& \bar{P}_{g,t} \le P_g^{\min} u_{g,t} + p_{g,t-1} + RU_g + (P_g^{\max} - P_g^{\min})(1 - u_{g,t-1}), && \forall g \in G,\ t \ge 2, && (7)\\
& \bar{P}_{g,t} \le P_g^{\min} u_{g,t} + (P_g^{\max} - P_g^{\min}) u_{g,t-1} - a_g^{\mathrm{SU}} v_{g,t-1} + RU_g, && \forall g \in G,\ t \ge 2, && (8)\\
& \bar{P}_{g,t} \le P_g^{\min} u_{g,t} + (P_g^{\max} - P_g^{\min}) u_{g,t-1} - a_g^{\mathrm{SD}} w_{g,t} + RU_g, && \forall g \in G,\ t \ge 2, && (9)\\
& D_t^{\mathrm{net}} \le \sum_{g \in G} \bar{P}_{g,t}, && \forall t \in \mathcal{T}. && (10)
\end{aligned}
$$

The analogous $t = 1$ ramp-reachability rows use the initial status and initial output fields from the instance data, as in the base SHUC model.

Proposition 5.1 (Post-hoc validity of the selected SHUC acceleration inequality family by SAGE)**.** *For the SHUC formulation in Appendix F.7, the acceleration inequality family from Eq.* (3) *through Eq.* (10) *is valid as an extended inequality family: every feasible integer solution of the original SHUC formulation can be extended with auxiliary variables* $(\bar{P}, b)$ *so that all added constraints are satisfied.*

Proof. Take any feasible integer solution of the original SHUC formulation. Set

$$\widehat{P}_{g,t} := P_g^{\min} u_{g,t} + p_{g,t} + r_{g,t}, \qquad \widehat{b}_{g,t} := v_{g,t} w_{g,t+1}.$$

Because $v_{g,t}$ and $w_{g,t+1}$ are binary, the three inequalities in Eq. (3) are the standard linearization of the product $b_{g,t} = v_{g,t} w_{g,t+1}$.

The base SHUC startup and shutdown capacity rows give

$$p_{g,t} + r_{g,t} \le (P_g^{\max} - P_g^{\min}) u_{g,t} - a_g^{\mathrm{SU}} v_{g,t}$$

and

$$p_{g,t} + r_{g,t} \le (P_g^{\max} - P_g^{\min}) u_{g,t} - a_g^{\mathrm{SD}} w_{g,t+1}.$$

Adding $P_g^{\min} u_{g,t}$ to both inequalities proves Eq. (4) and Eq. (5) for $\bar{P}_{g,t} = \widehat{P}_{g,t}$. Moreover, when $v_{g,t}, w_{g,t+1} \in \{0, 1\}$ and $b_{g,t} = v_{g,t} w_{g,t+1}$, the right-hand side of Eq. (6) equals the tighter of the two startup/shutdown upper envelopes. Hence Eq. (6) is also satisfied.

For the ramp-reachability rows, the base ramp-up constraint gives

$$p_{g,t} + r_{g,t} - p_{g,t-1} \le RU_g.$$

Adding $P_g^{\min} u_{g,t}$ yields

$$\widehat{P}_{g,t} \le P_g^{\min} u_{g,t} + p_{g,t-1} + RU_g,$$

which implies Eq. (7). Combining the same ramp-up constraint with the startup-capacity upper bound at time $t - 1$ gives

$$p_{g,t-1} \le (P_g^{\max} - P_g^{\min}) u_{g,t-1} - a_g^{\mathrm{SU}} v_{g,t-1},$$

and therefore Eq. (8). Combining the ramp-up constraint with the shutdown-capacity upper bound at time $t - 1$ gives

$$p_{g,t-1} \le (P_g^{\max} - P_g^{\min}) u_{g,t-1} - a_g^{\mathrm{SD}} w_{g,t},$$

and therefore Eq. (9).

Table 4: Post-hoc source attribution for the selected SHUC acceleration inequality family. Table 4 categorizes each component by whether it has a direct literature analog, follows from the base formulation, or appears to be a formulation-specific aggregate component.

Component	What it does	Source status
Binary product rows Eq. (3)	Encode $b_{g,t} = v_{g,t} w_{g,t+1}$, i.e., whether unit g starts at t and shuts down at $t+1$.	Standard MILP linearization of binary products, not SHUC-specific (Glover, 1975).
Startup/shutdown capacity envelopes Eq. (4) to Eq. (5)	Upper-bound reachable thermal capacity when the unit is starting up or shutting down.	Closely aligned with startup/shutdown ramping and strong UC production upper bounds (Morales-España et al., 2012; Knueven et al., 2020).
Lifted adjacent start/shutdown envelope Eq. (6)	Combines the two derating envelopes and uses $b_{g,t}$ to avoid double-counting when both transition indicators are active.	Built from known lifting and UC derating ideas. Closest references are strong UC upper-bound and ramping formulations (Knueven et al., 2018; 2020).
Ramp-reachability envelopes Eq. (7) to Eq. (9)	Bound what unit g can contribute at t using previous dispatch/status and ramp-up capability.	Related to ramping polytope and ramp-aware UC formulations (Knueven et al., 2018; 2020).
Aggregate auxiliary reachability inequality Eq. (10)	Projects unit-level startup, shutdown, and ramp-reachability envelopes into one system-level condition: net load plus reserve must be coverable by reachable thermal capacity.	In our targeted review, close analogs exist for the unit-level components in the UC formulation literature, including reserve and ramp-capability formulations (Morales-España et al., 2013; 2015), but we did not identify Eq. (10) as this same static aggregate inequality added before solving for the SHUC formulation in Appendix F.7.

Finally, the base power-balance constraint gives

$$L_t - \sum_{w \in W} p^w_{w,t} = \sum_{g \in G} \left(P^{\min}_g u_{g,t} + p_{g,t} \right),$$

and the base reserve constraint gives

$$R_t \leq \sum_{g \in G} r_{g,t}.$$

Adding these two relations gives

$$D^{\text{net}}_t = L_t - \sum_{w \in W} p^w_{w,t} + R_t \leq \sum_{g \in G} \left(P^{\min}_g u_{g,t} + p_{g,t} + r_{g,t} \right) = \sum_{g \in G} \widehat{P}_{g,t},$$

which proves Eq. (10). Thus every feasible integer solution of the base SHUC model admits feasible auxiliary variables, so the added family is valid in the extended space. □

This case study shows that this selected SHUC acceleration inequality family is interpretable at two levels. At the unit level, SAGE recovers known modeling motifs: binary-product linearization, transition derating, and ramp-aware capacity envelopes. At the system level, it composes these motifs into the aggregate reachability inequality in Eq. (10). Our targeted review did not identify this exact static aggregate inequality for SHUC. Rather than adding a runtime separator or a full ramping polytope, this selected acceleration inequality family can be appended before the solver is called.

6 Discussion

Starting with RQ1, Table 1 and Appendix E.1 (Appendix Fig. 7) show that SAGE can reduce the optimality gap at successive checkpoints under the same time budgets. The trajectory checks in Appendix E.8, Appendix E.9, and Appendix E.10 show the same effect for individual instances. Table 1 also indicates that

different LLMs lead to different families and levels of improvement. Therefore, choosing an LLM among the multiple LLM options of SAGE matters. The main LLM result is that Gemini 3 Pro has been generally the strongest backend in our experiments. This pattern justifies our use of the Gemini 3 Pro acceleration inequality family as the selected alternative in the rest of the experiments. Complementing this, Table 2 shows faster times to reach target gaps across benchmarks. SAGE often increases the number of instances that reach the target gap within the time limit. We have also observed on rare occasions that SAGE acceleration inequality families may have negative impact. When a discovered acceleration inequality family fails to improve the relaxed model for a particular instance, it increase model size and overhead nonetheless, leading to a slight deterioration of the overall performance (Appendix Fig. 11, Appendix E.11).

To test robustness beyond gap improvements, we reported on four complementary experiments. First, the post-hoc screening against proven-optimal solutions and the held-out LP separation check in Section 5.3 indicate a reduced chance that the observed speedups come from excluding good integer solutions. Second, the cross-solver ablation in Section 5.7 suggests that the same discovered families can transfer beyond one solver. Third, the internal-cut ablation in Section 5.7 shows that SAGE remains useful when cut generation inside Gurobi changes. Fourth, the learned-policy factorial in Section 5.6 evaluates the same modeling claim against learned cut-selection and separator-configuration policies. The learned-policy results are consistent with complementarity between SAGE's model-layer augmentation and the learned solver-layer policies. Together, these results suggest that SAGE provides a lightweight augmentation before solving that is separate from solver cut policies and learned runtime policies: it changes the formulation before the solver is called, while those policies act during solving. We also observe that the magnitude of improvement depends on the interaction between the compact formulation, the selected acceleration inequality family, and the solver. Mature benchmarks can still show large gains when the compact baseline leaves useful model structure. Finally, we focus on improving a given symbolic formulation without callbacks or domain-specific separation code. Accordingly, our baselines use standard compact formulations (Appendix F), since we did not have the goal of matching a tailored branch-and-bound method for each benchmark problem.

The static literature augmentation comparison in Section 5.5 and Appendix E.3 clarifies how SAGE's model-layer gains compare with existing expert valid inequalities. Valid inequalities can be strong under the same static augmentation protocol, as shown in Appendix Table 7. The SAGE acceleration inequality families, however, yield even larger mean speedups than the instantiated static literature valid inequalities on CWLP, JSSP, and TSP. On MCND both methods are essentially equivalent. Taken together, our analysis suggests that SAGE performs symbolic search for useful acceleration inequality families at the modeling level. The selected SAGE families include both familiar forms of inequalities and more specialized algebraic forms. This search can recover high-value structure that expert formulations of classic problems also exploit, including cut-set capacity, assignment and opening links, cover logic, ordering constraints, and workload bounds. For TSP, the smaller margin is consistent with the literature baseline already adding lifted MTZ ordering rows that overlap with the selected SAGE family. For the lesser explored benchmarks, such as IMO6 and SHUC, the same search can produce more specialized families tied to the symbolic formulation. Thus, SAGE is a symbolic search procedure whose empirical screening determines which acceleration inequality family is useful for the given compact formulation.

Turning to RQ2, evolutionary search produces more effective acceleration inequality families than a one-shot initializer, as measured by admissibility success and inequality impact scores. Initializer proposals were shown to have a lower success rate than evolved offspring (Appendix C.1), while both the ancestry trace (Appendix Fig. 4) and the population-level inequality-impact-score trajectories (Appendix Fig. 5) show sustained improvement over generations. Together, these results support the value of coupling LLM generation with evolutionary refinement for producing more effective acceleration inequality families.

Finally, for RQ3, SAGE derives acceleration inequality families from underlying problem logic rather than patterns specific to a dataset. In our experiments, these families remain effective even when the test distribution or size differs from those used for admissibility screening and impact assessment. This is consistent with our choice of split: D_{imp} and D_{adm} are synthetic, while D_t is a held-out public benchmark set. Table 3 suggests that small admissibility sets may suffice, while larger impact-assessment sets generally yield stronger families. As a practical consideration, the sizes of D_{imp} and D_{adm} create a tradeoff between runtime and quality. Larger impact-assessment sets tend to improve deployment performance. Larger admissibility sets

can also help, but to a lesser extent. Both increase SAGE development runtime because each candidate family is tested on more instances (Section 5.8). Appendix Table 13 reports the per-instance scale of the synthetic sources used for D_{imp} and D_{adm}.

7 Conclusion and Future Direction

SAGE automates acceleration inequality family generation, a step that augments a MILP model before solving and is typically driven by experts. Across seven MILP benchmarks, SAGE discovers reusable and interpretable families, ranging from expert strengthening patterns on mature benchmarks to more specialized families on less standard models. These families can be appended to MILPs in standard solver pipelines and often improve solver progress under time limits. We expect the approach to be particularly useful for custom or newly formulated MILPs, where established formulation ideas may be limited.

Our study has two limitations. First, the screening is empirical and does not provide a formal guarantee that a learned family is globally valid or optimality preserving. Establishing such guarantees may still require human expert proofs. Second, we currently focus on acceleration inequality families that can be added statically before solving, rather than inequalities that require runtime separation and dynamic addition.

These limitations motivate two directions for future work: (1) The current screening relies on recorded solutions on a small admissibility set. Integrating automated formal proof tooling could provide theoretical guarantees (e.g., discovering acceleration inequality families composed of provably valid inequalities or families that provably preserve optimality) Yang et al. (2025). (2) SAGE currently inserts instantiated acceleration inequality families into the MILP model before the solution process begins. Dynamically separating and adding inequalities via callbacks during the solve could further improve solver performance, but requires generating an efficient separation algorithm for each discovered family.

Broader Impact Statement

SAGE is a study of acceleration inequality families for MILP solving. It uses benchmark and synthetic MILP instances and no human subject data. By improving solver progress under fixed budgets, the method may reduce compute time and energy use, but it may also improve the performance of downstream optimization systems in high-stakes domains such as energy, logistics, scheduling, and public-sector resource allocation. The main technical risk is that empirically screened acceleration inequality families are not guaranteed to be globally valid or optimality preserving on all unseen instances. Users should validate any generated family in the target domain, keep a baseline fallback, monitor for regressions, and not treat empirical screening as proof of validity. Releasing reproducible evaluation code and synthetic generators helps others inspect the workflow and test generated families under their own assumptions.

References

Karen Aardal, Yves Pochet, and Laurence A Wolsey. Capacitated facility location: valid inequalities and facets. *Mathematics of Operations Research*, 20(3):562–582, 1995.

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*, pp. 577–596, 2024.

Tasnim Ahmed and Salimur Choudhury. Lm4opt: Unveiling the potential of large language models in formulating mathematical optimization problems. *INFOR: Information Systems and Operational Research*, 62(4):559–572, 2024.

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

David Applegate and William Cook. A computational study of the job-shop scheduling problem. *ORSA Journal on computing*, 3(2):149–156, 1991.

Samin Aref, Andrew J Mason, and Mark C Wilson. A modeling and computational study of the frustration index in signed networks. *Networks*, 75(1):95–110, 2020.

Egon Balas. Facets of the knapsack polytope. *Mathematical programming*, 8(1):146–164, 1975.

Egon Balas. Disjunctive programming. *Annals of discrete mathematics*, 5:3–51, 1979.

Clayton Barrows, Aaron Bloom, Ali Ehlen, Jussi Ikäheimo, Jennie Jorgenson, Dheepak Krishnamurthy, Jessica Lau, Brendan McBennett, Matthew O'Connell, Eugene Preston, et al. The ieee reliability test system: A proposed 2019 update. *IEEE Transactions on Power Systems*, 35(1):119–127, 2019.

John E Beasley. An algorithm for solving large capacitated warehouse location problems. *European Journal of Operational Research*, 33(3):314–325, 1988.

John E Beasley. Or-library: distributing test problems by electronic mail. *Journal of the operational research society*, 41(11):1069–1072, 1990.

Hongyu Cheng and Amitabh Basu. Learning cut generating functions for integer programming. In *Advances in Neural Information Processing Systems*, volume 37, pp. 61455–61480, 2024.

Didier Chételat and Andrea Lodi. Continuous cutting plane algorithms in integer programming. *Operations Research Letters*, 51(4):439–445, 2023.

