EVOLUTIONARY AI AGENTS FOR AUTOMATED COMBINATORIAL OPTIMIZATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Recent advances in large language models (LLMs) and AI agents have opened up new opportunities for automated combinatorial optimization. However, existing LLM/agent-based automated algorithm/heuristic design methods heavily rely on users to manually configure problem-dependent tools, heuristics, and iterative pipelines. To go beyond such design-level automation, we introduce Automated Problem Solving (APS), which enables AI agents to autonomously discover and organize the entire problem-solving process for combinatorial optimization. We further propose EvoAge, a self-evolving agentic framework for APS that explores, composes, and evolves reusable skills via evolutionary optimization. Inspired by natural evolution, EvoAge treats problem-solving trajectories, rather than individual heuristics or algorithms, as evolvable units and employs evolutionary search to automatically discover novel problem-solving strategies. It further distills successful experience into reusable skills and continuously evolves its skill library, enabling capabilities to accumulate, adapt, and expand across diverse tasks in an open-ended environment. Experiments across four classes of combinatorial optimization problems show that EvoAge obtains skills that can autonomously solve them without any human design or intervention and consistently outperforms state-of-the-art LLM/agent-based methods, demonstrating the effectiveness and huge potential of APS as a new paradigm for combinatorial optimization.

1 INTRODUCTION

Combinatorial optimization problems (COPs) widely exist in industrial and real-world applications, such as scheduling, routing, and allocation (Mohseni et al., 2022). Efficiently solving COPs is crucial for improving operational efficiency and supporting intelligent decision-making. Traditionally, solving COPs relies heavily on human expertise to manually design problem-specific heuristics, search strategies, and algorithmic components. Although handcrafted methods can achieve strong performance on certain problem classes, developing and configuring them typically requires substantial domain knowledge and engineering effort. Moreover, a heuristic that performs well for one problem or scale may not transfer effectively to others (Cappart et al., 2023). To alleviate such limitations, automated heuristic design (AHD) has emerged as a promising paradigm.

Recent advances in large language models (LLMs) have provided new opportunities for AHD (Da Ros et al., 2026). Benefiting from their strong capabilities in language understanding, reasoning, planning, and code generation, LLMs enable heuristic search to be conducted in a more flexible and expressive space. More recently, AI agents have extended this paradigm by equipping LLMs with tools, external environments, memory, and iterative interaction capabilities to autonomously decompose a task, design heuristics, execute solutions, observe the performance, and iteratively refine based on feedback (Zhang et al., 2026).

Despite the advances, the limitations of existing AHDs become increasingly apparent when moving from designing heuristics to solving problems. First, AHD typically searches within a predefined capability space consisting of human-designed heuristic components, prompts, or program structures. The quality and diversity of the resulting heuristics are therefore fundamentally bounded, making it difficult to discover problem-solving capabilities that fall outside the assumptions and building blocks. Second, AHD focuses primarily on optimizing specific heuristics for particular problems, rather than accumulating and reusing experience across problem-solving processes. Ex-

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.

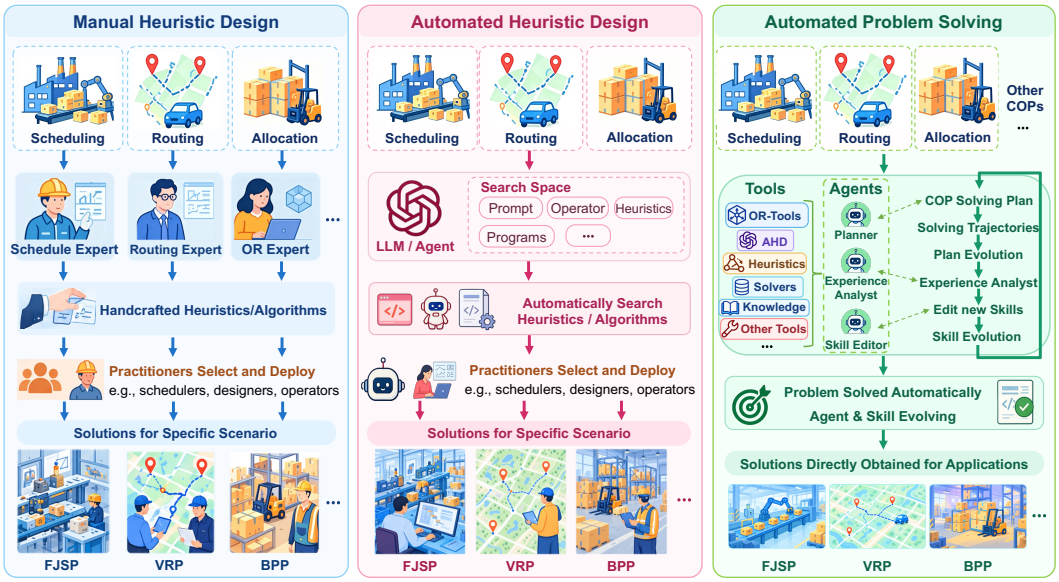

Figure 1: Comparisons between three paradigms: Manual Heuristic Design, Automated Heuristic Design, and our proposed Automated Problem Solving.

isting LLM/agent-based AHD methods lack the mechanisms to distill successful problem-solving experience into reusable skills and to continuously evolve such skills. In addition, AHD often leaves practitioners responsible for selecting, configuring, and combining appropriate heuristic components for different problem settings, introducing additional expertise and engineering overhead.

Therefore, a paradigm advance from AHD toward automated problem solving (APS) is desired. Instead of treating heuristic construction as the final objective, APS aims to enable an intelligent system (an agent) to directly understand a COP, organize and leverage available heuristics and tools, construct an effective problem-solving process, and iteratively improve itself. Thereby, the agent is no longer required to commit to a fixed heuristic configuration before solving the problem, but can dynamically compose existing tools, strategies, and learned skills according to the problem and feedback obtained during the solving process. Fig. 1 presents an intuitive comparison between manual heuristic design, AHD, and APS.

Nevertheless, realizing APS agents requires exploring problem-solving capabilities and accumulating such capabilities over time. Evolutionary optimization (Eiben & Smith, 2015) offers a natural solution. Evolutionary search can explore a large and open-ended space of candidate skills through variation, selection, and iterative improvement, without requiring the capability space to be specified in advance. Successful problem-solving trajectories can be retained, recombined, and refined to discover increasingly effective skills for COPs. Consequently, an agent can automatically generate a problem-solving strategy and accumulate such knowledge across tasks (Ren et al., 2026).

Based on these insights, we aim to establish a new APS paradigm for COPs and propose a self-evolving agent framework to achieve APS. The main contributions are as follows:

- We introduce EvoAge, an evolutionary agentic AI framework for realizing APS in COPs, to eliminate the reliance on human experts or practitioners by enabling agents to autonomously explore, acquire, and evolve new problem-solving skills.
- We propose an evolutionary skill search method that treats problem-solving trajectories as evolvable units to explore beyond fixed or predefined capability spaces, along with a skill management mechanism for continuously accumulating and reusing self-evolved skills.
- We demonstrate the consistent superiority of EvoAge over advanced LLM/agent-based methods across four categories of COPs, showing that EvoAge can efficiently solve COPs without relying on any human experts or practitioner design or intervention.

2 Related Work

2.1 Automated Heuristic Design

With the emergence of LLMs, recent studies have explored increasingly autonomous approaches to AHD, evolving from LLM-based heuristic generation toward agentic systems.

LLM-based AHD: Early efforts have demonstrated the potential of LLMs for AHD by combining their code-generation and reasoning capabilities with search. EoH (Liu et al., 2024) represents heuristic ideas as language thoughts and evolves them jointly with executable programs. ReEvo (Ye et al., 2024) introduces reflective evolution, using LLM-generated reflections to guide heuristic evolution. Experience-guided reflective co-evolution further evolves prompts and heuristics jointly based on accumulated experience. MCTS-AHD (Zheng et al., 2025) employs Monte Carlo Tree Search to improve the exploration of the heuristic space and alleviate premature convergence. EvoPH (Liu et al., 2025) jointly evolves prompts and heuristics by leveraging accumulated experience and reflective feedback to progressively improve automatic algorithm design. MoH (Shi et al., 2026) moves beyond individual heuristic generation by allowing LLMs to iteratively construct and refine the heuristic optimizer itself. HeurAgenix (Yang et al., 2025) integrates heuristic generation, evaluation, and evolution into an iterative LLM-based optimization framework. These studies demonstrate the potential of LLMs for AHD by leveraging their reasoning and programming capabilities. However, these methods primarily rely on LLMs as heuristic generators or search operators, with limited ability to autonomously interact with optimization environments and accumulate reusable problem-solving capabilities, which has motivated the emergence of agent-based AHD.

Agent-based AHD: AI agents extend LLM-based AHD with environment interaction, tool use, planning, memory, and iterative feedback. DRAGON (Chen et al., 2026b) employs LLM-driven decomposition and reconstruction agents to solve large-scale COPs by iteratively optimizing localized subproblems and integrating accumulated experience. HeuriGym (Chen et al., 2026a) provides an agentic benchmark for evaluating LLM-crafted heuristics through code execution and feedback across diverse COPs. RefineEvo (Wu et al., 2026) incorporates planning and bidirectional experience into heuristic evolution, enabling heuristic refinement based on previous search outcomes. PathWise (Gungordu et al., 2026) formulates heuristic generation as a sequential decision process, where multiple agents collaboratively plan, generate, and critique evolutionary actions using a stateful entailment graph. AFL (Zhang et al., 2026) employs specialized agents to collaboratively generate, execute, and refine heuristics for complex vehicle routing problems through iterative environment interaction and feedback. Agent-based AHD moves toward interactive and experience-driven heuristic design, but existing methods still primarily focus on heuristic optimization and lack mechanisms for managing experience as persistent, reusable skills.

2.2 Self-evolving Agents

Beyond COPs, recent studies have explored self-evolving agents that move beyond fixed prompts, tools, and search procedures through experience, reflection, memory, and evolutionary search.

Evolutionary Search for Self-Evolution: GEPA (Agrawal et al., 2025) evolves prompts through reflective feedback, using lessons from previous trajectories to guide subsequent optimization. EvoX (Liu et al., 2026) evolves the search strategy together with candidate solutions, enabling adaptive exploration during optimization. CORAL (Qu et al., 2026) employs autonomous multi-agent evolution with persistent shared memory to support long-horizon open-ended discovery. AVO (Chen et al., 2026c) replaces fixed variation operators with agentic coding processes that generate, evaluate, and refine modifications. BES (Xu et al., 2026b) combines forward evolutionary exploration with backward goal decomposition to expand the reachable capability space of language models.

Beyond Fixed Capability Spaces: These studies demonstrate a growing trend toward evolving not only solutions but also prompts, search strategies, variation mechanisms, and agent behaviors. However, existing approaches generally focus on improving algorithm design or model capability on specific tasks, with limited enhancement for distilling problem-solving experience into explicit, reusable skills. In particular, learned capabilities are rarely managed as skills that can be selectively reused, recombined, and continuously evolved across heterogeneous tasks. Differently, our proposed methodology extends these efforts by evolving reusable problem-solving skills to achieve APS.

