

EvoCut: Strengthening Integer Programs via Evolution Guided Language Models

Milad Yazdani¹ Mahdi Mostajabdaveh² Samin Aref³ Zirui Zhou²

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 acceleration cuts, i.e., inequalities that speed up solving. However, this creative process requires deep expertise and has been difficult to automate. Our proposed framework, **EvoCut**, automates the generation of acceleration cuts at the symbolic modeling level: it reasons over a symbolic MILP model and a natural language description of the problem to discover a reusable set of acceleration cuts that can be used for each concrete instance of the model. **EvoCut** (i) initializes a population of candidate cuts via an initializer agent that uses an LLM, (ii) empirically screens candidates on a small verification set by checking that reference solutions remain feasible and that at least one stored LP relaxation solution is cut off, and (iii) iteratively refines the population through evolutionary crossover and mutation agents. Compared to baseline MILP formulations solved with a fixed time budget, **EvoCut** reduces optimality gaps by up to 76% and reaches target gaps up to $7.2\times$ faster (shifted geometric mean speedup). Ablations show its robustness across different LLM backends and across solver/cut settings. Code: <https://github.com/milad1378yz/EvoCut>.

ing 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., 2024; 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., 2024a; 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 strength of the formulation (Klotz & Oberdieck, 2024).

In this study, we focus on automated formulation tightening for Mixed Integer Linear Programming (MILP). Starting from a base MILP formulation, we generate additional inequalities (acceleration cuts) intended to improve solver efficiency. Many widely studied problem classes admit problem-specific cuts derived in the OR literature. However, we aim for a general pipeline that operates directly on a symbolic MILP model shared across instances. We refer to this shared formulation as the symbolic model and to its specialization to a particular instance (via data instantiation) as the concrete model. Our proposed method, **EvoCut**, uses LLM agents guided by evolutionary search to discover and refine such cuts by reasoning over a symbolic MILP model and a natural language description of the problem. In particular, **EvoCut** learns a reusable set of acceleration cuts (with a symbolic formulation) during development and instantiates them for each concrete instance at deployment time (Fig. 1). **EvoCut** is a general framework that requires no tuning for a specific domain and integrates into standard

1. Introduction

Operations Research (OR) workflows usually consist of two main phases: formulating an optimization model and solving it.

¹Department of Electrical and Computer Engineering, University of British Columbia, Vancouver, BC, Canada ²Huawei Technologies Canada, Burnaby, BC, Canada ³Department of Mechanical and Industrial Engineering, University of Toronto, Toronto, ON, Canada. Correspondence to: Mahdi Mostajabdaveh <mahdi.mostajabdaveh1@huawei.com>.

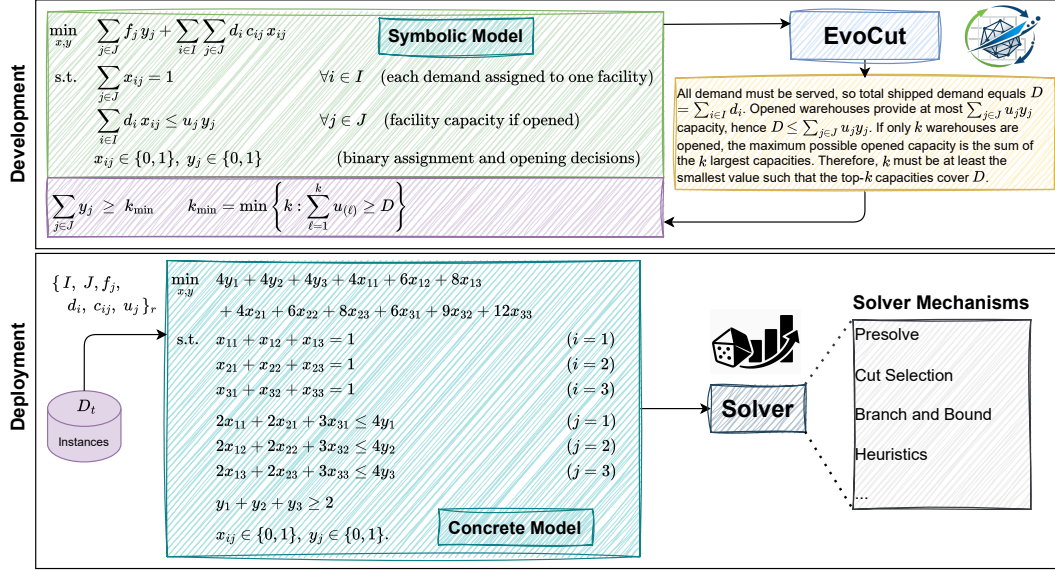


Figure 1. View of EVOCUT in two stages: during development, EVOCUT learns a reusable set of acceleration cuts for a symbolic model using a natural language description. During deployment, the cuts are instantiated for a concrete instance and appended to its MILP before solving.

solver pipelines across different MILP models.

Contributions. (i) We introduce EVOCUT, a general framework for generating *acceleration cuts at the symbolic modeling level*. Given a symbolic MILP model and a small verification set, EVOCUT discovers a *reusable* set of acceleration cuts (with symbolic formulation) that can be instantiated for each concrete instance. (ii) We evaluate EVOCUT on seven diverse MILP problems and show consistent solver speedups on unseen instances, including up to 76% smaller optimality gaps at fixed time budgets and up to $7.2\times$ faster time to target gaps. (iii) We provide an ablation and robustness analysis of EVOCUT: (a) data efficiency by varying the sizes of the evaluation and verification sets (Section 5.5), (b) LLM sensitivity across DeepSeek-R1, Gemini Pro 3, and GPT-5.1 (Table 1), (c) solver robustness by transferring the same discovered cuts across solvers (Appendix E.2), and (d) interaction with solver internal cut generation by comparing Gurobi with internal cuts disabled vs. default settings (Appendix E.3).

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., 2024b; 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; Li et al., 2025). Re-

cent 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., 2024) and the staged agent architecture of (Mostajabdaveh et al., 2024). These efforts focus on modeling fidelity, ensuring correct formulations and code generation from text. In contrast, EVOCUT targets efficiency after formulation: we automatically generate acceleration cuts and experimentally verify preservation of optimal solutions while selecting cuts based on their measured impact on the solver’s optimality gap. To our knowledge, prior research has not considered automating the generation of problem-specific acceleration cuts, a process that requires both modeling expertise and a deep understanding of combinatorial logic.

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., 2024a; 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., 2024a) 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, EVOCUT evolves acceleration cuts for MILPs and deploys them to reduce the optimality gap faster. By targeting efficiency after formulation, it takes advantage of LLMs’

reasoning capabilities to improve solver performance.

Several learning methods have been proposed to improve cut selection in MILP solvers. Early work framed it as a reinforcement learning (RL) task to choose cutting planes within branch and cut (Tang et al., 2020), while imitation learning approximated a lookahead oracle scoring cuts by exact bound improvement (Paulus et al., 2022). Hierarchical RL jointly optimized the number and order of cuts for additional speedups (Wang et al., 2023), and more recent LLM approaches use natural language problem descriptions to selectively activate separators built into the solver (Lawless et al., 2025). Unlike these methods, which select or tune solver-generated cuts and parameters, EVOCUT synthesizes new acceleration cuts that are *reusable* at the symbolic modeling level.

3. Preliminaries

To justify the practical relevance of an automatic yet reliable generator of acceleration cuts, we briefly recall MILPs, their linear relaxations, and the notions of valid and optimality preserving cuts. We also highlight that empirically verified acceleration cuts can be useful in practice even without traditional proofs establishing them as valid or optimality preserving cuts.

We consider an MILP of the form

$$\max\{c^\top x + h^\top y : Ax + Gy \leq b, x \in \mathbb{Z}_{\geq 0}^n, y \in \mathbb{R}_{\geq 0}^p\}, \quad (1)$$

with integer variables $x \in \mathbb{Z}_{\geq 0}^n$ and continuous variables $y \in \mathbb{R}_{\geq 0}^p$. Its feasible set is $S = \{(x, y) \in \mathbb{Z}_{\geq 0}^n \times \mathbb{R}_{\geq 0}^p : Ax + Gy \leq b\}$. Relaxing integrality yields the linear programming (LP) relaxation

$$\max\{c^\top x + h^\top y : Ax + Gy \leq b, x \in \mathbb{R}_{\geq 0}^n, y \in \mathbb{R}_{\geq 0}^p\}, \quad (2)$$

whose feasible set (the linear relaxation set) is $P = \{(x, y) \in \mathbb{R}_{\geq 0}^n \times \mathbb{R}_{\geq 0}^p : Ax + Gy \leq b\}$. Let $\text{conv}(S)$ 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, EVOCUT produces additional inequalities 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 Fig. 6 in Appendix D). Optimality preserving cuts 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 cut off 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 non-trivial 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 cuts that are proved to satisfy this.

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

Definition 3.1 (Acceleration cut). A cut added to an MILP with the explicit aim of reducing solution time. It is not necessarily a proven valid inequality or a proven optimality preserving cut, and is considered useful based on experiments after empirically screening that (i) it preserves the optimal objective value on a small verification set and (ii) it yields a tangible speedup.

Modern MILP solvers routinely inject cuts that are general purpose during solving and that apply to many formulations (Chvátal, 1973; Balas, 1979). These are enabled by default in commercial software like Gurobi (Gurobi Optimization LLC, 2024). By contrast, acceleration cuts tailored to a specific problem class exploit structure in a given formulation and can yield larger speedups when designed and checked carefully (Marchand et al., 2002). Examples include 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, EVOCUT automatically proposes, experimentally verifies, and evolves such cuts to improve solve time.

4. Proposed Method

We propose EVOCUT, an evolutionary algorithm powered by multiple LLM agents to iteratively generate and refine acceleration cuts that speed up solving across instances of a given MILP model. Fig. 2 illustrates the flow diagram of EVOCUT 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 (3) evolution (crossover and mutation driven by LLM agents). Next, we describe these phases, the verifier, and the fitness score.

We begin with a data preprocessing step by splitting available instances for a given MILP into an evaluation set D_e and a verification set D_v . We use D_v for verification set optimal solution screening (OSS) and usefulness checks, and D_e to evaluate cut quality. For every instance $i \in D_e$, we run the baseline MILP under a fixed computational budget (e.g. Gurobi’s `WorkLimit` (Gurobi Optimization LLC, 2025) set to 10) and record its terminal optimality gap, $\text{gap}_{\text{ref}}(i)$. We construct D_e so that $\text{gap}_{\text{ref}}(i) > 0$ for all $i \in D_e$. This *reference gap* provides the baseline to evaluate the impact of new cuts on solver performance within EVOCUT. For the smaller subset D_v , we also store (i) a solver solution $(\hat{x}_i^*, \hat{y}_i^*)$ from solving the MILP to an optimality gap tolerance of 10^{-4} (for the OSS check) and (ii) the LP relaxation optimum $(x_i^{\text{LP}}, y_i^{\text{LP}})$ (for the usefulness check). Each $i \in D_v$ therefore carries its reference gap $(\text{gap}_{\text{ref}}(i))$ together with both solutions. Specifically, we have $D_e = \{i \mapsto \text{gap}_{\text{ref}}(i)\}$ and $D_v = \{i \mapsto ((\hat{x}_i^*, \hat{y}_i^*), (x_i^{\text{LP}}, y_i^{\text{LP}}), \text{gap}_{\text{ref}}(i))\}$. With these datasets in place, EVOCUT seeds an initial population of candidate cuts using an initializer agent. At each iteration, the LLM agent receives three inputs: (1) the code of the complete MILP formulation alongside a natural language description of all model components (Fig. 1), (2) the set of cuts generated so far, and (3) explicit instructions to propose novel and distinct acceleration cuts that tighten the LP relaxation in Eq. (2). The complete prompt templates are provided in Appendix B. The LLM response includes a brief description (an idea) of the proposed cut together with executable code (e.g., Pyomo (Hart et al., 2011)). We immediately run the verifier described next. Failures trigger a diagnostic prompt and allow the agent to retry up to a preset limit. We log each cut idea to prevent duplication. This loop continues until the population reaches its predefined size, at which point the verified and distinct cuts enter the evolutionary search phase (Fig. 2). To decide whether a candidate enters the population, each newly proposed cut C must pass three checks (Fig. 3): (i) a code check, (ii) an OSS check, and (iii) a usefulness 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 change the optimal objective on D_v . Concretely, for each $i \in D_v$, 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 eval-

uate the inequalities on the fixed solution. Instead, we ask whether there exists z that makes the cut feasible. This solve is efficient since 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 test usefulness by discarding any cut that does not separate any stored LP optimum in D_v . For each $i \in D_v$, we relax all integrality constraints, fix $(x, y) = (x_i^{\text{LP}}, y_i^{\text{LP}})$, and test feasibility with C appended. We retain C iff there exists $i \in D_v$ such that $\nexists z$ with $A'_r x_i^{\text{LP}} + G'_r y_i^{\text{LP}} + H'_r z \leq \delta_r \quad \forall r \in R$ (i.e., the reduced LP becomes infeasible). Otherwise, C is discarded and the agent receives feedback (up to the retry limit).