Mervat Chouman, Teodor Gabriel Crainic, and Bernard Gendron. Commodity representations and cut-set-based inequalities for multicommodity capacitated fixed-charge network design. *Transportation Science*, 51 (2):650–667, 2017.

Vasek Chvátal. Edmonds polytopes and a hierarchy of combinatorial problems. *Discrete mathematics*, 4(4): 305–337, 1973.

CommaLAB. Multicommodity flow problem instances repository. `https://commalab.di.unipi.it/datasets/mmcf`, 2021. Accessed: 2025-05-01.

Alysson M Costa, Jean-François Cordeau, and Bernard Gendron. Benders, metric and cutset inequalities for multicommodity capacitated network design. *Computational Optimization and Applications*, 42(3):371–392, 2009.

Alexandre Salles da Cunha, Luidi Simonetti, and Abilio Lucena. Optimality cuts and a branch-and-cut algorithm for the k-rooted mini-max spanning forest problem. *European Journal of Operational Research*, 246(2):392–399, 2015.

Martin Desrochers and Gilbert Laporte. Improvements and extensions to the miller-tucker-zemlin subtour elimination constraints. *Operations research letters*, 10(1):27–36, 1991.

Michael R Garey, David S Johnson, and Ravi Sethi. The complexity of flowshop and jobshop scheduling. *Mathematics of operations research*, 1(2):117–129, 1976.

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.

Bernard Gendron, Teodor Gabriel Crainic, and Antonio Frangioni. Multicommodity capacitated network design. In *Telecommunications Network Planning*, pp. 1–19. Springer, 1999.

Fred Glover. Improved linear integer programming formulations of nonlinear integer problems. *Management science*, 22(4):455–460, 1975.

Google DeepMind. Gemini 3 pro model card. `https://storage.googleapis.com/deepmind-media/Model-Cards/Gemini-3-Pro-Model-Card.pdf`, 2025. Model card update: December 2025. Accessed: 2026-05-07.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, et al. Deepseek-r1 incentivizes reasoning in llms through reinforcement learning. *Nature*, 645(8081):633–638, 2025.

Gurobi Optimization LLC. Gurobi optimizer reference manual: Parameter reference. `https://docs.gurobi.com/projects/optimizer/en/current/reference/parameters.html`, 2026. Accessed: 2026-05-14.

William E Hart, Jean-Paul Watson, and David L Woodruff. Pyomo: modeling and solving mathematical programs in python. *Mathematical Programming Computation*, 3(3):219–260, 2011.

HiGHS Development Team. HiGHS options documentation. `https://ergo-code.github.io/HiGHS/dev/options/definitions/`, 2026. Accessed: 2026-05-14.

Chenyu Huang, Zhengyang Tang, Shixi Hu, Ruoqing Jiang, Xin Zheng, Dongdong Ge, Benyou Wang, and Zizhuo Wang. Orlm: A customizable framework in training large models for automated optimization modeling. *Operations Research*, 2025.

Sen Huang, Kaixiang Yang, Sheng Qi, and Rui Wang. When large language model meets optimization. *Swarm and Evolutionary Computation*, 90:101663, 2024.

International Mathematical Olympiad. International mathematical olympiad 2025 problems. `https://www.imo-official.org/problems.aspx`, 2025. Accessed: 2026-05-07.

Caigao Jiang, Xiang Shu, Hong Qian, Xingyu Lu, Jun Zhou, Aimin Zhou, and Yang Yu. Llmopt: Learning to define and solve general optimization problems from scratch. In *Proceedings of the Thirteenth International Conference on Learning Representations*, Singapore, Singapore, 2025. URL `https://openreview.net/pdf?id=9OMvtboTJg`.

Elias Khalil, Pierre Le Bodic, Le Song, George Nemhauser, and Bistra Dilkina. Learning to branch in mixed integer programming. In *Proceedings of the AAAI conference on artificial intelligence*, volume 30, 2016.

Ed Klotz and Richard Oberdieck. Converting weak to strong mip formulations: A practitioner's guide. In *Optimization Essentials: Theory, Tools, and Applications*, pp. 113–174. Springer, 2024.

Ben Knueven, Jim Ostrowski, and Jianhui Wang. The ramping polytope and cut generation for the unit commitment problem. *INFORMS Journal on Computing*, 30(4):739–749, 2018.

Bernard Knueven, James Ostrowski, and Jean-Paul Watson. On mixed-integer programming formulations for the unit commitment problem. *INFORMS Journal on Computing*, 32(4):857–876, 2020. doi: 10.1287/ijoc.2019.0944.

Thorsten Koch, Tobias Achterberg, Erling Andersen, Oliver Bastert, Timo Berthold, Robert E Bixby, Emilie Danna, Gerald Gamrath, Ambros M Gleixner, Stefan Heinz, et al. Miplib 2010: mixed integer programming library version 5. *Mathematical Programming Computation*, 3(2):103–163, 2011.

Eric Krall, Michael Higgins, and Richard P O'Neill. Rto unit commitment test system. *Federal Energy Regulatory Commission*, 98:126, 2012.

Gilbert Laporte and Frédéric Semet. An optimality cut for mixed integer linear programs. *European Journal of Operational Research*, 119(3):671–677, 1999.

Connor Lawless, Yingxi Li, Anders Wikum, Madeleine Udell, and Ellen Vitercik. Llms for cold-start cutting plane separator configuration. In *International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, pp. 51–69. Springer, 2025.

Sirui Li, Wenbin Ouyang, Max Paulus, and Cathy Wu. Learning to configure separators in branch-and-cut. *Advances in Neural Information Processing Systems*, 36:60021–60034, 2023.

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*, pp. 32201–32223, 2024.

Fei Liu, Yiming Yao, Ping Guo, Zhiyuan Yang, Xi Lin, Zhe Zhao, Xialiang Tong, Kun Mao, Zhichao Lu, Zhenkun Wang, et al. A systematic survey on large language models for algorithm design. *ACM Computing Surveys*, 58(8):1–32, 2026.

Jens Lysgaard, Adam N Letchford, and Richard W Eglese. A new branch-and-cut algorithm for the capacitated vehicle routing problem. *Mathematical programming*, 100:423–445, 2004.

Alan S Manne. On the job-shop scheduling problem. *Operations Research*, 8(2):219–223, 1960.

Clair E Miller, Albert W Tucker, and Richard A Zemlin. Integer programming formulation of traveling salesman problems. *Journal of the ACM (JACM)*, 7(4):326–329, 1960.

Germán Morales-España, Jesus M Latorre, and Andres Ramos. Tight and compact milp formulation of start-up and shut-down ramping in unit commitment. *IEEE Transactions on Power Systems*, 28(2): 1288–1296, 2012.

Germán Morales-España, Andres Ramos, and Javier García-González. An mip formulation for joint market-clearing of energy and reserves based on ramp scheduling. *IEEE Transactions on Power Systems*, 29(1): 476–488, 2013.

Germán Morales-España, Ross Baldick, Javier García-González, and Andres Ramos. Power-capacity and ramp-capability reserves for wind integration in power-based uc. *IEEE Transactions on Sustainable Energy*, 7(2):614–624, 2015.

Mahdi Mostajabdaveh, Timothy T Yu, Rindranirina Ramamonjison, Giuseppe Carenini, Zirui Zhou, and Yong Zhang. Optimization modeling and verification from problem specifications using a multi-agent multi-stage llm framework. *INFOR: Information Systems and Operational Research*, 62(4):599–617, 2024.

Mahdi Mostajabdaveh, Timothy Tin Long Yu, Samarendra Chandan Bindu Dash, Rindra Ramamonjison, Jabo Serge Byusa, Giuseppe Carenini, Zirui Zhou, and Yong Zhang. Evaluating llm reasoning in the operations research domain with orqa. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pp. 24902–24910, 2025.

OpenAI. Gpt-5.1 model documentation. `https://platform.openai.com/docs/models/gpt-5.1`, 2025. Accessed: 2026-01-28.

OpenRouter. Google: Gemini 3 pro preview. `https://openrouter.ai/google/gemini-3-pro-preview`, 2025. Accessed: 2026-05-07.

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

Power Grid Lib. Pglib-uc v19.08. `https://github.com/power-grid-lib/pglib-uc`, 2019. Accessed: 2025-05-01.

Maurice Queyranne and Yaoguang Wang. Single-machine scheduling polyhedra with precedence constraints. *Mathematics of Operations Research*, 16(1):1–20, 1991.

Rindranirina Ramamonjison, Timothy Yu, Raymond Li, Haley Li, Giuseppe Carenini, Bissan Ghaddar, Shiqi He, Mahdi Mostajabdaveh, Amin Banitalebi-Dehkordi, Zirui Zhou, et al. Nl4opt competition: Formulating optimization problems based on their natural language descriptions. In *NeurIPS 2022 competition track*, pp. 189–203. PMLR, 2023.

Daniel Rehfeldt, Thorsten Koch, and Yuji Shinano. Faster exact solution of sparse maxcut and qubo problems. *Mathematical Programming Computation*, 15(3):445–470, 2023.

Gerhard Reinelt. Tsplib—a traveling salesman problem library. *ORSA journal on computing*, 3(4):376–384, 1991.

Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Matej Balog, M Pawan Kumar, Emilien Dupont, Francisco JR 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.

SCIP Optimization Suite. SCIP parameter documentation. `https://www.scipopt.org/doc/html/PARAMETERS.php`, 2026. Accessed: 2026-05-14.

Yunzhuang Shen, Yuan Sun, Andrew Eberhard, and Xiaodong Li. Learning primal heuristics for mixed integer programs. In *2021 international joint conference on neural networks (ijcnn)*, pp. 1–8. IEEE, 2021.

FJ Srour, T Máhr, MMd Weerdt, and RA Zuidwijk. Miplib truckload pdptw instances derived from a real-world drayage case. erim report series research in management. *Finance and Accounting. Erasmus Research Institute of Management (ERIM)*, 2010.

Eric Taillard. Benchmarks for basic scheduling problems. *European Journal of Operational Research*, 64(2): 278–285, 1993.

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

Paolo Toth and Daniele Vigo. *The vehicle routing problem*. SIAM, 2002.

Mark Turner, Thorsten Koch, Felipe Serrano, and Michael Winkler. Adaptive cut selection in mixed-integer linear programming. *Open Journal of Mathematical Optimization*, 4:1–28, 2023.

Zhihai Wang, Xijun Li, Jie Wang, Yufei Kuang, Mingxuan Yuan, Jia Zeng, Yongdong Zhang, and Feng Wu. Learning cut selection for mixed-integer linear programming via hierarchical sequence model. In *International Conference on Learning Representations*, 2023. URL `https://openreview.net/forum?id=Zob4P9bRNcK`.

Ziyang Xiao, Dongxiang Zhang, Yangjun Wu, Lilin Xu, Yuan Jessica Wang, Xiongwei Han, Xiaojin Fu, Tao Zhong, Jia Zeng, Mingli Song, et al. Chain-of-experts: When llms meet complex operations research problems. In *The twelfth international conference on learning representations*, 2023.

Kaiyu Yang, Gabriel Poesia, Jingxuan He, Wenda Li, Kristin E. Lauter, Swarat Chaudhuri, and Dawn Song. Position: Formal mathematical reasoning—a new frontier in AI. In *Forty-second International Conference on Machine Learning Position Paper Track*, 2025. URL `https://openreview.net/forum?id=HuvAM5x2xG`.

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 *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. URL `https://openreview.net/forum?id=483IPG0HWL`.

Mingxuan Ye, Jie Wang, Fangzhou Zhu, Zhihai Wang, Yufei Kuang, Xijun Li, Weilin Luo, Jianye HAO, and Feng Wu. Dynamic configuration for cutting plane separators via reinforcement learning on incremental graph. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2026. URL `https://openreview.net/forum?id=iXjVis9Cwn`.

A SAGE Pseudocode

Algorithm 1 outlines the full SAGE procedure.

Algorithm 1 Pseudocode of SAGE

Require: (i) Baseline MILP Eq. (1), (ii) MILP data instances for impact assessment and admissibility, and (iii) hyperparameters: population size μ, crossover rate $\mathbb{P}_c$, mutation rate $\mathbb{P}_m = 1 - \mathbb{P}_c$, elitism ratio r_e, number of generations T, and maximum attempts for a failed generation or admissibility-screening cycle.

Ensure: A set of discovered acceleration inequality families with corresponding inequality impact scores.

Phase 1: Data Preprocessing
Set up $D_{imp} = \{\, i \mapsto \mathrm{gap}_{\mathrm{ref}}(i) \,\}$, $D_{adm} = \{\, i \mapsto ((\hat{x}_i^*, \hat{y}_i^*), (x_i^{\mathrm{LP}}, y_i^{\mathrm{LP}}), \mathrm{gap}_{\mathrm{ref}}(i)) \,\}$.
Phase 2: Population Initialization
$\mathcal{P}_1 \leftarrow \emptyset$.
while $|\mathcal{P}_1| < \mu$ **do**
  Call initializer LLM $\rightarrow$ propose candidate acceleration inequality family C.
  $s_{imp}(C) \leftarrow$ ScreenAndScoreInequality(C).
  if $s_{imp}(C) \neq$ `reject` **then**
    $\mathcal{P}_1 \leftarrow \mathcal{P}_1 \cup \{C\}$.
  else
    Provide feedback prompt to LLM and retry (up to max attempts).
  end if
end while
Phase 3: Evolution
for $t \leftarrow 1$ to T **do**
  Elitism: transfer top $\lceil r_e \mu \rceil$ from $\mathcal{P}_t$ to $\mathcal{P}_{t+1}$.
  while $|\mathcal{P}_{t+1}| < \mu$ **do**
    Select two parents (C_p, C_q) from $\mathcal{P}_t$ using tournament selection based on inequality impact score.
    Draw $u \sim \mathrm{Uniform}(0, 1)$.
    if $u < \mathbb{P}_c$ **then**
      Agent Selection: randomly choose a crossover agent from the available pool.
      Call the selected Crossover LLM with $(C_p, C_q) \rightarrow C_o$.
      $s_{imp}(C_o) \leftarrow$ ScreenAndScoreInequality(C_o).
      if $s_{imp}(C_o) \neq$ `reject` **then**
        $\mathcal{P}_{t+1} \leftarrow \mathcal{P}_{t+1} \cup \{C_o\}$.
        if $|\mathcal{P}_{t+1}| = \mu$ **then**
          break
        end if
      else
        Provide feedback prompt to the selected crossover LLM and retry (up to max attempts).
      end if
    else
      Agent Selection: randomly choose a mutation agent from the available pool.
      Call the selected Mutation LLM with one parent $C_p \rightarrow C_m$.
      $s_{imp}(C_m) \leftarrow$ ScreenAndScoreInequality(C_m).
      if $s_{imp}(C_m) \neq$ `reject` **then**
        $\mathcal{P}_{t+1} \leftarrow \mathcal{P}_{t+1} \cup \{C_m\}$.
      else
        Provide feedback prompt to the selected mutation LLM and retry (up to max attempts).
      end if
    end if
  end while
end for
return final population $\mathcal{P}_{T+1}$.

In Algorithm 1, ScreenAndScoreInequality(C) denotes the local routine that first runs InequalityAdmissibilityScreen(C) on D_{adm}. If any admissibility check fails, the routine returns `reject`. Otherwise, InequalityImpactScorer(C) computes and returns the inequality impact score $s_{imp}(C)$ on D_{imp}.

