HINTS: GENERATING MIP WARM STARTS VIA AN EVOLVING EXPERIENCE LIBRARY

Anonymous authors
Paper under double-blind review

ABSTRACT

Mixed-integer programming (MIP) solvers provide a general framework for combinatorial optimization, but their performance can depend heavily on how quickly they find a strong incumbent—the best feasible solution found so far. Specialized heuristics can produce good solutions quickly, yet they lack the systematic search and optimality guarantees of MIP solvers. We ask whether LLM-generated heuristics can combine these strengths by providing better starting solutions while leaving the solver unchanged. We propose **HINTS** (**H**euristic **IN**jection of **T**ight **S**tarts), which uses an LLM to generate a construction heuristic that produces a feasible MIP start for each instance. The start is injected into an unchanged solver, which then improves it through its native branch-and-bound search. To adapt construction strategies across heterogeneous instances, HINTS maintains a growing library of scoped experiences that record when a construction applies and how to build the corresponding start. Experiences are evaluated by the solver's progress after injection rather than by the heuristic solution alone. Across eight CO-Bench tasks and 1,611 instances, HINTS reduces the mean MIP gap across three solvers from 0.375 to 0.172 and consistently outperforms fixed textbook warm starts. With HINTS, OR-Tools further outperforms solver-only COPT and approaches solver-only Gurobi. These results show that LLM-generated warm starts can improve MIP solving without modifying the solver itself.

1 INTRODUCTION

Combinatorial optimization problems arise in scheduling, routing, packing, and many other operational settings (Wolsey, 2020; Bengio et al., 2021). Mixed-integer programming (MIP) provides a general framework for solving such problems, with modern solvers using branch-and-bound and related techniques to search for high-quality solutions and certify optimality (Land & Doig, 1960). Their performance, however, can depend strongly on how quickly a good incumbent is found. Modern MIP solvers already employ internal primal heuristics to discover feasible solutions during search, but they can also accept externally constructed solutions as MIP starts. Specialized heuristics offer a complementary strength: methods such as first-fit decreasing for bin packing (Coffman et al., 2013), NEH for flow shop (Fernández-Viagas et al., 2017), and nearest-neighbour with local search for TSP (Johnson & McGeoch, 1997) can exploit problem structure to quickly construct high-quality feasible solutions. However, these heuristics typically return a solution without the systematic search and optimality certification provided by MIP solvers. This complementarity motivates a natural question: *can a specialized heuristic construct a strong MIP start, while leaving subsequent search and proof to the solver?*

Large language models (LLMs) make this interface particularly attractive because they can generate executable heuristics from problem descriptions and feedback (Romera-Paredes et al., 2024; Liu et al., 2024). Much of LLM-based algorithm design focuses on generating or evolving stand-alone heuristics, while other approaches use LLMs to modify solver components or configurations (Zhao et al., 2026; Yu et al., 2026). We instead consider a simpler interface: an LLM generates a construction heuristic whose solution is supplied as a MIP start, while the mathematical model and solver remain unchanged. The goal is therefore not merely to produce a good heuristic solution in isolation, but to construct a start that enables effective subsequent solver progress. This objective also introduces an instance-adaptation challenge: instances within the same problem family can favor

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

different construction strategies, making both a single fixed heuristic and repeated generation from scratch undesirable.

We propose **HINTS** (**H**euristic **IN**jection of **T**ight **S**tarts), an LLM-based framework for generating instance-adaptive MIP warm starts. Given a problem and instance features, HINTS generates a construction heuristic, maps its feasible solution to the fixed MIP model, and injects it into an unchanged solver. The construction is evaluated by the solver's progress after injection—using terminal MIP gap and gap-AUC rather than the heuristic objective alone. To accumulate useful construction knowledge across instances, HINTS maintains a growing experience library in which each experience records *when* a construction applies and *how* to build the corresponding start. For each new instance, a lightweight router selects an applicable experience or generates a construction without stored experience; after observing solver progress, HINTS creates or updates experiences as needed. The resulting library captures distinct construction strategies for different instance regimes without forcing an entire problem family into a single heuristic.

We evaluate HINTS on eight CO-Bench tasks comprising 1,611 instances with Gurobi, COPT, and OR-Tools. Across the three solvers, HINTS reduces the mean terminal MIP gap from 0.375 for solver-only runs to 0.172, while also consistently improving over fixed textbook warm starts. The benefit is also visible across solver strengths: OR-Tools with HINTS achieves a lower mean terminal gap than solver-only COPT and narrows the difference to solver-only Gurobi. Further analysis reveals two complementary mechanisms: HINTS can provide an already strong incumbent, or a structured start that the solver can substantially improve during subsequent search. Our contributions are threefold:

- We introduce an LLM–MIP cooperation paradigm in which the LLM constructs a warm start while an unchanged MIP solver performs subsequent search and proof.
- We propose HINTS, which combines instance-adaptive warm-start generation with a growing library of scoped construction experiences and evaluates these experiences by solver progress after injection.
- Across eight CO-Bench tasks and three MIP solvers, HINTS consistently improves solution quality and time-to-quality over solver-only runs and fixed heuristic starts, through both stronger initial incumbents and starts that support effective subsequent search.

2 METHOD

We propose HINTS (**H**euristic **IN**jection of **T**ight **S**tarts), where an LLM generates a construction heuristic that produces a feasible MIP start. HINTS supplies this start to the solver, which then performs its native search without modification. A growing experience library supports cross-instance reuse and progressively adapts the construction strategy over the instance stream. Figure 1 summarises the procedure.

2.1 WARM-START INJECTION

HINTS does not replace the MIP solver or modify the mathematical formulation. Each task uses a fixed formulation, instantiated for instance x as

$$M(x) = (X_x, J_x, \mathcal{C}_x), \tag{1}$$

where X_x denotes the decision variables, J_x the objective, and $\mathcal{C}_x$ the constraint set. The instantiated model is passed unchanged to the solver. HINTS supplies only a MIP start: a feasible assignment that the solver can use as an initial incumbent and subsequently improve upon.

Constructing the start. HINTS does not ask the LLM to emit an assignment directly. Instead, it asks the LLM to design a construction heuristic that produces the start. Generating a construction program rather than a one-shot assignment provides an executable mechanism for constructing and validating a structured candidate solution. The prompt is assembled from three pieces: a task description D; the fixed formulation $M(x) = (X_x, J_x, \mathcal{C}_x)$; and a task-specific feature vector $u(x) \in [0,1]^d$ for the current instance (Appendix C). Using D, $M(x)$, and $u(x)$, the LLM writes a deterministic construction heuristic: executable code that returns a structured candidate solution for x. If a stored experience is selected (Section 2.2), its method sketch guides the construction;