After a cut C passes the verification checks of Section 4, we evaluate it on D_e . We append it to the baseline MILP and solve every evaluation instance $i \in D_e$ under the same budget, record the resulting optimality gap $\text{gap}_{\text{cut}}(i)$, and compare it to $\text{gap}_{\text{ref}}(i)$. The key intuition is that a cut should receive higher fitness when it reduces the gap on average, and lower fitness when it increases it. We implement this by computing the signed relative gap change per instance, averaging over D_e , and then applying a monotone mapping so that more effective cuts receive higher fitness scores. The fitness equation is provided in Appendix H. With the initial population of verified and evaluated cuts ready, the EVOCUT algorithm evolves the population for T generations. The main steps of the evolution are as follows: **(i)** Elitism: The top r_e fraction of cuts with the highest fitness carry over to the next generation. **(ii)** Selection: Parent cuts are picked via tournament selection based on fitness. **(iii)** Crossover: With probability $\mathbb{P}_c$, two parents are passed to a Crossover LLM, which attempts to produce a new child cut 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)** Verification & Evaluation: Each newly created candidate (C_o or C_m) is checked according to the verification checks in Section 4. Failed candidates are discarded, and a feedback prompt is provided to the LLM (up to a predefined maximum retry limit). Verified candidates are then forwarded to the evaluation stage (Section 4). Their fitness is computed and then they are added to the next generation. At each iteration, reproduction (i.e., the generation of a new cut) continues until the next population reaches its required size. This evolutionary process then repeats for T generations. In each reproduction step, EVOCUT 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.

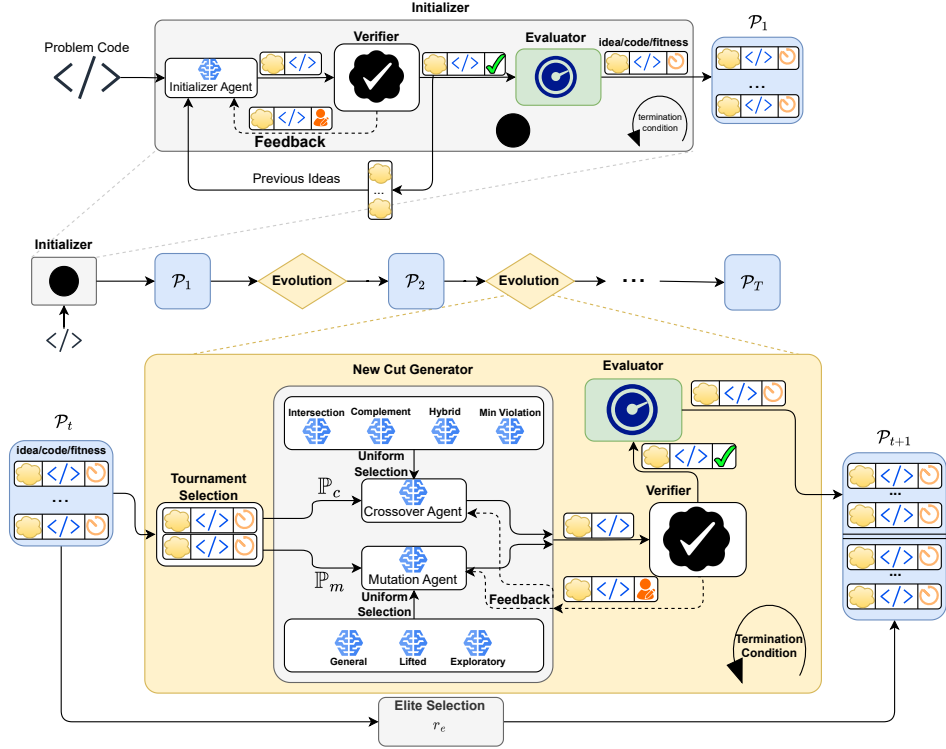


Figure 2. EVOCUT workflow: an initializer agent proposes a population of candidate acceleration cuts. Each candidate is screened for code validity, verification set optimal solution screening (OSS) on D_v , and usefulness via LP separation, then evaluated on D_e to obtain a fitness score based on the optimality gap. Tournament/elite selection and crossover/mutation driven by LLM agents generate the next population.

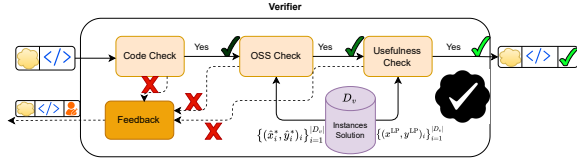


Figure 3. EVOCUT verifier: candidate cut code must compile (code check), pass OSS check on verification set D_v (i.e., not cut off the recorded reference solution), and cut off at least one stored fractional LP optimum (usefulness check). Failures trigger feedback and regeneration.

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: Can EVOCUT improve solver efficiency and to what extent?** We measure reductions in optimality gaps and wall clock time across diverse unseen instances. **RQ2: Does evolutionary search improve cuts generated by LLMs beyond a few shot?** We ablate the evolutionary component

by comparing against cuts produced solely by the initializer agent. **RQ3: How sensitive is EVOCUT to the sizes of evaluation (D_e) and verification (D_v) sets?** We vary $|D_e|$ and $|D_v|$ and report test set gap metrics for the strongest discovered cut.

5.1. Experimental Setup

We evaluate EVOCUT by comparing the baseline MILP to the baseline augmented with a single EVOCUT cut 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 considered to be among the fastest MILP solvers (Miltenerberger, 2025). We access Gurobi via the Pyomo interface (Hart et al., 2011) using eight threads; unless otherwise stated, solver parameters are defaults, except we fix `Seed=42` and `Method=4` (deterministic concurrent algorithm for continuous models and the initial root relaxation), use `WorkLimit` (Gurobi Optimization LLC, 2025) for EVOCUT’s internal evaluation runs (Section 4), and `TimeLimit=2000s` for test set benchmarking. Unless otherwise stated, we use each solver’s default parameter settings. In EVOCUT, we run $T = 20$ generations with

a population size $|\mathcal{P}| = 8$, using a mutation probability $\mathbb{P}_m = 0.3$, an elitism ratio $r_e = 0.2$, and a crossover probability $\mathbb{P}_c = 0.7$. Agents may retry cut generation up to three times if verification checks fail; this configuration required roughly 20 hours of wall clock time on our hardware. We use Gemini Pro 3 (Team et al., 2023) as the main LLM in EVOCUT (full API configuration and approximate costs are given in Appendix G); to assess sensitivity to the LLM, we also run EVOCUT with DeepSeek-R1 (Guo et al., 2025) and GPT-5.1 (OpenAI, 2025) (Table 1).

Our four core benchmarks span routing, network design, facility location, and scheduling: Traveling Salesman Problem (TSP), Multi-Commodity Network Design (MCND), Capacitated Warehouse Location Problem (CWLP), and Job Shop Scheduling Problem (JSSP). We additionally 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 derived from an IMO problem, so LLMs are unlikely to have seen 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 TSP (54), MCND (81), CWLP (36), JSSP (80), PDPTW (165), IMO6 (34), and SHUC (56). Appendix F provides a dedicated subsection for each benchmark, including its formulation, instance source, and dataset details. During evolution, EVOCUT relies on relatively small datasets D_e (default size 10) and D_v (default size 2), as detailed in Section 4, and we do not use benchmark test instances: for all problems except IMO6, both D_e and D_v are drawn from synthetic generators (released alongside our code) while public benchmark collections are held out as the test set D_t . For IMO6, all instances are generated and split into D_e , D_v , and D_t .

5.2. Generalization to Unseen Instances

We compare the baseline MILP to the baseline MILP augmented with the strongest EVOCUT cut 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 added cut. 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 relative gap improvement: $\Delta_g = (g_{\text{ref}} - g_{\text{cut}})/g_{\text{ref}}$. We report $\bar{\Delta}_g$, the average of Δ_g over all instances in D_t . At a checkpoint, $\Delta_g > 0$ iff $g_{\text{cut}} < g_{\text{ref}}$. 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. The table also reports that different LLMs yield different cuts and different $\bar{\Delta}_g$ values. In Table 1, $\bar{\Delta}_g$ at 2000 s is positive for all problem/LLM pairs. Among these entries,

the 2000 s mean ranges from 4.3% (SHUC, Gemini Pro 3) to 59.2% (JSSP, Gemini Pro 3). Appendix Fig. 7 plots $\bar{\Delta}_g(t)$ over time. In the SHUC panel, the mean curve approaches 0% within the first few hundred seconds. Appendix Fig. 11 shows four MCND instances where the augmented gap trace lies slightly above the baseline trace over most of the plotted horizon. Tables 5 and 6 report additional checkpoint $\bar{\Delta}_g$ results for the Gemini Pro 3 cuts under solver and parameter changes. In Table 5, the HiGHS values are smaller than the Gurobi values for TSP, MCND, JSSP, PDPTW, and IMO6, while CWLP and SHUC have larger HiGHS values at the later checkpoints. In Table 6, disabling internal cuts increases the reported $\bar{\Delta}_g$ for TSP and CWLP at all checkpoints, whereas MCND and PDPTW show the opposite ordering. SHUC and IMO6 show mixed ordering across checkpoints.

5.3. Post-hoc Screening Against Proven Optimal Solutions

EVOCUT screens candidate cuts using the small verification set D_v during development (Section 4). As an additional experiment on the held out test sets, we ask a simple question: Does the strongest EVOCUT cut ever cut off a proven optimal solution that we can compute for that instance? This is an offline experiment (not a deployment time procedure of EVOCUT). 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 36/36, JSSP 58/80, MCND 77/81, TSP 54/54, IMO6 9/34, and SHUC 53/56. For each such instance, we fix the original decision variables to the proven optimal solution, append the learned cut, and solve only for any auxiliary variables introduced by the cut to test feasibility. In all of these cases, the feasibility check succeeds, i.e., we find no counterexample in which the strongest EVOCUT cut 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 is not available. We therefore report a necessary bound consistency check: 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 165/165. These results provide additional empirical evidence on the test sets; they do not constitute a formal guarantee of global validity or optimality preservation.

5.4. Time to Target Optimality Gap

To complement the checkpoint gap analysis (Section 5.2), we measure how much EVOCUT 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. For each instance and method (baseline vs. augmented), we compute the time to reach

Table 1. Test set mean relative gap improvement $\bar{\Delta}_g$ (%) from adding the strongest EVO CUT cut (Gurobi, three LLMs). Mean $\pm \sigma$. Positive values correspond to $g_{\text{cut}} < g_{\text{ref}}$ at that checkpoint.

Problem	LLM	5 s	10 s	50 s	150 s	300 s	1000 s	2000 s
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)
	Gemini Pro 3	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)
	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)
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)
	Gemini Pro 3	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)
	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)
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)
	Gemini Pro 3	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)
	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)
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)
	Gemini Pro 3	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)
	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)
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)
	Gemini Pro 3	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)
	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)
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)
	Gemini Pro 3	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)
	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)
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)
	Gemini Pro 3	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)
	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)

Table 2. Time to reach target gaps g : baseline vs. EVO CUT. B/A: instances reaching g within 2000 s (Baseline/EVO CUT). SGM: shifted geometric mean speedup (> 1 better).