B SAGE Agents

This section outlines the agents used in the SAGE process and presents their prompt structures, including roles, tasks, requirements, and inputs/outputs. The agent library includes a single initializer and several Mutation and Crossover agents. Consistent with the framing in Section 4, these prompts seek useful acceleration inequality families, while the Inequality Admissibility Screen later decides which candidates survive based on empirical admissibility checks. The prompt structure for the initializer agent is as follows:

initializer agent

Role: You are an MILP optimization expert with extensive knowledge in proposing acceleration inequality families for MILP models.
Task:

- Propose a useful acceleration inequality family that is likely to improve solver performance.
- Provide a clear, concise explanation of the inequality intuition and its expected impact.

Requirements:

- Do not assume a formal proof of global validity is available.
- The proposal should be broadly reusable across instances of the symbolic MILP formulation.
- Introduce new variables or constructs if needed.
- The idea should be conceptually distant from previous ideas.

Input:

- *< Baseline MILP (Pyomo model code) >*.
- *< List of previous ideas (if applicable) >*.

Output:

```
{"code": "<Only python code snippet for the added inequality family using Pyomo syntax>",
"idea": "Concise technical explanation of the inequality intuition and expected impact"}
```

We employ three mutation agents, General Mutation, Lifted Mutation, and Exploratory Mutation, each of which follows the prompt structure described below.

Mutation Agents

Role: MILP optimization expert tasked with analyzing and modifying an individual's acceleration inequality family in an Evolutionary Algorithm for MILP acceleration inequality family generation.
Task:

- Propose a useful acceleration inequality family by building on the idea behind the provided individual family.
- Provide a concise explanation of the inequality intuition and expected impact.
- *<Agent specific instruction>*

Requirements:

- Do not assume a formal proof of global validity is available.
- The new family should improve on the provided input family in spirit, structure, or empirical promise.
- Introduce new variables if needed.

Input:

- *< Baseline MILP (Pyomo model code) >.*
- *< Input inequality family (code, idea, and score) >.*

Output:

```
{"code": "<Only python code snippet for the added inequality family using Pyomo syntax>",
"idea": "Concise technical explanation of the inequality intuition and expected impact"}
```

Agent specific instruction: Each mutation agent performs the same core task, with the following variations:

- *General Mutation*: Proposes a new family based on the individual family while modifying its structure or coefficients.
- *Lifted Mutation*: Enhances the family by borrowing lifting-style ideas or auxiliary structure when useful.
- *Exploratory Mutation*: Generates an exploratory family that diverges from the provided family to explore new regions of the design space.

Similarly, SAGE includes four crossover agents (Intersection, Complementary, Hybrid, and Min Violation) with the following prompt design.

Crossover Agents

Role: MILP optimization expert tasked with performing crossover between two parent acceleration inequality families.
Task:

- Generate a useful acceleration inequality family by combining elements from the parent families.
- Provide a concise explanation of the new family's idea and expected impact.
- *<Agent specific instruction>*

Requirements:

- Do not assume a formal proof of global validity is available.
- Combine elements from both parent families in a distinct and effective way.
- Introduce new variables or auxiliary constructs if needed.

Input:

- *< Baseline MILP (Pyomo model code) >.*
- *< First parent inequality family (code, idea, and score) >.*
- *< Second parent inequality family (code, idea, and score) >.*

Output:

```
{"code": "<Only python code snippet for the added inequality family using Pyomo syntax>",
"idea": "Concise technical explanation of the inequality intuition and expected impact"}
```

Agent specific instruction: Each crossover agent performs the same core task, with the following variations:

- Intersection Crossover: Combines elements of both parents to generate a family that preserves useful structural motifs from each.
- Complementary Crossover: Generates a family that complements both parents, creating a more distinct solution.
- Hybrid Crossover: Combines structural elements from one parent with numerical or conditional features from the other.
- Min Violation Crossover: Selects a crossover that minimizes the joint violation of both parents in previous runs.

C Detailed Evolutionary Diagnostics

C.1 Agent-Level Performance Statistics

We report agent-level outcomes for each SAGE agent. For each generated offspring, we compute the relative improvement rate $\Delta_{imp} = (s_{imp,\text{child}} - s_{imp,\text{parent}})/s_{imp,\text{parent}}$, where $s_{imp,\text{parent}}$ is the average inequality impact score of the parent population (a single value for mutation and the mean of two parents for crossover). For the initializer agent, the reference value is the neutral inequality impact score 10 that corresponds to "no added acceleration inequality family".

Table 5 reports five statistics collected over the full run on all benchmarks: code generation failures, OSS failures, LP-separation rejections, overall success rate, and the average improvement $\bar{\Delta}_{imp}$ of successful offspring.

C.2 Evolutionary Generation of a Novel Acceleration Inequality Family

Fig. 4 shows how SAGE with DeepSeek-R1 mutates and recombines candidate acceleration inequality families to obtain a strong family for the JSSP. The blue panel on the right shows the baseline MILP (minimizing makespan $C_{\max}$ under big-M disjunctive constraints). Green boxes list candidate families with their inequality impact scores. Arrows connect parents to offspring, and yellow panels quote the agent instructions that produced the offspring.

C.3 Population-Level Inequality Impact Score Progress

Fig. 5 plots the best (blue, dashed) and mean (orange, dash-dotted) inequality impact scores observed over 20 generations of SAGE on the TSP, MCND, CWLP, JSSP, IMO6, PDPTW, and SHUC benchmarks. Shaded bands indicate one standard deviation.

D Supplementary Mathematical Preliminaries

D.1 Geometric intuition for valid inequalities

Fig. 6 provides a schematic view of the integer feasible set S, its LP relaxation P, and the convex hull $\text{conv}(S)$. It also contrasts (i) valid inequalities that remove fractional LP optima while preserving all points in $\text{conv}(S)$, and (ii) optimality-preserving inequalities that may remove some feasible points but keep the optimal objective value unchanged. SAGE instead generates acceleration inequality families that are empirically checked on an admissibility set (Section 4).

Table 5: SAGE agent outcomes across three LLMs: code fail, OSS fail, LP-separation rejection, success rate, and mean inequality-impact-score gain $\bar{\Delta}_{imp} \pm \sigma$ (%) (successful offspring only).

Agent	LLM	Code Fail.%	OSS Fail.%	Not Useful %	Success Rate %	$\bar{\Delta}_{imp} \pm \sigma$ (%)
			Initializer			
Main	DeepSeek	38.1	11.1	25.4	25.4	12.3 ± 20.8
	Gemini 3 Pro	5.6	12.8	45.1	36.5	14.8 ± 19.9
	GPT-5.1	8.9	14.7	46.8	29.6	10.4 ± 19.5
			Mutation			
General	DeepSeek	17.6	13.2	2.9	66.2	15.5 ± 16.6
	Gemini 3 Pro	3.1	10.2	3.4	83.3	17.2 ± 15.1
	GPT-5.1	5.6	12.5	4.8	77.1	12.8 ± 15.3
Exploratory	DeepSeek	3.3	13.1	1.6	82.0	17.9 ± 20.8
	Gemini 3 Pro	1.4	11.3	2.1	85.2	19.1 ± 18.0
	GPT-5.1	3.2	13.4	2.9	80.5	13.7 ± 17.4
Lifted	DeepSeek	12.2	18.4	6.1	63.3	8.1 ± 13.1
	Gemini 3 Pro	2.8	15.9	6.4	74.9	9.6 ± 12.1
	GPT-5.1	4.9	18.2	8.1	68.8	6.1 ± 11.7
			Crossover			
Hybrid	DeepSeek	13.2	15.1	3.8	67.9	3.0 ± 11.0
	Gemini 3 Pro	2.5	12.2	4.6	80.7	4.4 ± 10.2
	GPT-5.1	4.3	14.8	6.1	74.8	2.0 ± 9.5
Intersect	DeepSeek	7.7	9.9	5.5	76.9	1.3 ± 8.7
	Gemini 3 Pro	1.9	8.7	5.0	84.4	2.2 ± 8.1
	GPT-5.1	3.7	10.9	6.4	79.0	1.1 ± 7.8
Complement	DeepSeek	4.2	14.3	6.7	74.8	14.7 ± 13.9
	Gemini 3 Pro	1.6	11.4	6.9	80.1	15.6 ± 12.7
	GPT-5.1	3.3	14.2	8.1	74.4	10.8 ± 13.6
Min Violation	DeepSeek	10.1	11.9	9.2	68.8	-0.4 ± 9.6
	Gemini 3 Pro	2.3	10.8	9.1	77.8	0.6 ± 8.7
	GPT-5.1	4.1	13.6	10.2	72.1	-0.3 ± 9.0

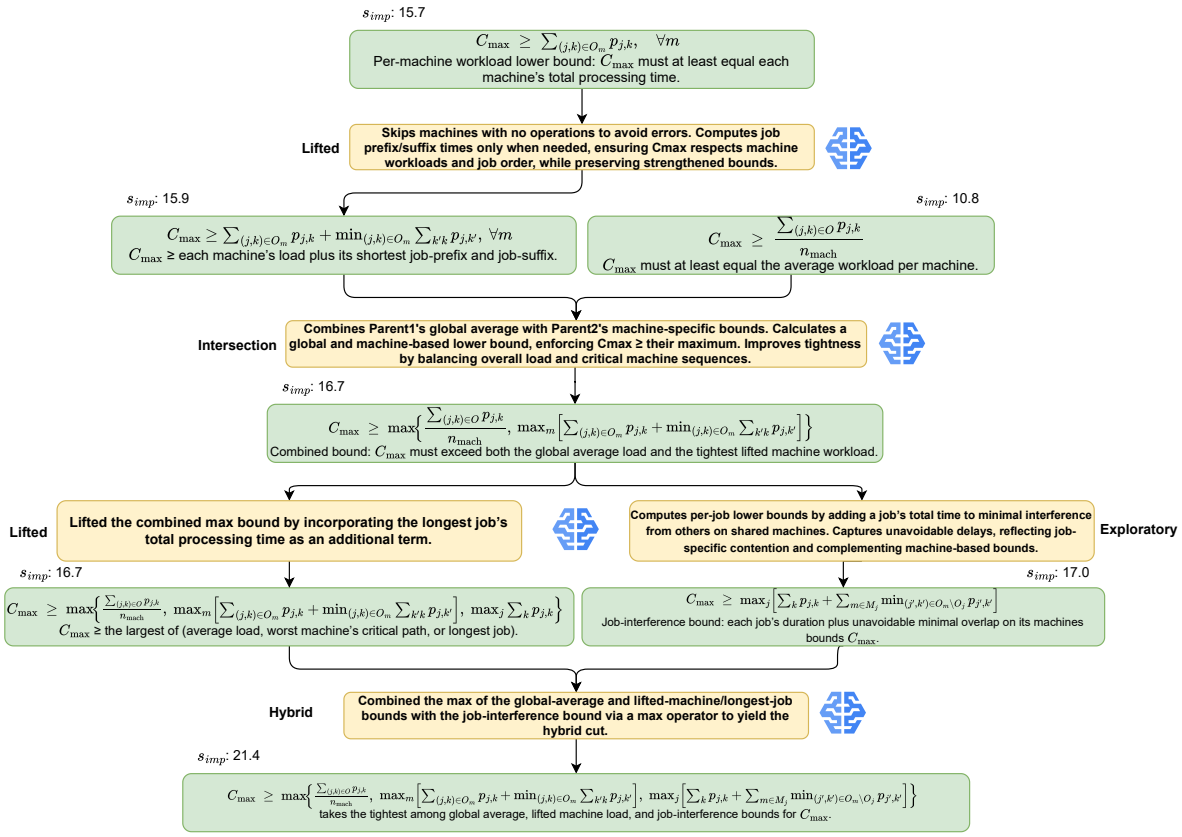

Figure 4: Example JSSP evolutionary trajectory. The blue panel shows the baseline formulation, the green panels show candidate families with annotated inequality impact scores, and the yellow panels show the agent instructions that produced each offspring.

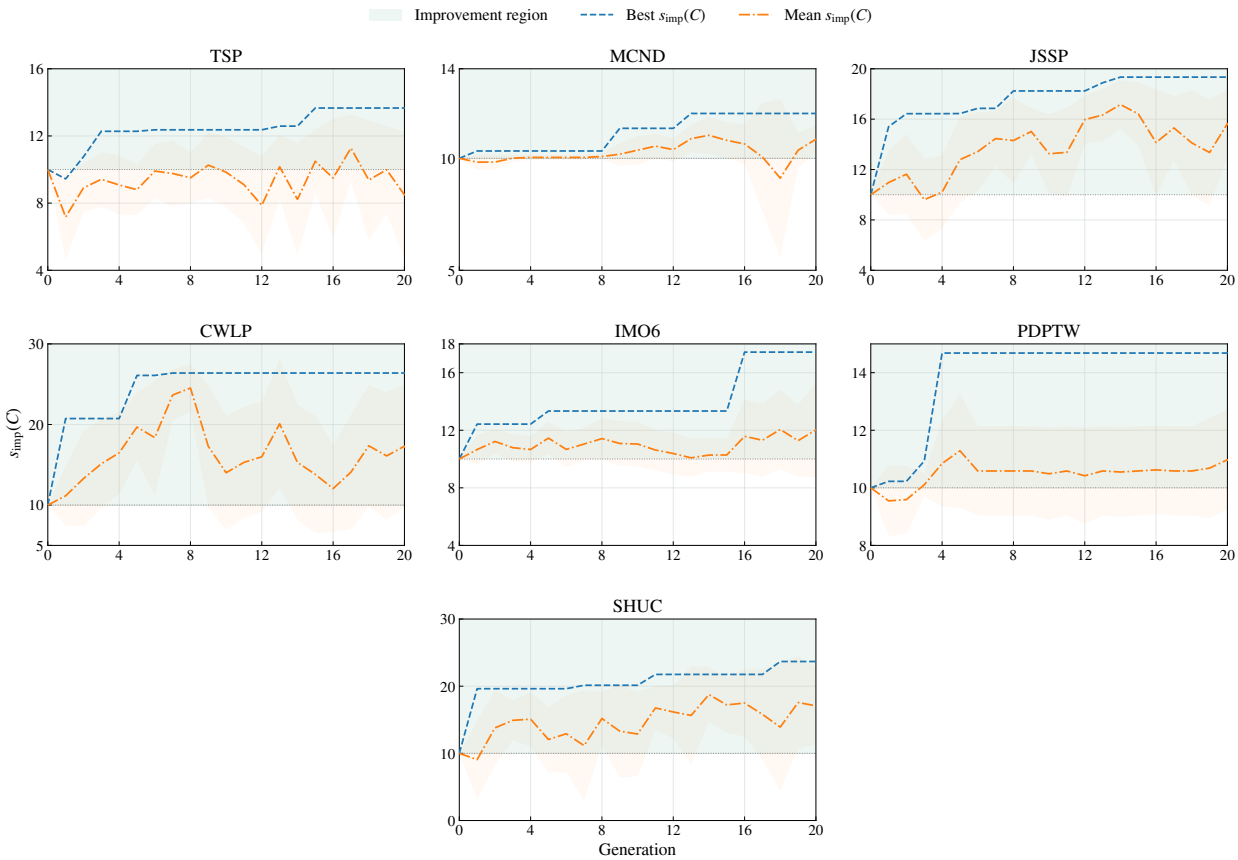

Figure 5: Best (blue, dashed) and mean (orange, dash-dotted) inequality impact scores across 20 generations for the TSP, MCND, CWLP, JSSP, IMO6, PDPTW, and SHUC benchmarks. Shaded regions show one standard deviation.