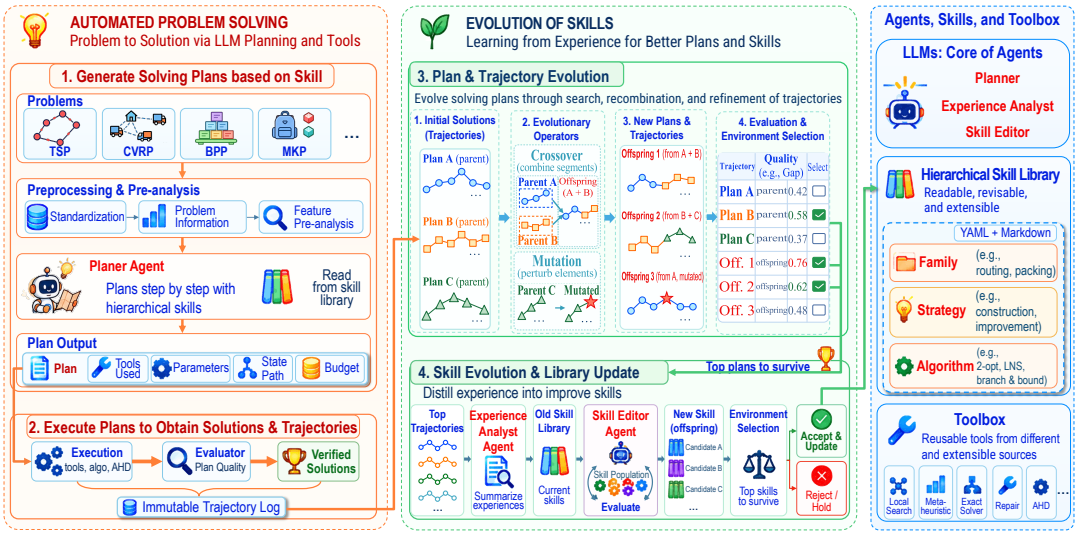

Figure 2: The overall workflow of the proposed EvoAge framework.

3 METHODOLOGY

3.1 OVERALL WORKFLOW

The workflow in Fig. 2 comprises four stages. **(1) Skill-guided planning:** Given a problem instance, the **Planner agent** identifies its characteristics and retrieves relevant guidance from the hierarchical Skill Library. It then constructs a Plan that specifies how permitted tools will be organized and used to solve the instance. The Library guides planning but does not prescribe a fixed pipeline. **(2) Plan execution:** The executor checks and runs the Plan with the available tools. The evaluator verifies the resulting solution and records both its measured quality and the execution trajectory, linking the Plan's decisions to their observed effects. **(3) Plan and trajectory evolution:** These records form an instance-level archive. Evolutionary operators select promising Plans and generate offspring by recombining or modifying their decisions. Each offspring returns to Stage 2 for execution and verification; the updated archive then guides the next evolutionary iteration. This feedback loop searches for effective ways to compose tools for the current instance. **(4) Skill evolution:** After instance-level search, the **Experience Analyst agent** compares verified trajectories across training instances and extracts reusable, conditional experience. The **Skill Editor agent** uses this evidence to propose new or revised Skills. Candidate Library updates are evaluated against the active Library on paired validation instances, and eligible revisions are selected for the next epoch. This second feedback loop transfers knowledge from completed searches to future planning without treating an unverified trajectory or a proposed Skill as an established improvement.

EvoAge therefore evolves at two connected levels. Plan evolution explores solving workflows within individual instances and supplies measured execution evidence; Skill evolution converts supported findings into guidance shared across instances. During final evaluation, the accepted Library is frozen, and the Planner constructs one Plan for each unseen instance.

3.2 EVOLUTION OF PROBLEM SOLVING PLAN AND TRAJECTORY

Let x be a training instance, $\mathcal{L}_e$ the active Skill Library at epoch e, and $\mathcal{T}_e(x)$ the operations permitted for x. The Library organizes guidance into Family, Strategy, and Algorithm Skills. Using the instance profile $\phi(x)$ and relevant Skills, the Planner constructs a Plan

$$p = (u_1, \ldots, u_m), \qquad u_t = (a_t, \theta_t, c_t, h_t), \tag{1}$$

where $a_t \in \mathcal{T}_e(x)$ is an operation, θ_t its parameters, c_t its execution condition, and h_t its state-handoff and update rule. A Plan also specifies an overall budget and may include step-level budgets, stopping conditions, or a fallback. Plan search therefore explores how registered operations are composed and controlled within the permitted toolbox and budget.

Algorithm 1 takes x, $\mathcal{L}_e$, $\mathcal{T}_e(x)$, and the per-Plan budget B as inputs. P is the requested number of initial Plans, whereas $\mathcal{P}$ denotes the initialized Plan set. G is the total number of generations, and H_x contains available history for this training instance. The algorithm maintains an archive $\mathcal{A}_{x,e}$ and returns its final state.

Algorithm 1 Instance-level plan evolution

Require: Instance x; Skill Library $\mathcal{L}_e$; permitted operations $\mathcal{T}_e(x)$; per-Plan budget B, initial Plan count P; generations G; instance history H_x
Ensure: Evaluated plan archive $\mathcal{A}_{x,e}$
$\mathcal{P} \leftarrow Initialize(x, \mathcal{L}_e, \mathcal{T}_e(x), B, P)$
$\mathcal{P} \leftarrow Validate(\mathcal{P}, \mathcal{T}_e(x), B)$
$\mathcal{A}_{x,e} \leftarrow Evaluate(\mathcal{P}, x, B)$
for $g = 1$ to $G - 1$ **do**
  $\mathcal{P}_p \leftarrow SelectParents(\mathcal{A}_{x,e})$
  $\mathcal{P}_c \leftarrow GenerateOffspring(\mathcal{P}_p, \mathcal{A}_{x,e}, H_x)$
  $\mathcal{P}_c \leftarrow Validate(\mathcal{P}_c, \mathcal{T}_e(x), B)$
  $\mathcal{A}_{x,e} \leftarrow \mathcal{A}_{x,e} \cup Evaluate(\mathcal{P}_c, x, B)$
end for
return $\mathcal{A}_{x,e}$

Initialization combines Planner-generated Plans conditioned on $\phi(x)$ and the retrieved Skills with program-side exploratory workflows. If enabled, an empty-Library proposal provides a within-instance reference for assessing Skill guidance. The initial Plans are checked against operation contracts and the budget before being independently executed on x. Execution of a Plan p produces a trajectory

$$\tau(p, x) = \left\{(a_t, z_t, o_t)\right\}_{t=1}^{T_p}, \tag{2}$$

where a_t is the operation specified at step t, z_t the associated search state, and o_t the observed result or skipped status. The record includes solution changes, runtime, and termination status. During execution, the best feasible incumbent is preserved separately from the mutable state passed between operations.

The initial evaluations form the generation-zero archive. After generation g, the current archive can be written as

$$\mathcal{A}_{x,e}^{(g)} = \{(p_i, \tau_i, m_i)\}_{i=1}^{N_g}, \tag{3}$$

where m_i summarizes feasibility, objective value, and execution cost. In each later generation, $\mathcal{P}_p$ denotes parents selected from the current archive using measured quality and runtime while maintaining behavioral diversity. Structural operators generate the offspring set $\mathcal{P}_c$ by changing operation choices, parameters, ordering, or control flow. When enabled, the Planner also proposes revisions targeting weaknesses observed in parent trajectories. Valid offspring are executed and added to the archive without discarding earlier records.

This trajectory-level search expands the strategies EvoAge can explore. Crossover recombines decisions from different Plans, while mutation changes operation choices, parameters, ordering, or control conditions. Their offspring can therefore form solving workflows that were absent from the initial population and Skill Library. Execution feedback determines whether these new combinations are useful, rather than assuming that an initial heuristic template or fixed pipeline is sufficient.

When an instance recurs, H_x may guide novelty and offspring generation. Historical trajectories remain discovery history: parent selection and current-epoch outcomes rely on executions under the current Library and tool policy. After G generations, the returned archive satisfies $\mathcal{A}_{x,e} = \mathcal{A}_{x,e}^{(G-1)}$ and provides instance-specific evidence for Skill evolution.

3.3 Evolution-based Skill Management

Plan evolution produces an archive for each training instance; Skill management uses the aggregated archive $\mathcal{A}_e$ to improve guidance across instances. As shown in Algorithm 2, the procedure takes the active Skill Library $\mathcal{L}_e$, experience memory $\mathcal{X}_e$, trial bank $\mathcal{B}_e$, fresh validation batch $\mathcal{V}_e$, and evaluation protocol Π_e as inputs. It returns the updated Library $\mathcal{L}_{e+1}$, memory $\mathcal{X}_{e+1}$, and trial bank $\mathcal{B}_{e+1}$.

The first step converts executed Plan trajectories into evidence cards $\mathcal{E}_e$. Each card links instance characteristics and Plan decisions to operations actually executed and their measured outcomes. The Experience Analyst compares trajectories where execution conditions permit, separates proposed decisions from realized ones, and derives conditional experiences from supported observations. These experiences are added to $\mathcal{X}_e$ to form $\mathcal{X}_{e+1}$. Grounding the analysis in execution records prevents a plausible account of a Plan from being treated as evidence of its effectiveness. Retaining experiences across epochs also allows later Skill revisions to draw on outcomes from multiple instances rather than an isolated successful run.

The Skill Editor uses $\mathcal{X}_{e+1}$ and the active Library to propose revisions $\mathcal{C}_e$. A revision may create, modify, link, or delete a Skill; decision-bearing guidance specifies its applicability, recommended action, stop-or-switch condition, and boundary of use. Proposed revisions are checked against the Skill schema, hierarchy, dependencies, and cited evidence before being registered in $\mathcal{B}_{e+1}$. Because the Editor can create new conditional guidance from successful trajectories, the Library can acquire strategies absent from its initial contents. The evidence and schema checks constrain this expansion to guidance supported by observed behavior.

For each active trial c compatible with the current Library and protocol, the procedure selects a validation panel $\mathcal{V}_c$ covering its affected families and applies the revision to $\mathcal{L}_e$ to obtain a complete candidate Library $\mathcal{L}_c$. The parent and candidate are then evaluated on the same fresh instances under matched solver seeds, tool access, planning settings, and budgets, yielding paired outcomes R_p^c and R_c. The Planner is held deterministic, and repeated seeds are aggregated within each instance rather than counted as independent instances. Paired evidence accumulates in $\mathcal{B}_{e+1}[c]$ while the parent Library and protocol remain unchanged. This comparison tests whether a proposed Skill improves actual problem solving under controlled conditions, rather than relying on its wording or the performance of the trajectory from which it was derived.