Problem	$g = 10^{-5}$		$g = 10^{-4}$		$g = 10^{-3}$		$g = 10^{-2}$		$g = 10^{-1}$	
	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
MCND	56/57	1.04	62/62	1.04	62/62	1.03	69/69	1.04	76/74	0.92
CWLP	26/23	2.44	35/36	4.24	35/36	4.42	35/36	5.19	35/36	3.83
JSSP	7/8	2.18	7/8	2.31	7/8	2.42	7/8	2.55	10/19	7.20
PDPTW	0/5	5.18	0/5	5.18	0/5	5.19	0/5	5.25	0/20	6.77
IMO6	7/8	1.23	7/8	1.23	7/8	1.23	7/8	1.24	7/8	1.39
SHUC	2/2	1.47	6/6	1.28	17/17	1.26	39/39	1.25	53/54	1.34

gap $t(g)$ by linearly interpolating the solver trace. If a run never reaches the target gap, we assign a penalized time $t(g) = 2T_{\text{max}}$. We aggregate using the shifted geometric mean (SGM, with shift $s=1$): $\text{SGM}(t) = (\prod_{i \in \mathcal{I}_g} (t_i + s))^{1/|\mathcal{I}_g|} - s$, where $\mathcal{I}_g$ is the set of instances for which at least one method reaches g . Table 2 reports $\text{SGM}(t_{\text{ref}})/\text{SGM}(t_{\text{cut}})$ and solved counts (B/A). In Table 2, $\text{SGM}(t_{\text{ref}})/\text{SGM}(t_{\text{cut}}) > 1$ for all problems at $g \in \{10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}\}$, and for six of seven problems at $g = 10^{-1}$.

5.5. Sensitivity to $|D_e|$ and $|D_v|$

We vary $|D_e|$ and $|D_v|$ on the JSSP benchmark using DeepSeek-R1. Table 3 reports test set $\bar{\Delta}_g \pm \sigma$ of the strongest cut found under each configuration. We ran two experiments. In the first, we fixed $|D_e|=10$ and varied $|D_v|$ over $\{2, 5, 10\}$. In the second, we held $|D_v|=10$ constant and varied $|D_e|$ over the same three values. In every configuration, the best cut found by EVO CUT was evaluated on the test set

Table 3. JSSP: test $\bar{\Delta}_g \pm \sigma$ (%) of the strongest EVO CUT cut vs. $|D_v|$ and $|D_e|$. Left varies $|D_v|$ ($|D_e|=10$). Right varies $|D_e|$ ($|D_v|=10$).

$ D_v $	$\bar{\Delta}_g \pm \sigma$ (%)	$ D_e $	$\bar{\Delta}_g \pm \sigma$ (%)
2	37.3 $\pm$ 22.0	2	7.1 $\pm$ 16.4
5	37.3 $\pm$ 22.0	5	37.3 $\pm$ 22.0
10	40.4 $\pm$ 22.3	10	40.4 $\pm$ 22.3

D_t , which we held out. Following the post-hoc screening in Section 5.3, we checked that none of the generated cuts (under any configuration) cut off the proven-optimal JSSP solutions; all setups pass at 100% of test instances (58/58).

5.6. Evolutionary Diagnostics

We report three diagnostics of the evolutionary process: (i) agent outcome statistics (Table 4), (ii) a traced ancestry graph for the best JSSP cut (Fig. 4), and (iii) population fitness trajectories over generations (Fig. 5). Full details are in Appendix C. In Table 4, the mutation and crossover agents have higher success rates than the initializer agent across the three LLMs. The table reports mean fitness gains $\bar{\Delta}_f$ that include both negative and positive values. Second, we trace the development of the best cut. For the best acceleration cut discovered by EVO CUT for the JSSP, we tracked its ancestry, acting agents, and fitness scores. Fig. 4 (Appendix C.2) shows the traced graph for this best cut. In Appendix F, we also include the best novel cut discovered by EVO CUT for each problem. Third, we plot fitness curves at the population level. For each of the seven problems,

we tracked the maximum and mean fitness scores of the population over 20 generations. Fig. 5 (in Appendix C.3) shows the trajectories of these values across generations. For each benchmark, the best fitness curve increases over the 20 generations.

6. Discussion

Starting with RQ1, Table 1 and Appendix E.1 (Appendix Fig. 7) show that EVOCUT can reduce the optimality gap at successive checkpoints under the same time budgets. Across the TSP experiments, this appears as earlier gap closure (Appendix E.4, Appendix Fig. 8), faster tightening of the best bound¹ (Klotz & Oberdieck, 2024) (Appendix E.5, Appendix Fig. 9), and substantially smaller PDI trajectories on the same instances (Appendix E.6, Appendix Fig. 10). Table 1 also indicates that different LLMs lead to different cuts and levels of improvement, so LLM choice matters even though multiple LLMs can be used in EVOCUT. Complementing this, Table 2 shows faster times to reach target gaps across benchmarks (up to $7.2\times$ shifted geometric mean speedup) and often increases the number of instances that reach the target gap within the time limit. At the same time, EVOCUT cuts are not always beneficial: when a discovered cut does not tighten the LP relaxation for an instance, it can still increase model size and overhead, leading to slightly worse performance (Appendix Fig. 11, Appendix E.7).

To probe robustness beyond gap improvements, we report three complementary experiments. First, in a post-hoc screening against proven-optimal solutions (Section 5.3), we find no counterexample in which the strongest EVOCUT cuts exclude a proven-optimal solution (and we also report the PDPTW bound consistency check). Second, the same discovered cuts (Gemini Pro 3) continue to improve performance when instantiated in a different solver (HiGHS), suggesting solver transfer (Table 5). Third, the gains persist under different solver cut ecosystems: Gurobi with internal cuts disabled vs. default (Table 6). Together, these results suggest EVOCUT provides a lightweight, pre-solve augmentation that can complement solver internal cut selection. We also observe that the magnitude of improvement depends on how much headroom the baseline model leaves for augmentation: for problem classes with decades of specialized formulations and cuts, a strong baseline leaves less room for additional gains, whereas for less standardized or newly formulated MILPs, EVOCUT tends to yield larger gains. 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), and we do not aim to match problem tailored branch-and-cut method for each benchmark.

¹Higher best bound curves in minimization imply tighter lower bounds, which directly shrink the optimality gap.

Turning to RQ2, evolutionary search improves cut quality beyond a one-shot initializer. Initializer proposals have a lower success rate than evolved offspring (Appendix C.1), while both the ancestry trace (Fig. 4) and the population level fitness trajectories (Fig. 5) show sustained improvement over generations. Together, these results support the value of coupling LLM generation with evolutionary refinement for producing stronger acceleration cuts.

Finally, for RQ3, EVOCUT derives cuts from underlying problem logic rather than patterns specific to a dataset. Accordingly, cuts remain effective even when the test distribution or size differs from those used for verification and evaluation. This is consistent with our split: D_e and D_v are synthetic, while D_t is a held-out public benchmark set. Table 3 suggests that small verification sets can suffice, while larger evaluation sets generally yield stronger cuts. As a practical consideration, there is a runtime–quality tradeoff in the sizes of D_e and D_v : larger evaluation sets (and, to a lesser extent, larger verification sets) tend to improve deployment performance, but increase EVOCUT development runtime since each candidate cut is tested on more instances (Section 5.5). Appendix F reports instance sizes in D_e and D_v .

7. Conclusion and Future Direction

EVOCUT automates acceleration cut generation, a step in MILP formulations improvement that is typically driven by experts. Across seven MILP benchmarks, EVOCUT discovers reusable and interpretable cuts that can be appended to MILPs in standard solver pipelines and often improve time-limited solver progress. We expect the approach to be particularly useful for custom or newly formulated MILPs, where established cuts may be limited.

Limitations. Our screening is empirical and does not provide a formal guarantee that a learned cut is globally valid or optimality preserving; establishing such guarantees may still require human expert proofs. Moreover, we currently focus on cuts that can be added statically before solving, rather than cuts 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 verification set. Integrating automated formal proof tooling could provide theoretical guarantees (e.g., discovering cuts that are provably valid or that provably preserve optimality) (Yang et al., 2025). (2) EVOCUT currently inserts cuts into the MILP model before the solution process begins. Dynamically separating and adding cuts via callbacks during the solve could further improve solver performance, but requires generating an efficient separation algorithm for each discovered cut.

Impact Statement

EvoCUT is a method for proposing reusable, human-interpretable acceleration cuts for symbolic MILP formulations using LLM-based agents and evolutionary search. When such cuts improve solver progress under fixed budgets, they can reduce compute time and energy use and enable practitioners to solve larger or more frequent planning instances. Because problem-specific cuts are often expert-designed, EvoCUT, as a generic cut generation framework, may reduce the manual effort required to propose candidate strengthening constraints for new or customized MILP formulations. The approach is most naturally applied in settings where the same symbolic model is solved repeatedly (e.g., logistics, manufacturing, and energy planning), and where users can validate added constraints within their existing modeling and testing workflow. EvoCUT is not a formal verification tool: it relies on empirical screening during development phase (e.g., checking feasibility of stored optimal solutions and separation of stored LP solutions on a verification set) and therefore does not guarantee global validity or optimality preservation on all unseen instances. In production settings, we recommend expert review (and, when possible, formal validation) of any generated cut before deployment. Accordingly, deployment should be conservative (e.g., keep a baseline fallback and monitor for regressions), and future work on proof generation could further reduce risk and broaden applicability. As with any optimization accelerator, benefits depend on the application domain, and improved efficiency could also amplify undesirable uses if applied inappropriately.

References

- AhmadiTeshnizi, A., Gao, W., and Udell, M. 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.
- Ahmed, T. and Choudhury, S. Lm4opt: Unveiling the potential of large language models in formulating mathematical optimization problems. *INFOR: Information Systems and Operational Research*, 62(4):559–572, 2024.
- Alvarez, A. M., Louveaux, Q., and Wehenkel, L. A machine learning-based approximation of strong branching. *INFORMS Journal on Computing*, 29(1):185–195, 2017.
- Aref, S., Mason, A. J., and Wilson, M. C. A modeling and computational study of the frustration index in signed networks. *Networks*, 75(1):95–110, 2020.
- Art of Problem Solving Wiki. 2025 IMO problem-6. https://artofproblemsolving.com/wiki/index.php/2025_IMO_Problems/Problem_6, 2025. Accessed: 14 Aug 2025.
- Balas, E. Disjunctive programming. *Annals of discrete mathematics*, 5:3–51, 1979.
- Beasley, J. E. An algorithm for solving large capacitated warehouse location problems. *European Journal of Operational Research*, 33(3):314–325, 1988.
- Beasley, J. E. Or-library: distributing test problems by electronic mail. *Journal of the operational research society*, 41(11):1069–1072, 1990.
- Chvátal, V. 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/mmc>, 2021. Accessed: 2025-05-01.
- Costa, A. M., Cordeau, J.-F., and Gendron, B. Benders, metric and cutset inequalities for multicommodity capacitated network design. *Computational Optimization and Applications*, 42(3):371–392, 2009. doi: 10.1007/s10589-007-9122-0.
- da Cunha, A. S., Simonetti, L., and Lucena, A. 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.
- Gendron, B., Crainic, T. G., and Frangioni, A. Multicommodity capacitated network design. In *Telecommunications Network Planning*, pp. 1–19. Springer, 1999.
- Guo, D., Yang, D., Zhang, H., Song, J., Zhang, R., Xu, R., Zhu, Q., Ma, S., Wang, P., Bi, X., et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.
- Gurobi Optimization LLC. Gurobi Optimizer Reference Manual, 2024. URL <https://www.gurobi.com>.
- Gurobi Optimization LLC. Worklimit parameter in Gurobi optimizer. <https://docs.gurobi.com/projects/optimizer/en/current/reference/parameters.html>, 2025. Accessed: 2025-05-14.
- Hart, W. E., Watson, J.-P., and Woodruff, D. L. Pyomo: modeling and solving mathematical programs in python. *Mathematical Programming Computation*, 3(3):219–260, 2011.
- Huang, C., Tang, Z., Hu, S., Jiang, R., Zheng, X., Ge, D., Wang, B., and Wang, Z. Orlm: A customizable framework in training large models for automated optimization modeling. *Operations Research*, 2025.