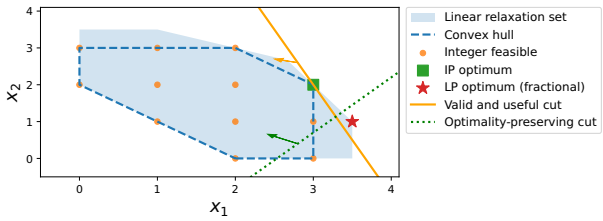

Figure 6: Relationship between the integer feasible set S, its LP relaxation P, the set $\text{conv}(S)$, a valid inequality that removes the fractional LP optimum (useful), and an optimality-preserving inequality that may remove some feasible points while keeping the optimal value.

D.2 Polyhedra and Facets

A polyhedron is any set Q in $\mathbb{R}^n$ that can be described by finitely many linear inequalities. The constraints $Ax \leq b$ are the linear inequalities that define Q.

$$Q \;=\; \{\, x \in \mathbb{R}^n : Ax \leq b\}.$$

Polyhedra are convex, and if Q is bounded, it is called a polytope.

A face of a polyhedron Q is a set of the form $F := Q \cap \{x \in \mathbb{R}^n : cx = \beta\}$ where $cx \leq \beta$ is a valid inequality for Q. We say that the valid inequality $cx \leq \beta$ defines the face F of the polyhedron Q. The equation $cx = \beta$ defines a supporting hyperplane that intersects Q. A facet is a face of maximum dimension (i.e., one dimension less than that of the polyhedron).

For example, consider a cube as a 3D polytope in $\mathbb{R}^3$ and a plane as the supporting hyperplane. Intersecting a cube with a supporting hyperplane may result in a 0D corner point, a 1D straight line, or a 2D plane. The corner point, the line (edge of a cube), and the 2D plane are all faces of the 3D cube. However, only the 2D plane is a facet of the 3D cube.

D.3 Facet-Defining Valid Inequalities

In integer programming, valid inequalities that are facet-defining for $\mathrm{conv}(S)$ are especially important because they are necessary and sufficient for characterizing $\mathrm{conv}(S)$. Once $\mathrm{conv}(S)$ is characterized, solving the MILP over S reduces to the linear program of maximizing $c^\top x + h^\top y$ over $\mathrm{conv}(S)$, which is polynomially solvable.

E Additional Experiments

E.1 Aggregate gap improvement curves

Fig. 7 reports the mean stabilized relative gap improvement $\bar{\Delta}_g(t)$ as a function of time after augmenting the baseline MILP with the selected SAGE acceleration inequality family, averaged across held-out test instances (TSP, MCND, CWLP, JSSP, PDPTW, IMO6, and SHUC). This plot complements the checkpoint summaries in Table 1 by showing the full 2000 s profile rather than only selected checkpoints.

E.2 Held-Out LP Separation Rates

As a robustness and admissibility check, we also test whether the selected SAGE acceleration inequality family separates the baseline LP optimum on held-out test instances. This repeats the LP-separation check from Section 4, but uses D_t instead of the small admissibility set D_{adm}. For each test instance, we fix the original decision variables to the baseline LP optimum, append the selected acceleration inequality family, and solve only for any auxiliary variables introduced by the family. If this reduced model is infeasible, the family separates that LP optimum.

Table 6 shows that the selected acceleration inequality family separates the LP optimum on all TSP, CWLP, JSSP, PDPTW, and IMO6 test instances. For MCND and SHUC, the measured separation rates are 88.9% and 85.7%, respectively. This check is empirical and uses one LP optimum per instance, so it does not prove validity or separation of the whole LP relaxation.

E.3 Static Literature Augmentation Baselines

This experiment compares the selected static SAGE acceleration inequality families with a static set of valid inequalities from the literature. We keep the same base formulations, test sets, solver budget, and time-to-target evaluation protocol used in Section 5.4. The only change is that the SAGE acceleration inequality families are replaced by the corresponding published valid inequalities. Since several of these inequalities are indexed by node subsets, this is a static model augmentation comparison rather than a dynamic separation implementation.

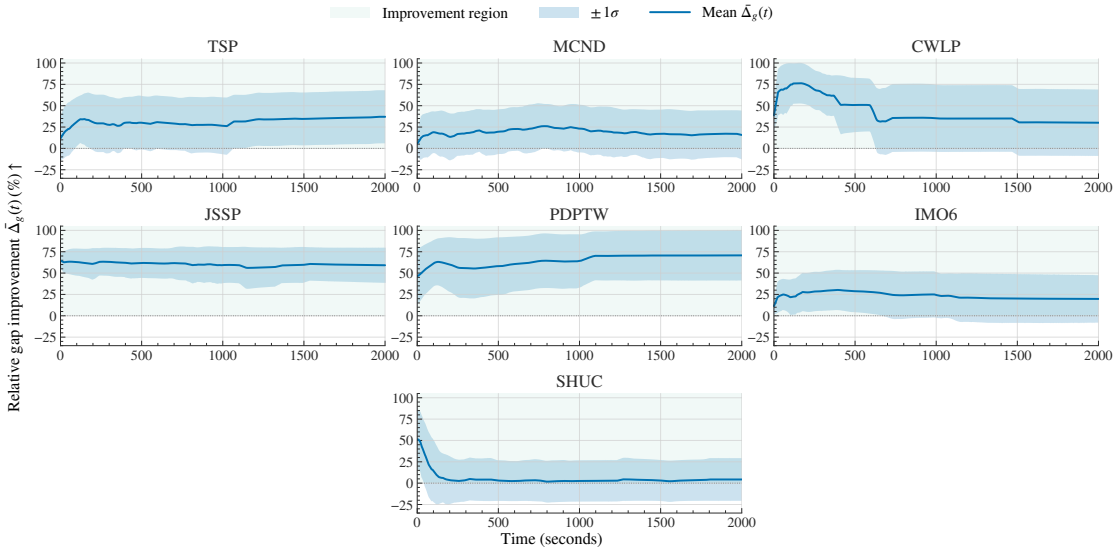

Figure 7: Change over time (up to 2000 s) of the mean stabilized relative gap improvement $\bar{\Delta}_g(t)$ after adding the selected SAGE acceleration inequality family to the baseline MILP for TSP, MCND, CWLP, JSSP, PDPTW, IMO6, and SHUC. The solid line is the mean across test instances at each wall-clock time and the shaded band indicates one standard deviation (σ). Positive values mean the augmented model has a smaller optimality gap at that wall-clock time.

Table 6: Held-out test-set LP separation rate of the selected SAGE acceleration inequality family.

Problem	Separated / Total	Rate (%)
TSP	54/54	100.0
MCND	72/81	88.9
CWLP	36/36	100.0
JSSP	80/80	100.0
PDPTW	165/165	100.0
IMO6	34/34	100.0
SHUC	48/56	85.7

Let $\mathcal{G} = \{10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}\}$. For an instantiated static literature valid inequality set L, Table 7 reports

$$\bar{s}_L = \frac{1}{|\mathcal{G}|} \sum_{g \in \mathcal{G}} \frac{\mathrm{SGM}(t_{\mathrm{ref}}(g))}{\mathrm{SGM}(t_L(g))},$$

using the same shifted geometric mean convention as Table 2. Values larger than one mean that the model augmented with static literature valid inequalities reaches the target gaps faster than the baseline MILP on average.

Table 7: Mean SGM speedup for static literature valid inequalities and SAGE acceleration inequality families. Each value is averaged over the same five target gaps used in Table 2.

Problem	Static literature valid inequalities	SAGE families
CWLP	3.01×	4.02×
JSSP	1.91×	3.33×
MCND	1.01×	1.01×
TSP	2.57×	2.87×

Table 7 shows that the SAGE acceleration inequality families yield a larger mean speedup than the instantiated static literature valid inequalities on CWLP, JSSP, and TSP, and are essentially tied on MCND. The average over these four rows is 2.81× for SAGE and 2.13× for the static literature valid inequalities. The largest absolute gap is on JSSP (3.33× vs. 1.91×, a 1.42× margin), followed by CWLP (4.02× vs. 3.01×, a 1.01× margin). TSP is close (2.87× vs. 2.57×, a 0.30× margin), and MCND is tied (1.01× vs. 1.01×).

CWLP inequalities. For each warehouse j, the capacity row in Eq. (CWLP) induces a 0–1 knapsack structure over customer-assignment variables x_{ij}. Following the 0–1 knapsack inequality in Balas Balas (1975) and the facility-location inequalities in Aardal et al. Aardal et al. (1995), any set $C \subseteq I$ with $\sum_{i \in C} d_i > u_j$ gives

$$\begin{cases} \displaystyle\sum_{i \in C} x_{ij} \le (|C| - 1) y_j & \forall j \in J,\ C \subseteq I : \displaystyle\sum_{i \in C} d_i > u_j, \\ \displaystyle\sum_{i \in I:\ d_i > u_j/(q+1)} x_{ij} \le q\, y_j & \forall j \in J,\ q = 1, \ldots, |I| - 1. \end{cases} \tag{LIT-CWLP}$$

The second row is the threshold form of the same inequality: if each selected customer has demand greater than $u_j/(q+1)$, then no feasible integer solution can assign $q+1$ such customers to warehouse j. This is the same local capacity logic present in the CWLP SAGE acceleration inequality family, but the SAGE version also includes global demand-threshold slot bounds.

JSSP inequalities. Using the JSSP notation from Appendix F.4, write an operation as $a = (j, k)$, its start time as $S_a := S_{j,k}$, and its processing time as $p_a := p_{j,k}$. For $Q \subseteq O_m$, define

$$P_Q := \sum_{a \in Q} p_a, \qquad H_Q := \min_{a=(j,k) \in Q} \sum_{t<k} p_{j,t}, \qquad T_Q := \min_{a=(j,k) \in Q} \sum_{t>k} p_{j,t},$$

and $\Phi_Q := \sum_{\{a,b\} \subseteq Q} p_a p_b$. Applegate and Cook Applegate & Cook (1991) give the two inequalities

$$\begin{cases} \displaystyle\sum_{a \in Q} p_a S_a \ge H_Q P_Q + \Phi_Q & \forall m,\ Q \subseteq O_m, \\ \displaystyle\sum_{a \in Q} p_a (C_{\max} - S_a) \ge T_Q P_Q + \sum_{a \in Q} p_a^2 + \Phi_Q & \forall m,\ Q \subseteq O_m. \end{cases} \tag{LIT-JSSP}$$

These inequalities use job heads and tails to bound how early a set of operations on one machine can start and how late they can finish. In our experiments they yield a meaningful mean speedup (Table 7), while the SAGE acceleration inequality family in Appendix F.4 acts directly as a makespan lower-bound family and produces a larger speedup on the same benchmark.

MCND inequalities. For a node subset $S \subseteq N$, let $\delta^+(S)$, $K(S)$, $B \subseteq K(S)$, and D_B be defined as in Appendix F.2. Chouman et al. Chouman et al. (2017) define $b_{ij}^B := \min\{u_{ij}, D_B\}$ for this node subset relaxation and include the following two inequalities:

$$\begin{cases} \sum_{(i,j)\in\delta^+(S)} \min\{u_{ij}, D_B\}\, y_{ij} \geq D_B & \forall S \subseteq N,\ B \subseteq K(S), \\ \sum_{(i,j)\in\delta^+(S)} y_{ij} \geq q_{S,B} & \forall S \subseteq N,\ B \subseteq K(S), \end{cases} \tag{LIT-MCND}$$

where $q_{S,B}$ is the smallest number of largest-capacity arcs in $\delta^+(S)$ whose total capacity can cover D_B. The selected SAGE MCND rows in Eq. (SAGE-MCND) use the same algebraic form, but they are added only for the generated pairs used by the implementation.

TSP inequalities. The compact TSP model in Eq. (MTZ) uses the Miller–Tucker–Zemlin ordering variables Miller et al. (1960). Desrochers and Laporte Desrochers & Laporte (1991) give the following lifted ordering rows in the notation of Appendix F.1:

$$\begin{cases} u_i - u_j + (n-1)x_{ij} + (n-3)x_{ji} \leq n-2 & \forall i,j \in V \setminus \{1\},\ i \neq j, \\ u_i \geq 3 - x_{1i} + (n-3)x_{i1} & \forall i \in V \setminus \{1\}, \\ u_i \leq (n-1) + x_{i1} - (n-3)x_{1i} & \forall i \in V \setminus \{1\}. \end{cases} \tag{LIT-TSP}$$

The TSP SAGE acceleration inequality family in Eq. (SAGE-TSP) contains these rows and augments them with explicit two-cycle and depot-anchored short-cycle eliminators.

E.4 Complementarity with Learned Cut-Selection and Separator Policies

We run a factorial CWLP experiment on the held-out 36-instance OR-Library benchmark, with one random seed and a 2000 s SCIP 10.0 time limit, to check how SAGE interacts with two learned runtime methods that act inside branch-and-bound: the Turner et al. adaptive cut selector Turner et al. (2023) and the Lawless et al. cold-start LLM separator configuration Lawless et al. (2025). The factorial crosses SAGE (off / on, at the model layer) with the ML policy (off / Turner / Lawless-LLM(0) / Lawless-LLM(5), at the solver layer). Turner changes SCIP's cut selector by scoring generated cuts before SCIP decides which cuts to keep. The Turner column uses the reinforcement learning setup from Turner et al. (2023). Lawless changes SCIP's separator settings before solving by choosing configurations generated by GPT-4o. The Lawless columns use 100 separator configurations generated by GPT-4o, clustered into LLM(0) (largest-cluster medoid) and LLM(5) (k-medoids with k=5), exactly as in Lawless et al. (2025). We report the paper's shifted-geometric-mean speedup (SGM, shift s=1, PAR-2 penalty at 2000 s) from Section 5.4 at the five target gaps used in Table 2, and the row-averaged speedup.

Three observations follow from Table 8. (i) Each method gives benefit on its own on CWLP at the paper's time budget: Turner reaches the target gaps 2.12× faster than plain SCIP on average, and Lawless et al. LLM(5) reaches them 2.60× faster. (ii) SAGE alone is the strongest single intervention at 3.13×, consistent with its model-layer effect of adding empirically screened acceleration inequality families to the formulation before solving (Section 5.4). (iii) Combining SAGE with the learned solver-layer policies gives higher row-average SGM speedups than SAGE alone in this CWLP experiment: Turner reaches 3.45×, Lawless et al. LLM(0) reaches the fastest cell at 3.51×, and Lawless et al. LLM(5) reaches 3.43×. All three combined cells (3.43× to 3.51×) are above SAGE alone (3.13×) and above the matching cells without SAGE (2.12×, 2.58×, and 2.60×). This experiment separates the stages: SAGE modifies the formulation before SCIP is called, while Turner and Lawless change choices inside SCIP.

E.5 Solver Default Parameters

Table 9 lists important solver defaults that were not set by our experiments but can affect MILP search. We omit `MIPGap`, `Seed`, `Method`, `WorkLimit`, `TimeLimit`, and the stated Gurobi thread setting because these are already given in Section 5.1. The listed values are documented defaults, not tuned settings (Gurobi Optimization LLC, 2026; SCIP Optimization Suite, 2026; HiGHS Development Team, 2026).