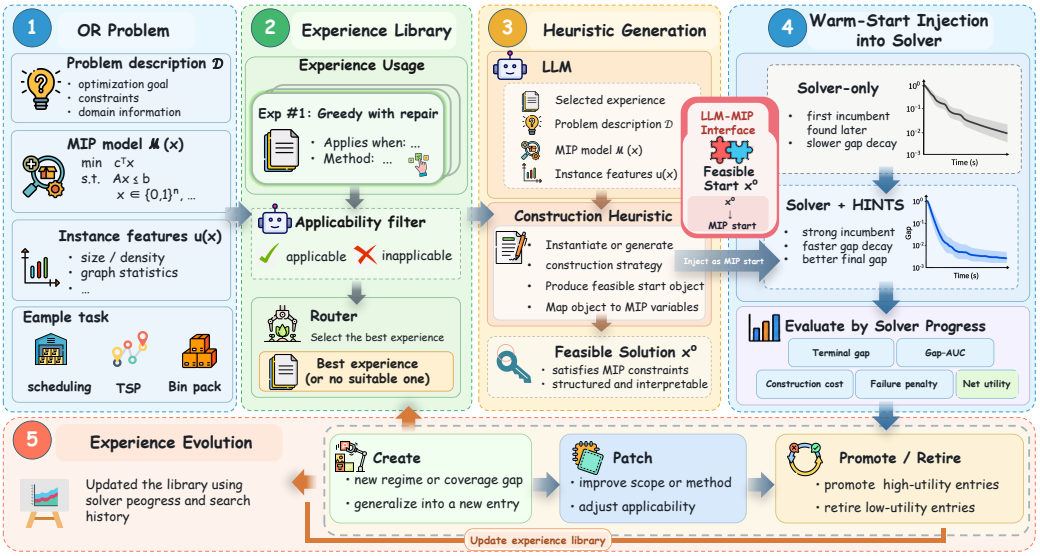

Figure 1: **Overview of HINTS.** For a combinatorial problem, HINTS selects a suitable experience from the library based on the problem type and instance features, then generates a construction heuristic that returns a feasible start. The start is injected into an unchanged solver; subsequent solver progress is observed and scored, and the library is updated after each wave.

otherwise, the LLM constructs a heuristic from the instance and its features alone. The generated code is executed once; if execution fails, the LLM is allowed up to three repair rounds.

Injecting the start. A fixed, task-specific map writes the returned object onto the variables of $M(x)$ and sets the solver's start attributes. For each run, the model is built once and is not modified thereafter. If the mapped assignment is feasible and accepted by the solver, it provides an initial incumbent for subsequent search. If the heuristic fails, the mapping is incomplete, or the solver rejects the start, the solver proceeds without an externally supplied MIP start. Presolve, cuts, branching, primal heuristics, and dual proof remain entirely those of the underlying solver.

Scoring and improvement. After injection, the solver runs for a fixed time limit T. An accepted start provides an initial primal bound and can enable earlier pruning, while the solver's native search continues to seek better feasible solutions. HINTS evaluates the construction by the solver's subsequent progress—the terminal MIP gap and gap-AUC—rather than by the heuristic objective alone. Let $g(t) \in [0, 1]$ denote the clipped MIP gap at time t, and define

$$\mathrm{AUC} = \frac{1}{T} \int_0^T g(t)\, dt. \tag{2}$$

Let t_{ex} be the time spent constructing the start and t_{sol} the subsequent solver time. We use $\tau > 0$ as a small constant that keeps the time ratio well defined, ν as the penalty when $\mathbf{1}_{\mathrm{fail}}$ marks an infeasible or rejected start, and κ as the weight on construction cost. The utilities are

$$B = 1 - \big(\omega\, \mathrm{AUC} + (1-\omega)\, g(T)\big), \quad C = \frac{t_{\mathrm{ex}}}{t_{\mathrm{ex}} + t_{\mathrm{sol}} + \tau} + \nu\, \mathbf{1}_{\mathrm{fail}}, \quad Y = B - \kappa C. \tag{3}$$

Here B rewards both rapid gap reduction and a good terminal gap; C penalises construction time and failed starts; and Y is the resulting net utility. These quantities are recorded for each instance and used to update and retrieve experiences (Section 2.2).

2.2 Experience Library

A single construction strategy need not perform well across a heterogeneous instance stream. HINTS therefore stores reusable experiences, each recording *when* a construction applies and *how* to build

a start that can be passed to the solver. Each problem maintains its own experience library, isolated from those of other problems. An experience is a short template with three fields:

- `<title>`: a short name that distinguishes the construction from others;
- `<applies_when>`: the feature regime in which the construction should be used;
- `<method>`: a natural-language sketch of how to build the start, rather than executable code.

Creating an experience. After each wave, the LLM may propose a new experience to cover a gap: an empty library, a newly observed feature regime, or a method family not covered by existing experiences. A proposal is accepted only when it addresses such a gap. For a non-empty library, the proposal must cite at least three distinct supporting instances from the current wave, preventing a single noisy instance from creating a new experience. Each run has already been scored as in Section 2.1; these utilities provide the observations used for subsequent experience selection and updates.

Using an experience. For each instance, the LLM marks every stored experience as applicable, uncertain, or inapplicable. An experience is marked inapplicable only when its stated scope explicitly contradicts the current instance. Let $\mathcal{A}(x)$ denote the eligible set, consisting of the null action together with all non-retired experiences that are not marked inapplicable. A lightweight contextual router based on Thompson-style posterior sampling (Thompson, 1933; Agrawal & Goyal, 2013) then selects

$$a^\star = \arg\max_{a \in \mathcal{A}(x)} \left(\widetilde{Y}_a(x) + \alpha(n_a) s_a(x) \right), \tag{4}$$

where $\widetilde{Y}_a(x)$ is a posterior draw of the predicted net utility for action a on instance x, $s_a(x)$ is its suitability score for the current instance, n_a is the number of times a has previously been selected, and $\alpha(n_a)$ is a small exploration boost that decays with use. The null action represents generation without a stored experience and has suitability zero. Posterior updates and the remaining details are given in Appendix A.

2.3 Experience update

After each wave, the LLM reads the current library together with a compact wave history—including the selected experience, instance features, observed utilities, and a short excerpt of the generated code—and proposes one of three actions: create, patch, or idle. HINTS accepts these proposals only when they satisfy conservative update gates, so noisy outcomes do not cause unnecessary changes to the library.

Patching an experience. A patch may revise the title, scope, or method sketch of an existing experience, but not all three at once, and each experience may be patched at most once per wave. A proposal that accumulates multi-regime caveats in the scope is rejected in favour of creating a separate experience, while a scope-only patch must remain a light refinement of the same regime. Rejected proposals leave the library unchanged. When a patch is accepted, the posterior associated with the target experience is discounted toward its prior,

$$A \leftarrow A_0 + \gamma(A - A_0), \qquad h \leftarrow h_0 + \gamma(h - h_0), \tag{5}$$

where A_0 and h_0 denote the prior parameters and $\gamma \in (0, 1)$ controls how much previously accumulated evidence is retained. We use a smaller γ when the scope changes than when only the method sketch changes, reflecting the greater uncertainty introduced by modifying where an experience applies. After an accepted patch, the experience returns to pending status. The numerical discount factors are given in Appendix B.

Promotion and retirement. A newly created experience starts in the pending state. It is promoted once it has accumulated enough valid trials, covers more than one feature region, has sufficiently reduced posterior uncertainty, and produces accepted starts reliably. A pending experience is retired after repeated injection failures or after sufficiently many trials without any locally positive region. An active experience is retired after repeated severe failures, or when it is frequently eligible but rarely selected and has negligible posterior probability of being the best available action. These rules encourage coarse library growth: HINTS maintains a small set of experiences that cover distinct instance regimes rather than creating one experience per instance. The promotion and retirement thresholds are specified in Appendix B.

Table 1: **Overall performance of three solvers on eight CO-Bench tasks.** Gap is the terminal MIP gap at 600s, and AUC is the time-normalised integral of the gap. Average is the macro-average over tasks. Lower is better. Best results are highlighted in **bold**, and second-best values are underlined.