Algorithm 2 Paired validation and skill evolution

Require: Skill Library $\mathcal{L}_e$; aggregated Plan archive $\mathcal{A}_e$, experience memory $\mathcal{X}_e$; trial bank $\mathcal{B}_e$, validation batch $\mathcal{V}_e$; protocol Π_e
Ensure: updated Library $\mathcal{L}_{e+1}$; memory $\mathcal{X}_{e+1}$, updated trial bank $\mathcal{B}_{e+1}$

$\mathcal{E}_e \leftarrow BuildEvidence(\mathcal{A}_e)$
$\mathcal{X}_{e+1} \leftarrow \mathcal{X}_e \cup Analyze(\mathcal{E}_e, \mathcal{X}_e)$
$\mathcal{C}_e \leftarrow ProposeEdits(\mathcal{L}_e, \mathcal{X}_{e+1})$
$\mathcal{C}_e \leftarrow ValidateEdits(\mathcal{C}_e, \mathcal{E}_e)$
$\mathcal{B}_{e+1} \leftarrow Register(\mathcal{B}_e, \mathcal{C}_e, \mathcal{L}_e, \Pi_e)$
$\mathcal{Q}_e \leftarrow \emptyset$
for all active trials c compatible with $\mathcal{L}_e$ and Π_e **do**
  $\mathcal{V}_c \leftarrow RestrictPanel(\mathcal{V}_e, scope(c))$
  $\mathcal{L}_c \leftarrow ApplyEdits(\mathcal{L}_e, c)$
  $(R_p^c, R_c) \leftarrow PairedEval(\mathcal{L}_e, \mathcal{L}_c, \mathcal{V}_c)$
  $\mathcal{B}_{e+1}[c] \leftarrow Accumulate(\mathcal{B}_{e+1}[c], R_p^c, R_c)$
  $q_c \leftarrow Assess(\mathcal{B}_{e+1}[c])$
  if $Accepts(q_c)$ **then**
    $\mathcal{Q}_e \leftarrow \mathcal{Q}_e \cup \{(c, \mathcal{L}_c, q_c)\}$
  else
    $\mathcal{B}_{e+1}[c] \leftarrow UpdateStatus(\mathcal{B}_{e+1}[c], q_c)$
  end if
end for
$\mathcal{L}_{e+1} \leftarrow SelectNonoverlapping(\mathcal{L}_e, \mathcal{Q}_e)$
$\mathcal{B}_{e+1} \leftarrow Reconcile(\mathcal{B}_{e+1}, \mathcal{L}_{e+1}, \Pi_e)$
return $\mathcal{L}_{e+1}, \mathcal{X}_{e+1}, \mathcal{B}_{e+1}$

The assessment q_c determines whether c has sufficient paired evidence, preserves feasibility, meets the configured quality and runtime tolerances, and is used in execution when applicable. Deletion trials do not require realized Skill use, and protocol failures are not interpreted as performance changes. Candidates satisfying these conditions enter $\mathcal{Q}_e$; other trials have their status updated. The procedure selects validated revisions with non-overlapping family scopes to form $\mathcal{L}_{e+1}$ and reconciles the remaining trial bank with the resulting Library. Separately tested revisions with overlapping scopes are not combined without evaluating their joint effect. If none qualifies, $\mathcal{L}_e$ remains active and valid incomplete trials may continue on later validation batches.

This process connects the two levels of evolution: instance-level Plan search discovers and tests operation compositions, while Skill management distills their outcomes into reusable guidance and admits revisions through paired validation. The accepted Library guides planning in the next training epoch. For final evaluation, it is frozen, and the Planner generates one Plan per unseen instance without further Skill updates or Plan evolution.

4 EXPERIMENTS

Training and Testing: We train EvoAge on synthetic instances of the traveling salesman problem (TSP), capacitated vehicle routing problem (CVRP), offline bin packing (BPP), and multiple knapsack problem (MKP). Training sizes are $\{100, 500, 1000, 2000\}$ cities for TSP, $\{100, 200, 500, 1000\}$ customers for CVRP, and $\{500, 1000, 5000, 10000\}$ items for BPP. MKP uses $\{200, 500, 1000, 2000\}$ items with $\{5, 10, 20, 40\}$ knapsacks, respectively. Each scale includes four different structures of 16 training instances per family. Testing uses 77 TSPLIB instances (Reinelt, 1991), 20 CVRPLIB instances (Uchoa et al., 2017; Queiroga et al., 2022), five Weibull BPP, and ten synthetic MKP instances from the DRAGON benchmark setting (Chen et al., 2026b).

LLMs Configuration and Implementation Details: All LLM-based methods use DeepSeek-V4-Flash (Xu et al., 2026a). EvoAge uses a sampling temperature of 1.0; baselines retain their native temperature settings. All experiments are conducted on a workstation equipped with an NVIDIA

GeForce RTX 5060 Ti GPU (8 GB) and an Intel Core i7-14650HX CPU, demonstrating that the EvoAge framework can be easily implemented and extended on most devices.

Baselines: We compare EvoAge with: 1). direct methods: OR-Tools (Furnon & Perron, 2026), CP-SAT (Perron & Didier, 2026), and constructive packing heuristics; 2). five AHD methods: EoH (Liu et al., 2024), ReEvo (Ye et al., 2024), MCTS-AHD (Zheng et al., 2025), PathWise (Gungordu et al., 2026), RefineEvo (Wu et al., 2026), and DRAGON (Chen et al., 2026b). AHD comparisons cover TSP, CVRP, and BPP; MKP is compared with the applicable direct and agent-based methods. DRAGON performs online optimization without offline training. Peer methods are evaluated on the same benchmark instances with a 60-second per-instance time budget, where the final heuristics on different COPs, trained using their open-source code, are adopted. Since DRAGON needs more time to obtain feasible results, we adopt two settings: a 60-second limit and the same setting as in its paper, one hour per instance.

4.1 Comparisons to SOTA LLM/Agent-based Solvers

Table 1 compares APS and EvoAge over AHD paradigms across heterogeneous COPs. Numbers under each problem title indicate benchmark size ranges; $k = 10^3$ and $M = 10^6$. "Succ." denotes the percentage of instances for which a valid feasible solution is obtained. "Gap" denotes the mean relative gap over successfully solved instances, computed against published optima or best-known solutions for TSP and CVRP, a lower bound for BPP, and optima or upper bounds for MKP. "–" denotes a non-applicable setting or zero successful solution from which a gap can be computed. Best results are shown in **bold**; second-best results are underlined, with ties formatted identically.

Under the same 60-second per-instance computational budget, the LLM/Agent-based AHD methods generally exhibit poor performance when evaluated following their original settings or using the source code provided by the authors. They either achieve relatively low success rates or produce solutions with large gaps, and several methods fail to obtain feasible solutions on some problem families. For the agent-based DRAGON, its performance is relatively poor even with a one-hour budget. This indicates that AHD alone remains insufficient for reliably solving heterogeneous COPs under a limited computational budget. In contrast, EvoAge maintains consistently high success rates across all four problem families while achieving strong solution quality, demonstrating the advantage of APS over AHD. Rather than restricting automation to the search for heuristics within a predefined design process, APS allows the agent to autonomously determine how a problem should be solved, including the exploration, composition, and refinement of problem-solving capabilities. Moreover, the substantial performance improvement over existing LLM/Agent-based methods highlights the effectiveness of self-evolution, enabling EvoAge to continuously accumulate experience and evolve its problem-solving Skills instead of repeatedly searching within a fixed capability space.

4.2 Evolution of Agent Skills

Table 1: Performance on public benchmarks. Higher success rates and lower mean reference gaps indicate better performance; both metrics are reported in percent.

Category	Method	TSP (51–18.5k)		CVRP (109–5.0k)		BPP (5.0k)		MKP (1.97k–3.0M)	
		Succ.↑	Gap↓	Succ.↑	Gap↓	Succ.↑	Gap↓	Succ.↑	Gap↓
Direct	OR-Tools	89.6	4.30	80.0	8.01	–	–	–	–
	CP-SAT	–	–	–	–	**100.0**	**0.02**	40.0	0.19
LLM-based AHD	EoH	37.7	10.47	35.0	27.43	**100.0**	0.14	–	–
	ReEvo	62.3	29.93	35.0	23.75	**100.0**	1.02	–	–
	MCTS-AHD	32.5	37.22	15.0	125.08	0.0	–	–	–
Agent-based AHD	RefineEvo	62.3	17.78	35.0	79.64	0.0	–	–	–
	PathWise	64.9	25.08	35.0	155.48	**100.0**	1.35	–	–
Agentic	DRAGON-60s	41.6	12.47	0.0	–	0.0	–	70.0	0.21
	DRAGON-1h	98.7	10.95	**100.0**	19.74	80.0	0.33	**100.0**	0.16
	EvoAge	**100.0**	**2.92**	**100.0**	**5.43**	**100.0**	0.31	**100.0**	**0.03**

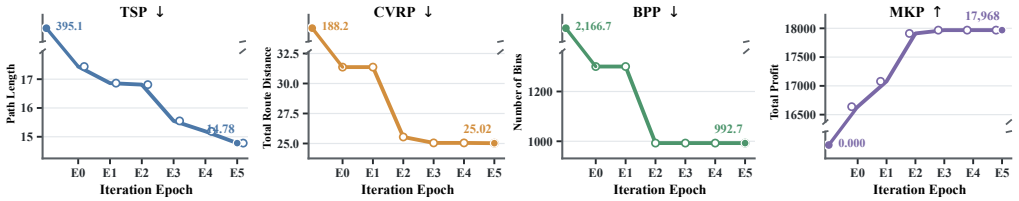

Figure 3: The evolution of skills on different COPs.

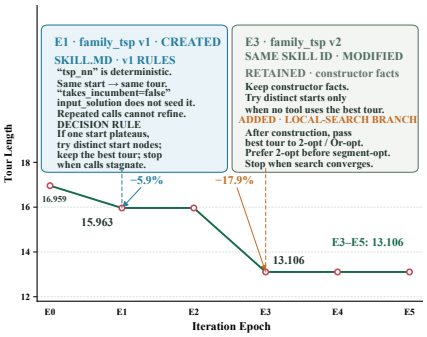

Figure 4: Visualization of the skill evolution on an 800-node TSP case.

Fig. 3 depicts the self-evolution behavior of EvoAge across successive evolutionary iterations. Overall, as the evolution proceeds from E_0 to E_5, the agent continuously accumulates problem-solving experience and evolves its Skills, resulting in consistent performance improvements across different COP families. Despite their distinct problem structures and challenges, substantial improvements can be achieved within only a few evolutionary iterations, while the subsequent evolution gradually stabilizes as high-quality Skills are established. For BPP, EvoAge exhibits particularly fast evolution, reaching a high-quality Skill as early as E_2, after which only marginal improvements are observed. For TSP, the performance improves progressively over the evolutionary process, indicating that the agent continuously refines its routing strategies through accumulated experience. A similar evolutionary pattern can be observed on CVRP, where successive iterations progressively improve the learned Skill and eventually lead to stable performance. For MKP, EvoAge also achieves substantial improvement during the early evolutionary iterations, followed by a slower refinement stage as the Skill approaches convergence. The different evolutionary trajectories across these families further indicate that EvoAge does not enforce an identical evolution pattern, but allows its Skills to autonomously develop according to the problem.