- Huang, S., Yang, K., Qi, S., and Wang, R. When large language model meets optimization. *Swarm and Evolutionary Computation*, 90:101663, 2024. ISSN 2210-6502. doi: <https://doi.org/10.1016/j.swevo.2024.101663>.
- Jiang, C., Shu, X., Qian, H., Lu, X., Zhou, J., Zhou, A., and Yu, Y. LLMOPT: Learning to define and solve general optimization problems from scratch. *arXiv preprint arXiv:2410.13213*, 2024.
- Klotz, E. and Oberdieck, R. Converting weak to strong mip formulations: A practitioner’s guide. In *Optimization Essentials: Theory, Tools, and Applications*, pp. 113–174. Springer, 2024.
- Koch, T., Achterberg, T., Andersen, E., Bastert, O., Berthold, T., Bixby, R. E., Danna, E., Gamrath, G., Gleixner, A. M., Heinz, S., et al. Miplib 2010: mixed integer programming library version 5. *Mathematical Programming Computation*, 3(2):103–163, 2011.
- Laporte, G. The traveling salesman problem: An overview of exact and approximate algorithms. *European Journal of Operational Research*, 59(2):231–247, 1992.
- Laporte, G. and Semet, F. An optimality cut for mixed integer linear programs. *European Journal of Operational Research*, 119(3):671–677, 1999.
- Lawless, C., Li, Y., Wikum, A., Udell, M., and Vitercik, E. 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.
- Li, S., Ouyang, W., Paulus, M., and Wu, C. Learning to configure separators in branch-and-cut. *Advances in Neural Information Processing Systems*, 36:60021–60034, 2023.
- Li, S., Kulkarni, J., Wu, C., Menache, I., and Li, B. Towards foundation models for mixed integer linear programming. In *The Thirteenth International Conference on Learning Representations*, 2025.
- Liu, F., Tong, X., Yuan, M., Lin, X., Luo, F., Wang, Z., Lu, Z., and Zhang, Q. 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, 2024a.
- Liu, F., Yao, Y., Guo, P., Yang, Z., Zhao, Z., Lin, X., Tong, X., Yuan, M., Lu, Z., Wang, Z., and Zhang, Q. A systematic survey on large language models for algorithm design, 2024b. URL <https://arxiv.org/abs/2410.14716>.
- Lysgaard, J., Letchford, A. N., and Eglese, R. W. A new branch-and-cut algorithm for the capacitated vehicle routing problem. *Mathematical programming*, 100:423–445, 2004.
- Manne, A. S. On the job-shop scheduling problem. *Operations Research*, 8(2):219–223, 1960.
- Marchand, H., Martin, A., Weismantel, R., and Wolsey, L. Cutting planes in integer and mixed integer programming. *Discrete Applied Mathematics*, 123(1-3):397–446, 2002.
- Miltenberger, M. Interactive visualizations of mittelmann benchmarks. <https://github.com/mattmilten/mittelmann-plots>, 2025. Accessed: 2025-05-01.
- Mostajabdaveh, M., Yu, T. T., Ramamonjison, R., Carenini, G., Zhou, Z., and Zhang, Y. 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.
- Mostajabdaveh, M., Yu, T. T. L., Dash, S. C. B., Ramamonjison, R., Byusa, J. S., Carenini, G., Zhou, Z., and Zhang, Y. 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.
- Ostrowski, J., Anjos, M. F., and Vannelli, A. Tight mixed integer linear programming formulations for the unit commitment problem. *IEEE transactions on power systems*, 27(1):39–46, 2011.
- Paulus, M. B., Zarpellon, G., Krause, A., Charlin, L., and Maddison, C. 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.
- Queyranne, M. and Wang, Y. Single-machine scheduling polyhedra with precedence constraints. *Mathematics of Operations Research*, 16(1):1–20, 1991.
- Ramamonjison, R., Yu, T., Li, R., Li, H., Carenini, G., Ghaddar, B., He, S., Mostajabdaveh, M., Banitalebi-Dehkordi, A., Zhou, Z., et al. NL4Opt competition: Formulating optimization problems based on their natural language descriptions. In *NeurIPS 2022 Competition Track*, pp. 189–203, 2023.

- Reinelt, G. TSPLIB—a traveling salesman problem library. *ORSA Journal on Computing*, 3(4):376–384, 1991.
- Romera-Paredes, B., Barekatin, M., Novikov, A., Balog, M., Kumar, M. P., Dupont, E., Ruiz, F. J., Ellenberg, J. S., Wang, P., Fawzi, O., Kohli, P., and Fawzi, A. Mathematical discoveries from program search with large language models. *Nature*, 625(7995):468–475, 2024.
- Srour, J., Máhr, T., De Weerd, M., and Zuidwijk, R. Miplib truckload pdptw instances derived from a real-world drayage case. Technical report, 2010.
- Taillard, E. Benchmarks for basic scheduling problems. *European Journal of Operational Research*, 64(2):278–285, 1993.
- Tang, Y., Agrawal, S., and Faenza, Y. Reinforcement learning for integer programming: Learning to cut. In *International conference on machine learning*, pp. 9367–9376. PMLR, 2020.
- Team, G., Anil, R., Borgeaud, S., Alayrac, J.-B., Yu, J., Soricut, R., Schalkwyk, J., Dai, A. M., Hauth, A., Millican, K., et al. Gemini: a family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805*, 2023.
- Toth, P. and Vigo, D. *The vehicle routing problem*. SIAM, 2002.
- Wang, Z., Li, X., Wang, J., Kuang, Y., Yuan, M., Zeng, J., Zhang, Y., and Wu, F. Learning cut selection for mixed-integer linear programming via hierarchical sequence model. *arXiv preprint arXiv:2302.00244*, 2023.
- Xiao, Z., Zhang, D., Wu, Y., Xu, L., Wang, Y. J., Han, X., Fu, X., Zhong, T., Zeng, J., Song, M., et al. Chain-of-experts: When llms meet complex operations research problems. In *ICLR*, 2024.
- Yang, K., Poesia, G., He, J., Li, W., Lauter, K. E., Chaudhuri, S., and Song, D. Position: Formal mathematical reasoning—a new frontier in ai. In *Forty-second International Conference on Machine Learning Position Paper Track*, 2025.
- Ye, H., Wang, J., Cao, Z., Berto, F., Hua, C., Kim, H., Park, J., and Song, G. ReEvo: Large language models as hyperheuristics with reflective evolution. In *Proceedings of the 38th Conference on Neural Information Processing (NeurIPS 2024)*, pp. 10–15, Vancouver, Canada, 2024.

A. EVO CUT Pseudocode

The full procedure is outlined in Algorithm 1, which presents the pseudocode for the proposed EVO CUT framework.

B. EVO CUT Agents

This section outlines the agents used in the EVO CUT process and presents their prompt structures, including roles, tasks, requirements, and inputs/outputs. The agent library comprises a single **initializer** and several **Mutation** and **Crossover** agents, each endowed with specific instructions to generate valid and useful cuts. The prompt structure for the initializer agent is as follows:

initializer agent

Role: You are an MILP optimization expert with extensive knowledge in designing valid inequalities for MILP models.

Task:

- Propose a valid, effective constraint (cut) that tightens the feasible region and improves solver performance.
- Provide a clear, concise explanation of the cut’s derivation, validity, and impact.

Requirements:

- Cut must be valid for all instances of the problem.
- Introduce new variables or constructs if needed.
- Cut should be conceptually distant from previous ideas.

Input:

- i Baseline MILP (Pyomo model code) i .
- i List of previous ideas (if applicable) i .

Output:

```
{ "code": "<Only python code snippet for the added cut using Pyomo syntax>",
  "idea": "Concise technical explanation of cut derivation, validity, and 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 constraint in an Evolutionary Algorithm for MILP cut generation.

Task:

- Propose a valid and effective constraint (cut) that reduces the LP relaxation feasible region but remains valid for any integer feasible solution by using the idea behind the provided individual cut.
- Provide a concise explanation of the cut’s derivation, validity, and impact.
- i Agent specific instruction i

Requirements:

- The cut must be valid for all instances.
- The new constraint should enhance the provided Input Cut.

Algorithm 1 Pseudocode of EVO CUT

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

Ensure: A set of discovered acceleration cuts with corresponding fitness.

```

1: Phase 1: Data Preprocessing
2: Set up  $D_e = \{i \mapsto \text{gap}_{\text{ref}}(i)\}$ ,  $D_v = \{i \mapsto ((\hat{x}_i^*, \hat{y}_i^*), (x_i^{\text{LP}}, y_i^{\text{LP}}), \text{gap}_{\text{ref}}(i))\}$ .
3: Phase 2: Population Initialization
4:  $\mathcal{P}_1 \leftarrow \emptyset$ .
5: while  $|\mathcal{P}_1| < \mu$  do
6:   Call initializer LLM  $\rightarrow$  propose candidate cut  $C$ .
7:   if VerifyAndEvaluate( $C$ ) = True then
8:      $\mathcal{P}_1 \leftarrow \mathcal{P}_1 \cup \{C\}$ .
9:   else
10:    Provide feedback prompt to LLM and retry (up to max attempts).
11:   end if
12: end while
13: Phase 3: Evolution
14: for  $t \leftarrow 1$  to  $T$  do
15:   Elitism: Transfer top  $\lceil r_e \mu \rceil$  from  $\mathcal{P}_t$  to  $\mathcal{P}_{t+1}$ .
16:   while  $|\mathcal{P}_{t+1}| < \mu$  do
17:     Select two parents  $(C_p, C_q)$  from  $\mathcal{P}_t$  using tournament selection based on fitness.
18:     if rand() <  $\mathbb{P}_c$  then
19:       Agent Selection: randomly choose a crossover agent from the available pool.
20:       Call the selected Crossover LLM with  $(C_p, C_q) \rightarrow C_o$ .
21:       if VerifyAndEvaluate( $C_o$ ) = True then
22:          $\mathcal{P}_{t+1} \leftarrow \mathcal{P}_{t+1} \cup \{C_o\}$ .
23:         if  $|\mathcal{P}_{t+1}| = \mu$  then
24:           break
25:         end if
26:       else
27:         Provide feedback prompt to the selected crossover LLM and retry (up to max attempts).
28:       end if
29:     else if rand() <  $\mathbb{P}_m$  then
30:       Agent Selection: randomly choose a mutation agent from the available pool.
31:       Call the selected Mutation LLM with one parent  $C_p \rightarrow C_m$ .
32:       if VerifyAndEvaluate( $C_m$ ) = True then
33:          $\mathcal{P}_{t+1} \leftarrow \mathcal{P}_{t+1} \cup \{C_m\}$ .
34:       else
35:         Provide feedback prompt to the selected mutation LLM and retry (up to max attempts).
36:       end if
37:     end if
38:   end while
39: end for
40: return final population  $\mathcal{P}_{T+1}$ .
    
```

- Introduce new variables if needed.

Input:

- i *Baseline MILP (Pyomo model code)* i .
- i *Input Cut (code, idea, and score)* i .

Output:

```
{ "code": "<Only python code snippet for the added cut using Pyomo syntax>",
  "idea": "Concise technical explanation of cut derivation, validity, and impact" }
```

Agent specific instruction: Each mutation agent performs the same core task, but with variations in the approach:

- *General Mutation:* Proposes a new cut based on the individual constraint while improving the feasible region.
- *Lifted Mutation:* Enhances the cut by applying lifting techniques to tighten the feasible region further.
- *Exploratory Mutation:* Generates an exploratory cut that diverges from the provided constraint to explore new regions of the solution space.

Similarly, EVO CUT 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 constraint sets.

Task:

- Generate a valid and effective constraint (cut) by combining elements from the parent cuts.
- Provide a concise explanation of the new cut's idea, validity, and impact.
- i *Agent specific instruction* i .

Requirements:

- The cut must be valid for all instances.
- It should combine elements from both parent cuts in a novel and effective way.
- Introduce new variables or auxiliary constructs if needed.

Input:

- i *Baseline MILP (Pyomo model code)* i .
- i *First Parent Cut (code, idea, and score)* i .
- i *Second Parent Cut (code, idea, and score)* i .

Output:

```
{ "code": "<Only python code snippet for the added cut using Pyomo syntax>",
  "idea": "Concise technical explanation of cut derivation, validity, and impact" }
```

Agent specific instruction: Each crossover agent performs the same core task but with variations in the approach:

- *Intersection Crossover:* Combines elements of both parents to generate a cut that ensures both parent cuts are respected.

- Complementary Crossover: Generates a cut 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 quantify how effectively each EVO_{CUT} agent improves cut fitness. For each generated offspring, we compute the relative improvement rate $\Delta_f = (f_{\text{child}} - f_{\text{parent}})/f_{\text{parent}}$, where f_{parent} is the average fitness 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 fitness 10 that corresponds to "no cut".

Table 4 reports five statistics collected over the full run on all benchmarks: code generation failures, OSS failures, usefulness rejections, overall success rate, and the average improvement $\bar{\Delta}_f$ of successful offspring.

Table 4. EVO_{CUT} agent outcomes across three LLMs: code fail, OSS fail, not useful, success rate, and mean fitness gain $\bar{\Delta}_f \pm \sigma$ (%) (successful offspring only).

Agent	LLM	Code Fail. %	OSS Fail. %	Not Useful %	Success Rate %	$\bar{\Delta}_f \pm \sigma$ (%)
Initializer						
Main	DeepSeek	38.1	11.1	25.4	25.4	12.3 ± 20.8
	Gemini Pro 3	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 Pro 3	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 Pro 3	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 Pro 3	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 Pro 3	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 Pro 3	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 Pro 3	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 Pro 3	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

C.2. Evolutionary Generation of a Novel Cut

Fig. 4 shows how EVO_{CUT} using Deepseek mutates and recombines cuts to obtain a high quality acceleration cut 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 cuts with their fitness score. Arrows connect parents to offspring, and yellow panels quote the agent instructions that produced the offspring.

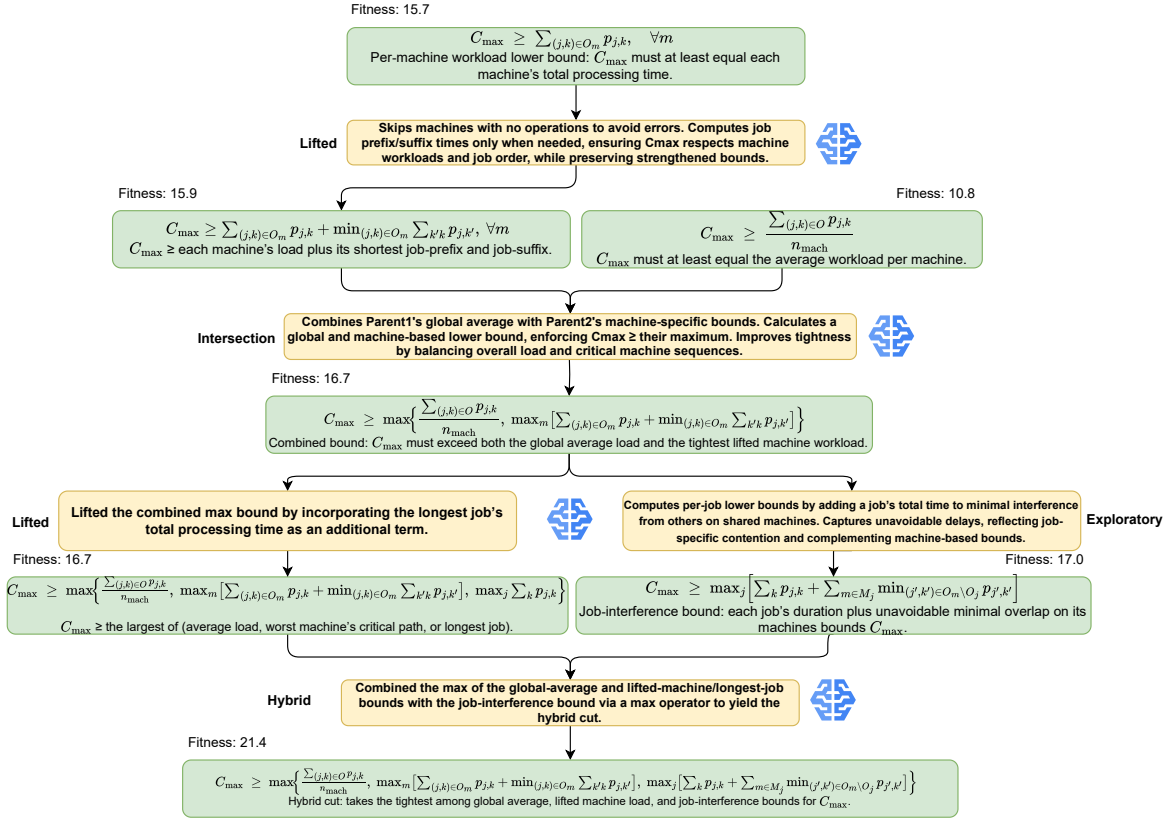


Figure 4. Steps in the development of the best cut for the JSSP. **Blue panel:** baseline formulation. **Green panels:** candidate cuts with annotated fitness. **Yellow panels:** agent instructions that produced each offspring.

C.3. Population Level Fitness Progress

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

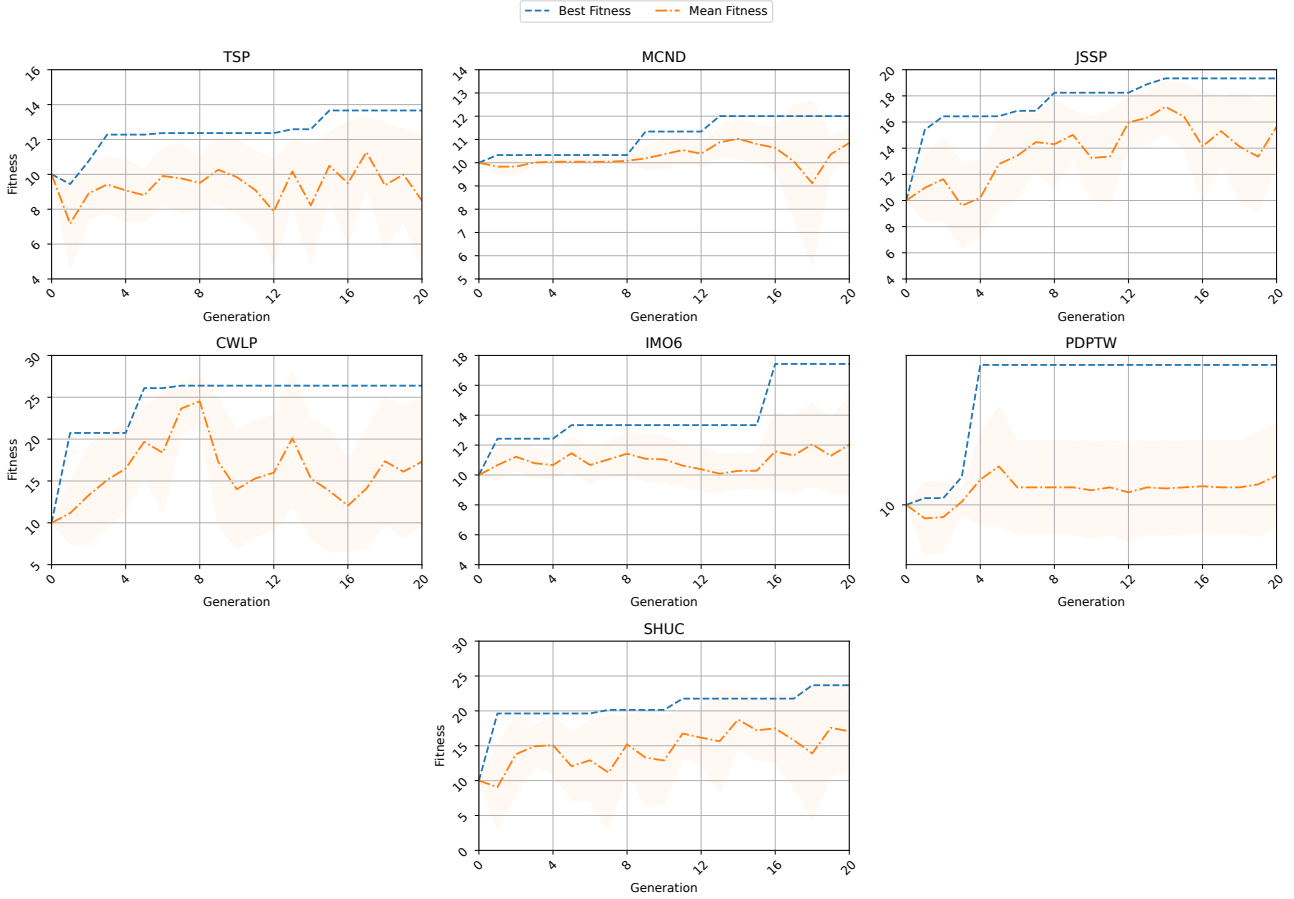


Figure 5. Best (blue, dashed) and mean (orange, dash dotted) fitness across 20 generations for the TSP, MCND, CWLP, JSSP, IMO6, PDPTW, and SHUC benchmarks. Shaded regions show one standard deviation.

D. Supplementary Mathematical Preliminaries

D.1. Geometric intuition for cuts

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 cuts that remove fractional LP optima while preserving all points in $\text{conv}(S)$, and (ii) optimality preserving cuts that may remove some feasible points but keep the optimal objective value unchanged. EVO CUT instead generates acceleration cuts that are empirically checked on a verification set (Section 4).

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 . Note that $cx = \beta$ is a supporting hyperplane

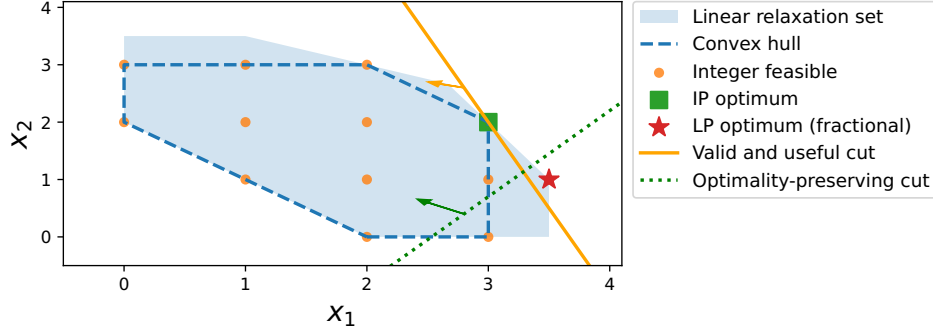


Figure 6. Relationship between the integer feasible set S , its LP relaxation P , the set $\text{conv}(S)$, a valid cut that removes the fractional LP optimum (useful), and an optimality preserving cut that may remove some feasible points while keeping the optimal value.

(has at least one point common with 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 the context of integer programming, valid inequalities that are facet defining for $\text{conv}(S)$ are especially important because they are necessary and sufficient for characterizing $\text{conv}(S)$. When $\text{conv}(S)$ is characterized, solving the MILP on set S (that is generally NP-hard) reduces to the linear program of maximizing $c^\top x + h^\top y$ on $\text{conv}(S)$ (which is polynomially solvable).

E. Additional Experiments

E.1. Aggregate gap improvement curves

Fig. 7 reports the mean relative gap improvement $\bar{\Delta}_g(t)$ as a function of time after augmenting the baseline MILP with the best EVOCUT cut, averaged across test instances that were held out (TSP, MCND, CWLP, JSSP, PDPTW, IMO6, and SHUC). This plot complements the checkpoint summaries in Table 1 by showing whether improvements occur early (better bounds sooner) and how consistently they persist throughout the solve.

E.2. Cross solver results for Gemini Pro 3 cuts

Table 5 compares $\bar{\Delta}_g$ when instantiating the same cuts discovered with Gemini Pro 3 but solving with two different MILP solvers (Gurobi vs. HiGHS). This ablation checks whether the observed gains are specific to a particular solver or persist under a different branch-and-bound implementation and default cut ecosystem.

E.3. Internal Cut Settings (Gemini Pro 3 Cuts)

Table 6 reports $\bar{\Delta}_g$ under Gurobi with internal cuts disabled versus enabled. This isolates how much of EVOCUT’s benefit remains when the solver’s own cutting plane machinery is reduced, and whether the interaction between user provided cuts and solver generated cuts changes the net improvement.

E.4. Representative TSP instance trajectories

To give a more detailed view of EVOCUT’s behavior, Fig. 8 plots the full gap traces over time for four challenging instances from the Traveling Salesman Problem Library (TSPLIB). Each panel compares the baseline MILP (blue line) with the same model augmented by the single strongest EVOCUT acceleration cut found during evolution (orange line). The line plots confirm that the added acceleration cut drives a markedly faster gap closure. Note, the acceleration cut used in Fig. 8 for the

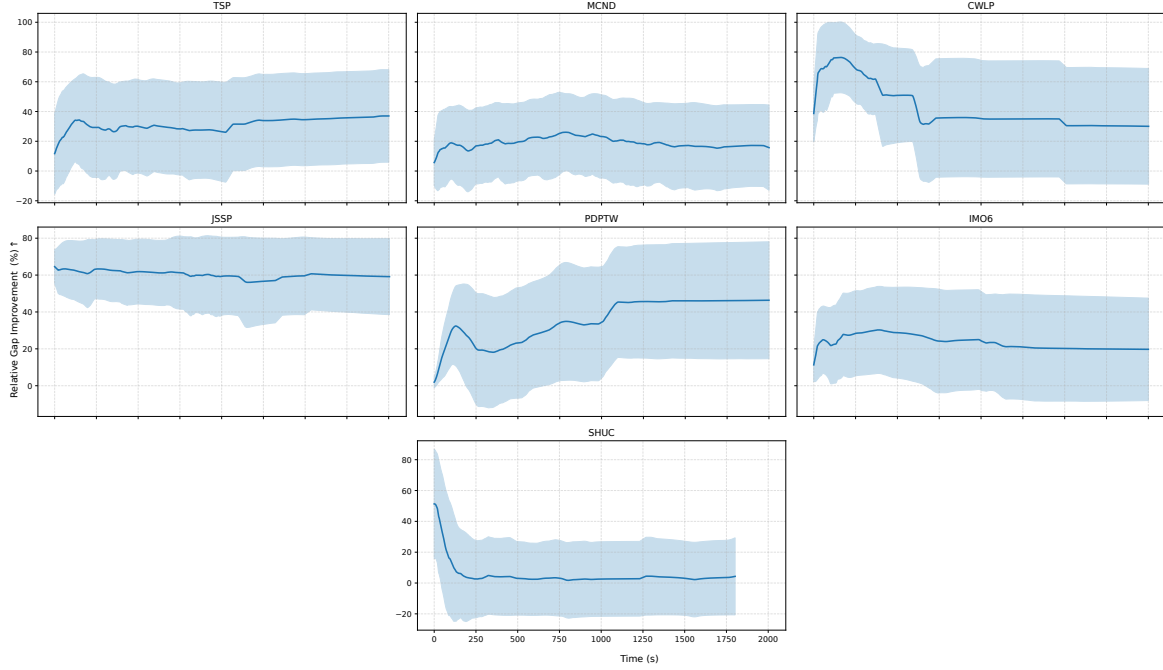


Figure 7. Time evolution (up to 2000 s) of the mean relative gap improvement $\bar{\Delta}_g(t)$ after adding the best EVOCUT cut to the baseline MILP for TSP, MCND, CWLP, JSSP, PDPTW, IMO6, and SHUC. The solid line is the mean across test instances and the shaded band indicates one standard deviation (σ). Positive values mean the augmented model exhibits a smaller optimality gap at that wall clock time.

TSP model is described in F.

E.5. Best Bound Trajectories for Representative TSP Instances

Fig. 9 complements the gap results of Section 5.2 by plotting the solver 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 best EVOCUT acceleration cut. Solver logs are sampled when it 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.6. Primal-Dual Integral (PDI) Trajectories for Representative TSP Instances

Where Fig. 8 focused on the (instantaneous) gap and Fig. 9 on the solve, reported best lower bound, 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.² As before, the blue curve shows the baseline MILP. The orange curve adds the best EVOCUT cut. Across all four cases, the model augmented by EVOCUT reduces the total PDI by roughly 75–95%, demonstrating that adding acceleration cuts is enough to deliver faster, and ultimately better, overall search progress than the handcrafted baseline.

E.7. MCND failure cases with slight gap increase