		Bin (160)		CDD (70)		Cont. (762)		Flow (120)		Job (80)		Open (60)		Set (87)		TSP (272)		Average	
Solver	Method	Gap	AUC	Gap	AUC	Gap	AUC	Gap	AUC	Gap	AUC	Gap	AUC	Gap	AUC	Gap	AUC	Gap	AUC
Gurobi	Baseline	.015	.032	.359	.447	.445	.496	.225	.285	.401	.481	.182	.229	.035	.055	.058	.117	.215	.268
	Heuristic-1	.010	.025	.252	.337	.387	.438	.123	.149	.393	.467	.176	.213	**.031**	.047	.028	.055	.175	.216
	Heuristic-2	.010	.025	.183	.233	.481	.522	.127	.158	.337	.387	.179	.216	**.031**	.047	.052	.087	.175	.209
	HINTS (ours)	**.003**	**.015**	**.066**	**.075**	**.252**	**.262**	**.113**	**.136**	**.323**	**.386**	**.158**	**.181**	**.031**	**.045**	**.021**	**.038**	**.121**	**.142**
COPT	Baseline	.226	.272	.361	.497	.759	.996	.107	.181	.621	.677	.378	.412	.048	.059	.817	.928	.415	.503
	Heuristic-1	.083	.153	.373	.504	.333	.485	.100	.189	.656	.693	.296	.321	**.044**	.057	.145	.192	.254	.324
	Heuristic-2	.083	.154	.285	.471	.400	.642	.111	.195	.457	.518	.297	.320	.045	.057	.131	.160	.226	.315
	HINTS (ours)	**.064**	**.130**	**.093**	**.351**	**.239**	**.301**	**.076**	**.169**	**.331**	**.373**	**.213**	**.225**	**.044**	**.053**	**.089**	**.111**	**.144**	**.214**
OR-Tools	Baseline	.139	.241	.456	.576	.993	.997	.413	.473	.920	.925	.580	.611	.103	**.119**	.344	.428	.494	.546
	Heuristic-1	.166	.265	.423	.580	.508	.535	.389	.441	.890	.899	.537	.551	.100	.123	.088	.127	.388	.440
	Heuristic-2	.166	.264	.333	.518	.662	.681	.395	.450	.600	.612	.520	.536	.108	.124	.244	.293	.379	.435
	HINTS (ours)	**.104**	**.224**	**.066**	**.367**	**.310**	**.346**	**.377**	**.423**	**.498**	**.507**	**.483**	**.505**	**.099**	.120	**.059**	**.093**	**.250**	**.323**

3 EXPERIMENTS

3.1 EXPERIMENT SETUP

Benchmarks. We evaluate HINTS on eight CO-Bench tasks (Sun et al., 2026) from the OR-Library tradition (Beasley, 1990): one-dimensional bin packing, common due-date scheduling (CDD), container loading, flow shop, job shop, open shop, set covering, and TSP. We retain the harder subset by removing instances that Gurobi solves within 100s, leaving 1,611 instances. We use this filtered collection for evaluation without a train–test split. For each task, the same MIP formulation is used across solvers; the decision variables, constraints, and required solution format are given in Appendix D.

Baselines. We compare HINTS against solver-only runs and fixed construction heuristics.

- **Solver.** We evaluate Gurobi (Gurobi Optimization, LLC, 2024), COPT (Ge et al., 2022), and OR-Tools with the SCIP backend (Google, 2024; Achterberg, 2009). Each engine uses its default parameter configuration. The solver-only baseline runs each engine without an externally supplied MIP start, relying on its native search and heuristics to obtain an incumbent.
- **Heuristic.** For each task, we also evaluate two classical construction heuristics whose feasible solutions are injected as MIP starts before the same solver continues its native search (Table 5). These baselines provide fixed task-specific starts. For bin packing, we use first-fit decreasing and best-fit decreasing; for CDD, shortest-processing-time ordering and the V-shaped construction. The remaining tasks use standard constructions for their respective problem families. Details are provided in Appendix E.

Implementation Details. Each solver run uses a 600s time limit and two threads. Across methods, the MIP formulation, solver configuration, and solver time limit are fixed; only the supplied MIP start differs. Heuristic synthesis and library evolution use DeepSeek-V4-Flash. Experiments were run on an Intel Xeon Silver 4310 at 2.10 GHz with 24 physical cores and 251 GB of memory under CentOS Linux 7. Routing, scoring, and library-update constants are listed in Appendix B. MIP gaps are clipped to $[0, 1]$; runs without a feasible solution are assigned a gap of 1. Gap-AUC is the time-normalised integral of this clipped gap over the 600s horizon.

3.2 MAIN RESULTS

Across all three solvers, HINTS improves both terminal gap and gap-AUC over solver-only runs (Table 1). The average terminal gap drops from 0.215 to 0.121 on Gurobi, from 0.415 to 0.144 on

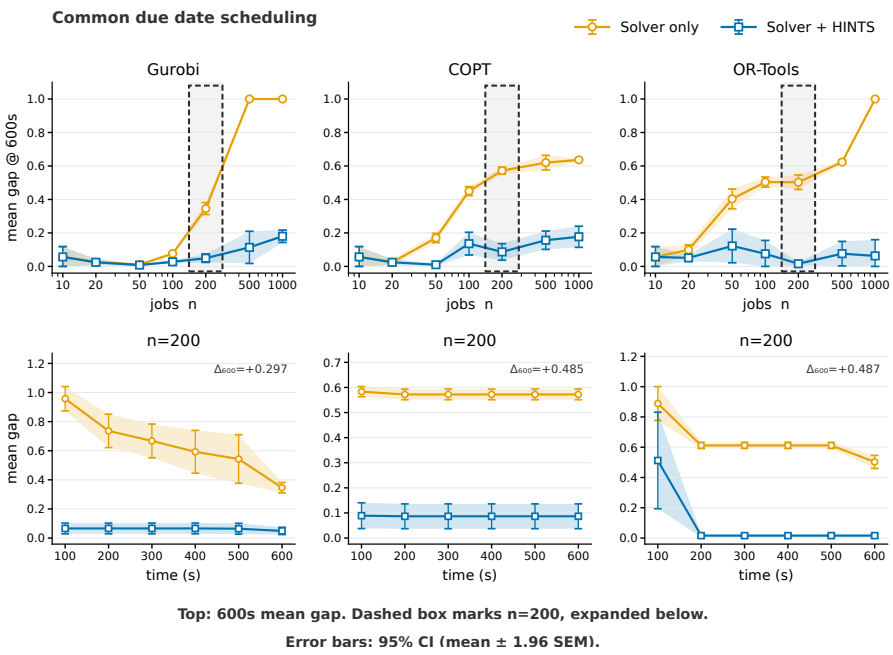

Figure 2: Top: mean CDD MIP gap at 600s versus the number of jobs for each solver. Bottom: 600s gap trajectories at the boxed size n=200. Error bars show 95% confidence intervals.

COPT, and from 0.494 to 0.250 on OR-Tools, corresponding to relative reductions of **44%**, **65%**, and **49%**. Average gap-AUC decreases from 0.268 to 0.142, 0.503 to 0.214, and 0.546 to 0.323, respectively. HINTS also outperforms both fixed heuristic starts in average gap and gap-AUC on every solver, indicating that its gains extend beyond simply supplying a feasible start. The benefit is also visible across solver strengths: OR-Tools with HINTS achieves an average gap of 0.250, below solver-only COPT at 0.415 and close to solver-only Gurobi at 0.215.

When generation and heuristic-execution time are charged against the same 600s end-to-end budget, the gains remain substantial (Appendix F). Under this accounting, HINTS reduces the mean terminal gap relative to the corresponding solver-only baseline by **43%**, **55%**, and **42%** on Gurobi, COPT, and OR-Tools, respectively.

3.3 Analysis of the Initial Solution

Aggregate results do not show *how* a HINTS start helps: it may provide a strong incumbent immediately, support substantial later improvement, or both. Figure 2 examines these effects on CDD, where solver-only performance deteriorates clearly with instance size. At small n, HINTS and solver-only runs perform similarly, but the difference becomes pronounced from n=200. At n=500, solver-only Gurobi fails to find a feasible solution within 600s, while HINTS maintains a mean gap below 0.2. The bottom row shows the full trajectories for n=200.