Table 8: CWLP factorial with learned runtime policies, held-out 36-instance OR-Library benchmark, SCIP 10.0, 2000 s time limit, single seed. The reported value is the average over the five target gaps $g \in \{10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}\}$ of the time-to-target SGM speedup vs the reference cell (SAGE off, plain SCIP defaults), following the protocol of Table 2. Values > 1 are faster. The reference cell is $1.00\times$ by construction.

SAGE	Solver-layer policy	Avg SGM speedup
Off	SCIP default	1.00
Off	Turner et al.	2.12
Off	Lawless et al. LLM(0)	2.58
Off	Lawless et al. LLM(5)	2.60
On	SCIP default	3.13
On	Turner et al.	3.45
On	Lawless et al. LLM(0)	**3.51**
On	Lawless et al. LLM(5)	3.43

Table 9: Important default solver parameters not otherwise listed in the experimental setup.

Solver	Parameter	Default	Role
Gurobi	`MIPGapAbs`	`1e-10`	Absolute optimality gap tolerance.
Gurobi	`Presolve`	`-1`	Automatic presolve control.
Gurobi	`Cuts`	`-1`	Automatic global cut generation control.
Gurobi	`Heuristics`	`0.05`	Target fraction of runtime for primal heuristics.
Gurobi	`MIPFocus`	`0`	Balanced MIP search focus.
Gurobi	`Symmetry`	`-1`	Automatic symmetry detection.
SCIP 10.0	`limits/gap`	`0`	Relative optimality gap limit.
SCIP 10.0	`limits/absgap`	`0`	Absolute optimality gap limit.
SCIP 10.0	`presolving/maxrounds`	`-1`	Automatic number of presolving rounds.
SCIP 10.0	`separating/maxrounds`	`-1`	Automatic separation rounds after root.
SCIP 10.0	`separating/maxroundsroot`	`-1`	Automatic separation rounds at the root node.
SCIP 10.0	`separating/maxcuts`	`100`	Cut pool limit outside the root node.
SCIP 10.0	`separating/maxcutsroot`	`2000`	Cut pool limit at the root node.
SCIP 10.0	`misc/usesymmetry`	`7`	Symmetry handling mode.
SCIP 10.0	`parallel/maxnthreads`	`8`	Maximum number of parallel threads.
HiGHS	`mip_rel_gap`	`1e-4`	Relative optimality gap tolerance.
HiGHS	`mip_abs_gap`	`1e-6`	Absolute optimality gap tolerance.
HiGHS	`presolve`	`choose`	Automatic presolve control.
HiGHS	`mip_heuristic_effort`	`0.05`	Effort level for MIP heuristics.
HiGHS	`mip_allow_cut_separation_at_nodes`	`true`	Enables cut separation at branch-and-bound nodes.
HiGHS	`mip_detect_symmetry`	`true`	Enables MIP symmetry detection.
HiGHS	`threads`	`0`	Automatic thread count.
HiGHS	`parallel`	`choose`	Automatic parallel execution mode.

Table 10: `Gemini 3 Pro` selected acceleration inequality family: test stabilized $\bar{\Delta}_g$ (%) with Gurobi vs. HiGHS. Mean $\pm\sigma$ is reported at each checkpoint. Average reports the row average of checkpoint means.

Problem	Solver	5 s	10 s	50 s	150 s	300 s	1000 s	2000 s	Average
TSP	Gurobi	13.1 (± 27.5)	15.0 (± 27.9)	22.6 (± 31.3)	34.2 (± 30.3)	27.6 (± 33.1)	26.4 (± 33.3)	37.0 (± 31.1)	25.1
	HiGHS	2.5 (± 13.8)	2.9 (± 14.9)	11.3 (± 23.3)	18.6 (± 26.5)	18.5 (± 27.2)	21.4 (± 28.1)	21.9 (± 28.7)	13.9
MCND	Gurobi	6.8 (± 17.7)	8.2 (± 19.9)	15.1 (± 25.8)	17.4 (± 26.9)	17.8 (± 25.5)	23.2 (± 27.9)	15.8 (± 28.6)	14.9
	HiGHS	6.0 (± 15.1)	7.0 (± 16.9)	6.6 (± 22.5)	1.4 (± 27.7)	5.5 (± 28.4)	2.0 (± 29.3)	5.7 (± 29.0)	4.9
CWLP	Gurobi	43.8 (± 20.7)	49.4 (± 22.3)	68.8 (± 28.6)	76.3 (± 23.8)	64.8 (± 23.0)	35.2 (± 39.3)	30.1 (± 38.9)	52.6
	HiGHS	25.5 (± 24.6)	25.3 (± 24.6)	23.0 (± 24.8)	28.1 (± 29.8)	39.1 (± 36.2)	62.3 (± 32.3)	64.2 (± 31.9)	38.2
JSSP	Gurobi	64.1 (± 9.9)	63.7 (± 10.7)	63.3 (± 14.4)	61.8 (± 16.6)	63.2 (± 16.5)	59.4 (± 21.2)	59.2 (± 20.6)	62.1
	HiGHS	12.8 (± 16.5)	14.1 (± 17.2)	18.9 (± 22.4)	24.6 (± 24.8)	21.0 (± 25.6)	26.7 (± 26.5)	27.3 (± 27.5)	20.8
PDPTW	Gurobi	2.7 (± 3.4)	3.8 (± 4.1)	16.4 (± 12.3)	31.4 (± 23.4)	19.1 (± 30.5)	34.4 (± 30.2)	46.4 (± 31.7)	22.0
	HiGHS	-0.4 (± 4.1)	0.3 (± 4.6)	3.1 (± 12.8)	7.5 (± 21.7)	4.2 (± 26.9)	9.8 (± 28.4)	12.3 (± 21.9)	5.3
IMO6	Gurobi	13.6 (± 11.2)	15.7 (± 13.5)	24.6 (± 18.4)	25.0 (± 21.4)	28.9 (± 23.0)	24.3 (± 26.8)	19.7 (± 27.8)	21.7
	HiGHS	5.3 (± 12.9)	6.7 (± 14.5)	12.1 (± 19.4)	14.8 (± 21.3)	16.0 (± 22.7)	13.5 (± 25.6)	18.6 (± 24.8)	12.4
SHUC	Gurobi	51.2 (± 35.1)	50.6 (± 34.8)	32.2 (± 37.7)	6.3 (± 29.4)	3.7 (± 24.9)	2.6 (± 24.2)	4.3 (± 25.0)	21.6
	HiGHS	1.9 (± 9.2)	1.8 (± 9.7)	-2.7 (± 11.4)	3.8 (± 23.0)	17.8 (± 32.3)	24.3 (± 28.2)	20.4 (± 30.5)	9.6

Table 11: `Gemini 3 Pro` selected acceleration inequality family: test stabilized $\bar{\Delta}_g$ (%) with Gurobi internal cuts disabled vs. default. Mean $\pm\sigma$ is reported at each checkpoint. Average reports the row average of checkpoint means.

Problem	Solver	5 s	10 s	50 s	150 s	300 s	1000 s	2000 s	Average
TSP	No internal cuts	34.9 (± 25.1)	37.8 (± 25.5)	53.5 (± 23.3)	61.6 (± 18.1)	64.2 (± 16.5)	64.0 (± 16.8)	62.0 (± 20.2)	54.0
	Default internal cuts	13.1 (± 27.5)	15.0 (± 27.9)	22.6 (± 31.3)	34.2 (± 30.3)	27.6 (± 33.1)	26.4 (± 33.3)	37.0 (± 31.1)	25.1
MCND	No internal cuts	2.5 (± 10.0)	2.9 (± 11.0)	4.3 (± 14.8)	3.4 (± 15.3)	1.7 (± 17.1)	4.1 (± 17.2)	5.1 (± 13.6)	3.4
	Default internal cuts	6.8 (± 17.7)	8.2 (± 19.9)	15.1 (± 25.8)	17.4 (± 26.9)	17.8 (± 25.5)	23.2 (± 27.9)	15.8 (± 28.6)	14.9
CWLP	No internal cuts	53.6 (± 19.8)	60.6 (± 21.2)	86.7 (± 19.8)	84.3 (± 28.3)	94.8 (± 8.8)	94.5 (± 8.9)	93.9 (± 9.2)	81.2
	Default internal cuts	43.8 (± 20.7)	49.4 (± 22.3)	68.8 (± 28.6)	76.3 (± 23.8)	64.8 (± 23.0)	35.2 (± 39.3)	30.1 (± 38.9)	52.6
JSSP	No internal cuts	64.2 (± 14.6)	62.8 (± 16.1)	67.5 (± 20.3)	71.7 (± 23.5)	64.9 (± 22.7)	69.3 (± 23.9)	63.6 (± 19.7)	66.3
	Default internal cuts	64.1 (± 9.9)	63.7 (± 10.7)	63.3 (± 14.4)	61.8 (± 16.6)	63.2 (± 16.5)	59.4 (± 21.2)	59.2 (± 20.6)	62.1
PDPTW	No internal cuts	1.7 (± 6.2)	2.9 (± 6.9)	12.4 (± 16.8)	22.5 (± 28.9)	16.0 (± 33.5)	27.8 (± 34.2)	34.2 (± 32.6)	16.8
	Default internal cuts	2.7 (± 3.4)	3.8 (± 4.1)	16.4 (± 12.3)	31.4 (± 23.4)	19.1 (± 30.5)	34.4 (± 30.2)	46.4 (± 31.7)	22.0
IMO6	No internal cuts	11.9 (± 13.6)	13.7 (± 15.1)	21.5 (± 19.2)	23.4 (± 21.1)	26.8 (± 22.5)	20.6 (± 26.3)	24.3 (± 26.7)	20.3
	Default internal cuts	13.6 (± 11.2)	15.7 (± 13.5)	24.6 (± 18.4)	25.0 (± 21.4)	28.9 (± 23.0)	24.3 (± 26.8)	19.7 (± 27.8)	21.7
SHUC	No internal cuts	44.3 (± 36.8)	41.6 (± 35.9)	28.2 (± 38.4)	10.7 (± 31.6)	8.2 (± 28.9)	4.6 (± 27.8)	3.4 (± 26.9)	20.1
	Default internal cuts	51.2 (± 35.1)	50.6 (± 34.8)	32.2 (± 37.7)	6.3 (± 29.4)	3.7 (± 24.9)	2.6 (± 24.2)	4.3 (± 25.0)	21.6

E.6 Cross-Solver Robustness for `Gemini 3 Pro` Families

Table 10 compares $\bar{\Delta}_g$ when using the same acceleration inequality families discovered with `Gemini 3 Pro` but solving with two different MILP solvers (Gurobi vs. HiGHS). This cross-solver robustness ablation checks whether the observed gains are specific to a particular solver or persist under a different branch-and-bound implementation and default cut setting.

In Table 10, the HiGHS 2000 s means are positive for 7/7 benchmarks and range from 5.7% (MCND) to 64.2% (CWLP). At 2000 s, HiGHS is larger than Gurobi on CWLP (64.2% vs. 30.1%) and SHUC (20.4% vs. 4.3%) and smaller than Gurobi on the other 5 benchmarks.

E.7 Cut-Setting Interaction (`Gemini 3 Pro` Families)

Table 11 reports $\bar{\Delta}_g$ under Gurobi with internal cuts disabled versus default settings. This check shows how much of SAGE's benefit remains when the solver's own cutting-plane machinery is reduced and whether the interaction between instantiated acceleration inequality families and solver-generated cuts changes the net improvement.

In Table 11, disabling internal cuts gives larger $\bar{\Delta}_g$ than default settings for TSP and CWLP at 7/7 checkpoints. MCND and PDPTW have smaller values with internal cuts disabled at 7/7 checkpoints. At 2000 s, the no-internal-cuts and default values are 62.0% vs. 37.0% for TSP, 93.9% vs. 30.1% for CWLP, 5.1% vs. 15.8% for MCND, and 34.2% vs. 46.4% for PDPTW. JSSP, IMO6, and SHUC have mixed ordering across the 7 checkpoints. The row averages remain positive for all seven benchmarks in both settings: with cuts disabled they range from 3.4% (MCND) to 81.2% (CWLP), and with default cuts they range from 14.9% (MCND) to 62.1% (JSSP). This ablation tests cut-policy interaction: SAGE acceleration inequality families are added before Gurobi is called, while the cut parameter changes solver-generated cuts inside Gurobi.

E.8 Illustrative TSP Instance Trajectories

Fig. 8 plots the full gap traces over time for four instances from the Traveling Salesman Problem Library (TSPLIB). Each panel compares the baseline MILP (blue line) with the same model augmented by the selected SAGE acceleration inequality family found during evolution (orange line). In the four panels, the augmented gap curve lies below the baseline curve for most of the plotted horizon. The acceleration inequality family used in Fig. 8 for the TSP model is described in Appendix F.1.

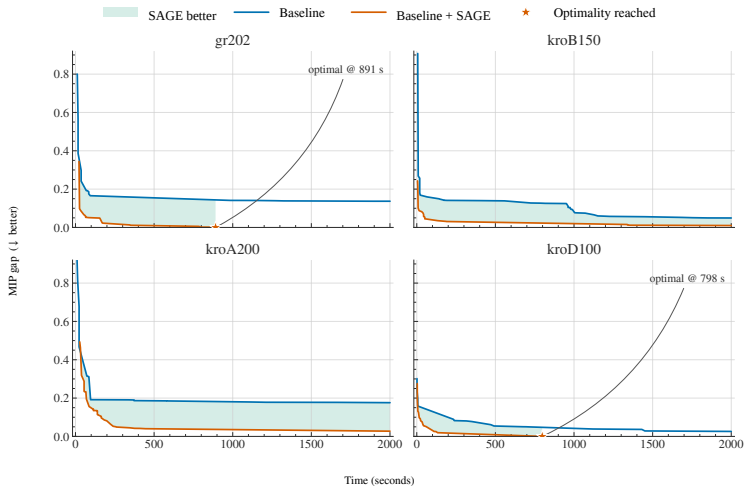

Figure 8: Gap trajectories over time for four representative TSP instances (`gr202`, `kroB150`, `kroA200`, and `kroD100`) from Reinelt (1991). Orange: baseline MILP augmented with the selected SAGE acceleration inequality family. Blue: baseline MILP.

E.9 Best Bound Trajectories for Representative TSP Instances

Fig. 9 complements the gap results of Section 5.2 by plotting the solver's reported best bound over time for four representative TSP test instances drawn from D_t. For each instance, we show two curves: the baseline MILP (reference) and the MILP augmented with the selected SAGE acceleration inequality family. Solver logs are sampled when the solver finds an MILP incumbent, up to the same 2000 s wall-clock limit. Because TSP is formulated as a minimization problem, higher curves correspond to tighter lower bounds.

E.10 Primal-Dual Integral (PDI) Trajectories for Representative TSP Instances

Fig. 8 reports gap trajectories, and Fig. 9 reports solver best bounds. Fig. 10 complements both views by plotting the PDI, a cumulative metric that rewards closing the gap early and penalizes every second the gap remains open.[1] As before, the blue curve shows the baseline MILP. The orange curve adds the selected SAGE acceleration inequality family. Across all four cases, the model augmented by SAGE reduces the

[1]PDI is the time integral of the optimality gap, hence lower curves are better and an earlier flat curve indicates faster convergence.