In a small number of MCND instances, the added cut does not help. Fig. 11 shows four failure cases to see how it fails. The gap curves have a similar shape, but the model with the cut stays slightly higher at most times. This shows that the larger model can slightly slow gap reduction on some instances.

²PDI is the time integral of the optimality gap, hence lower curves are better and earlier plateauing indicates faster convergence.

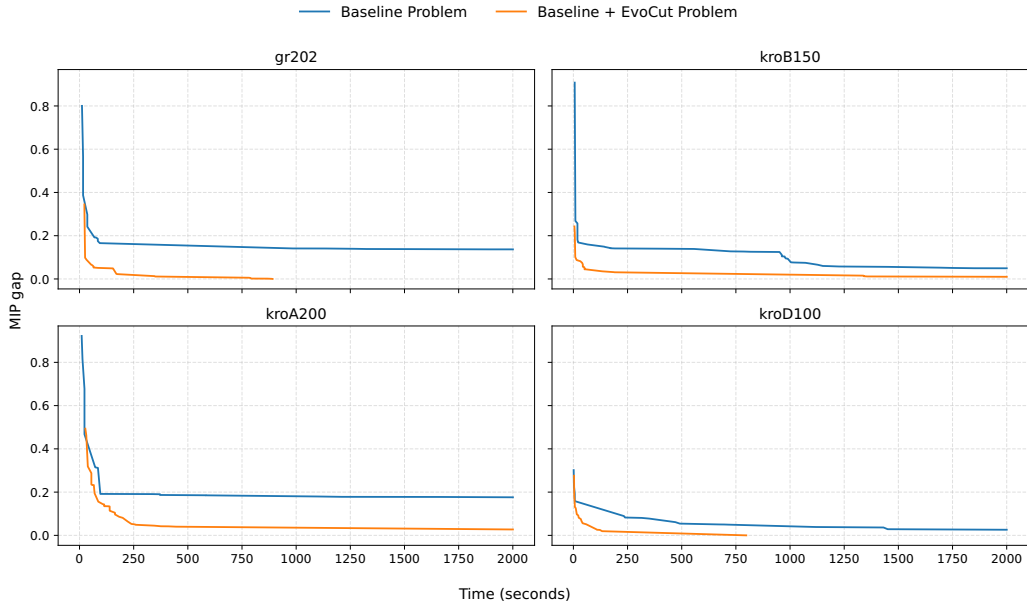


Figure 8. Gap trajectories over time for four representative TSP instances (gr202, kroB150, kroA200, and kroD100) from (Reinelt, 1991). Orange: baseline MILP augmented with the best EVOCUT acceleration cut. Blue: baseline MILP.

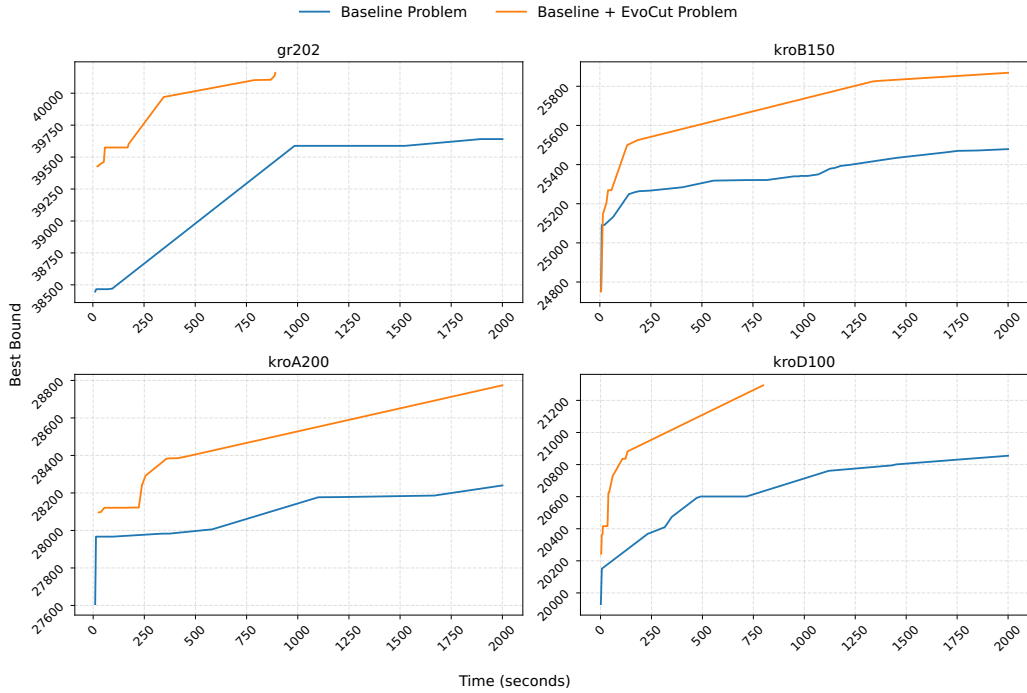


Figure 9. Evolution of the solver's best bound for four representative TSP instances. Blue lines: baseline MILP (reference). Orange lines: MILP with the strongest EVOCUT acceleration cut.

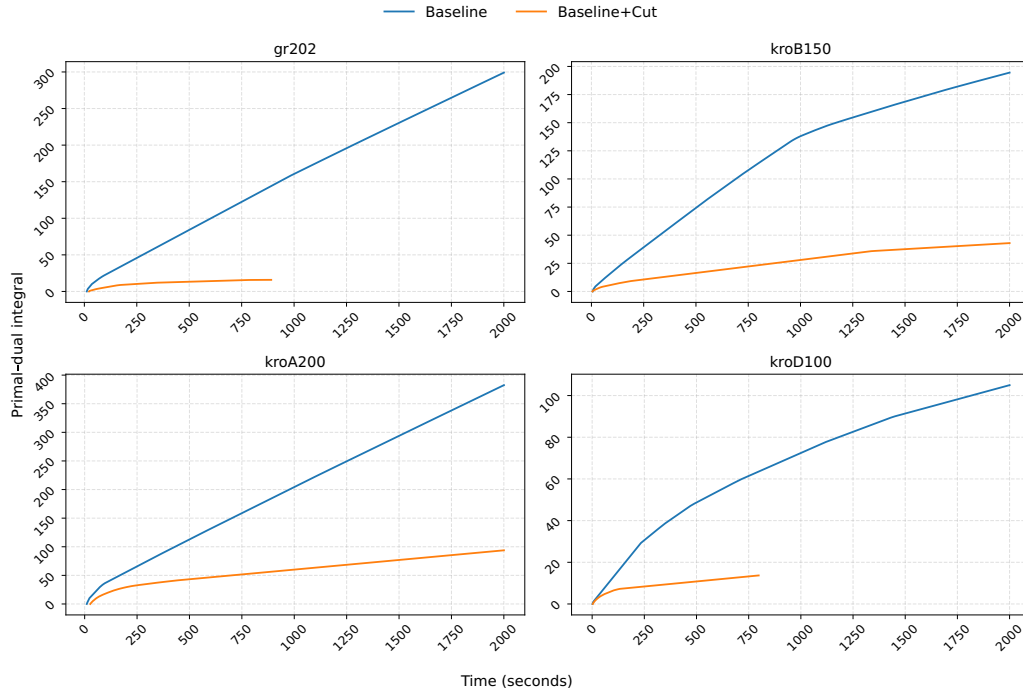


Figure 10. PDI trajectories for the same four TSPLIB instances used in Figs. 8-9. Lower curves are better. Orange: baseline + strongest EVOCUT cut. Blue: baseline MILP.

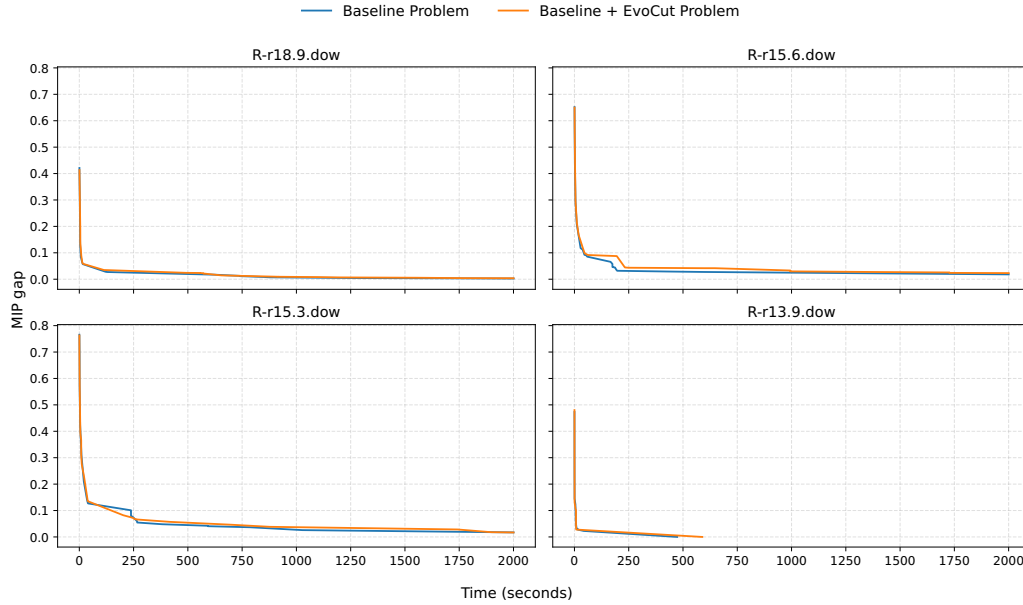


Figure 11. Gap trajectories for four MCND instances where the added cut slightly hurts the gap (R-r18.9.dow, R-r15.6.dow, R-r15.3.dow, and R-r13.9.dow). Orange: baseline MILP augmented with the best EVOCUT cut. Blue: baseline MILP.

Table 5. Gemini Pro 3 strongest cut: test $\bar{\Delta}_g$ (%) with Gurobi vs. HiGHS (mean $\pm \sigma$).

Problem	Solver	5 s	10 s	50 s	150 s	300 s	1000 s	2000 s
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)
	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)
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)
	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)
CWLTP	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)
	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)
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)
	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)
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)
	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)
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)
	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)
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)
	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)

 Table 6. Gemini Pro 3 strongest cut: test $\bar{\Delta}_g$ (%) with Gurobi internal cuts disabled vs. default (mean $\pm \sigma$).

Problem	Solver	5 s	10 s	50 s	150 s	300 s	1000 s	2000 s
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)
	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)
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)
	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)
CWLTP	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)
	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)
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)
	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)
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)
	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)
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)
	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)
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)
	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)

F. Benchmark MILP Problems and Best EVO CUT Cuts

F.1. Traveling Salesman Problem

TSP aims to find the shortest cycle in a graph that visits every node exactly once. It is a classic NP-hard combinatorial optimization problem (Laporte, 1992) that admits a pure MILP model. For our experiments, we used instances from TSPLIB (Reinelt, 1991), a widely used benchmark, together with the Miller-Tucker-Zemlin (MTZ) formulation in Eq. (MTZ).

Compact MILP formulation for TSP.

$$\left\{ \begin{array}{ll} \min & \sum_{(i,j) \in A} c_{ij} x_{ij} \\ \text{s.t.} & \sum_{j \in V \setminus \{i\}} x_{ij} = 1 \quad \forall i \in V \\ & \sum_{i \in V \setminus \{j\}} x_{ij} = 1 \quad \forall j \in V \\ & u_i - u_j + n x_{ij} \leq n - 1 \quad \forall i, j \in V \setminus \{1\}, i \neq j \\ & x_{ij} \in \{0, 1\} \quad \forall (i, j) \in A \\ & 1 \leq u_i \leq n \quad \forall i \in V, \quad u_1 = 1 \end{array} \right. \quad (\text{MTZ})$$

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.

Inequalities (EC) tighten the LP relaxation of Eq. (MTZ) by combining elimination of short cycles, a lifted Desrochers and Laporte strengthening of MTZ ordering, and envelope bounds on the MTZ position variables anchored at the depot. This set of inequalities was the single most effective cut selected by EVOCUT according to our fitness metric based on gap reduction.

Strongest EVOCUT acceleration cut.

$$\left\{ \begin{array}{ll} x_{ij} + x_{ji} \leq 1 & \forall i, j \in V, i < j \\ x_{1i} + x_{i1} + x_{1j} + x_{j1} + x_{ij} + x_{ji} \leq 2 & \forall i, j \in V \setminus \{1\}, i < j \\ u_i - u_j + (n-1)x_{ij} + (n-3)x_{ji} \leq n-2 & \forall i, j \in V \setminus \{1\}, i \neq j \\ u_i \geq 3 - x_{1i} + (n-3)x_{i1} & \forall i \in V \setminus \{1\} \\ u_i \leq (n-1) + x_{i1} - (n-3)x_{1i} & \forall i \in V \setminus \{1\} \end{array} \right. \quad (\text{EC})$$

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 publicly available set of MCND instances (group **R** from the CommaLAB dataset) (CommaLAB, 2021), which provides a range of network sizes, cost structures, and capacity tightness scenarios. We used the MILP formulation in Eq. (MCND).

Compact MILP formulation for MCND.

$$\left\{ \begin{array}{ll} \min & \sum_{(i,j) \in A} \sum_{k \in K} c_{ij} x_{ijk} + \sum_{(i,j) \in A} f_{ij} y_{ij} \\ \text{s.t.} & \sum_{j: (i,j) \in A} x_{ijk} = d_k \quad \forall k \in K, i = O_k \\ & \sum_{j: (j,i) \in A} x_{jik} = d_k \quad \forall k \in K, i = D_k \\ & \sum_{j: (i,j) \in A} x_{ijk} - \sum_{j: (j,i) \in A} x_{jik} = 0 \quad \forall k \in K, i \in N \setminus \{O_k, D_k\} \\ & \sum_{k \in K} x_{ijk} \leq u_{ij} y_{ij} \quad \forall (i,j) \in A \\ & x_{ijk} \geq 0 \quad \forall (i,j) \in A, k \in K \\ & y_{ij} \in \{0, 1\} \quad \forall (i,j) \in A \end{array} \right. \quad (\text{MCND})$$

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) .
- $x_{ijk} \geq 0$ (variable): flow of commodity k on arc (i, j) .
- $y_{ij} \in \{0, 1\}$ (variable): equals 1 iff arc (i, j) is activated.

Strongest EVOCUT acceleration cut.

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

Here, for any node subset $S \subseteq N$, $\delta^+(S) := \{(i, j) \in A : i \in S, j \notin S\}$ 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 any bundle $B \subseteq K(S)$, let $D_B := \sum_{k \in B} d_k$. The first inequality in (EC) is a knapsack-cover strengthening of the standard cut-set capacity constraint. The truncation $\min\{u_{ij}, D_B\}$ prevents the LP relaxation from satisfying the cut using tiny fractional y_{ij} on very high-capacity arcs. The second inequality 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 . In our implementation, EVOCUT instantiates a small subset of these inequalities by expanding concentric breadth-first search zones from commodity origins and sampling small bundles B .

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 $\mathbf{rx.y}$, where x determines the network size and number of commodities, and y encodes the fixed cost and capacity regime. Table 7 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 7. CommaLAB **R** MCND instance sizes by x in $\mathbf{rx.y}$.