Better early incumbent. On Gurobi and COPT, HINTS establishes a strong incumbent early in the search: by 100s, the mean gaps are 0.066 and 0.089, compared with 0.957 and 0.583 for the first feasible incumbents found by the solver-only runs. Subsequent improvement is comparatively modest, suggesting that much of the final advantage is established early. Solver-only COPT changes little after 100s, while solver-only Gurobi continues to improve but does not catch the HINTS trajectory within the 600s limit. HINTS therefore primarily helps these solvers establish a strong incumbent early, which is then maintained and refined during the remaining search.

Better structure. OR-Tools shows a different pattern. At n=200, the HINTS run still has a mean gap of 0.512 at 100s, so its early advantage is limited. The solver-only run remains near 0.6 and finishes at 0.503, whereas OR-Tools continues improving from the HINTS start and reaches 0.016

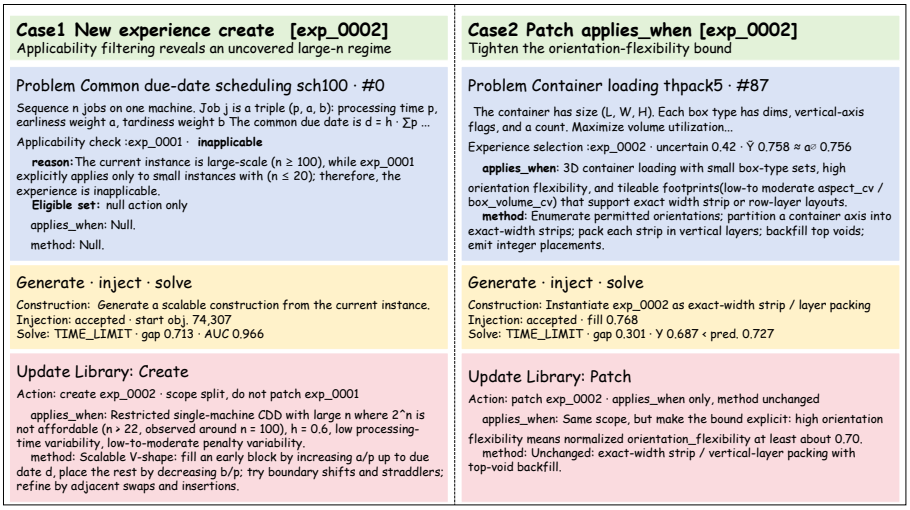

Figure 3: **Illustrative experience updates after observing instance outcomes.** Each card shows the selected experience (σ, μ), the injected start and solver outcome, and the resulting library update.

by 200s. Thus, the value of a start is not limited to its initial quality: some starts are followed by substantially greater progress during subsequent solver search.

Case study. Figure 3 shows how such outcomes feed back into the library. On CDD `sch100#0`, the only stored experience, exp_0001, is an exact subset-DP construction scoped to $n \leq 20$. Because the current instance has $n=100$, the experience is marked inapplicable and HINTS falls back to the null action, generating a new scalable construction. The update then creates exp_0002 for the uncovered large-n regime while retaining exp_0001 for small instances.

On Container `thpack5#87`, HINTS retrieves exp_0002, a strip/layer construction whose scope specifies "high orientation flexibility." The start is accepted, but its observed utility falls below the router's prediction. HINTS therefore narrows the scope by making the orientation-flexibility condition explicit, while leaving the method unchanged. Together, the two cases illustrate the two main update modes: creating an experience for an uncovered regime and refining the scope of an existing one.

3.4 Evolution of the Experience Library

Library composition. Figure 4(a) shows the 59 Gurobi experiences retained across all tasks at the end of the stream. The library remains small relative to the number of instances: TSP and CDD retain two experiences, Bin packing four, while Container retains a larger set. The library therefore grows by feature regime rather than by instance, avoiding both an instance-level log and a single global construction.

Stepwise accumulation. Figure 4(b) traces Bin packing over its 160-instance stream. New experiences appear only at distinct regime changes: the initially empty library around $n \approx 120$, size splits near $n \approx 500$ and $n \approx 1000$, and a later homogeneous regime with CV≈ 0.05. Between these points, library size remains flat, indicating that later instances reuse or lightly patch existing experiences instead of creating new entries.

3.5 Ablation

We ablate experience reuse and routing to isolate their contributions (Table 2). The *w/o experience* variant generates each start independently without storing or retrieving experiences. The *w/o sampling* variant keeps the library but places all eligible experiences in the prompt instead of selecting one through the router.

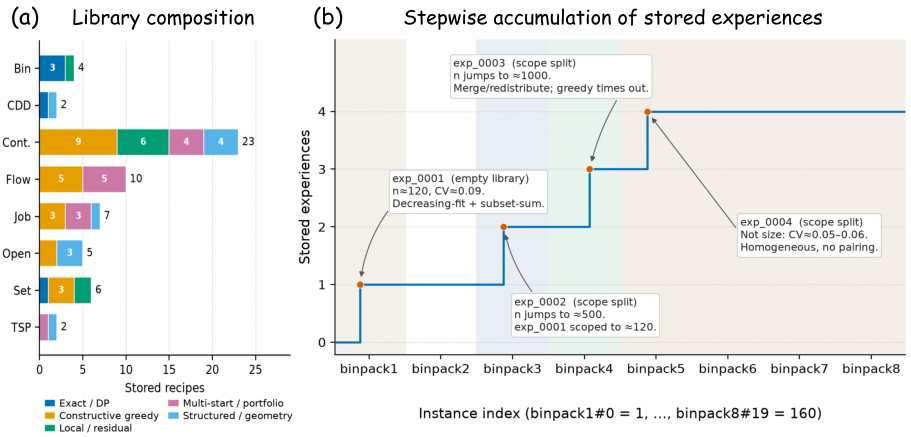

Figure 4: (a) Final number of experiences for each task. (b) Stepwise creation of experiences on Bin packing over the instance stream.

Table 2: **Ablation of the main HINTS components.** Each cell reports Gap / AUC. Average is the macro-average over tasks. Lower is better. Best results are highlighted in **bold**.

Method	Bin	CDD	Cont.	Flow	Job	Open	Set	TSP	Average
HINTS (ours)	**0.003 / 0.015**	**0.066 / 0.075**	**0.252 / 0.262**	**0.113 / 0.136**	0.323 / 0.386	**0.158 / 0.181**	**0.031 / 0.045**	**0.021 / 0.038**	**0.121 / 0.142**
w/o experience	0.009 / 0.040	0.171 / 0.201	0.515 / 0.625	0.118 / 0.163	0.371 / 0.414	0.167 / 0.204	0.033 / 0.069	0.095 / 0.124	0.185 / 0.230
w/o sampling	0.010 / 0.026	0.068 / 0.083	0.300 / 0.320	0.117 / 0.141	**0.319 / 0.362**	0.160 / 0.187	**0.031** / 0.046	0.045 / 0.068	0.131 / 0.154

Without experience. Removing the library raises the average gap from 0.121 to 0.185 and gap-AUC from 0.142 to 0.230. The largest degradations occur on Container, CDD, and TSP, while Flow and Set change little. This indicates that reusable experiences are particularly useful when different instance regimes favour different constructions.

Library without sampling. Keeping the library but removing router-based selection recovers most of the performance, with an average gap of 0.131 and AUC of 0.154, but remains worse overall than full HINTS. The largest degradation occurs on Container, whereas Job improves slightly. These results suggest that the library provides the main cross-instance benefit, while routing adds value when several eligible experiences compete for the same instance.