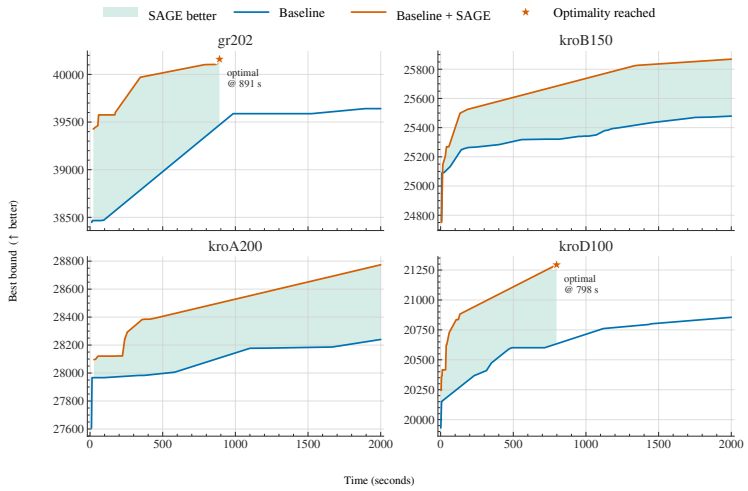

Figure 9: Evolution of the solver's best bound for four representative TSP instances. Blue lines: baseline MILP (reference). Orange lines: MILP with the selected SAGE acceleration inequality family.

total PDI by roughly 75−95%, showing that acceleration inequality families can improve search progress relative to the baseline.

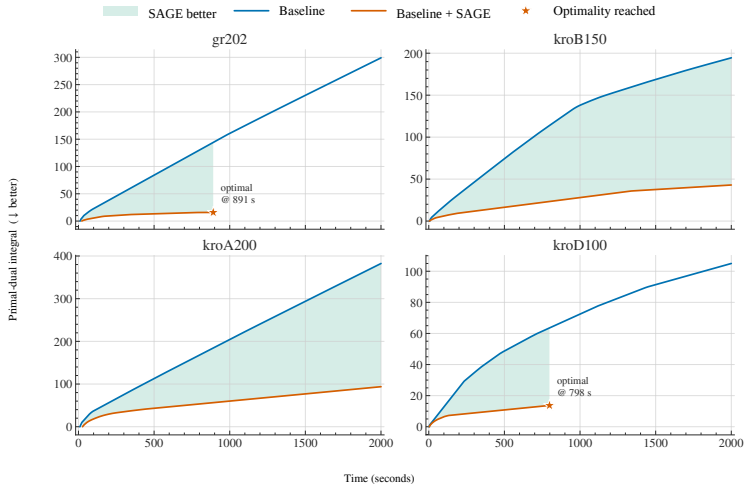

Figure 10: PDI trajectories for the same four TSPLIB instances used in Figs. 8-9. Lower curves are better. Orange: baseline + selected SAGE acceleration inequality family. Blue: baseline MILP.

E.11 Failure Analysis: MCND Cases with Slight Gap Increase

In a small number of MCND instances, the selected acceleration inequality family does not help. Fig. 11 shows four such cases. The gap curves have similar shapes, but the model with the selected acceleration inequality family stays slightly higher for most of the horizon. This indicates that the larger model can slightly slow gap reduction on some instances.

F Benchmark MILP Problems and Selected SAGE Families

Table 12 reports the scale of the held-out benchmark sets. For each benchmark, we show the number of test instances and the mean and standard deviation of the variable and constraint counts after model

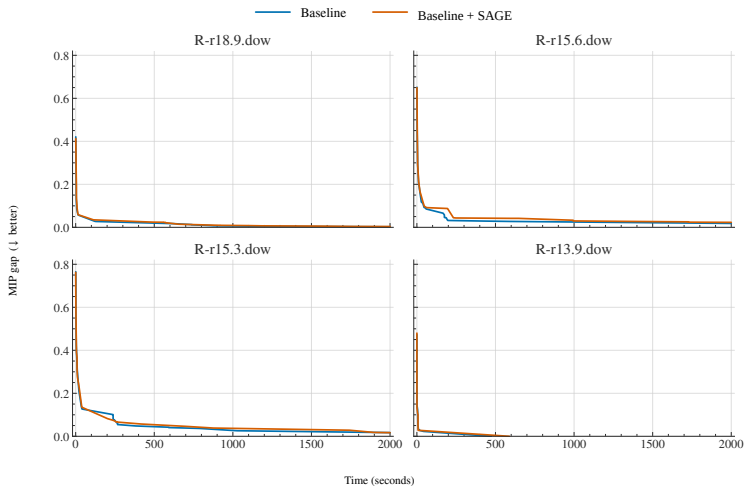

Figure 11: Gap trajectories for four MCND instances where the selected acceleration inequality family slightly hurts the gap (`R-r18.9.dow`, `R-r15.6.dow`, `R-r15.3.dow`, and `R-r13.9.dow`). Orange: baseline MILP augmented with the selected SAGE acceleration inequality family. Blue: baseline MILP.

construction. The selected acceleration inequality families below show the symbolic output of SAGE: on mature benchmarks they sometimes recover expert strengthening structure, while on less standard models they can produce more specialized families tied to the formulation.

Benchmark	# Instances	Variables	Constraints
TSP	54	15821.65 ± 15593.23	15931.00 ± 15655.59
MCND	81	24991.89 ± 17962.83	2485.22 ± 1330.64
CWLP	36	34398.00 ± 47121.26	420.22 ± 487.64
JSSP	80	21382.25 ± 31271.97	42046.88 ± 62013.04
PDPTW	165	11090.00 ± 0.00	4565.00 ± 0.00
IMO6	34	20759.50 ± 22078.88	7524.50 ± 7842.23
SHUC	56	276709.71 ± 147182.35	369301.00 ± 194012.45

Table 12: Held-out benchmark scale. Variable and constraint counts are reported as mean ± standard deviation over the test instances.

Table 13 reports the corresponding per-instance scale for the synthetic sources used to form D_{imp} and D_{adm}. For a given problem, D_{imp} and D_{adm} are drawn from the same generator distribution, so their per-instance scale distribution is the same. The default difference is only cardinality.

Synthetic source for D_{imp} / D_{adm}	Variables	Constraints
TSP synthetic	16737.80 ± 15413.50	16852.61 ± 15473.23
MCND synthetic	24988.19 ± 17868.66	2485.22 ± 1322.35
CWLP synthetic	19690.59 ± 31363.09	420.22 ± 449.19
JSSP synthetic	21382.25 ± 31075.91	42046.88 ± 61624.24
PDPTW synthetic	8645.00 ± 1038.60	3627.50 ± 548.92
IMO6 synthetic	41740.75 ± 44521.34	14863.25 ± 15606.55
SHUC synthetic	48944.00 ± 30308.70	65436.33 ± 40031.25

Table 13: Synthetic development source scale for D_{imp} and D_{adm}. Variable and constraint counts are reported as mean ± standard deviation over generated instances.

F.1 Traveling Salesman Problem

Compact MILP formulation for TSP.

$$\begin{cases} \min \displaystyle\sum_{(i,j)\in A} c_{ij}\, x_{ij} & \\ \text{s.t.} \displaystyle\sum_{j\in V\setminus\{i\}} x_{ij} = 1 & \forall i \in V \\ \displaystyle\sum_{i\in V\setminus\{j\}} x_{ij} = 1 & \forall j \in V \\ u_i - u_j + n\, x_{ij} \le n-1 & \forall i,j \in V\setminus\{1\},\ i \ne j \\ x_{ij} \in \{0,1\} & \forall (i,j) \in A \\ 1 \le u_i \le n & \forall i \in V, \quad u_1 = 1 \end{cases} \tag{MTZ}$$

Notation glossary.

- V (set): cities to be visited (index i, j).
- $A \subseteq V \times V$ (set): directed arcs (i, j) that can be used.
- c_{ij} (parameter): travel cost from city i to city j.
- $n = |V|$ (parameter): number of cities.
- $x_{ij} \in \{0, 1\}$ (variable): equals 1 iff the tour goes directly from city i to city j.
- $u_i \in [1, n]$ (variable): MTZ position of city i in the tour. u_1 is fixed to 1 to anchor the numbering.

The acceleration inequality family in Eq. (SAGE-TSP) combines short-cycle elimination, a lifted Desrochers–Laporte variant of MTZ ordering, and depot-anchored envelope bounds on the MTZ position variables. This acceleration inequality family had the highest inequality impact score selected by SAGE under our gap-reduction metric.

Selected SAGE **acceleration inequality family.**

$$\begin{cases} x_{ij} + x_{ji} \le 1 & \forall i,j \in V,\ i<j \\ x_{1i} + x_{i1} + x_{1j} + x_{j1} + x_{ij} + x_{ji} \le 2 & \forall i,j \in V\setminus\{1\},\ i<j \\ u_i - u_j + (n-1)\, x_{ij} + (n-3)\, x_{ji} \le n-2 & \forall i,j \in V\setminus\{1\},\ i \ne j \\ u_i \ge 3 - x_{1i} + (n-3)\, x_{i1} & \forall i \in V\setminus\{1\} \\ u_i \le (n-1) + x_{i1} - (n-3)\, x_{1i} & \forall i \in V\setminus\{1\} \end{cases} \tag{SAGE-TSP}$$

Benchmark sizes. We considered 54 TSPLIB instances with $n < 250$ cities.

F.2 Multi-Commodity Network Design (MCND)

The MCND problem involves selecting a set of network links and assigning multiple flow demands (commodities) at minimal cost. Each commodity must be routed from its source to destination without exceeding link capacities, and activating a link incurs a fixed cost. The MCND problem is an NP-hard combinatorial problem Gendron et al. (1999) that can be formulated as an MILP. We used a public set of MCND instances (group `R` from the CommaLAB dataset) CommaLAB (2021), which provides a range of network sizes, cost structures, and capacity settings. We used the MILP formulation in Eq. (MCND).

Compact MILP formulation for MCND.

$$
\begin{cases}
\min \displaystyle\sum_{(i,j)\in A}\sum_{k\in K} c_{ij}\, x_{ijk} + \sum_{(i,j)\in A} f_{ij}\, y_{ij} & \\
\text{s.t.} \displaystyle\sum_{j:(i,j)\in A} x_{ijk} = d_k & \forall k \in K,\ i = O_k \\
\displaystyle\sum_{j:(j,i)\in A} x_{jik} = d_k & \forall k \in K,\ i = D_k \\
\displaystyle\sum_{j:(i,j)\in A} x_{ijk} - \sum_{j:(j,i)\in A} x_{jik} = 0 & \forall k \in K,\ i \in N \setminus \{O_k, D_k\} \\
\displaystyle\sum_{k\in K} x_{ijk} \le u_{ij}\, y_{ij} & \forall (i,j) \in A \\
x_{ijk} \ge 0 & \forall (i,j) \in A,\ k \in K \\
y_{ij} \in \{0,1\} & \forall (i,j) \in A
\end{cases}
\tag{MCND}
$$

Notation glossary.

- N (set): all network nodes (index i).
- $A \subseteq N \times N$ (set): candidate directed arcs (i, j).
- K (set): commodities to be routed (index k).
- O_k / D_k (nodes): origin and destination of commodity k.
- d_k (parameter): demand (quantity) of commodity k that must be shipped from O_k to D_k.
- c_{ij} (parameter): unit transportation cost on arc (i, j).
- f_{ij} (parameter): fixed cost to activate arc (i, j).
- u_{ij} (parameter): capacity of arc (i, j).
- $x_{ijk} \ge 0$ (variable): flow of commodity k on arc (i, j).
- $y_{ij} \in \{0, 1\}$ (variable): equals 1 iff arc (i, j) is activated.

Selected SAGE **acceleration inequality family.** Let $\mathcal{I}_{\text{MCND}}$ be the finite set of generated pairs (S, B) used in the static augmentation. Each pair has $S \subseteq N$ and $B \subseteq K(S)$. In our implementation, S ranges over concentric breadth-first search zones expanded from commodity origins. The bundles B are sampled small bundles in $K(S)$.

$$
\begin{aligned}
\sum_{(i,j)\in\delta^+(S)} \min\{u_{ij},\, D_B\}\, y_{ij} \;&\ge\; D_B \qquad \forall (S,B) \in \mathcal{I}_{\text{MCND}} \\
\sum_{(i,j)\in\delta^+(S)} y_{ij} \;&\ge\; q_{S,B} \qquad \forall (S,B) \in \mathcal{I}_{\text{MCND}}
\end{aligned}
\tag{SAGE-MCND}
$$

Here, for each generated pair $(S, B) \in \mathcal{I}_{\text{MCND}}$, $\delta^+(S) := \{(i, j) \in A : i \in S,\ j \notin S\}$ denotes the arcs leaving S, and $K(S) := \{k \in K : O_k \in S,\ D_k \notin S\}$ are the commodities that must cross this cut. For a generated bundle B, let $D_B := \sum_{k\in B} d_k$. The first row in Eq. (SAGE-MCND) is a knapsack-cover variant of the standard graph cut-set capacity inequality. The truncation $\min\{u_{ij}, D_B\}$ prevents the LP relaxation from satisfying the capacity constraint using tiny fractional y_{ij} on very high-capacity arcs. The second row is a cardinality companion, where $q_{S,B}$ is the smallest integer such that the sum of the $q_{S,B}$ largest capacities in $\delta^+(S)$ reaches D_B. These rows have the same form as the full cut-set family over all $S \subseteq N$ and $B \subseteq K(S)$. The MILP receives only the finite generated subset $\mathcal{I}_{\text{MCND}}$.

Benchmark sizes. We used the group R instances from the CommaLAB dataset (81 instances), which cover diverse network topologies and demand settings. Each instance is identified as r$x.y$, where x determines the network size and number of commodities, and y encodes the fixed cost and capacity regime. Table 14 summarizes the key characteristics of the x configurations used in our benchmark dataset.

x	# Nodes	# Arcs	# Commodities
01-09	10	35-83	10-50
10-12	20	120	40-200
13-15	20	220	40-200
16-18	20	314-318	40-200

Table 14: CommaLAB R MCND instance sizes by x in r$x.y$.

The second index y in r$x.y$ controls the fixed cost and capacity setting. Higher fixed costs increase the relative importance of arc selection costs, while higher capacities relax arc flow constraints.

F.3 Capacitated Warehouse Location Problem (CWLP)

The CWLP is an NP-hard combinatorial optimization problem, formulated as an MILP Beasley (1988). It involves selecting a subset of candidate warehouse locations to open and assigning each customer to one open warehouse. The problem has capacity constraints at each facility, with the goal of minimizing total fixed opening and transportation costs. We used a public set Beasley (1990) of CWLP instances. For our experiments, we use the formulation in Eq. (CWLP).

Compact MILP formulation.

$$\begin{cases} \min \sum_{j \in J} f_j \, y_j \; + \sum_{i \in I} \sum_{j \in J} c_{ij} \, x_{ij} & \\ \text{s.t.} \sum_{j \in J} x_{ij} \; = \; 1 & \forall i \in I \quad \text{(each customer is assigned)} \\ \sum_{i \in I} d_i \, x_{ij} \; \leq \; u_j \, y_j & \forall j \in J \quad \text{(capacity)} \\ x_{ij} \in \{0,1\} & \forall i \in I, \, j \in J \\ y_j \in \{0,1\} & \forall j \in J \end{cases} \tag{CWLP}$$

Glossary of notation.