The fixed cost and capacity configuration is governed by the second index y in $\mathbf{rx.y}$. 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 is subjected to capacity constraints at each facility, with the goal of minimizing total fixed opening and transportation costs. We used a publicly available set (Beasley, 1990) of CWLP instances. For our experiments, we use the formulation in Eq. (CWLP).

Compact MILP formulation.

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

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).

Strongest EVOCUT acceleration cut. The most effective cut we found for CWLP 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 sharply limit how many large customers can be assigned to a single facility, together with strong links between assignment and opening.

$$\left\{ \begin{array}{ll} \sum_{j \in J} u_j y_j \geq \sum_{i \in I} d_i \\ \sum_{j \in J} \left\lfloor \frac{u_j}{v} \right\rfloor y_j \geq |\{i \in I : d_i \geq v\}| \quad \forall v \in \mathcal{V} \\ x_{ij} \leq y_j \quad \forall i \in I, \forall j \in J \\ x_{ij} = 0 \quad \forall i \in I, \forall j \in J \text{ with } d_i > u_j \\ \sum_{i \in C_j} x_{ij} \leq y_j \quad \forall j \in J \\ \sum_{i \in I : d_i > u_j/3} x_{ij} \leq 2 y_j \quad \forall j \in J \end{array} \right. \quad (\text{EC})$$

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 (Beasley, 1990) (100 facility locations and 1000 customers).

F.4. Job Shop Scheduling Problem (JSSP)

The JSSP is a classic NP-hard combinatorial optimization problem, typically formulated as an MILP (Manne, 1960). It involves scheduling a set of jobs on multiple machines, where each job comprises a sequence of operations that must be

processed in a specified order on designated machines. The objective is to minimize the makespan (the completion time of the last operation), ensuring that each machine handles at most one operation at a time. We use the formulation in Eq. (JSSP) and test EVOCUT on the widely used Taillard benchmark (Taillard, 1993).

Compact MILP formulation.

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

Variable and set glossary.

- n_{mach} (parameter): number of machines (and operations per job), with $k \in \{1, \dots, n_{\text{mach}}\}$.
- $m \in \{1, \dots, n_{\text{mach}}\}$ (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 $((j_1, k_1), (j_2, k_2))$ 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 not selected sequencing inequality.

Strongest EVOCUT acceleration cut.

$$C_{\max} \geq \max \left\{ \frac{1}{n_{\text{mach}}} \sum_{(j,k) \in O} p_{j,k}, \max_{m \in \{1, \dots, n_{\text{mach}}\}} (\text{CP}_m), \max_j \sum_{k=1}^{n_{\text{mach}}} p_{j,k} \right\} \quad (\text{EC})$$

This hybrid inequality 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, ensuring that the makespan is at least the average workload.
- **Machine level critical path bound** (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)

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

Then we take

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

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 collectively tighten the LP relaxation by incorporating machine workloads and job chain precedence (heads/tails). The hybrid inequality was identified as the most effective cut by EVO CUT based on optimality gap reduction.

Benchmark sizes. We evaluated EVO CUT 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 $\times$ 15 machines (e.g., ta31-ta40),
- 30 jobs $\times$ 20 machines (e.g., ta41-ta50),
- 50 jobs $\times$ 15 machines (e.g., ta51-ta60),
- 50 jobs $\times$ 20 machines (e.g., ta61-ta70),
- 100 jobs $\times$ 20 machines (e.g., ta71-ta80).

These instances are widely used in the literature to assess the performance of scheduling algorithms on problems of varying complexity.

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 is a real world, constraint rich problem, and some of the concrete instances used in our benchmark suite are included in MIPLIB (Koch et al., 2011). Here we use the truckload PDPTW instances derived from a real world drayage case at the Port of Rotterdam (Srouf et al., 2010). We download the compressed MPS (Mathematical Programming System) files from the MIPLIB 2010 contribution page (e.g., `wget -q https://miplib2010.zib.de/contrib/submission2010/f_jordan_srouf/R0_1.mps.gz`) (Srouf et al., 2010), and convert them into our parameterized model by reading each MPS file in Gurobi (Gurobi Optimization LLC, 2024) and recovering $(K, N, d_{0i}^k, d_{ij}, d_{iH}^k, v^k, \tau_i^-, \tau_i^+)$ from the instance coefficients.

Compact MILP formulation. We use the MILP formulation of (Srouf 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} :

$$\left\{ \begin{array}{ll} \min & \sum_{k=1}^K \sum_{i=1}^N d_{0i}^k x_{k, K+i} + \sum_{i=1}^N \sum_{j=1}^N d_{ij} x_{K+i, K+j} + \sum_{i=1}^N \sum_{k=1}^K d_{iH}^k x_{K+i, k} \\ \text{s.t.} & \sum_{v=1}^{K+N} x_{uv} = 1 \quad \forall u = 1, \dots, K+N \\ & \sum_{v=1}^{K+N} x_{vu} = 1 \quad \forall u = 1, \dots, K+N \\ & \delta_i - \sum_{k=1}^K (d_{0i}^k + v^k) x_{k, K+i} \geq 0 \quad \forall i = 1, \dots, N \\ & \delta_j - \delta_i - M x_{K+i, K+j} + (d_{ii} + d_{ij}) x_{K+i, K+i} \geq d_{ii} + d_{ij} - M \quad \forall i, j = 1, \dots, N \\ & \tau_i^- \leq \delta_i \leq \tau_i^+ \quad \forall i = 1, \dots, N \\ & \delta_i \in \mathbb{R}_{\geq 0} \quad \forall i = 1, \dots, N \\ & x_{uv} \in \{0, 1\} \quad \forall u, v = 1, \dots, K+N \end{array} \right. \quad (\text{PDPTW})$$

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_{0i}^k (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, $M = 2 \max_{i,j} d_{ij}$ (Srou 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.

Strongest EVOCut acceleration cut. Our strongest PDPTW cut is an integrated temporal bundle (implemented as `additional_cut` in our Pyomo code) that tightens the model using time window logic. Let $\text{EST}_i := \max\{\tau_i^-, \min_k(v^k + d_{0i}^k)\}$ and $\mathcal{A}^- := \{(i, j) : \text{EST}_i + d_{ii} + d_{ij} > \tau_j^+\}$. We then add the following four cuts:

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

Here, $\mathcal{F}_2 := \{(i, j) : i \neq j, (i, j) \notin \mathcal{A}^-, (j, i) \notin \mathcal{A}^-\}$ are mutually feasible pairs. $\mathcal{C}$ is the set of cliques with $|C| > |K(C)|$ in the conflict graph that connects pairs with both directions infeasible, where $K(C) := \{k \in \{1, \dots, K\} : \exists i \in C \text{ s.t. } v^k + d_{0i}^k \leq \tau_i^+\}$ are the trucks that can reach at least one job in C by its latest pickup time. $\mathcal{Q}$ is the set of triples (i, k, j) with $(i, k) \notin \mathcal{A}^-, (k, j) \notin \mathcal{A}^-$, and $\max\{\text{EST}_k, \text{EST}_i + d_{ii} + d_{ik}\} + d_{kk} + d_{kj} > \tau_j^+$.

Benchmark sizes. We use the five MIPLIB truckload PDPTW groups R0, R25, R50, R75, and R100 (Srou 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).

E.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 (Art of Problem Solving Wiki, 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 cuts 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 $I = \{(a, b) \in C \times C : a \leq b\}$ the set of horizontal strips. Binary variables: h_{ij} indicate the hole position, x_i^{ab} indicate that strip (a, b) is active on row i . $s_i^{ab}, t_j^{ab} \in \{0, 1\}$ are the start and end markers of the rows where strip (a, b) is active. The objective is to count the number of rectangles, since 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.