Effect on construction diversity. Figure 5 shows the construction types used on small, medium, and large Bin packing and Container instances. Without experience reuse, the LLM concentrates on one dominant construction across sizes. The w/o sampling variant retains multiple experiences but still concentrates on a small number of constructions. HINTS instead shifts among scoped constructions as the instance regime changes, yielding diversity that is structured by instance features rather than by a large undifferentiated pool.

4 Related Work

Improving the exact MIP pipeline. Modern MIP solvers (Gurobi Optimization, LLC, 2024; Ge et al., 2022; Achterberg, 2009) rely on branch-and-cut for search and proof. Recent learning-based methods augment this pipeline by targeting branching, cutting, node selection, primal heuristics, and solver configuration (Scavuzzo et al., 2026). On the primal side, Neural Diving (Nair et al., 2020) and Predict-and-Search (Han et al., 2023) predict partial assignments and pass them to the solver as starts or restricted search regions. FunL2O (Li et al., 2026) evolves feature functions, LLM4Branch (Hou et al., 2026) generates branching programs, and GRIMIP (Yu et al., 2026) generates instance-specific solver parameters. These methods modify or guide components of the optimization pipeline. HINTS instead leaves the formulation and solver mechanisms unchanged and interacts with the solver only through an injected MIP start.

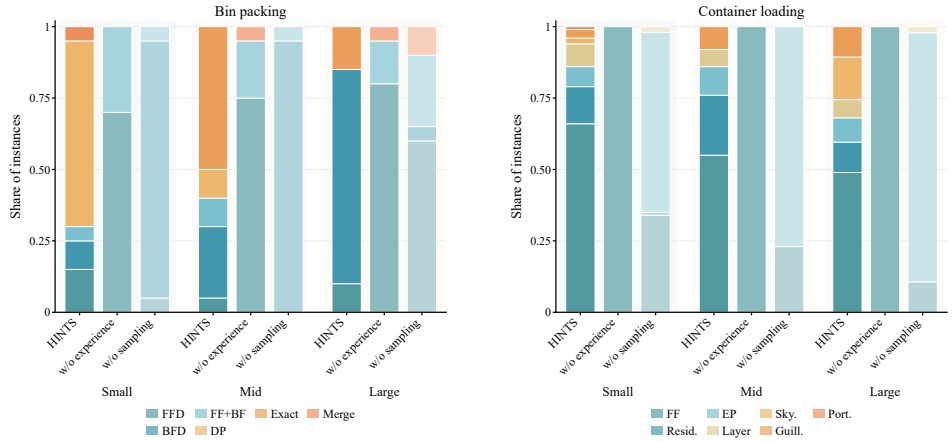

Figure 5: **Distribution of construction types under HINTS and its two ablations across instance sizes.** HINTS shifts among regime-specific constructions, while the ablated variants concentrate on fewer construction types.

LLM heuristics and exact solving. A growing line of work uses LLMs to design heuristic programs. FunSearch, Evolution of Heuristics, ReEvo, and LLaMEA (Romera-Paredes et al., 2024; Liu et al., 2024; Ye et al., 2024; van Stein & Bäck, 2025) evolve high-performing heuristics, typically offline and at the problem level. EoH-S (Liu et al., 2026) and DHEvo (Zhang et al., 2025) extend this idea to complementary heuristic sets or data–algorithm pairs, while still treating the generated heuristic as the optimization procedure. CO-Bench (Sun et al., 2026) benchmarks such agents on combinatorial optimization tasks. Other methods retain an exact solver: OptiMUS (AhmadiTeshnizi et al., 2024) uses an LLM to construct optimization models, while Rare-MIP (Wang et al., 2024) generates feasible assignments for a MIP solver. HINTS also keeps the exact solver, but operates specifically at the incumbent interface: the LLM generates a construction heuristic whose output is injected as a MIP start, without replacing the solver or changing the formulation.

Experience libraries. Reusable memory has also been explored to avoid solving each instance from scratch. Voyager (Wang et al., 2023) stores executable skills, while ExpeL (Zhao et al., 2024) retrieves verbal insights from past trajectories; in both cases, the memory primarily supports the agent itself. Related ideas appear in algorithm selection and hyper-heuristics (Rice, 1976; Xu et al., 2008; Burke et al., 2013), which maintain portfolios and select methods from instance features, but these portfolios are typically fixed offline and evaluated by the selected method's own performance. EoH-S (Liu et al., 2026) is closer in maintaining a small complementary heuristic set, but that set is evolved offline and used as a stand-alone optimizer. HINTS instead stores scoped experiences that specify *when* a construction applies and *how* to build a MIP start, and evaluates them by the solver's progress after injection. The library is updated only when evidence supports a new regime or a revision to an existing one, keeping it compact rather than growing into either an instance log or a single global heuristic.

5 Conclusion

HINTS builds a library of scoped experiences that generate MIP starts for an unchanged solver. Experiences are evaluated by subsequent solver progress—terminal gap, gap-AUC, and construction cost—rather than by the heuristic solution alone. Across Gurobi, COPT, and OR-Tools, the library drives most of the cross-instance adaptation, while routing provides an additional gain. OR-Tools with HINTS also outperforms solver-only COPT and narrows the gap to solver-only Gurobi. Our study uses a single LLM family, hand-designed task features, and one instance ordering, so transfer across models and stream orders remains open. HINTS also cannot compensate for a weak MIP formulation: it improves the starting incumbent and subsequent search, but leaves the underlying model unchanged.

AI USE STATEMENT

We used generative AI tools to draft parts of the manuscript, to edit the prose for readability, and to create or modify scientific figures. The LLM in the experiments is part of the method (Section 2), not a writing assistant. Authors reviewed all AI-assisted text and figures and take responsibility for the final content.

REPRODUCIBILITY STATEMENT

Solver settings and hyperparameters are in Appendix B; instance counts are in Section 3. The MIP attached to each of the eight tasks, and the start the heuristic must return, are in Appendix D; per-task instance features are in Appendix C; the fixed heuristic constructions are in Appendix E. Instances come from CO-Bench (Sun et al., 2026). We will include the HINTS loop, the injection interface, and the evaluation scripts as supplementary material.The anonymous repository at `https://anonymous.4open.science/r/HINTS-FCAF` provides the code for reproducing the experiments.

REFERENCES

Tobias Achterberg. SCIP: solving constraint integer programs. *Mathematical Programming Computation*, 1(1):1–41, 2009. doi: 10.1007/s12532-008-0001-1.

Shipra Agrawal and Navin Goyal. Thompson sampling for contextual bandits with linear payoffs. In *International Conference on Machine Learning*, volume 28, pp. 127–135, 2013. URL `https://proceedings.mlr.press/v28/agrawal13.html`.

Ali AhmadiTeshnizi, Wenzhi Gao, and Madeleine Udell. OptiMUS: Scalable optimization modeling with (MI)LP solvers and large language models. In *International Conference on Machine Learning*, volume 235, pp. 577–596, 2024. URL `https://proceedings.mlr.press/v235/ahmaditeshnizi24a.html`.

John E Beasley. OR-library: distributing test problems by electronic mail. *Journal of the Operational Research Society*, 41(11):1069–1072, 1990. doi: 10.1057/jors.1990.166.

Yoshua Bengio, Andrea Lodi, and Antoine Prouvost. Machine learning for combinatorial optimization: a methodological tour d'horizon. *European Journal of Operational Research*, 290(2): 405–421, 2021. doi: 10.1016/j.ejor.2020.07.063.

Edmund K Burke, Michel Gendreau, Matthew Hyde, Graham Kendall, Gabriela Ochoa, Ender Özcan, and Rong Qu. Hyper-heuristics: A survey of the state of the art. *Journal of the Operational Research Society*, 64(12):1695–1724, 2013. doi: 10.1057/jors.2013.71.