Overall, these results demonstrate that EvoAge is not limited to executing a fixed problem-solving strategy or heuristic improvement. Instead, it can progressively discover, accumulate, and refine problem-solving knowledge through self-evolution, while naturally adapting the evolution process to different COP families.

4.3 Illustration of Skills

Fig. 4 presents a specific case to show how EvoAge progressively improves its TSP Skill on a fixed 800-node corridor instance. From E_0 to E_1, the learned Skill improves nearest-neighbor construction by exploring multiple starting nodes, reducing the objective from 16.959 to 15.963. At E_3, the Skill further incorporates incumbent-aware local search, decreasing the objective to 13.106. Performance then remains stable through E_5, indicating convergence of the learned skill. Overall, Skill evolution yields a 22.7% reduction in tour length. This visualization of Skill evolution reveals the dynamic behavior of the agent as it progressively refines its problem-solving strategy, providing intuitive evidence of the effectiveness of self-evolution in EvoAge.

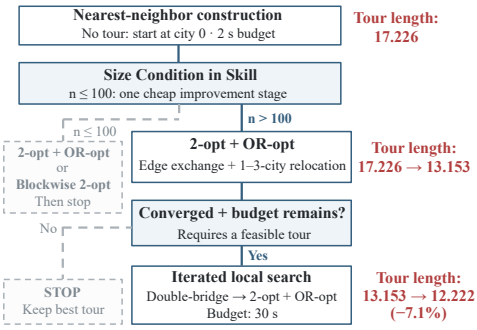

Figure 5: An interpretable illustration of the obtained skill after TSP training.

While Fig. 4 illustrates how a Skill evolves, Fig. 5 further reveals what an evolved Skill actually encodes in an interpretable way. Fig. 5 presents an evolved skill on TSP, illustrating that the evolved

skill is not simply a fixed sequence of heuristics but an adaptive problem-solving strategy. The skill first constructs a feasible solution using nearest-neighbor construction, then selects appropriate improvement strategies based on the problem scale. For small instances ($n \leq 100$), it performs only a lightweight improvement stage, such as 2-opt with OR-opt or blockwise 2-opt, to avoid unnecessary computational overhead.

For larger instances ($n > 100$), it applies stronger local improvements through edge exchange and 1–3-city relocation, reducing the tour length from 17.226 to 13.153. More importantly, the skill further adapts its solving behavior according to the online optimization status rather than following a predetermined sequence: after local improvement, it checks solution feasibility, convergence, and the remaining computational budget. If further search is worthwhile, it switches to iterated local search, using double-bridge perturbation followed by 2-opt and OR-opt to escape the current local optimum; otherwise, it terminates and retains the best solution found. In the illustrated case, this adaptive stage further improves the tour from 13.153 to 12.222 (7.1%). This example demonstrates that the evolved skills can encode interpretable decision logic that jointly considers problem characteristics and runtime search dynamics, enabling scale-aware strategy selection, conditional execution, and adaptive allocation of the search budget.

4.4 Comparison between APS and AHD

Fig. 6 illustrates the convergence of the best objective values achieved by peer AHD methods over heuristic evaluations across different COPs. Meanwhile, we also report the best objective value obtained by EvoAge after one iteration as a benchmark. The results show that different AHD methods exhibit varying applicability across problem types, and no single AHD method generalizes well to all problems, even though we strictly follow their source code, which was provided specifically for TSP, CVRP, and BPP, in our experiments. In contrast, EvoAge can identify suitable tools and formulate effective solution plans for different types of problems efficiently, thereby achieving competitive objective values across diverse COPs. These results further demonstrate the effectiveness and efficiency of the proposed APS paradigm.

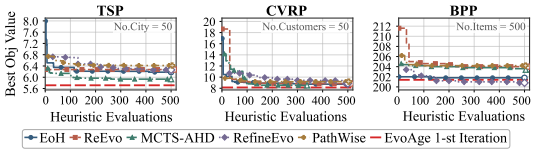

Figure 6: Verification of the proposed agent-based APS paradigm.

5 Conclusion

This work introduced a new paradigm for advancing combinatorial optimization from AHD to APS, and presented EvoAge, an evolutionary agentic framework that autonomously searches over problem-solving Plans and distills execution experience into reusable Skills. By evolving trajectories and validating candidate Skill updates, EvoAge can discover and retain effective ways to compose available operations across problem families without relying on any human experts or practitioners ad-hoc design. On four classes of COPs, EvoAge consistently outperformed the state-of-the-art LLM/agent-based AHD methods under a 60-second per-instance budget. It also achieved stronger results than the agentic DRAGON baseline evaluated with a one-hour budget. These findings support the value of evolving and reusing complete solving strategies for APS as well as shifting the research focus from AHD to APS. Future work will extend the available toolsets and examine how learned Skills transfer to new COPs.

The source code of EvoAge can be obtained from ***https://github.com/Anonymous-cloud-paper/ICLR27-EvoAge/releases/tag/EvoAge***

AI Usage Statements

In this work, we used generative AI tools to assist with writing and presentation. Specifically, these tools were used to improve grammatical correctness, sentence fluency, clarity, and consistency of academic expression, as well as to polish the wording of selected passages without altering their intended technical meaning. Generative AI tools were also used to assist with the formatting

and organization of large tables, including adjusting table layouts, aligning entries, refining column structures, and improving their readability and presentation. Some of the icons of FIg. 1 and Fig. 2 are also generated by Generative AI. All AI-assisted content and formatting were subsequently reviewed, verified, and revised by the authors, who take full responsibility for the final manuscript.

REFERENCES

Lakshya A Agrawal, Shangyin Tan, Dilara Soylu, Noah Ziems, Rishi Khare, Krista Opsahl-Ong, Arnav Singhvi, Herumb Shandilya, Michael J Ryan, Meng Jiang, et al. Gepa: Reflective prompt evolution can outperform reinforcement learning, 2025. *URL https://arxiv. org/abs/2507.19457*, 2(3):4, 2025.

Quentin Cappart, Didier Chételat, Elias B Khalil, Andrea Lodi, Christopher Morris, and Petar Veličković. Combinatorial optimization and reasoning with graph neural networks. *Journal of Machine Learning Research*, 24(130):1–61, 2023.

Hongzheng Chen, Yingheng Wang, Yaohui Cai, Hins Hu, Jiajie Li, Shirley Huang, Chenhui Deng, Rongjian Liang, Shufeng Kong, Haoxing Ren, et al. Heurigym: An agentic benchmark for llm-crafted heuristics in combinatorial optimization. In *International Conference on Learning Representations*, volume 2026, pp. 27466–27520, 2026a.

Shengkai Chen, Zhiguang Cao, Jianan Zhou, Yaoxin Wu, Senthilnath Jayavelu, Zhuoyi Lin, Xiaoli Li, and Shili Xiang. Dragon: Llm-driven decomposition and reconstruction agents for large-scale combinatorial optimization. *arXiv preprint arXiv:2601.06502*, 2026b.

Terry Chen, Zhifan Ye, Bing Xu, Zihao Ye, Timmy Liu, Ali Hassani, Tianqi Chen, Andrew Kerr, Haicheng Wu, Yang Xu, et al. Avo: Agentic variation operators for autonomous evolutionary search. *arXiv preprint arXiv:2603.24517*, 2026c.

Francesca Da Ros, Michael Soprano, Luca Di Gaspero, and Kevin Roitero. Large language models for combinatorial optimization: A systematic review. *ACM Computing Surveys*, 58(11):1–53, 2026.

Agoston E. Eiben and Jim Smith. From evolutionary computation to the evolution of things. *Nature*, 521(7553):476–482, 2015. doi: 10.1038/nature14544.

Vincent Furnon and Laurent Perron. OR-Tools routing library, version 9.15. Google Optimization Tools, 2026. URL `https://developers.google.com/optimization/routing/`.

Oguzhan Gungordu, Siheng Xiong, and Faramarz Fekri. Pathwise: Planning through world model for automated heuristic design via self-evolving llms. *arXiv preprint arXiv:2601.20539*, 2026.

Fei Liu, Xialiang Tong, Mingxuan Yuan, Xi Lin, Fu Luo, Zhenkun Wang, Zhichao Lu, and Qingfu Zhang. Evolution of heuristics: Towards efficient automatic algorithm design using large language model. *arXiv preprint arXiv:2401.02051*, 2024.

Shu Liu, Shubham Agarwal, Monishwaran Maheswaran, Mert Cemri, Zhifei Li, Qiuyang Mang, Ashwin Naren, Ethan Boneh, Audrey Cheng, Melissa Z Pan, et al. Evox: Meta-evolution for automated discovery. *arXiv preprint arXiv:2602.23413*, 2026.

Yihong Liu, Junyi Li, Wayne Xin Zhao, Hongyu Lu, and Ji-Rong Wen. Experience-guided reflective co-evolution of prompts and heuristics for automatic algorithm design. *arXiv preprint arXiv:2509.24509*, 2025.

Naeimeh Mohseni, Peter L McMahon, and Tim Byrnes. Ising machines as hardware solvers of combinatorial optimization problems. *Nature Reviews Physics*, 4(6):363–379, 2022.

Laurent Perron and Frédéric Didier. OR-Tools CP-SAT, version 9.15. Google Optimization Tools, 2026. URL `https://developers.google.com/optimization/cp/cp_solver/`.

Ao Qu, Han Zheng, Zijian Zhou, Yihao Yan, Yihong Tang, Shao Yong Ong, Fenglu Hong, Kaichen Zhou, Chonghe Jiang, Minwei Kong, et al. Coral: Towards autonomous multi-agent evolution for open-ended discovery. *arXiv preprint arXiv:2604.01658*, 2026.

Eduardo Queiroga, Ruslan Sadykov, Eduardo Uchoa, and Thibaut Vidal. 10,000 optimal CVRP solutions for testing machine learning based heuristics. In *AAAI-22 Workshop on Machine Learning for Operations Research*, 2022. URL https://openreview.net/forum?id=yHiMXKN6nTl.

Gerhard Reinelt. TSPLIB—a traveling salesman problem library. *ORSA Journal on Computing*, 3 (4):376–384, 1991. doi: 10.1287/ijoc.3.4.376.

Zhe Ren, Yimeng Chen, Dandan Guo, Guowei Rong, Tonghui Li, RB Xiong, Qingfeng Lan, Wenyi Wang, Li Nanbo, Yibo Yang, et al. Self-improvements in modern agentic systems: A survey. *arXiv preprint arXiv:2607.13104*, 2026.

Yiding Shi, Jianan Zhou, Wen Song, Jieyi Bi, Yaoxin Wu, Zhiguang Cao, and Jie Zhang. Generalizable heuristic generation through llms with meta-optimization. In *International Conference on Learning Representations*, volume 2026, pp. 119314–119349, 2026.