- I (index i): set of customers.
- J (index j): set of candidate warehouses.
- d_i: demand of customer i.
- u_j: capacity of warehouse j.
- f_j: fixed cost to open warehouse j.
- c_{ij}: total transportation cost incurred if all of customer i's demand is served by warehouse j (so c_{ij} already accounts for d_i).
- $y_j \in \{0,1\}$: decision variable that equals 1 iff warehouse j is opened (0 otherwise).
- $x_{ij} \in \{0,1\}$: decision variable that equals 1 iff customer i is assigned to warehouse j (0 otherwise).

Selected SAGE **acceleration inequality family.** The selected CWLP acceleration inequality family found by SAGE is a hybrid of global bin-packing bounds and local lifted capacity covers. It combines (i) global slot count lower bounds (at multiple demand thresholds) and (ii) per warehouse clique/cover inequalities that limit how many large customers can be assigned to a single facility, together with strong links between assignment and opening.

$$\begin{cases} \sum_{j \in J} u_j \, y_j \geq \sum_{i \in I} d_i & \\ \sum_{j \in J} \left\lfloor \frac{u_j}{v} \right\rfloor y_j \geq \left| \{ i \in I : d_i \geq v \} \right| & \forall v \in \mathcal{V} \\ x_{ij} \leq y_j & \forall i \in I, \ \forall j \in J \\ x_{ij} \leq 0 & \forall i \in I, \ \forall j \in J \text{ with } d_i > u_j \\ \sum_{i \in C_j} x_{ij} \leq y_j & \forall j \in J \\ \sum_{i \in I : d_i > u_j/3} x_{ij} \leq 2 \, y_j & \forall j \in J \end{cases} \tag{SAGE-CWLP}$$

Here, $\mathcal{V} \subseteq \{d_i : i \in I\}$ is a small set of demand thresholds (we use up to the 8 largest distinct demand values in each instance) to instantiate the global slot count bounds. For the local clique inequality, $C_j \subseteq I$ is a lifted conflict set constructed by starting from $\{i \in I : d_i > u_j/2\}$ and sequentially adding smaller customers that still conflict with every customer already in the set (so $d_i + d_{i'} > u_j$ for all distinct $i, i' \in C_j$).

Benchmark sizes. We used 36 CWLP instances from the OR-Library benchmark (100 facility locations and 1000 customers).

F.4 Job Shop Scheduling Problem (JSSP)

The JSSP is a classic NP-hard combinatorial optimization problem Garey et al. (1976) and admits a MILP formulation Manne (1960). It involves scheduling a set of jobs on multiple machines, where each job has a sequence of operations that must be processed in a specified order on specified machines. The objective is to minimize the makespan (the completion time of the last operation), while ensuring that each machine handles at most one operation at a time. We use the formulation in Eq. (JSSP) and test SAGE on the widely used Taillard benchmark Taillard (1993).

Compact MILP formulation.

$$\begin{cases} \min C_{\max} & \\ \text{s.t.} \quad S_{j,k+1} \geq S_{j,k} + p_{j,k} & \forall j, \ k = 1, \ldots, n_{\text{mach}} - 1 \\ \quad S_{j_1,k_1} + p_{j_1,k_1} \leq & \\ \qquad S_{j_2,k_2} + M(1 - y_{j_1,k_1,j_2,k_2}) & \forall \left((j_1,k_1), (j_2,k_2) \right) \in \mathcal{E} \\ \quad S_{j_2,k_2} + p_{j_2,k_2} \leq & \\ \qquad S_{j_1,k_1} + M \, y_{j_1,k_1,j_2,k_2} & \forall \left((j_1,k_1), (j_2,k_2) \right) \in \mathcal{E} \\ \quad C_{\max} \geq S_{j,n_{\text{mach}}} + p_{j,n_{\text{mach}}} & \forall j \\ \quad S_{j,k} \geq 0 & \forall j, k \\ \quad y_{j_1,k_1,j_2,k_2} \in \{0,1\} & \forall \left((j_1,k_1), (j_2,k_2) \right) \in \mathcal{E} \end{cases} \tag{JSSP}$$

Variable and set glossary.

- n_{mach} (parameter): number of machines (and operations per job), with $k \in \{1, \ldots, n_{\text{mach}}\}$.
- $m \in \{1, \ldots, n_{\text{mach}}\}$ (index): machine index.

- O (set): set of all operations (j, k).
- O_m (set): set of operations processed on machine m.
- $S_{j,k} \in \mathbb{R}_{\geq 0}$ (variable): start time of the k-th operation of job j.
- $p_{j,k}$ (parameter): processing time of operation (j, k).
- $\mathcal{E}$ (set): ordered pairs $\big((j_1, k_1), (j_2, k_2)\big)$ of distinct operations that require the same machine. For every such pair, exactly one of the two order enforcing inequalities becomes active.
- $y_{j_1,k_1,j_2,k_2} \in \{0, 1\}$: binary variable that equals 1 if operation (j_1, k_1) is scheduled before (j_2, k_2) on their shared machine, 0 otherwise.
- $C_{\max} \in \mathbb{R}_{\geq 0}$: continuous makespan (completion time of the last operation). The objective minimizes this value.
- M: a sufficiently large constant ("big-M") that deactivates the unselected sequencing inequality.

Selected SAGE **acceleration inequality family.**

$$
\begin{aligned}
&C_{\max} \geq \frac{1}{n_{\text{mach}}} \sum_{(j,k)\in O} p_{j,k} && \\
&C_{\max} \geq \texttt{CP}_m && \forall m \in \{1, \ldots, n_{\text{mach}}\} \\
&C_{\max} \geq \sum_{k=1}^{n_{\text{mach}}} p_{j,k} && \forall j
\end{aligned}
\tag{SAGE-JSSP}
$$

This hybrid family combines three lower bounds on the makespan:

- **Average load bound**: $\frac{1}{n_{\text{mach}}} \sum_{(j,k)\in O} p_{j,k}$ represents the average total processing time per machine, so the makespan is at least the average workload.
- **Machine-level critical path bound** ($\texttt{CP}_m$): For each machine m, let O_m be the set of operations processed on m. For any operation (j, k), define the job prefix (head) and job suffix (tail)

  $$h_{j,k} := \sum_{t<k} p_{j,t}, \qquad t_{j,k} := \sum_{t>k} p_{j,t}.$$

  Then we take

  $$\texttt{CP}_m \;:=\; \sum_{(j,k)\in O_m} p_{j,k} \;+\; \min_{(j,k)\in O_m} h_{j,k} \;+\; \min_{(j,k)\in O_m} t_{j,k},$$

  which lower bounds the completion time of any schedule by accounting for the total load on m plus the earliest possible release (head) of the first operation on m and the smallest remaining tail after the last operation on m.
- **Longest job bound**: $\max_j \sum_k p_{j,k}$ lower bounds the makespan by the total processing time of the longest job chain.

These three bounds together use machine workloads and job chain precedence (heads/tails). The hybrid acceleration inequality family was selected by SAGE based on optimality gap reduction.

Benchmark sizes. We evaluated SAGE on the Taillard JSSP benchmark (`ta01` to `ta80`, 80 instances). This includes instances with the following job and machine counts:

- 15 jobs $\times$ 15 machines (e.g., ta01-ta10),
- 20 jobs $\times$ 15 machines (e.g., ta11-ta20),
- 20 jobs $\times$ 20 machines (e.g., ta21-ta30),

- 30 jobs × 15 machines (e.g., ta31-ta40),
- 30 jobs × 20 machines (e.g., ta41-ta50),
- 50 jobs × 15 machines (e.g., ta51-ta60),
- 50 jobs × 20 machines (e.g., ta61-ta70),
- 100 jobs × 20 machines (e.g., ta71-ta80).

These instances are widely used in the literature to assess scheduling algorithms on problems with different sizes and difficulty levels.

F.5 Pickup and Delivery Problem with Time Windows (PDPTW)

The PDPTW is a vehicle routing problem in which each request consists of a pickup location and a delivery location that must be served in precedence order, subject to time window and vehicle capacity constraints. The objective is to route vehicles to serve all requests while minimizing total travel cost. PDPTW is NP-hard and admits MILP formulations. It comes from practice and has many constraints, and some benchmark instances used in our suite are included in MIPLIB Koch et al. (2011). We use the truckload PDPTW instances derived from a real-world drayage case at the Port of Rotterdam Srour et al. (2010). We obtained the compressed MPS files from the MIPLIB 2010 contribution page Srour et al. (2010). We convert them into our parameterized model by reading each MPS file in Gurobi and recovering $(K, N, d^k_{0i}, d_{ij}, d^k_{iH}, v^k, \tau^-_i, \tau^+_i)$ from the instance coefficients.

Compact MILP formulation. We use the MILP formulation of Srour et al. (2010), which models trucks and jobs as nodes in a complete directed graph and permits explicit job rejection via self-loops $x_{K+i,K+i} = 1$ with penalty d_{ii}:

$$\begin{cases} \min \displaystyle\sum_{k=1}^{K}\sum_{i=1}^{N} d^k_{0i}\, x_{k,K+i} + \sum_{i=1}^{N}\sum_{j=1}^{N} d_{ij}\, x_{K+i,K+j} + \sum_{i=1}^{N}\sum_{k=1}^{K} d^k_{iH}\, x_{K+i,k} & \\ \text{s.t.} \displaystyle\sum_{v=1}^{K+N} x_{uv} = 1 & \forall u = 1,\dots,K+N \\ \displaystyle\sum_{v=1}^{K+N} x_{vu} = 1 & \forall u = 1,\dots,K+N \\ \delta_i - \displaystyle\sum_{k=1}^{K} (d^k_{0i} + v^k)\, x_{k,K+i} \geq 0 & \forall i = 1,\dots,N \\ \delta_j - \delta_i - M\, x_{K+i,K+j} + (d_{ii} + d_{ij})\, x_{K+i,K+i} \geq d_{ii} + d_{ij} - M & \forall i,j = 1,\dots,N \\ \tau^-_i \leq \delta_i \leq \tau^+_i & \forall i = 1,\dots,N \\ \delta_i \in \mathbb{R}_{\geq 0} & \forall i = 1,\dots,N \\ x_{uv} \in \{0,1\} & \forall u,v = 1,\dots,K+N \end{cases} \tag{PDPTW}$$

Notation glossary.

- K (parameter): number of trucks/vehicles in the fleet (truck nodes $k \in \{1,\dots,K\}$).
- N (parameter): number of pickup and delivery demands/jobs (job nodes $K+i$ for $i \in \{1,\dots,N\}$).
- $u, v \in \{1,\dots,K+N\}$ (indices): nodes in the complete directed graph.
- d_{ij} (parameter): travel time from demand i's return terminal to demand j's pickup terminal. For $i = j$, d_{ii} is the loaded time of job i and is used as the rejection penalty.
- d^k_{0i} (parameter): travel time from the home terminal of truck k to demand i's pickup terminal.

- d_{iH}^k (parameter): travel time from demand i's return terminal to the home terminal of truck k.
- v^k (parameter): time at which truck k becomes available.
- τ_i^-, τ_i^+ (parameters): earliest/latest permissible arrival times at demand i's pickup terminal.
- M (parameter): big-M constant. For the MIPLIB instances we use, $M = 2\max_{i,j} d_{ij}$ Srour et al. (2010).
- $x_{uv} \in \{0, 1\}$ (variable): equals 1 iff arc (u, v) is selected in the routing cycle cover.
- $\delta_i \in \mathbb{R}_{\geq 0}$ (variable): arrival time at demand i's pickup terminal.

Selected SAGE **acceleration inequality family.** The selected PDPTW acceleration inequality family adds four time-window inequalities (implemented as `additional_cut` in our Pyomo code). Let

$$\mathrm{EST}_i := \max\{\tau_i^-, \min_k (v^k + d_{0i}^k)\}$$

and

$$\mathcal{A}^- := \{(i, j) : \mathrm{EST}_i + d_{ii} + d_{ij} > \tau_j^+\}.$$

We then add the following four inequalities:

$$\begin{cases} x_{K+i,K+j} \leq 0 & \forall (i,j) \in \mathcal{A}^- \\ x_{K+i,K+j} + x_{K+j,K+i} + x_{K+i,K+i} \leq 1 & \forall (i,j) \in \mathcal{F}_2 \\ \displaystyle\sum_{i \in C} x_{K+i,K+i} \geq |C| - |K(C)| & \forall C \in \mathcal{C} \\ x_{K+i,K+k} + x_{K+k,K+j} + x_{K+k,K+k} \leq 1 & \forall (i,k,j) \in \mathcal{Q} \end{cases} \tag{SAGE-PDPTW}$$

Here, $\mathcal{F}_2$ is the set of mutually feasible pairs:

$$\mathcal{F}_2 := \{(i, j) : i \neq j,\ (i, j) \notin \mathcal{A}^-,\ (j, i) \notin \mathcal{A}^-\}.$$

$\mathcal{C}$ is the set of cliques with $|C| > |K(C)|$ in the conflict graph that connects pairs with both directions infeasible, where

$$K(C) := \{k \in \{1, \ldots, K\} : \exists i \in C \text{ s.t. } v^k + d_{0i}^k \leq \tau_i^+\}$$

is the set of trucks that can reach at least one job in C by its latest pickup time. Finally,

$$\mathcal{Q} := \{(i, k, j) : (i, k) \notin \mathcal{A}^-,\ (k, j) \notin \mathcal{A}^-,\ \max\{\mathrm{EST}_k, \mathrm{EST}_i + d_{ii} + d_{ik}\} + d_{kk} + d_{kj} > \tau_j^+\}.$$

Benchmark sizes. We use the five MIPLIB truckload PDPTW groups `R0`, `R25`, `R50`, `R75`, and `R100` Srour et al. (2010). Each group contains 33 instances (one per day) for a total of 165 instances. Each instance has $K = 40$ trucks and $N = 65$ jobs. The groups differ by the fraction of jobs that are released after the start of the day (0%, 25%, 50%, 75%, 100%, respectively).

F.6 International Mathematical Olympiad 2025 Problem 6 (IMO6)

This benchmark is based on IMO 2025 Problem 6: given an $N \times N$ grid of unit squares, place axis-aligned rectangular tiles (no overlaps) so that each row and each column contains exactly one uncovered unit square (a hole). The objective is to minimize the number of tiles used International Mathematical Olympiad (2025). We include IMO6 as a newly formulated MILP benchmark (derived from a math olympiad problem), so it is not drawn from standard IP benchmark literature and LLMs are unlikely to have seen established formulation ideas for it. We model the problem by defining fixed-length strips and tracking their vertical continuation over a 2D board. The following IP model matches the Pyomo code used in our experiments.