Edward G Coffman, János Csirik, Gábor Galambos, Silvano Martello, and Daniele Vigo. Bin packing approximation algorithms: Survey and classification. In Panos M Pardalos, Ding-Zhu Du, and Ronald L Graham (eds.), *Handbook of Combinatorial Optimization*, pp. 455–531. Springer, 2013. doi: 10.1007/978-1-4419-7997-1_35.

Victor Fernández-Viagas, Rubén Ruiz, and Jose M Framiñán. A new vision of approximate methods for the permutation flowshop to minimise makespan: State-of-the-art and computational evaluation. *European Journal of Operational Research*, 257(3):707–721, 2017. doi: 10.1016/j.ejor.2016.09.055.

Dongdong Ge, Qi Huangfu, Zizhuo Wang, Jian Wu, and Yinyu Ye. Cardinal optimizer (COPT) user guide, 2022. URL `https://arxiv.org/abs/2208.14314`.

Google. OR-tools, 2024. URL `https://developers.google.com/optimization`.

Gurobi Optimization, LLC. Gurobi optimizer reference manual, 2024. URL `https://www.gurobi.com`.

Qingyu Han, Linxin Yang, Qian Chen, Xiang Zhou, Dong Zhang, Akang Wang, Ruoyu Sun, and Xiaodong Luo. A GNN-guided predict-and-search framework for mixed-integer linear programming. In *International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=pSLtpcqmdt.

Zhinan Hou, Xingchen Li, Yankai Zhang, Tianxun Li, and Keyou You. LLM4Branch: Large language model for discovering efficient branching policies of integer programs, 2026. URL https://arxiv.org/abs/2605.10401.

David S Johnson and Lyle A McGeoch. The traveling salesman problem: A case study in local optimization. In Emile H L Aarts and Jan Karel Lenstra (eds.), *Local Search in Combinatorial Optimization*, pp. 215–310. John Wiley and Sons, 1997.

Ailsa H Land and Alison G Doig. An automatic method of solving discrete programming problems. *Econometrica*, 28(3):497–520, 1960. doi: 10.2307/1910129.

Bingheng Li, Junyang Cai, Yupeng Zhang, Bistra Dilkina, Jayant Kalagnanam, and Dzung T. Phan. FunL2O: LLM-guided feature function design for learning to optimize, 2026. URL https://arxiv.org/abs/2607.27389.

Fei Liu, Tong Xialiang, Mingxuan Yuan, Xi Lin, Fu Luo, Zhenkun Wang, Zhichao Lu, and Qingfu Zhang. Evolution of heuristics: Towards efficient automatic algorithm design using large language model. In *International Conference on Machine Learning*, volume 235, pp. 32201–32223, 2024. URL https://proceedings.mlr.press/v235/liu24bs.html.

Fei Liu, Yilu Liu, Qingfu Zhang, Tong Xialiang, and Mingxuan Yuan. EoH-S: Evolution of heuristic set using LLMs for automated heuristic design. *Proceedings of the AAAI Conference on Artificial Intelligence*, 40(43):37090–37098, 2026. doi: 10.1609/aaai.v40i43.41038.

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

John R Rice. The algorithm selection problem. *Advances in Computers*, 15:65–118, 1976. doi: 10.1016/S0065-2458(08)60520-3.

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

Lara Scavuzzo, Karen Aardal, Andrea Lodi, and Neil Yorke-Smith. Machine learning augmented branch and bound for mixed integer linear programming. *Mathematical Programming*, 217(1–2): 123–166, 2026. doi: 10.1007/s10107-024-02130-y.

Weiwei Sun, Shengyu Feng, Shanda Li, and Yiming Yang. CO-bench: Benchmarking language model agents in algorithm search for combinatorial optimization. *Proceedings of the AAAI Conference on Artificial Intelligence*, 40(39):33126–33134, 2026. doi: 10.1609/aaai.v40i39.40596.

William R Thompson. On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika*, 25(3/4):285–294, 1933. doi: 10.1093/biomet/25.3-4.285.

Niki van Stein and Thomas Bäck. LLaMEA: A large language model evolutionary algorithm for automatically generating metaheuristics. *IEEE Transactions on Evolutionary Computation*, 29 (2):331–345, 2025. doi: 10.1109/TEVC.2024.3497793.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. In *Advances in Neural Information Processing Systems*, 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/3bc91d1966490e89583107b424e6eb0e-Abstract-Conference.html.

Teng Wang, Wing-Yin Yu, Ruifeng She, Wenhan Yang, Taijie Chen, and Jianping Zhang. Leveraging large language models for solving rare MIP challenges, 2024. URL https://arxiv.org/abs/2409.04464.

Laurence A Wolsey. *Integer Programming*. Wiley, 2 edition, 2020. doi: 10.1002/9781119606475.

Lin Xu, Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. SATzilla: portfolio-based algorithm selection for SAT. *Journal of Artificial Intelligence Research*, 32:565–606, 2008. doi: 10.1613/jair.2490.

Haoran Ye, Jiarui Wang, Zhiguang Cao, Federico Berto, Chuanbo Hua, Haeyeon Kim, Jinkyoo Park, and Guojie Song. ReEvo: Large language models as hyper-heuristics with reflective evolution. In *Advances in Neural Information Processing Systems*, 2024. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/4ced59d480e07d290b6f29fc8798f195-Abstract-Conference.html.

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

Zhihao Zhang, Siyuan Li, Chenxi Li, Feifan Liu, Mengjing Chen, Kai Li, Tao Zhong, Bo An, and Peng Liu. DHEvo: Data-algorithm based heuristic evolution for generalizable MILP solving, 2025. URL https://arxiv.org/abs/2507.15615.

Andrew Zhao, Daniel Huang, Qingtao Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. ExpeL: LLM agents are experiential learners. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 19632–19642, 2024. doi: 10.1609/aaai.v38i17.29936.

Baoyun Zhao, He Wang, and Liang Zeng. G-LNS: Generative large neighborhood search for LLM-based automatic heuristic design, 2026.

A ROUTING DETAILS

This appendix records the symbols used by the router; the main text uses only equation 3 and equation 4.

Experience. An experience is

$$e = (\iota, r, \ell, \sigma, \mu), \tag{6}$$

an identifier, a revision index, a short title, a natural-language scope, and a natural-language method sketch. Its status is $\psi(e) \in \{\mathrm{pend}, \mathrm{act}, \mathrm{ret}\}$. The standing action set is the null action $a_\emptyset$ together with every non-retired experience.

Features and gap. Each task has a fixed feature schema of d deterministic statistics of the instance x (Appendix C). Raw values are mapped to the unit interval and centred, then stacked as

$$\phi(x) = \big(1, z_1, \ldots, z_d\big)^\top. \tag{7}$$

The MIP gap $g(t) \in [0, 1]$ is right-continuous: it is 1 until the first valid report and 0 after a proof. Its time average over the limit T is

$$\mathrm{AUC} = \frac{1}{T} \int_0^T g(t)\, dt. \tag{8}$$

In equation 3, t_{ex} is construction time, t_{sol} is the subsequent solver time, and $\mathbf{1}_{\mathrm{fail}}$ marks an infeasible or rejected start.