Eduardo Uchoa, Diego Pecin, Artur Pessoa, Marcus Poggi, Thibaut Vidal, and Anand Subramanian. New benchmark instances for the capacitated vehicle routing problem. *European Journal of Operational Research*, 257(3):845–858, 2017. doi: 10.1016/j.ejor.2016.08.012.

Yang Wu, Junran Pan, Yifan Zhang, Ning Xu, Fanshuo Zeng, and Jian Cheng. Refineevo: Planning-guided heuristic evolution with bidirectional experience. *arXiv preprint arXiv:2607.11358*, 2026.

Anyi Xu, Bangcai Lin, Bing Xue, Bingxuan Wang, Bingzheng Xu, Bochao Wu, Bowei Zhang, Chaofan Lin, Chen Dong, Chenchen Ling, et al. Deepseek-v4: Towards highly efficient million-token context intelligence. *arXiv preprint arXiv:2606.19348*, 2026a.

Guowei Xu, Zhenting Qi, Huangyuan Su, Weirui Ye, Himabindu Lakkaraju, Sham M Kakade, and Yilun Du. Self-improving language models with bidirectional evolutionary search. *arXiv preprint arXiv:2605.28814*, 2026b.

Xianliang Yang, Ling Zhang, Haolong Qian, Lei Song, and Jiang Bian. Heuragenix: Leveraging llms for solving complex combinatorial optimization challenges. *arXiv preprint arXiv:2506.15196*, 2025.

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. *Advances in neural information processing systems*, 37:43571–43608, 2024.

Ni Zhang, Zhiguang Cao, Jianan Zhou, Cong Zhang, and Yew-Soon Ong. An agentic framework with llms for solving complex vehicle routing problems. In *International Conference on Learning Representations*, volume 2026, pp. 114166–114206, 2026.

Zhi Zheng, Zhuoliang Xie, Zhenkun Wang, and Bryan Hooi. Monte carlo tree search for comprehensive exploration in llm-based automatic heuristic design. *arXiv preprint arXiv:2501.08603*, 2025.

A Appendix

A.1 Detailed EvoAge Workflow

Algorithm 3 integrates instance-level Plan search with cross-instance Skill evolution. Its inputs are a training set D, a disjoint validation set V, the number of epochs E, and an initial Skill Library $\mathcal{L}_1$. As in Section 3.2, P is the initial Plan population size, C the requested offspring count per subsequent generation, $G \geq 1$ the number of generations including initialization, M the maximum Plan length, and B the execution budget for each Plan. The set $\mathcal{T}_e(x)$ contains the operations available for instance x at epoch e. The validation protocol Π_e specifies panel selection, matched solver seeds, planning settings, tool access, execution budgets, and acceptance criteria. The persistent states are the experience memory $\mathcal{X}_e$ and candidate trial bank $\mathcal{B}_e$; both are initially empty.

Algorithm 3: EvoAge: Plan Search and Persistent Skill Evolution

Require: Training set D, validation set V, number of epochs E
Require: Initial population size P (number of Plans), offspring count C (per generation)
Require: Generations G, maximum Plan length M, per-Plan budget B
Require: Initial Library $\mathcal{L}_1$, available operations $\mathcal{T}_e(x)$, validation protocol Π_e
Ensure: Learned Library $\mathcal{L}_{E+1}$ and experience memory $\mathcal{X}_{E+1}$

$\mathcal{X}_1 \leftarrow \emptyset$; $\mathcal{B}_1 \leftarrow \emptyset$
for $e = 1, \ldots, E$ **do**
  $\mathcal{A}_e \leftarrow \emptyset$
  for all $x \in D$ **do**
    $(f, \phi(x)) \leftarrow$ Identify and Profile(x)
    $\mathcal{S}_x \leftarrow$ Retrieve Hierarchical Skills$(\mathcal{L}_e, f, \phi(x))$
    $H_x \leftarrow$ Retrieve Instance History(x, e)
    $\mathcal{P} \leftarrow$ Initialize Plans$(\phi(x), \mathcal{S}_x, \mathcal{T}_e(x), B, P)$ (initial Plan population)
    $\mathcal{P} \leftarrow$ Validate Plans$(\mathcal{P}, \mathcal{T}_e(x), B, M)$
    $\mathcal{A}_{x,e}^{(0)} \leftarrow$ Execute and Verify$(\mathcal{P}, x, B)$
    for $g = 1, \ldots, G-1$ **do**
      $\mathcal{P}_p \leftarrow$ Select Parents$\left(\mathcal{A}_{x,e}^{(g-1)}\right)$ (parent Plan set)
      $\mathcal{P}_c \leftarrow$ Generate Offspring$\left(\mathcal{P}_p, \mathcal{A}_{x,e}^{(g-1)}, H_x, C\right)$ (offspring Plan set)
      $\mathcal{P}_c \leftarrow$ Validate Plans$(\mathcal{P}_c, \mathcal{T}_e(x), B, M)$
      $\mathcal{A}_{x,e}^{(g)} \leftarrow \mathcal{A}_{x,e}^{(g-1)} \cup$ Execute and Verify$(\mathcal{P}_c, x, B)$
    end for
    $\mathcal{A}_e \leftarrow \mathcal{A}_e \cup \mathcal{A}_{x,e}^{(G-1)}$
  end for
  $\mathcal{X}_{e+1} \leftarrow \mathcal{X}_e$; $\mathcal{C}_e \leftarrow \emptyset$
  for all problem families f represented in $\mathcal{A}_e$ **do**
    $\mathcal{E}_{e,f} \leftarrow$ Build Evidence$(\mathcal{A}_{e,f})$
    $\Delta\mathcal{X}_{e,f} \leftarrow$ Analyze Experience$(\mathcal{E}_{e,f}, \mathcal{X}_{e,f})$
    $\Delta\mathcal{X}_{e,f} \leftarrow$ Check Evidence Links$(\Delta\mathcal{X}_{e,f}, \mathcal{E}_{e,f})$
    $\mathcal{X}_{e+1} \leftarrow \mathcal{X}_{e+1} \cup \Delta\mathcal{X}_{e,f}$
    $\mathcal{U}_{e,f} \leftarrow$ Summarize Skill Use$(\mathcal{A}_{e,f}, \mathcal{L}_e)$
    $\mathcal{F}_{e,f} \leftarrow$ Collect Trial Feedback$(\mathcal{B}_e, f)$
    $\mathcal{C}_{e,f} \leftarrow$ Propose Edits$(\mathcal{L}_e, \Delta\mathcal{X}_{e,f}, \mathcal{X}_{e,f}, \mathcal{U}_{e,f}, \mathcal{F}_{e,f})$
    $\mathcal{C}_{e,f} \leftarrow$ Validate Edits$(\mathcal{C}_{e,f}, \mathcal{E}_{e,f})$
    $\mathcal{C}_e \leftarrow \mathcal{C}_e \cup \mathcal{C}_{e,f}$
  end for
  $\mathcal{B}_{e+1} \leftarrow$ Register$(\mathcal{B}_e, \mathcal{C}_e, \mathcal{L}_e, \Pi_e)$
  $\mathcal{V}_e \leftarrow$ Select Validation Panel(V, Π_e); $\mathcal{Q}_e \leftarrow \emptyset$
  for all active trials c in $\mathcal{B}_{e+1}$ compatible with $\mathcal{L}_e$ and Π_e **do**
    $\mathcal{V}_c \leftarrow$ Restrict Panel$(\mathcal{V}_e, \mathrm{scope}(c))$
    $\mathcal{L}_c \leftarrow$ Apply Edits$(\mathcal{L}_e, c)$
    $(R_p^c, R_c) \leftarrow$ Paired Evaluation$(\mathcal{L}_e, \mathcal{L}_c, \mathcal{V}_c, \Pi_e)$
    $\mathcal{B}_{e+1}[c] \leftarrow$ Accumulate$\left(\mathcal{B}_{e+1}[c], R_p^c, R_c\right)$
    $q_c \leftarrow$ Assess$(\mathcal{B}_{e+1}[c], \Pi_e)$
    if Accepts(q_c, Π_e) **then**
      $\mathcal{Q}_e \leftarrow \mathcal{Q}_e \cup \{(c, \mathcal{L}_c, q_c)\}$
    else
      $\mathcal{B}_{e+1}[c] \leftarrow$ Update Status$(\mathcal{B}_{e+1}[c], q_c)$

43: **end if**
44: **end for**
45: $\mathcal{L}_{e+1} \leftarrow$ Select Nonoverlapping$(\mathcal{L}_e, \mathcal{Q}_e)$
46: $\mathcal{B}_{e+1} \leftarrow$ Reconcile$(\mathcal{B}_{e+1}, \mathcal{L}_{e+1}, \Pi_e)$
47: **end for**
48: **return** $\mathcal{L}_{E+1}, \mathcal{X}_{E+1}$

Skill-guided Plan search. At epoch e, EvoAge holds the parent Library $\mathcal{L}_e$ fixed and initializes the epoch archive $\mathcal{A}_e$. For each training instance $x \in D$, preprocessing identifies its problem family f and extracts the profile $\phi(x)$. Hierarchical retrieval returns the relevant Skill set $\mathcal{S}_x$, while H_x contains records from earlier epochs for the same instance and is empty if none exist. The Planner uses the profile, retrieved Skills, available operations, and budget to construct the initial Plan population $\mathcal{P}$. Each Plan is checked for valid operations and parameters, length, and budget before execution. Execution and solution verification produce the initial archive $\mathcal{A}_{x,e}^{(0)}$, whose records contain Plans, trajectories, and measured feasibility, objective values, and costs, including failed executions.

At each subsequent generation g, selection draws a parent Plan set $\mathcal{P}_p$ from the evaluated archive according to performance and behavioral diversity. Recombination and mutation generate the offspring Plan set $\mathcal{P}_c$, with historical records H_x available to guide proposal generation. Here, subscripts p and c denote parents and children; the sets $\mathcal{P}_p$ and $\mathcal{P}_c$ are distinct from the population-size parameters P and C. Offspring undergo the same validity checks before execution, so fewer than C children may be evaluated. Their records extend the cumulative archive $\mathcal{A}_{x,e}^{(g)}$; prior-epoch records inform proposals but are not used as current evaluations. After G generations, the instance archives are combined into $\mathcal{A}_e$ and retained for experience extraction and future history retrieval.

Experience extraction and Skill revision. For each family f, EvoAge extracts the corresponding records $\mathcal{A}_{e,f} \subseteq \mathcal{A}_e$ and summarizes them as evidence $\mathcal{E}_{e,f}$ connecting instance characteristics and Plan decisions to outcomes. The Experience Analyst considers this evidence alongside the prior family-specific memory $\mathcal{X}_{e,f} \subseteq \mathcal{X}_e$ to derive new experiences $\Delta\mathcal{X}_{e,f}$. After their evidence links are checked, these experiences are added to the next memory $\mathcal{X}_{e+1}$, initialized from $\mathcal{X}_e$.