Compact MILP formulation. Let $R = \{1, \dots, N\}$ be the rows, $C = \{1, \dots, N\}$ the columns, and $I = \{(a, b) \in C \times C : \ a \le b\}$ the set of horizontal strips. Binary variables: h_{ij} indicate the hole position, and x_i^{ab} indicates that strip (a, b) is active on row i. The variables s_i^{ab}, $t_i^{ab} \in \{0, 1\}$ mark the start and end rows of strip (a, b). The objective counts rectangles, because each rectangle contributes exactly one start. The flow equalities ensure each active strip (a, b) has one start and one end, so summation over start variables equals the number of rectangles.

$$
\begin{cases}
\min \displaystyle\sum_{i \in R} \sum_{(a,b) \in I} s_i^{ab} & \\
\text{s.t.} \displaystyle\sum_{j \in C} h_{ij} = 1 & \forall i \in R \\
 & \text{(one hole per row)} \\
\displaystyle\sum_{i \in R} h_{ij} = 1 & \forall j \in C \\
 & \text{(one hole per column)} \\
\displaystyle\sum_{\substack{(a,b) \in I \\ a \le j \le b}} x_i^{ab} + h_{ij} = 1 & \forall i \in R, \ \forall j \in C \\
 & \text{(each cell is covered by exactly one strip or a hole)} \\
x_1^{ab} - s_1^{ab} = 0 & \forall (a, b) \in I \\
 & \text{(top flow)} \\
x_i^{ab} - x_{i-1}^{ab} - s_i^{ab} + t_{i-1}^{ab} = 0 & \forall i = 2, \dots, N, \ \forall (a, b) \in I \\
 & \text{(mid flow)} \\
x_N^{ab} - t_N^{ab} = 0 & \forall (a, b) \in I \\
 & \text{(bottom flow)} \\
h_{ij} \in \{0, 1\} & \forall i \in R, \ \forall j \in C \\
x_i^{ab}, \ s_i^{ab}, \ t_i^{ab} \in \{0, 1\} & \forall i \in R, \ \forall (a, b) \in I
\end{cases}
\tag{RT-2DFlow}
$$

Notation glossary.

- $R = \{1, \dots, N\}$, $C = \{1, \dots, N\}$: row and column index sets ($i \in R$, $j \in C$).
- $I = \{(a, b) \in C^2 : \ a \le b\}$: all contiguous column intervals (index (a, b)).
- $h_{ij} \in \{0, 1\}$: $= 1$ iff (i, j) is the unique hole in row i and in column j.
- $x_i^{ab} \in \{0, 1\}$: $= 1$ iff on row i the columns $a, a{+}1, \dots, b$ are covered by the same tile and strip (a, b) is active.
- $s_i^{ab}, t_i^{ab} \in \{0, 1\}$: start/end flags of the vertical strip for interval (a, b) at row i. The number of rectangles equals $\sum_{i,(a,b)} s_i^{ab}$.

Selected SAGE **acceleration inequality family.** The selected IMO6 acceleration inequality family uses both uniqueness conditions on holes. It forbids an active strip from containing the row hole, tightens row activity to the number of connected covered components in the row, and enforces mandatory new starts when a hole in the previous row is vacated. The last two inequalities also restrict the new start to the same side of

the new hole as the covered column.

$$
\begin{cases}
x_i^{ab} + \sum_{\ell=a}^{b} h_{i\ell} \le 1 & \forall i \in R,\ \forall (a,b) \in I \\
\sum_{(a,b)\in I} x_i^{ab} \ge 2 - h_{i1} - h_{iN} & \forall i \in R \\
\sum_{(a,b)\in I} s_i^{ab} \ge 1 & \forall i \in \{2,\dots,N\} \\
\sum_{\substack{(a,b)\in I \\ a\le j\le b}} s_i^{ab} \ge h_{i-1,j} & \forall i \in \{2,\dots,N\},\ \forall j \in C \\
\sum_{\substack{(a,b)\in I \\ a\le j\le b<k}} s_i^{ab} \ge \sum_{\substack{(a,b)\in I \\ a\le j \\ b\ge k}} x_{i-1}^{ab} + h_{ik} - 1 & \forall i \in \{2,\dots,N\},\ \forall j,k \in C,\ j<k \\
\sum_{\substack{(a,b)\in I \\ k<a\le j\le b}} s_i^{ab} \ge \sum_{\substack{(a,b)\in I \\ a\le k \\ b\ge j}} x_{i-1}^{ab} + h_{ik} - 1 & \forall i \in \{2,\dots,N\},\ \forall j,k \in C,\ j>k
\end{cases}
\tag{SAGE-RT-Disrupt}
$$

The first inequality prevents any active strip from crossing the row's unique hole. The second inequality counts row components: if the hole is in an interior column, the covered cells split into two disconnected intervals. Otherwise they form one interval. The third inequality follows from column uniqueness: the column that was a hole in row $i-1$ must be covered in row i, so at least one rectangle starts in every row after the first. The fourth tightens the vacated-column inequality to require a start covering the exact column vacated by the previous row's hole. The final two inequalities handle intervals broken by the new hole: if the new hole at k breaks an interval from row $i-1$, the new start covering j cannot cross k. Together with the top-flow equality, these inequalities give the LP relaxation the objective lower bound $\sum_{i,(a,b)} s_i^{ab} \ge N$.

Benchmark sizes. We generate one instance per $N \in \{3,\dots,46\}$ (44 instances total). We randomly sample 10 instances as the development pool for SAGE. In this pool, D_{imp} contains all 10 instances and D_{adm} is a 2-instance subset used for admissibility screening. The remaining 34 instances form the held-out test set.

F.7 Sub-Hour Unit Commitment (SHUC)

The SHUC benchmark is a unit commitment problem with a horizon under one hour that decides generator commitment (on/off) and dispatch levels to meet demand and reserve. SHUC is a problem motivated by practice with many constraints. Some benchmark instances used in our suite are included in MIPLIB Koch et al. (2011). We use the PGLib-UC v19.08 JavaScript Object Notation (JSON) benchmark Power Grid Lib (2019) and follow its reference formulation with piecewise production costs, startup categories, ramping limits, and minimum up and down times Knueven et al. (2020). For source attribution, we use the recommended documents for the `ca` cases (Knueven et al., 2020), the `ferc` cases (Knueven et al., 2020; Krall et al., 2012), and the `rts_gmlc` cases (Barrows et al., 2019).

Compact MILP formulation.

$$
\begin{cases}
\min \displaystyle\sum_{g\in G}\sum_{t\in\mathcal{T}}\Big(c_{g,t}+C_{g,1}\,u_{g,t}\Big)+\sum_{g\in G}\sum_{s\in S_g}\sum_{t\in\mathcal{T}} C^{su}_{g,s}\,d_{g,s,t} & \\
\text{s.t.} \displaystyle\sum_{g\in G}\Big(p_{g,t}+P^{\min}_g\,u_{g,t}\Big)+\sum_{w\in W} p^{w}_{w,t}=L_t & \forall t\in\mathcal{T}\\
\displaystyle\sum_{g\in G} r_{g,t}\ge R_t & \forall t\in\mathcal{T}\\
u_{g,t}-u_{g,t-1}=v_{g,t}-w_{g,t} & \forall g\in G,\ t=2,\dots,T\\
\displaystyle\sum_{\tau=t-U_g+1}^{t} v_{g,\tau}\le u_{g,t} & \forall g\in G,\ t\ge U_g\\
\displaystyle\sum_{\tau=t-D_g+1}^{t} w_{g,\tau}\le 1-u_{g,t} & \forall g\in G,\ t\ge D_g\\
v_{g,t}=\displaystyle\sum_{s\in S_g} d_{g,s,t} & \forall g\in G,\ t\in\mathcal{T}\\
d_{g,s,t}\le \displaystyle\sum_{i=\ell_{g,s}}^{\ell_{g,s+1}-1} w_{g,t-i} & \forall g\in G,\ s<|S_g|,\ t\ge \ell_{g,s+1}\\
u_{g,t}\ge \mathrm{MR}_g & \forall g\in G,\ t\in\mathcal{T}\\
p_{g,t}+r_{g,t}\le (P^{\max}_g-P^{\min}_g)\,u_{g,t}-\max(P^{\max}_g-SU_g,0)\,v_{g,t} & \forall g\in G,\ t\in\mathcal{T}\\
p_{g,t}+r_{g,t}\le (P^{\max}_g-P^{\min}_g)\,u_{g,t}-\max(P^{\max}_g-SD_g,0)\,w_{g,t+1} & \forall g\in G,\ t<T\\
p_{g,t}+r_{g,t}-p_{g,t-1}\le RU_g & \forall g\in G,\ t=2,\dots,T\\
p_{g,t-1}-p_{g,t}\le RD_g & \forall g\in G,\ t=2,\dots,T\\
p_{g,t}=\displaystyle\sum_{l\in L_g}(P_{g,l}-P_{g,1})\,\lambda_{g,l,t} & \forall g\in G,\ t\in\mathcal{T}\\
c_{g,t}=\displaystyle\sum_{l\in L_g}(C_{g,l}-C_{g,1})\,\lambda_{g,l,t} & \forall g\in G,\ t\in\mathcal{T}\\
u_{g,t}=\displaystyle\sum_{l\in L_g}\lambda_{g,l,t} & \forall g\in G,\ t\in\mathcal{T}\\
P^{w,\min}_{w,t}\le p^{w}_{w,t}\le P^{w,\max}_{w,t} & \forall w\in W,\ t\in\mathcal{T}\\
u_{g,t},v_{g,t},w_{g,t},d_{g,s,t}\in\{0,1\},\ \lambda_{g,l,t}\in[0,1] & \\
p_{g,t},r_{g,t},c_{g,t},p^{w}_{w,t}\ge 0 &
\end{cases}
\tag{SHUC}
$$

We also enforce initial on status, initial output, and remaining up or down time at $t=1$ using the dataset fields `unit_on_t0`, `power_output_t0`, `time_up_t0`, and `time_down_t0`, matching the PGLib-UC reference model.

Notation glossary.

- $\mathcal{T}=\{1,\dots,T\}$: time periods (index t).
- G: thermal generators, W: renewable generators.
- S_g: startup categories for generator g, with lag $\ell_{g,s}$ and cost $C^{su}_{g,s}$.
- L_g: piecewise production points $(P_{g,l},C_{g,l})$, with $C_{g,1}$ the fixed on cost per period.

- L_t, R_t: demand and reserve requirements at time t.
- $P_g^{\min}$, $P_g^{\max}$: minimum and maximum thermal output, and $P_{w,t}^{w,\min}$, $P_{w,t}^{w,\max}$ are renewable bounds.
- RU_g, RD_g: ramp up and ramp down limits, and SU_g, SD_g: startup and shutdown ramp limits.
- U_g, D_g: minimum up and minimum down times, and MR_g: must run flag.
- $u_{g,t}$: on status, $v_{g,t}$: startup, and $w_{g,t}$: shutdown.
- $p_{g,t}$: thermal output above minimum, $r_{g,t}$: spinning reserve, and $p_{w,t}^{w}$: renewable output.
- $d_{g,s,t}$: startup category selection, $\lambda_{g,l,t}$: piecewise weights, and $c_{g,t}$: variable production cost above the base cost.

Selected SAGE **acceleration inequality family.**

$$
\begin{cases}
b_{g,t} \le v_{g,t} & \forall g \in G,\ t < T \\
b_{g,t} \le w_{g,t+1} & \forall g \in G,\ t < T \\
b_{g,t} \ge v_{g,t} + w_{g,t+1} - 1 & \forall g \in G,\ t < T \\
\overline{P}_{g,t} \le P_g^{\max}\, u_{g,t} - \max(P_g^{\max} - SU_g, 0)\, v_{g,t} & \forall g \in G,\ t \in \mathcal{T} \\
\overline{P}_{g,t} \le P_g^{\max}\, u_{g,t} - \max(P_g^{\max} - SD_g, 0)\, w_{g,t+1} & \forall g \in G,\ t < T \\
\overline{P}_{g,t} \le P_g^{\max}\, u_{g,t} - \max(P_g^{\max} - SU_g, 0)\, v_{g,t} - \max(P_g^{\max} - SD_g, 0)\, w_{g,t+1} & \forall g \in G,\ t < T \\
\qquad + \min\Big(\max(P_g^{\max} - SU_g, 0), \max(P_g^{\max} - SD_g, 0)\Big)\, b_{g,t} & \\
\overline{P}_{g,t} \le P_g^{\min}\, u_{g,t} + p_{g,t-1} + RU_g + (P_g^{\max} - P_g^{\min})\,(1 - u_{g,t-1}) & \forall g \in G,\ t = 2, \ldots, T \\
\overline{P}_{g,t} \le P_g^{\min}\, u_{g,t} + (P_g^{\max} - P_g^{\min})\, u_{g,t-1} - \max(P_g^{\max} - SU_g, 0)\, v_{g,t-1} + RU_g & \forall g \in G,\ t = 2, \ldots, T \\
\overline{P}_{g,t} \le P_g^{\min}\, u_{g,t} + (P_g^{\max} - P_g^{\min})\, u_{g,t-1} - \max(P_g^{\max} - SD_g, 0)\, w_{g,t} + RU_g & \forall g \in G,\ t = 2, \ldots, T \\
L_t - \displaystyle\sum_{w \in W} p_{w,t}^{w} + R_t \le \sum_{g \in G} \overline{P}_{g,t} & \forall t \in \mathcal{T}
\end{cases}
\tag{SAGE-SHUC}
$$

This acceleration inequality family requires that demand minus renewable output plus reserve can be met by thermal capacity that is reachable under ramp limits, with derating during startups and shutdowns. We also apply the $t = 1$ form of the ramp reachability bounds using the dataset $t0$ fields. Here, $\overline{P}_{g,t}$ and $b_{g,t}$ are auxiliary variables introduced by the selected acceleration inequality family.

Benchmark sizes. We use all 56 PGLib-UC v19.08 instances: 20 `ca`, 24 `ferc`, and 12 `rts_gmlc` cases. Each instance has $T = 48$ periods, 73 to 978 thermal generators, and 0 to 81 renewable generators. We set $|D_{imp}| = 10$ and $|D_{adm}| = 2$ from our synthetic generator, and use the full 56-instance PGLib-UC benchmark as D_t.

G LLM API Configuration

G.1 Parameters and token usage

Table 15 lists the exact parameters used in every call to the three LLM APIs (`deepseek-reasoner`, `google/gemini-3-pro-preview`, and `gpt-5.1`) throughout our SAGE pipeline, together with approximate token usage.

H Inequality Impact Score Definition

We assume $\mathrm{gap}_{\mathrm{ref}}(i) > 0$ for all $i \in D_{imp}$ (as ensured by the construction of D_{imp} in Section 4). Given the signed relative gap change

$$
d(i) \;=\; \frac{\mathrm{gap}_{\mathrm{aug}}(i) - \mathrm{gap}_{\mathrm{ref}}(i)}{\mathrm{gap}_{\mathrm{ref}}(i)},
$$

Parameter	DeepSeek	Gemini 3 Pro	GPT-5.1
Model	`deepseek-reasoner`	`google/gemini-3-pro-preview`	`gpt-5.1`
Maximum output tokens	10,000	10,000	10,000
Temperature	1.0	1.0	1.0
Frequency penalty	0.0	0.0	0.0
Presence penalty	0.0	0.0	0.0
Total input tokens	∼26 M	∼19 M	∼28 M
Total output tokens	∼13 M	∼14 M	∼17 M
Total tokens	∼39 M	∼33 M	∼45 M
Total cost (Jan 2026 pricing)	∼$50	∼$200	∼$150

Table 15: LLM API settings and estimated token costs used for all SAGE experiments. Costs use Jan 2026 OpenRouter list prices. The Gemini 3 Pro API identifier and pricing follow OpenRouter OpenRouter (2025).

its mean over D_{imp},

$$\bar{d} \;=\; \frac{1}{|D_{imp}|} \sum_{i \in D_{imp}} d(i),$$

is negative when C is beneficial and positive when it is harmful. We map $\bar{d}$ to an inequality impact score via

$$s_{imp}(C) \;=\; 10\, e^{-\bar{d}},$$

so larger (more negative) gap reductions yield exponentially higher inequality impact scores.