Eligibility. The LLM marks every non-retired experience with a label $\delta_e \in \{\mathrm{app}, \mathrm{unc}, \mathrm{inapp}\}$ and a suitability $s_e \in [0, 1]$; $s_{a_\emptyset} = 0$. The eligible set is

$$\mathcal{A}(x) = \big\{a_\emptyset\big\} \cup \big\{e : \psi(e) \neq \mathrm{ret},\ \delta_e \neq \mathrm{inapp}\big\}. \tag{9}$$

Posterior heads. Each action keeps two Bayesian linear heads, one for B and one for C. The prior is $A_0 = \lambda I$ and $h_0 = \lambda \mu_0$, where μ_0 is zero except an intercept $m_{0,B}$ or $m_{0,C}$. An observation (ϕ, y) with $y \in \{B, C\}$ updates

$$A \leftarrow A + \phi\phi^\top, \qquad h \leftarrow h + y\phi. \tag{10}$$

Selection in equation 4 draws one weight vector per head,

$$\tilde{\theta} \sim \mathcal{N}\big(A^{-1}h,\, T^2\sigma^2 A^{-1}\big)\,, \tag{11}$$

and sets $\tilde{B}_a = \phi^\top \tilde{\theta}_{a,B}$, $\tilde{C}_a = \phi^\top \tilde{\theta}_{a,C}$, and $\tilde{Y}_a = \tilde{B}_a - \kappa \tilde{C}_a$. $T = 1$ is the untempered posterior, also used for the probability of being best. Pending actions use $T = T_{\text{task}}$; active actions and $a_\emptyset$ use $T = T_{\text{act}} T_{\text{task}}$, with

$$T_{\text{task}} = T_{\min} + (1 - T_{\min})\sqrt{n_0/(n_0 + n_{\text{proc}})}, \tag{12}$$

where n_{proc} is the number of instances already processed. The decaying exploration boost is

$$\alpha(n) = \big[\alpha_{\max}\big(1 - n/N_\alpha\big)\big]_+. \tag{13}$$

After the wave, each selected arm (except bootstrap trials) is updated with the observed utilities.

B HYPERPARAMETERS

Table 3 lists the constants used in all experiments.

Table 3: Key hyperparameters used by HINTS in all experiments, grouped by solver runtime, protocol, scoring, router, and library update.

Stage	Type	Parameter	Value
Solver	Runtime	Time limit	600s
		Threads	2
	Version	Gurobi	13.0.1
		COPT	8.0.3
		OR-Tools / SCIP	SCIP backend
	Reproducibility	Random seed	Gurobi 0 (default); COPT, SCIP not set
		Deterministic	Gurobi on (default); COPT, SCIP not set
Protocol	Stream	Wave size W	8
	LLM	Model	DeepSeek-V4-Flash
		Prompt	fixed construction, repair, and wave templates
		Temperature	0.2 (construction, wave); 0 (repair, applicability)
		Max tokens	384000
		Repair rounds	3
Scoring	Utility	ω (gap-AUC weight)	0.8
		τ (time smoother)	1
		ν (failure penalty)	0.25
		κ (cost weight)	0.02
Router	Suitability	$\alpha_{\max}$	0.05
		N_α	4
	Bayesian heads	λ	1
		$m_{0,B}$	0.5
		$m_{0,C}$	0.1
		σ^2	0.04
	Temperature	$T_{\min}$	0.25
		n_0	32
		T_{act}	0.35
Library	Create	Labels	empty, split, uncovered
		Min. supporting instances	3
	Patch	Fields changed / wave	≤ 2 / once
		γ (method)	0.85
		γ (scope / title)	0.70 / 0.90

Promotion and retirement. Promotion in Section 2.3 requires at least 8 valid trials, at least two feature regions, a posterior standard deviation of Y at most 0.90 times the prior, and an injection success rate of at least 0.90. An active experience is retired after two consecutive severe failures—a timeout, an infeasible heuristic, or a rejected start—or when, since becoming active, it has been eligible at least 30 times, selected at most once, and its probability of being best is below 0.02.

C INSTANCE FEATURES

Each task owns a fixed schema of d deterministic statistics ($6 \leq d \leq 12$). Raw values are normalised to $u(x) \in [0,1]^d$ and centred to $z(x)$ for the router (Appendix A). Table 4 lists the features selected for each of the eight tasks.

Table 4: Features selected for each of the eight tasks.

Task	Features in $u(x)$
Bin	item count, capacity, size CV, large/small-item ratios, continuous bin LB, median capacity ratio, pairing pressure
CDD	job count, due tightness, processing CV, processing p_{75}, early-penalty CV, late-to-early penalty ratio
Container	box types, box count, volume pressure, volume CV, aspect CV, orientation flexibility
Flow	jobs, machines, processing CV, machine-load CV, job-load CV, bottleneck ratio, processing p_{75}
Job	jobs, machines, processing CV, machine-load CV, bottleneck ratio, route diversity, job-load CV
Open	jobs, machines, processing CV, machine-load CV, bottleneck ratio, route diversity
Set	rows, columns, matrix density, row/column degree CV, fragile-row fraction, cost-efficiency CV
TSP	node count, distance p_{25}/p_{75}, nearest-neighbour ratio, anisotropy, cluster score, outlier ratio, MST length, distance CV

D MIP FORMULATIONS

Each of the eight tasks uses one fixed MIP. HINTS never rewrites that model: the LLM returns a feasible object, which is mapped onto the variables below and injected as a start. The same model is handed to Gurobi, COPT, and OR-Tools (Gurobi Optimization, LLC, 2024; Ge et al., 2022; Google, 2024).

D.1 ONE-DIMENSIONAL BIN PACKING

Items $i{=}1\ldots n$ of sizes s_i must be packed into bins of capacity C. Binary y_b opens bin b; binary $x_{i,b}$ puts item i in bin b. Each item occupies one bin, $\sum_b x_{i,b}{=}1$; capacity is $\sum_i s_i x_{i,b} \leq C y_b$; symmetry breaking forces $y_b \leq y_{b-1}$. The objective minimises $\sum_b y_b$. The start is $\{\texttt{num_bins}{=}k,\ \texttt{bins}{=}[[i,\ldots],\ldots]\}$ with 1-based item IDs and no overloaded or duplicated items.

D.2 COMMON DUE-DATE SCHEDULING

Jobs $j{=}1\ldots n$ have processing times p_j and earliness/tardiness weights (a_j, b_j). A single machine sequences them around the common due date $d{=}h\sum_j p_j$ (default $h{=}0.6$). Binary $x_{j,k}$ assigns job j to position k; each job and each position appears once. Positional completion times satisfy $C^p_1{=}\sum_j p_j x_{j,1}$ and $C^p_k{=}C^p_{k-1}{+}\sum_j p_j x_{j,k}$. Job completion C_j is linked to C^p_k by big-M when $x_{j,k}{=}1$; earliness and tardiness obey $E_j \geq d{-}C_j$ and $T_j \geq C_j{-}d$. The objective minimises $\sum_j (a_j E_j{+}b_j T_j)$. The start is a 1-based permutation $\{\texttt{schedule}{=}[j_1,\ldots,j_n]\}$.

D.3 CONTAINER LOADING

Box types are expanded into individual boxes i with dimensions and orientation flags. Binary s_i places box i; continuous (x_i, y_i, z_i) is its lower-left-bottom corner; six orientation binaries choose a permutation of the three axes (flags may forbid some). Effective sizes $(\ell_i^x, \ell_i^y, \ell_i^z)$ are linear in those binaries. A placed box stays inside the container; pairwise six-direction binaries enforce 3D non-overlap through big-M. The objective maximises total placed volume. The start is a list of placements $\{$`box_type`, `container_id`=1, x, y, z, v, `hswap`$\}$.

D.4 FLOW SHOP SCHEDULING

Jobs j=1…n visit machines m=1…M in a fixed order, with processing times $p_{j,m}$. Binary $x_{j,k}$ puts job j in permutation position k; each job and each position appears once. $C_{k,m}$ is the completion time of the job in position k on machine m, and obeys $C_{k,m} \geq p_{k,m}$, $C_{k,m} \geq C_{k-1,m} + p_{k,m}$, and $C_{k,m} \geq C_{k,m-1} + p_{k,m}$, where $p_{k,m} = \sum_j p_{j,m} x_{j,k}$. The objective minimises $C_{\max} \geq C_{k,M}$. The start is a 1-based permutation $\{$`job_sequence`$=[j_1, \ldots, j_n]\}$.