The Skill Editor uses the new and prior experiences, a summary of observed Skill use $\mathcal{U}_{e,f}$, and feedback from earlier trials $\mathcal{F}_{e,f}$ to propose the family-specific revision set $\mathcal{C}_{e,f}$. Each revision contains coordinated Skill edits and an explicit family scope. Revisions passing schema, hierarchy, dependency, and evidence checks are collected in the epoch candidate set $\mathcal{C}_e$. Registration records them in $\mathcal{B}_{e+1}$ together with their parent Library and protocol, retaining the trial histories carried forward from $\mathcal{B}_e$.

Paired selection and persistent update. EvoAge selects a validation panel $\mathcal{V}_e \subseteq V$ under protocol Π_e and initializes the eligible set $\mathcal{Q}_e$. For each active revision c compatible with the parent Library and protocol, its affected family set $\mathrm{scope}(c)$ determines the restricted panel $\mathcal{V}_c$. Applying its edits to $\mathcal{L}_e$ yields a candidate Library $\mathcal{L}_c$. Parent and candidate are evaluated on the same instances with matched solver seeds, planning settings, operation access, and budgets. Their respective outcomes R_p^c and R_c are accumulated in the trial record $\mathcal{B}_{e+1}[c]$; repeated seeds refine within-instance estimates rather than constituting additional independent instances.

Assessment summarizes the accumulated evidence in q_c, covering feasibility, solution quality, cost, realized Skill use, and evaluation validity. Trials satisfying the acceptance rule of Section 3.3 enter $\mathcal{Q}_e$ as tuples $(c, \mathcal{L}_c, q_c)$; other trials receive an updated status distinguishing pending evidence, rejection, and evaluation failure. Selection applies eligible revisions with non-overlapping family scopes to obtain $\mathcal{L}_{e+1}$, or retains $\mathcal{L}_e$ if none qualifies. Overlapping revisions remain alternatives because their combined effect has not been validated. Reconciliation then updates the trial bank against the selected Library and protocol. This completes one epoch: subsequent search uses the updated Library while experience and compatible trial histories persist. The procedure returns $\mathcal{L}_{E+1}$ and $\mathcal{X}_{E+1}$ after E epochs. During testing, the Library is frozen and guides profiling, Skill retrieval, and Plan execution on unseen instances; test outcomes do not enter the learning loop.

A.2 PER-INSTANCE PUBLIC-BENCHMARK RESULTS

Each table compares all reported methods for one problem family, with one row per instance. Within each method cell, the upper value is Gap (%) and the lower, gray value is runtime (s). Instance names and sizes occupy separate columns on the same line; Ref. denotes the reference objective used in the main table. Gaps and runtime are displayed to two and one decimal places, respectively, whereas summary means are computed before rounding. The lowest successful Gap in each row is bold and shaded; displayed ties are marked identically. Runtime is not part of this ranking.

Timeout indicates that a run reached its time limit, and *Failed* indicates that it ended without a feasible solution for another reason. These labels replace the numeric Gap. Such runs are excluded from the mean Gap but counted as unsuccessful; their elapsed times are retained and may include termination overhead. Size n counts cities for TSP, customers excluding the depot for CVRP, and items for BPP. MKP reports items n and knapsacks m. BPP includes only the five Weibull instances, whose L_1 and L_2 lower bounds coincide. AHD methods use one selected training seed per method and family throughout the panel. DRAGON-60s and DRAGON-1h are separate runtime settings.

Table 2: TSP: per-instance results on 77 TSPLIB instances. Each cell reports Gap (%) above runtime (s).

Instance	n	Ref.	OR-Tools	EoH	ReEvo	MCTS-AHD	Refine Evo	PathWise	DRAGON 60s	DRAGON 1h	EvoAge
eil51	51	426	**0.00** 51.5	9.86 1.7	26.53 1.7	19.48 2.1	4.46 1.9	20.66 2.0	*Timeout* 60.0	2.11 3394.8	0.23 43.1
berlin52	52	7542	**0.00** 52.3	6.79 1.8	17.58 1.7	35.03 2.1	7.46 2.0	19.07 1.9	*Timeout* 60.0	1.54 3221.9	**0.00** 58.3
st70	70	675	**0.00** 51.3	11.85 2.5	24.74 1.7	25.04 3.5	16.59 1.9	18.52 2.0	*Timeout* 60.0	7.70 1086.8	**0.00** 54.9
eil76	76	538	**0.00** 52.3	18.03 2.6	32.16 1.7	44.98 4.0	4.65 2.0	19.33 1.9	6.51 47.6	6.51 3473.2	**0.00** 57.7
pr76	76	108159	0.02 52.3	17.81 3.0	36.59 1.8	38.84 4.0	11.13 1.9	41.89 2.1	*Timeout* 60.0	3.17 3639.0	**0.00** 58.2
rat99	99	1211	**0.08** 52.3	14.12 4.3	39.80 1.7	6.44 8.3	19.24 2.0	34.27 1.9	*Timeout* 60.0	8.84 3686.6	**0.08** 33.8
kroA100	100	21282	**0.00** 52.3	12.55 4.8	33.04 1.7	35.02 8.5	12.15 1.9	26.18 1.8	*Timeout* 60.0	6.28 3619.6	0.41 37.6
kroB100	100	22141	**0.00** 52.3	10.87 4.6	46.01 1.8	55.12 8.5	10.93 2.1	31.69 1.8	7.07 34.4	6.01 2350.8	0.98 55.1
kroC100	100	20749	**0.00** 52.3	12.54 4.8	39.30 1.8	34.85 9.0	11.30 2.1	26.40 1.9	8.88 47.3	5.20 2109.4	**0.00** 58.5
kroD100	100	21294	1.20 52.3	9.98 4.9	27.60 1.7	30.44 8.6	10.56 1.9	26.55 1.9	10.83 51.5	9.68 1006.0	**0.38** 55.9
kroE100	100	22068	0.42 52.3	6.76 4.7	21.56 1.8	17.15 8.4	5.79 2.0	25.00 2.0	*Timeout* 60.0	5.30 3712.5	**0.19** 57.8
rd100	100	7910	1.73 52.3	9.49 4.6	30.48 1.8	56.86 8.7	19.63 2.0	29.15 1.9	*Timeout* 60.0	7.79 3594.3	**0.00** 58.3
eil101	101	629	1.27 52.3	12.56 4.7	37.36 1.8	24.01 9.1	11.45 1.9	32.43 1.9	7.63 37.9	7.15 2191.1	**0.95** 59.5
lin105	105	14379	**0.00** 52.3	15.73 5.0	42.01 1.8	38.99 9.7	19.22 1.9	41.57 2.0	*Timeout* 60.0	7.64 3670.7	**0.00** 58.3
pr107	107	44303	**0.08** 52.3	1.68 5.3	29.61 1.8	5.02 10.2	6.01 1.9	11.42 1.9	*Timeout* 60.1	3.56 918.5	0.49 43.7
pr124	124	59030	0.55 52.3	6.01 8.0	19.37 1.8	62.74 17.1	30.49 2.1	21.63 1.8	*Timeout* 60.0	6.07 3653.7	**0.00** 58.4
bier127	127	118282	0.62 52.3	8.32 8.6	33.70 1.8	52.61 19.2	26.20 2.1	17.28 1.9	7.21 39.8	6.03 3608.8	**0.52** 39.5
ch130	130	6110	1.01 52.3	10.49 9.6	21.23 1.8	63.40 20.8	10.21 2.2	17.81 2.0	*Timeout* 60.0	6.68 3644.8	**0.98** 40.3
pr136	136	96772	**0.00** 52.3	11.58 11.0	37.07 2.0	64.10 24.6	9.92 2.3	31.74 2.0	*Timeout* 60.0	8.21 3677.5	1.61 57.6
pr144	144	58537	**0.00** 52.3	2.33 12.5	6.73 1.9	13.94 28.5	20.59 2.4	6.18 2.0	*Timeout* 60.0	3.44 3626.9	**0.00** 55.7
ch150	150	6528	0.54 52.3	6.57 16.2	28.98 2.1	52.63 34.9	14.89 2.4	25.47 2.1	9.85 52.7	8.66 3600.4	**0.32** 58.3
kroA150	150	26524	0.37 52.4	9.21 15.9	27.75 2.0	58.33 34.2	20.73 2.3	29.12 2.0	5.90 40.3	5.00 3595.0	**0.02** 58.5
kroB150	150	26130	1.38 52.3	13.56 16.3	27.66 2.0	61.70 34.4	17.66 2.4	25.65 2.1	6.37 35.2	4.80 3631.5	**0.38** 57.8

