REAL-AHD: PLANNING OVER RELATIONAL ALGORITHM LANDSCAPE FOR LLM-BASED AUTOMATIC HEURISTIC DESIGN

Anonymous authors
Paper under double-blind review

ABSTRACT

Large Language Models (LLMs) have enabled a new paradigm of Automatic Heuristic Design (AHD), in which heuristics for combinatorial optimization problems (COPs) are generated as executable programs and improved through performance feedback. Existing methods either compress the search history into an elite population, discarding the relations produced along the way, or impose a single structure that captures only one view of the algorithm space. Moreover, they underutilize this history, neither adaptively allocating evaluations across families of similar algorithms nor jointly selecting parents and operators. We propose ReAL-AHD, which represents the discovered algorithm space as a **Re**lational **A**lgorithm **L**andscape with two complementary relations: a derivation graph that records lineages and supports credit assignment, and a similarity graph that groups heuristics into dynamic algorithm families based on PreferSim, which compares action preferences under aligned decision states. On this landscape, hierarchical graph planning turns accumulated history into the explicit basis for search decisions: family planning allocates the evaluation budget across families, node–operator planning jointly selects the parent node and the operator of each expansion, and graph-aware memory reuses relevant past outcomes. Across five COPs and three heuristic paradigms, ReAL-AHD consistently outperforms state-of-the-art LLM-based AHD methods and converts the evaluation budget into improvement efficiently.

1 INTRODUCTION

Large Language Models (LLMs) have enabled a new paradigm of Automatic Heuristic Design (AHD) for combinatorial optimization problems (COPs): instead of labor-intensive, expertise-dependent manual design (Burke et al., 2013; Stützle & López-Ibáñez, 2018), LLMs generate candidate algorithms and evaluators execute them to provide verifiable performance feedback (Lehman et al., 2022; Romera-Paredes et al., 2024; Liu et al., 2024). This paradigm opens an effectively unbounded program space, yet the evaluation budget remains limited (Cui et al., 2025; Wu et al., 2025). The central question is therefore not only what to generate next, but how to organize and exploit everything discovered so that each evaluation is spent where it is most informative.

Most existing LLM-based AHD methods adopt *population-based* search (Liu et al., 2024; Ye et al., 2024; Liu et al., 2026), which keeps only a set of promising heuristics. When candidates are replaced or discarded, their derivation paths and relations to other algorithms are lost, so the accumulated search history cannot be reused. Recent *structure-based* methods make part of this information explicit: derivation trees trace *where an algorithm comes from* (Zheng et al., 2025), whereas similarity-based niches identify *which algorithms are related* (Hu & Zhang, 2025). A single search structure, however, captures only one view of the algorithm space. These two views are complementary, yet no existing method exploits them jointly.

Organizing the history, however, does not by itself exploit it. Most existing methods (Liu et al., 2024; Zheng et al., 2025; Hu & Zhang, 2025) select parents and operators by fixed rules, or decide only one of the two. This is a missed opportunity: the value of an expansion lies in their combination, since each operator draws on different properties of a parent, and a parent that is promising under one operator may be unproductive under another. At a coarser level, algorithms with simi-

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.

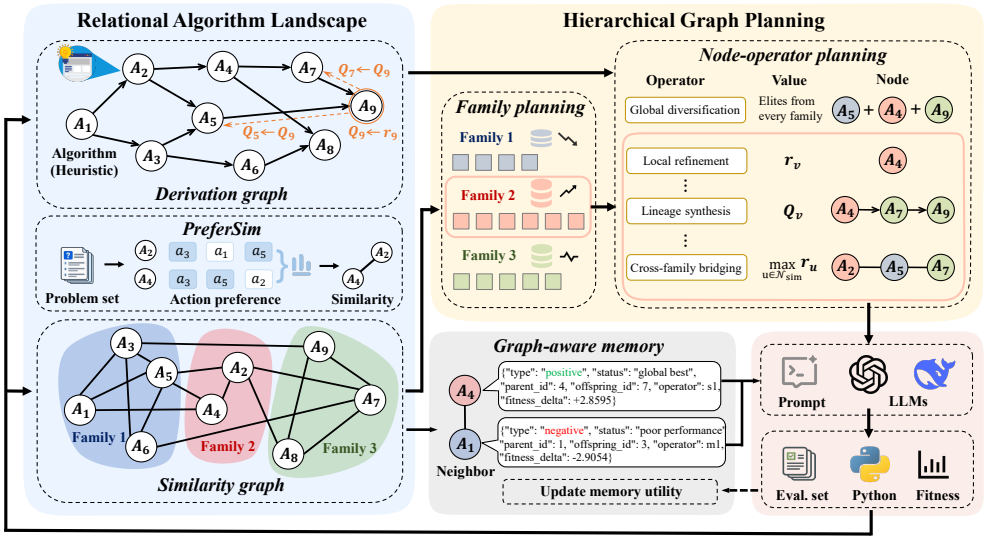

Figure 1: Overview of ReAL-AHD. The derivation graph records algorithm lineages and supports credit assignment, while the similarity graph uses PreferSim to group algorithms into dynamic families. At each round, family planning allocates evaluation budgets, and node–operator planning selects expansions. Evaluated offspring update the graphs and memory utility.

lar decision mechanisms form families, and the budget must balance exploiting productive families against exploring the rest; otherwise evaluations are wasted on saturated families. Existing methods therefore exploit an organized history only partially: they do not select parents and operators jointly, nor do they adaptively allocate the budget across algorithm families.