D.5 JOB SHOP SCHEDULING

Job j has a fixed operation order; operation k of job j has processing time $p_{j,k}$ and a prescribed machine. Continuous $s_{j,k}$ is that start time; job precedence is $s_{j,k+1} \geq s_{j,k} + p_{j,k}$. For every pair of operations that share a machine, a binary chooses their order and a pair of big-M inequalities forbids overlap. The objective minimises $C_{\max} \geq s_{j,K_j} + p_{j,K_j}$. The start is $\{$`start_times`$=[[s_{j,k}]]\}$, one row per job.

D.6 OPEN SHOP SCHEDULING

Each job has one operation per machine, but the operation order is free. Continuous $s_{j,k}$ is again a start time. Two families of binaries resolve conflicts: one for every pair of operations on the same physical machine, and one for every pair of operations of the same job. $C_{\max}$ bounds every operation completion. The start has the same `start_times` layout as job shop, and must avoid both machine overlap and job overlap.

D.7 SET COVERING

Binary x_j selects column j of cost c_j. For every row i, the selected columns that cover i must sum to at least one. The objective minimises $\sum_j c_j x_j$. The start is $\{$`selected_columns`$=[j, \ldots]\}$ with 1-based column IDs that cover every row.

D.8 TRAVELLING SALESMAN PROBLEM

Cities have Euclidean coordinates. Binary x_e selects an undirected edge; every node has degree two. Disconnected subtours are cut by lazy DFJ inequalities $\sum_{e \in \delta(S)} x_e \geq 2$ at integer solutions. For $n \leq 500$ the graph is complete; for $n > 500$ it is the 50-nearest-neighbour graph plus the edges of a nearest-neighbour tour. The objective minimises Euclidean tour length. The start is a 0-based permutation $\{$`tour`$=[v_0, \ldots, v_{n-1}]\}$; on sparse instances every consecutive edge, including the closing edge, must exist in the model edge set.

E FIXED HEURISTIC CONSTRUCTIONS

Each task uses the two constructions in Table 5. Both return a feasible object in the same format as Appendix D and are injected as MIP starts; the solver then continues unchanged. The constructions are the standard feasible methods for that family.

Table 5: Classical construction heuristics injected as MIP starts. Heuristic-1 and Heuristic-2 are the standard feasible pair for each task family; both return a start in the same format as the HINTS injection.

Task	Heuristic-1	Heuristic-2
Bin	first-fit decreasing	best-fit decreasing
CDD	shortest processing time	V-shape
Container	wall-building (large-first)	wall-building (small-first)
Flow	NEH	Johnson
Job	SPT list	most work remaining
Open	SPT list	longest remaining processing time
Set	greedy ratio	greedy prune
TSP	nearest neighbour	nearest neighbour + 2-opt

Bin packing. Items are sorted by decreasing size. First-fit decreasing places each item in the lowest-index bin that has residual capacity, and opens a new bin only if none fits. Best-fit decreasing places each item in the feasible bin with the least residual capacity, and likewise opens a new bin if needed.

Common due-date scheduling. Shortest processing time sequences jobs by increasing p_j. The V-shape recommended for a common due date puts early jobs in decreasing processing time before d and tardy jobs in increasing processing time after d.

Container loading. Both starts are wall-building packers: boxes are stacked into vertical layers along one container axis, then the next wall is opened. Large-first sorts boxes by decreasing volume; small-first reverses that order.

Flow shop. NEH inserts jobs in decreasing total processing time into the position that yields the best partial makespan. Johnson's rule is the two-machine (or pairwise) sequence used as a constructive start on the same permutation model.

Job shop. SPT list dispatches, at each idle machine, the ready operation with the shortest processing time. Most work remaining instead prefers the job whose unfinished processing is largest.

Open shop. SPT list is the same dispatch rule as in job shop. Longest remaining processing time prefers the job with the most unfinished work among those that can start.

Set covering. Greedy ratio repeatedly selects the column with the best cost per newly covered row (the Chvátal greedy). Greedy prune starts from that cover and drops any column whose removal leaves every row covered.

TSP. Nearest neighbour grows a tour by always appending the closest unused city and returning to the start. The second start applies 2-opt to that tour until no improving exchange remains.

F GENERATION TIME INSIDE THE LIMIT

Section 3 scores every run on a 600s solver clock. Table 6 reports the comparison on the same eight tasks when the LLM generation time and the heuristic running time are included in that 600s limit. Until both have elapsed the gap is 1. Baseline, Heuristic-1, and Heuristic-2 are unchanged. Gap and gap-AUC are defined as in Table 1: clipped to $[0, 1]$, with the AUC taken over $[0, 600]$ and divided by 600. Average is the macro-average over tasks.

Table 6: **Performance when the 600s limit includes HINTS generation and heuristic execution.** Gap and AUC follow Table 1. Baseline and the fixed heuristics are unchanged. Average is the macro-average over tasks. Lower is better. Best results are highlighted in **bold** and the second-best values are underlined.

		Bin (160)		CDD (70)		Cont. (762)		Flow (120)		Job (80)		Open (60)		Set (87)		TSP (272)		Average	
Solver	Method	Gap	AUC	Gap	AUC	Gap	AUC	Gap	AUC	Gap	AUC	Gap	AUC	Gap	AUC	Gap	AUC	Gap	AUC
Gurobi	Baseline	.015	.032	.359	.447	.445	.496	.225	.285	.401	.481	.182	.229	.035	.055	.058	.117	.215	.268
	Heuristic-1	.010	**.025**	.252	.337	.387	.438	.123	**.149**	.393	.467	.176	.213	**.031**	**.047**	.028	**.055**	.175	.216
	Heuristic-2	.010	**.025**	.183	.233	.481	.522	.127	.158	.337	**.387**	.179	.216	**.031**	**.047**	.052	.087	.175	.209
	HINTS (ours)	**.003**	.026	**.067**	**.094**	**.252**	**.274**	**.113**	.153	**.324**	.410	**.159**	**.197**	**.031**	.062	**.027**	.119	**.122**	**.167**
COPT	Baseline	.226	.272	.361	.497	.759	.996	.107	**.181**	.621	.677	.378	.412	.048	.059	.817	.928	.415	.503
	Heuristic-1	.083	.153	.373	.504	.333	.485	**.100**	.189	.656	.693	.296	.321	**.044**	**.057**	.145	.192	.254	.324
	Heuristic-2	.083	.154	**.285**	.471	.400	.642	.111	.195	.457	.518	.297	.320	.045	**.057**	.131	.160	.226	.315
	HINTS (ours)	**.064**	**.141**	.331	**.361**	**.284**	**.312**	.108	.183	**.332**	**.400**	**.213**	**.236**	**.044**	.067	**.121**	**.152**	**.187**	**.232**
OR-Tools	Baseline	.139	.241	.456	.576	.993	.997	.413	.473	.920	.925	.580	.611	.103	**.119**	.344	.428	.494	.546
	Heuristic-1	.166	.265	.423	.580	.508	.535	.389	**.441**	.890	.899	.537	.551	**.100**	.123	.088	**.127**	.388	.440
	Heuristic-2	.166	.264	**.333**	.518	.662	.681	.395	.450	.600	.612	.520	.536	.108	.124	.244	.293	.379	.435
	HINTS (ours)	**.111**	**.240**	**.333**	**.385**	**.311**	**.371**	**.377**	**.441**	**.498**	**.541**	**.484**	**.510**	.109	.146	**.082**	.133	**.288**	**.344**