Instance	n	Ref.	OR-Tools	EoH	ReEvo	MCTS-AHD	Refine Evo	PathWise	DRAGON 60s	DRAGON 1h	EvoAge
pr152	152	73682	0.27	7.43	20.80	17.94	14.71	16.31	6.57	4.92	**0.18**
			52.3	16.6	2.1	36.0	2.6	2.0	49.0	3632.5	43.2
u159	159	42080	1.82	14.22	34.50	15.87	20.14	32.10	8.91	6.01	**0.00**
			52.3	18.8	2.1	41.3	2.6	2.3	36.7	3553.6	46.3
rat195	195	2323	**1.94**	11.36	22.56	*Timeout*	16.10	13.30	*Timeout*	12.91	**1.94**
			52.3	40.6	2.4	60.1	2.8	2.3	60.0	3606.8	38.5
d198	198	15780	1.23	8.59	15.91	*Timeout*	35.34	24.90	*Timeout*	5.54	**0.44**
			52.3	42.1	2.4	60.1	2.9	2.2	60.0	3606.3	58.0
kroA200	200	29368	0.76	9.81	27.34	*Timeout*	13.52	22.44	8.48	6.75	**0.52**
			51.3	44.1	2.3	60.1	3.2	2.2	41.8	3602.4	56.0
kroB200	200	29437	2.60	13.52	23.65	*Timeout*	14.61	26.55	9.39	8.26	**1.89**
			52.3	44.6	2.4	60.1	3.1	2.4	50.2	3664.4	59.8
ts225	225	126643	0.88	*Timeout*	24.54	*Timeout*	3.05	14.87	*Timeout*	10.93	**0.00**
			52.3	60.1	2.7	60.0	3.2	2.6	60.0	3812.4	43.8
tsp225	225	3916	2.76	*Timeout*	46.22	*Timeout*	17.57	29.49	12.08	11.90	**0.46**
			52.3	60.1	2.6	60.1	3.5	2.4	24.5	3643.2	43.6
pr226	226	80369	1.33	*Timeout*	22.88	*Timeout*	36.26	18.05	*Timeout*	1.18	**0.29**
			52.3	60.1	2.6	60.1	3.4	2.4	60.0	3828.9	43.5
gil262	262	2378	**3.87**	*Timeout*	35.79	*Timeout*	24.47	27.75	*Timeout*	10.18	5.05
			52.3	60.1	3.1	60.1	4.1	2.9	60.0	3690.7	59.8
pr264	264	49135	3.58	*Timeout*	20.93	*Timeout*	21.59	19.33	*Timeout*	8.45	**1.04**
			52.3	60.1	3.2	60.1	4.1	2.7	60.0	3635.5	56.0
a280	280	2579	**0.70**	*Timeout*	38.15	*Timeout*	17.41	28.00	21.25	20.28	2.44
			52.3	60.0	3.5	60.1	4.5	3.0	34.0	3817.6	59.6
pr299	299	48191	**3.18**	*Timeout*	27.15	*Timeout*	18.06	23.67	*Timeout*	11.38	3.32
			52.3	60.1	3.9	60.1	5.1	3.3	60.0	1278.0	2.6
lin318	318	42029	2.87	*Timeout*	38.20	*Timeout*	31.82	28.53	8.01	*Timeout*	**2.26**
			52.3	60.1	3.9	60.1	5.9	3.5	59.1	3600.0	36.4
rd400	400	15281	2.44	*Timeout*	35.69	*Timeout*	13.69	27.71	*Timeout*	9.63	**1.06**
			51.3	60.1	6.4	60.1	9.9	5.1	60.0	3605.9	38.4
fl417	417	11861	**2.29**	*Timeout*	29.94	*Timeout*	42.63	34.09	*Timeout*	9.34	2.78
			52.3	60.1	7.2	60.1	9.5	5.1	60.0	3624.9	58.0
pr439	439	107217	6.35	*Timeout*	37.06	*Timeout*	19.41	23.33	*Timeout*	11.69	**2.51**
			52.3	60.1	8.3	60.1	11.3	6.1	60.0	3639.1	59.6
pcb442	442	50778	**1.87**	*Timeout*	28.78	*Timeout*	18.01	21.23	17.50	17.18	6.53
			51.3	60.1	7.8	60.1	11.3	5.8	45.2	3840.1	55.4
d493	493	35002	4.49	*Timeout*	25.11	*Timeout*	17.13	24.10	8.69	7.66	**1.75**
			52.4	60.1	10.2	60.1	15.2	7.4	45.6	3701.4	59.8
u574	574	36905	5.43	*Timeout*	36.06	*Timeout*	13.75	30.31	*Timeout*	11.60	**2.05**
			52.4	60.1	15.0	60.1	21.4	10.3	60.0	3630.9	57.8
rat575	575	6773	4.40	*Timeout*	28.92	*Timeout*	21.47	27.08	*Timeout*	13.07	**3.26**
			52.4	60.1	15.1	60.1	23.6	10.3	60.0	3587.2	45.6
p654	654	34643	**0.94**	*Timeout*	36.63	*Timeout*	46.35	23.44	5.70	5.70	2.95
			52.4	60.1	23.5	60.1	30.6	14.7	43.0	226.5	58.1
d657	657	48912	4.31	*Timeout*	31.63	*Timeout*	18.16	27.76	12.59	12.53	**2.15**
			52.3	60.1	20.7	60.1	31.2	15.3	54.0	3634.0	59.8
u724	724	41910	4.84	*Timeout*	30.34	*Timeout*	23.28	21.93	*Timeout*	13.83	**2.21**
			52.4	60.1	26.8	60.1	42.0	19.1	60.0	3612.8	36.9
rat783	783	8806	3.47	*Timeout*	30.89	*Timeout*	13.82	29.34	12.43	11.82	**2.49**
			52.4	60.1	34.2	60.1	51.2	24.8	33.2	3625.7	44.8
pr1002	1002	259045	**4.36**	*Timeout*	*Timeout*	*Timeout*	*Timeout*	27.98	*Timeout*	13.80	6.41
			52.4	60.1	60.1	60.1	60.1	47.2	60.0	3609.6	57.0
u1060	1060	224094	6.32	*Timeout*	*Timeout*	*Timeout*	*Timeout*	31.30	*Timeout*	11.38	**2.84**
			52.4	60.1	60.1	60.1	60.1	56.5	60.0	3679.4	57.5
vm1084	1084	239297	**4.54**	*Timeout*	*Timeout*	*Timeout*	*Timeout*	*Timeout*	*Timeout*	12.71	4.64
			52.4	60.1	60.1	60.1	60.1	60.0	60.0	3612.9	58.0
pcb1173	1173	56892	9.60	*Timeout*	*Timeout*	*Timeout*	*Timeout*	*Timeout*	17.40	17.03	**4.20**
			52.4	60.1	60.1	60.1	60.1	60.1	50.2	3613.8	39.7
d1291	1291	50801	9.66	*Timeout*	*Timeout*	*Timeout*	*Timeout*	*Timeout*	18.16	17.52	**3.00**
			52.4	60.1	60.1	60.1	60.1	60.1	52.6	3626.0	39.8
rl1304	1304	252948	11.10	*Timeout*	*Timeout*	*Timeout*	*Timeout*	*Timeout*	*Timeout*	17.87	**3.83**
			51.5	60.1	60.1	60.1	60.1	60.1	60.0	3617.2	58.0
rl1323	1323	270199	6.43	*Timeout*	*Timeout*	*Timeout*	*Timeout*	*Timeout*	20.64	20.59	**4.99**
			51.4	60.1	60.1	60.1	60.1	60.1	30.5	3609.3	44.9
nrw1379	1379	56638	6.76	*Timeout*	*Timeout*	*Timeout*	*Timeout*	*Timeout*	12.19	12.00	**3.87**
			52.5	60.1	60.1	60.1	60.1	60.1	59.7	3632.6	40.9
fl1400	1400	20127	6.24	*Timeout*	*Timeout*	*Timeout*	*Timeout*	*Timeout*	*Timeout*	**4.88**	5.27
			52.5	60.1	60.1	60.1	60.1	60.1	60.0	3626.3	57.8

Instance	n	Ref.	OR-Tools	EoH	ReEvo	MCTS-AHD	Refine Evo	PathWise	DRAGON 60s	DRAGON 1h	EvoAge
u1432	1432	152970	9.71 52.5	*Timeout* 60.1	*Timeout* 60.1	*Timeout* 60.1	*Timeout* 60.1	*Timeout* 60.1	*Timeout* 60.0	12.79 2827.1	**5.19** 47.6
fl1577	1577	22249	**5.82** 52.5	*Timeout* 60.1	*Timeout* 60.1	*Timeout* 60.0	*Timeout* 60.1	*Timeout* 60.1	13.70 42.5	13.61 3612.5	9.09 38.5
d1655	1655	62128	12.20 52.5	*Timeout* 60.1	*Timeout* 60.1	*Timeout* 60.1	*Timeout* 60.1	*Timeout* 60.1	17.07 53.6	16.54 3420.4	**4.54** 59.8
vm1748	1748	336556	12.42 52.5	*Timeout* 60.1	*Timeout* 60.1	*Timeout* 60.1	*Timeout* 60.1	*Timeout* 60.1	12.94 38.4	12.80 2956.4	**4.58** 55.3
u1817	1817	57201	10.80 51.6	*Timeout* 60.1	*Timeout* 60.1	*Timeout* 60.0	*Timeout* 60.1	*Timeout* 60.1	*Timeout* 60.0	19.92 2713.8	**4.59** 40.8
rl1889	1889	316536	11.61 52.5	*Timeout* 60.1	*Timeout* 60.1	*Timeout* 60.1	*Timeout* 60.1	*Timeout* 60.1	19.41 34.8	19.41 2886.5	**5.92** 57.8
d2103	2103	80450	6.06 52.6	*Timeout* 60.1	*Timeout* 60.1	*Timeout* 60.0	*Timeout* 60.1	*Timeout* 60.1	22.83 55.7	22.53 3452.7	**3.01** 58.7
u2152	2152	64253	16.86 52.6	*Timeout* 60.1	*Timeout* 60.1	*Timeout* 60.1	*Timeout* 60.1	*Timeout* 60.1	20.31 53.5	19.83 3010.0	**8.24** 57.2
u2319	2319	234256	12.56 52.6	*Timeout* 60.1	*Timeout* 60.1	*Timeout* 60.1	*Timeout* 60.1	*Timeout* 60.1	*Timeout* 60.0	7.42 3460.5	**2.66** 40.9
pr2392	2392	378032	19.54 52.6	*Timeout* 60.1	*Timeout* 60.1	*Timeout* 60.1	*Timeout* 60.1	*Timeout* 60.1	*Timeout* 60.0	16.62 3147.1	**4.92** 43.5
pcb3038	3038	137694	14.20 52.9	*Timeout* 60.1	*Timeout* 60.1	*Timeout* 60.1	*Timeout* 60.1	*Timeout* 60.1	*Timeout* 60.0	15.55 3712.8	**4.56** 35.2
fl3795	3795	28772	22.16 52.2	*Timeout* 60.1	*Timeout* 60.1	*Timeout* 60.1	*Timeout* 60.1	*Timeout* 60.1	*Timeout* 60.0	15.07 3575.5	**10.95** 48.8
fnl4461	4461	182566	*Timeout* 60.1	*Timeout* 60.1	*Timeout* 60.1	*Timeout* 60.1	*Timeout* 60.1	*Timeout* 60.1	*Timeout* 60.0	13.40 3655.2	**3.81** 48.8
rl5915	5915	565530	*Timeout* 60.1	*Timeout* 60.1	*Timeout* 60.1	*Timeout* 60.1	*Timeout* 60.1	*Timeout* 60.1	*Timeout* 60.0	23.09 3689.7	**5.26** 58.0
rl5934	5934	556045	*Timeout* 60.1	*Timeout* 60.1	*Timeout* 60.1	*Timeout* 60.1	*Timeout* 60.1	*Timeout* 60.1	22.57 44.5	22.54 3677.0	**5.82** 50.5
rl11849	11849	923288	*Timeout* 60.1	*Timeout* 60.1	*Timeout* 60.1	*Timeout* 60.1	*Timeout* 60.1	*Timeout* 60.1	*Timeout* 60.1	19.21 3789.8	**5.14** 59.8
usa13509	13509	19982859	*Timeout* 60.1	*Timeout* 60.1	*Timeout* 60.1	*Timeout* 60.1	*Timeout* 60.1	*Timeout* 60.1	*Timeout* 60.1	14.45 3847.6	**7.70** 51.2
brd14051	14051	469385	*Timeout* 60.1	*Timeout* 60.1	*Timeout* 60.1	*Timeout* 60.1	*Timeout* 60.1	*Timeout* 60.1	*Timeout* 60.1	13.02 3896.4	**4.58** 60.0
d15112	15112	1573084	*Timeout* 60.1	*Timeout* 60.1	*Timeout* 60.1	*Timeout* 60.1	*Timeout* 60.1	*Timeout* 60.1	*Timeout* 60.1	**13.01** 3862.2	24.05 7.1
d18512	18512	645238	*Timeout* 60.1	*Timeout* 60.1	*Timeout* 60.1	*Timeout* 60.1	*Timeout* 60.1	*Timeout* 60.2	*Timeout* 60.1	13.14 3501.4	**4.44** 60.0

Table 3: CVRP: per-instance results on 20 CVRPLIB instances. Each cell reports Gap (%) above runtime (s).