$$\left\{ \begin{array}{ll} \min \sum_{i \in R} \sum_{(a,b) \in I} s_i^{ab} & \\ \text{s.t. } \sum_{j \in C} h_{ij} = 1 & \forall i \in R \\ & \text{(one hole per row)} \\ \sum_{i \in R} h_{ij} = 1 & \forall j \in C \\ & \text{(one hole per column)} \\ \sum_{\substack{(a,b) \in I \\ a \leq j \leq b}} x_i^{ab} + h_{ij} = 1 & \forall i \in R, \forall j \in C \\ & \text{(each cell covered at most once either by a hole or stripe)} \\ x_1^{ab} - s_1^{ab} = 0 & \forall (a, b) \in I \\ & \text{(top flow)} \\ x_i^{ab} - x_{i-1}^{ab} - s_i^{ab} + t_{i-1}^{ab} = 0 & \forall i = 2, \dots, N, \forall (a, b) \in I \\ & \text{(mid flow)} \\ x_N^{ab} - t_N^{ab} = 0 & \forall (a, b) \in I \\ & \text{(bottom flow)} \\ h_{ij} \in \{0, 1\} & \forall i \in R, \forall j \in C \\ x_i^{ab}, s_i^{ab}, t_i^{ab} \in \{0, 1\} & \forall i \in R, \forall (a, b) \in I \end{array} \right. \quad (\text{RT-2DFlow})$$

Notation glossary.

- $R = \{1, \dots, N\}$, $C = \{1, \dots, N\}$: row and column index sets ($i \in R, j \in C$).
- $I = \{(a, b) \in C^2 : a \leq b\}$: 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, 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 $\sum_{i, (a,b)} s_i^{ab}$.

Strongest EVOCUT acceleration cut. The best performing cut enforces mandatory new interval starts when hole movement disrupts coverage continuity across rows. In particular, if a column is vacated by the hole or if the hole appears inside a previously connected interval, then the affected covered columns must initiate new rectangles in the next row.

$$\left\{ \begin{array}{ll} \sum_{\substack{(a,b) \in I \\ a \leq j \leq b}} s_i^{ab} \geq h_{i-1,j} - h_{ij} & \forall i \in \{2, \dots, N\}, \forall j \in C \\ \sum_{\substack{(a,b) \in I \\ a \leq j \leq b}} s_i^{ab} \geq \sum_{\substack{(a,b) \in I \\ a \leq \min\{j,k\} \\ b \geq \max\{j,k\}}} x_{i-1}^{ab} + h_{ik} - 1 & \forall i \in \{2, \dots, N\}, \forall j, k \in C, j \neq k \end{array} \right. \quad (\text{EC-RT-Disrupt})$$

The first inequality is the vacated column cut: if the hole leaves column j from row $i-1$ to row i , then column j must be covered by an interval that starts at row i . The second is the broken interval cut: if columns j and k were covered by the same interval on row $i-1$ and the hole appears at k on row i , then coverage at j must start anew at row i .

Benchmark sizes. We generate one instance per $N \in \{3, \dots, 46\}$ (44 instances total). We randomly sample 10 instances for evaluation and verification during EVOCUT, and use the remaining 34 instances as the held out test set.

E.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 practically motivated problem with many constraints.

Some of the concrete instances used in our benchmark 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 Pyomo formulation with piecewise production costs, startup categories, ramping limits, and minimum up and down times (Ostrowski et al., 2011).

Compact MILP formulation.

$$\begin{cases}
 \min \sum_{g \in G} \sum_{t \in \mathcal{T}} (c_{g,t} + C_{g,1} u_{g,t}) + \sum_{g \in G} \sum_{s \in S_g} \sum_{t \in \mathcal{T}} C_{g,s}^{su} d_{g,s,t} \\
 \text{s.t.} \sum_{g \in G} (p_{g,t} + P_g^{\min} u_{g,t}) + \sum_{w \in W} p_{w,t}^w = L_t & \forall t \in \mathcal{T} \\
 \sum_{g \in G} r_{g,t} \geq R_t & \forall t \in \mathcal{T} \\
 u_{g,t} - u_{g,t-1} = v_{g,t} - w_{g,t} & \forall g \in G, t = 2, \dots, T \\
 \sum_{\tau=t-U_g+1}^t v_{g,\tau} \leq u_{g,t} & \forall g \in G, t \geq U_g \\
 \sum_{\tau=t-D_g+1}^t w_{g,\tau} \leq 1 - u_{g,t} & \forall g \in G, t \geq D_g \\
 v_{g,t} = \sum_{s \in S_g} d_{g,s,t} & \forall g \in G, t \in \mathcal{T} \\
 d_{g,s,t} \leq \sum_{i=\ell_{g,s}}^{\ell_{g,s+1}-1} w_{g,t-i} & \forall g \in G, s < |S_g|, t \geq \ell_{g,s+1} \\
 u_{g,t} \geq MR_g & \forall g \in G, t \in \mathcal{T} \\
 p_{g,t} + r_{g,t} \leq (P_g^{\max} - P_g^{\min}) u_{g,t} - \max(P_g^{\max} - SU_g, 0) v_{g,t} & \forall g \in G, t \in \mathcal{T} \\
 p_{g,t} + r_{g,t} \leq (P_g^{\max} - P_g^{\min}) u_{g,t} - \max(P_g^{\max} - SD_g, 0) w_{g,t+1} & \forall g \in G, t < T \\
 p_{g,t} + r_{g,t} - p_{g,t-1} \leq RU_g & \forall g \in G, t = 2, \dots, T \\
 p_{g,t-1} - p_{g,t} \leq RD_g & \forall g \in G, t = 2, \dots, T \\
 p_{g,t} = \sum_{l \in L_g} (P_{g,l} - P_{g,1}) \lambda_{g,l,t} & \forall g \in G, t \in \mathcal{T} \\
 c_{g,t} = \sum_{l \in L_g} (C_{g,l} - C_{g,1}) \lambda_{g,l,t} & \forall g \in G, t \in \mathcal{T} \\
 u_{g,t} = \sum_{l \in L_g} \lambda_{g,l,t} & \forall g \in G, t \in \mathcal{T} \\
 P_{w,t}^{w,\min} \leq p_{w,t}^w \leq P_{w,t}^{w,\max} & \forall w \in W, t \in \mathcal{T} \\
 u_{g,t}, v_{g,t}, w_{g,t}, d_{g,s,t} \in \{0, 1\}, \lambda_{g,l,t} \in [0, 1] \\
 p_{g,t}, r_{g,t}, c_{g,t}, p_{w,t}^w \geq 0
 \end{cases} \quad (\text{SHUC})$$

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 $C_{g,s}^{su}$.
- 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.

Strongest EVOCUT acceleration cut.

$$\begin{cases}
 b_{g,t} \leq v_{g,t} & \forall g \in G, t < T \\
 b_{g,t} \leq w_{g,t+1} & \forall g \in G, t < T \\
 b_{g,t} \geq v_{g,t} + w_{g,t+1} - 1 & \forall g \in G, t < T \\
 \bar{P}_{g,t} \leq P_g^{\max} u_{g,t} - \max(P_g^{\max} - SU_g, 0) v_{g,t} & \forall g \in G, t \in \mathcal{T} \\
 \bar{P}_{g,t} \leq P_g^{\max} u_{g,t} - \max(P_g^{\max} - SD_g, 0) w_{g,t+1} & \forall g \in G, t < T \\
 \bar{P}_{g,t} \leq P_g^{\max} u_{g,t} - \max(P_g^{\max} - SU_g, 0) v_{g,t} - \max(P_g^{\max} - SD_g, 0) w_{g,t+1} \\
 \quad + \min(\max(P_g^{\max} - SU_g, 0), \max(P_g^{\max} - SD_g, 0)) b_{g,t} & \forall g \in G, t < T \\
 \bar{P}_{g,t} \leq P_g^{\min} u_{g,t} + p_{g,t-1} + RU_g + (P_g^{\max} - P_g^{\min})(1 - u_{g,t-1}) & \forall g \in G, t = 2, \dots, T \\
 \bar{P}_{g,t} \leq P_g^{\min} u_{g,t} + (P_g^{\max} - P_g^{\min}) u_{g,t-1} - \max(P_g^{\max} - SU_g, 0) v_{g,t-1} + RU_g & \forall g \in G, t = 2, \dots, T \\
 \bar{P}_{g,t} \leq P_g^{\min} u_{g,t} + (P_g^{\max} - P_g^{\min}) u_{g,t-1} - \max(P_g^{\max} - SD_g, 0) w_{g,t} + RU_g & \forall g \in G, t = 2, \dots, T \\
 L_t - \sum_{w \in W} p_{w,t}^w + R_t \leq \sum_{g \in G} \bar{P}_{g,t} & \forall t \in \mathcal{T}
 \end{cases} \quad (\text{EC})$$

This cut enforces 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. Note that $\bar{P}_{g,t}$ and $b_{g,t}$: auxiliary variables introduced by the strongest cut.

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 $|D_e| = 10$ and $|D_v| = 2$ from our synthetic generator, and use the full 56-instance PGLib-UC benchmark as D_t . We do not split SHUC into small, medium, and large size tiers.

G. LLM API Configuration

G.1. Parameters and token usage

Table 8 lists the exact parameters used in every call to the three LLM APIs (*deepseek-reasoner*, *gemini-pro-3*, and *gpt-5.1*) throughout our EVOCUT pipeline, together with approximate token usage.

Parameter	DeepSeek	Gemini Pro 3	GPT-5.1
Model	deepseek-reasoner	gemini-pro-3	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 8. LLM API settings and estimated token costs used for all EVOCUT experiments (Jan 2026 OpenRouter list prices).

H. Fitness definition

We assume $\text{gap}_{\text{ref}}(i) > 0$ for all $i \in D_e$ (as ensured by the construction of D_e in Section 4). Given the signed relative gap change $d(i) = \frac{\text{gap}_{\text{cut}}(i) - \text{gap}_{\text{ref}}(i)}{\text{gap}_{\text{ref}}(i)}$, its mean over D_e , $\bar{d} = \frac{1}{|D_e|} \sum_{i \in D_e} d(i)$, is negative when C is beneficial and positive when it is harmful. We map $\bar{d}$ to a fitness score via $\text{Fit}(C) = 10 e^{-\bar{d}}$, so larger (more negative) gap reductions yield exponentially higher fitness.