Instance	n	Ref.	OR-Tools	EoH	ReEvo	MCTS-AHD	Refine Evo	PathWise	DRAGON 60s	DRAGON 1h	EvoAge
X-n110-k13	109	14971	**1.68** 52.3	18.24 6.6	20.12 5.6	81.20 16.2	32.72 5.7	100.11 6.6	*Failed* 7.0	18.26 3821.2	6.50 45.3
X-n162-k11	161	14138	**3.20** 52.4	21.55 8.5	20.65 8.9	159.15 40.1	61.68 9.1	106.63 10.5	*Failed* 6.7	19.57 3758.6	7.77 60.0
X-n223-k34	222	40437	7.98 52.4	21.15 13.9	18.66 13.5	134.89 56.8	49.40 13.5	90.01 15.7	*Failed* 8.2	28.48 3697.9	**4.49** 57.6
X-n284-k15	283	20215	**8.26** 51.4	31.07 17.6	24.16 17.4	*Timeout* 60.1	92.84 17.3	148.29 19.2	*Failed* 7.6	29.92 2594.4	8.77 58.3
X-n420-k130	419	107798	5.65 52.4	14.19 41.3	12.78 39.9	*Timeout* 60.1	32.02 40.0	61.36 46.7	*Failed* 8.1	23.98 3772.5	**4.25** 58.2
X-n459-k26	458	24139	14.13 51.4	38.34 38.9	29.36 37.7	*Timeout* 60.1	110.62 37.7	155.14 46.0	*Failed* 7.7	31.12 3417.7	**8.11** 60.0
X-n513-k21	512	24201	12.33 52.4	47.49 46.6	40.50 45.1	*Timeout* 60.1	178.22 44.6	426.80 56.2	*Failed* 8.5	32.13 3331.6	**10.32** 59.8
X-n670-k130	669	146332	**4.68** 52.5	*Timeout* 60.1	*Timeout* 60.1	*Timeout* 60.1	*Timeout* 60.1	*Timeout* 60.1	*Failed* 7.2	44.77 3764.8	9.42 60.0
X-n701-k44	700	81923	8.94 51.4	*Timeout* 60.1	*Timeout* 60.1	*Timeout* 60.1	*Timeout* 60.1	*Timeout* 60.1	*Failed* 9.0	14.69 3626.4	**3.56** 59.5
X-n783-k48	782	72386	9.86 51.4	*Timeout* 60.1	*Timeout* 60.1	*Timeout* 60.1	*Timeout* 60.1	*Timeout* 60.1	*Failed* 8.8	30.47 3773.1	**5.46** 59.7
X-n801-k40	800	73305	**4.16** 52.4	*Timeout* 60.1	*Timeout* 60.1	*Timeout* 60.1	*Timeout* 60.1	*Timeout* 60.1	*Failed* 7.4	9.22 3772.8	4.18 59.8

Instance	n	Ref.	OR-Tools	EoH	ReEvo	MCTS-AHD	Refine Evo	PathWise	DRAGON 60s	DRAGON 1h	EvoAge
X-n1001-k43	1000	72355	9.74 52.4	*Timeout* 60.1	*Timeout* 60.1	*Timeout* 60.1	*Timeout* 60.1	*Timeout* 60.1	*Failed* 36.1	18.97 3732.7	**6.66** 59.8
XML1000_1335_01	1000	63968	8.33 52.4	*Timeout* 60.1	*Timeout* 60.1	*Timeout* 60.1	*Timeout* 60.1	*Timeout* 60.1	*Failed* 9.1	14.52 3690.1	**6.13** 59.8
XML1500_2114_01	1500	99781	5.08 52.5	*Timeout* 60.1	*Timeout* 60.1	*Timeout* 60.1	*Timeout* 60.1	*Timeout* 60.1	*Failed* 8.3	12.43 3768.9	**4.43** 58.5
XML2500_2145_01	2500	112858	19.90 51.8	*Timeout* 60.1	*Timeout* 60.1	*Timeout* 60.2	*Timeout* 60.1	*Timeout* 60.1	*Failed* 9.1	37.99 3621.3	**5.22** 59.9
XML3500_3335_01	3500	284329	4.28 52.2	*Timeout* 60.1	*Timeout* 60.1	*Timeout* 60.2	*Timeout* 60.1	*Timeout* 60.1	*Failed* 11.9	5.66 3614.6	**3.28** 59.8
XML4500_3233_01	4500	602565	*Timeout* 60.2	*Timeout* 60.1	*Timeout* 60.2	*Timeout* 60.3	*Timeout* 60.1	*Timeout* 60.2	*Failed* 51.7	3.68 3658.6	**2.57** 59.9
XML5000_1321_01	5000	1466910	*Timeout* 60.1	*Timeout* 60.2	*Timeout* 60.2	*Timeout* 60.3	*Timeout* 60.1	*Timeout* 60.2	*Failed* 12.6	6.90 3649.5	**2.80** 59.9
XML5000_2224_01	5000	315739	*Timeout* 60.1	*Timeout* 60.1	*Timeout* 60.2	*Timeout* 60.3	*Timeout* 60.2	*Timeout* 60.2	*Failed* 12.6	7.11 3675.9	**1.80** 59.9
XML5000_3135_01	5000	396487	*Timeout* 60.1	*Timeout* 60.2	*Timeout* 60.1	*Timeout* 60.3	*Timeout* 60.1	*Timeout* 60.2	*Failed* 16.1	4.88 3608.6	**2.91** 59.9

Table 4: BPP: per-instance results on five Weibull instances. Each cell reports Gap (%) above runtime (s).

Instance	n	Ref.	CP-SAT	EoH	ReEvo	MCTS-AHD	Refine Evo	PathWise	DRAGON 60s	DRAGON 1h	EvoAge
test_0	5000	2012	**0.00** 1.4	0.15 34.8	0.99 42.3	*Timeout* 60.1	*Timeout* 60.1	1.29 50.3	*Failed* 12.9	0.40 1707.7	0.40 46.8
test_1	5000	1983	**0.05** 1.2	0.15 34.5	1.01 46.8	*Timeout* 60.1	*Timeout* 60.1	1.31 51.8	*Timeout* 60.0	*Timeout* 3600.0	0.35 60.0
test_2	5000	1978	**0.00** 2.3	0.15 35.3	1.01 44.4	*Timeout* 60.1	*Timeout* 60.1	1.62 50.8	*Timeout* 60.0	0.30 3412.8	0.30 42.6
test_3	5000	1986	**0.00** 1.2	0.15 35.4	1.06 46.1	*Timeout* 60.1	*Timeout* 60.1	1.21 51.6	*Failed* 30.7	0.30 2654.7	0.25 21.1
test_4	5000	1980	**0.05** 1.0	0.10 37.0	1.01 45.6	*Timeout* 60.1	*Timeout* 60.1	1.31 50.8	*Timeout* 60.0	0.30 3258.2	0.25 30.0

Table 5: MKP: per-instance results on ten MKP instances. Each cell reports Gap (%) above runtime (s).

Instance	$n \times m$	Ref.	DRAGON 60s	DRAGON 1h	EvoAge
mkp_393_5_0	393×5	5455	0.51 56.6	0.51 2566.8	**0.00** 2.5
mkp_565_10_0	565×10	11478	0.55 35.5	0.55 3613.3	**0.03** 55.4
mkp_3594_20_0	3594×20	37706	0.04 50.7	0.04 813.2	**0.03** 57.4
mkp_3851_20_0	3851×20	37058	0.11 23.8	0.10 691.9	**0.04** 44.4
mkp_6870_50_0	6870×50	83880	0.06 43.7	0.06 568.5	**0.05** 53.7
mkp_6922_50_0	6922×50	80416	*Timeout* 60.0	0.10 834.4	**0.06** 53.9
mkp_8864_50_0	8864×50	85266	0.13 58.4	0.13 669.4	**0.05** 56.6
mkp_10729_80_0	10729×80	127756	*Timeout* 60.0	**0.03** 1255.4	**0.03** 58.1
mkp_12397_50_0	12397×50	105592	0.03 36.2	0.03 449.2	**0.02** 55.0
mkp_30018_100_0	30018×100	235483	*Timeout* 60.0	**0.01** 818.1	0.02 55.0

A.3 INTERPRETABLE SKILL EXAMPLES

TSP: Allocate the improvement budget

When to apply: A feasible tour is available for a medium-size instance with corridor, mixed-density, or clustered structure, and the remaining search budget is limited.

Principle:

1. Apply a short local-improvement stage to the initial tour, then use iterated local search to continue improving the best available solution.
2. End the first stage when it converges and give the remaining budget to iterated search, which may continue to benefit from additional time.
3. When the time limit is binding, omit a second local-improvement stage after iterated search if its marginal benefit is small; retain that budget for the search stage instead.

This Skill guides the Planner's ordering of improvement stages and allocation of search time. It links the decision to retain or remove a stage to its marginal benefit under the available budget.

CVRP: Start with a fast savings construction

When to apply: A large capacitated routing instance requires many vehicles, and an initial capacity-feasible routing solution is needed.

Principle:

1. Use a savings-based construction to build the initial routes, restricting merge candidates to a modest nearest-neighbour list.
2. Allocate enough time to finish construction, rather than treating the full solving budget as useful construction time.
3. If further improvement is worthwhile, pass the resulting incumbent to an improvement or diversification stage and spend the remaining budget there.

This Skill connects problem scale and vehicle demand to the choice of initial construction. It guides the Planner to obtain a feasible starting solution economically and reserve time for subsequent improvement.

BPP: Condition bin reduction on the initial packing

When to apply: A feasible packing already exists, and the next decision is whether to spend time reducing its bin count.

Principle:

1. When the packing comes from a weak sequential construction, apply a bounded bin-reduction procedure to the current best packing.
2. Retain the result only if it does not increase the number of bins, and stop the procedure when it converges.
3. When a strong construction already produced the incumbent, the same bin-reduction procedure may offer no improvement. If it converges without a gain, omit repeated calls and consider a different repacking approach only when additional search is justified.

This Skill makes refinement conditional on the source of the incumbent. It guides the Planner to distinguish a packing that may benefit from bin reduction from one for which repeating the same operation is unlikely to help.

MKP: Use bounded exchanges and stop at stagnation

When to apply: A feasible multiple-knapsack assignment exists, and unassigned items may fit or replace lower-profit packed items without violating capacity constraints.

Principle:

1. Start from the best feasible assignment. Perform a bounded refinement pass that adds unassigned items or makes feasible one-for-one replacements.
2. Accept a change only when total profit increases and all knapsack capacities remain satisfied; preserve the best assignment throughout the pass.
3. If the pass converges, avoid automatically chaining another pass, balancing step, or repeated construction. Continue only when unfinished search or measured progress gives a reason to spend more time.

This Skill combines a feasible improvement mechanism with a stopping rule. It guides refinement of the current assignment while protecting accumulated profit and avoiding repeated computation after convergence.