To bridge these gaps, we propose ReAL-AHD, which organizes discovered algorithms into a relational algorithm landscape that maintains both views jointly: a *derivation graph* records lineages and supports credit assignment, while a *similarity graph*, built on our *PreferSim* measure, groups algorithms into dynamic families. On this landscape, *hierarchical graph planning* exploits the relational search history at two levels: family planning adaptively allocates the evaluation budget across families to balance exploration against exploitation, and node–operator planning jointly selects the node and the operator for each expansion within a family. Graph-aware memory reuses relevant past outcomes. Search history thus becomes an explicit planning substrate rather than a passive archive.

Our contributions are summarized as follows:

- We introduce the relational algorithm landscape, a dual-graph representation that jointly models the derivation and similarity relations among discovered algorithms, together with PreferSim, which quantifies the similarity between two algorithms by comparing their action preferences under aligned decision states.
- We develop hierarchical graph planning, which couples family planning with node–operator planning in a two-level scheme and complements them with graph-aware memory to coordinate exploration and exploitation under a fixed search budget.
- We conduct extensive experiments on five COPs spanning three heuristic paradigms. ReAL-AHD consistently outperforms existing LLM-based AHD methods while improving the efficiency of the search process.

2 Related Work

LLM-based Automatic Heuristic Design. LLMs have emerged as powerful generators for heuristic design, extending the search space beyond manually predefined components. FunSearch (Romera-Paredes et al., 2024) couples LLM generation with evolutionary search, main-

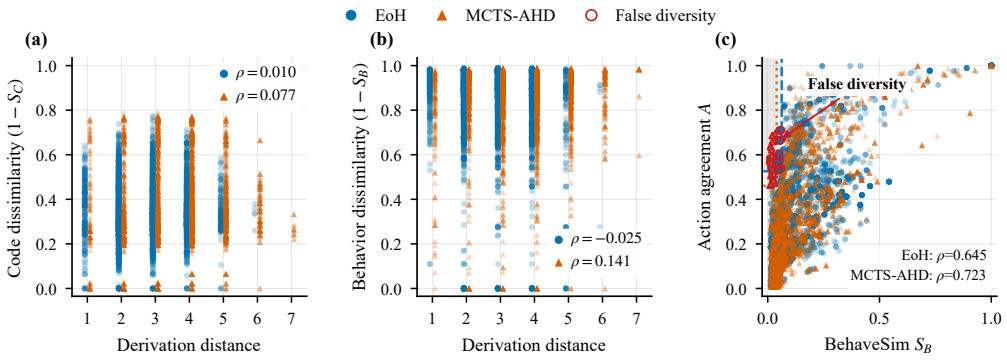

Figure 2: (a,b) Derivation distance versus CodeBLEU and BehaveSim dissimilarity. (c) BehaveSim versus action agreement measured on shared decision states. Each panel reports Spearman correlations for both archives. Red-outlined points fall in the bottom BehaveSim quartile but the top agreement quartile, revealing false diversity.

taining islands of programs and repeatedly prompting the LLM with high-performing candidates. EoH (Liu et al., 2024) represents each heuristic as a natural-language thought paired with executable code and evolves both jointly. Subsequent methods improve this population-based paradigm along different dimensions: ReEvo (Ye et al., 2024) adds verbal reflections on pairwise performance comparisons, HSEvo (Dat et al., 2025) balances exploration and exploitation with diversity-aware harmony search, and HiFo-Prompt (Chen et al., 2026) accumulates insights in a persistent pool while adaptively switching search regimes. More recently, structure-based methods make the algorithm-space structure explicit: MCTS-AHD (Zheng et al., 2025) replaces population management with Monte Carlo Tree Search, keeping all discovered heuristics in a derivation tree where temporarily underperforming nodes can be revisited and expanded, while PartEvo (Hu & Zhang, 2025) projects algorithms into a feature space and statically allocates search resources across similarity-based niches.

Algorithm Similarity. Measuring algorithm similarity helps identify redundancy, maintain diversity, and organize the growing collection of generated algorithms. Existing measures are mostly defined over code implementations: token-based methods such as BLEU (Papineni et al., 2002) capture textual overlap, embedding-based methods map programs into a continuous semantic space with pretrained code models such as CodeBERT (Feng et al., 2020), and CodeBLEU (Ren et al., 2020) further incorporates abstract syntax trees (ASTs) and data-flow structures. Beyond code implementations, Thought Embedding (Hu & Zhang, 2025) compares algorithmic ideas by encoding natural-language descriptions with BERT (Devlin et al., 2019). BehaveSim (Zhang & Lu, 2026) represents behavior as a sequence of intermediate solutions, termed a problem-solving trajectory (PSTraj), and compares trajectories using dynamic time warping (DTW).

Empirical Observation. We examine the search archives of EoH and MCTS-AHD on the TSP constructive heuristic paradigm. Ignoring link direction, we define the derivation distance d_D between two algorithms as the minimum number of parent–offspring links connecting them. For the same pairs, we compute the CodeBLEU and BehaveSim similarities S_C and S_B. Figure 2(a,b) reports the Spearman correlations $\rho(d_D, 1 - S_C)$ and $\rho(d_D, 1 - S_B)$, showing weak association between derivation distance and either dissimilarity measure in both archives. These results motivate modeling algorithm similarity explicitly alongside lineage. ReAL-AHD retains both relations and uses them jointly to plan search. We further compare BehaveSim with action agreement A, computed on a bank of 1,000 shared decision states. BehaveSim remains globally informative, attaining rank correlations of 0.645 and 0.723 with action agreement for EoH and MCTS-AHD, respectively. A considerable fraction of pairs, however, fall in the bottom BehaveSim quartile despite ranking in the top quartile of action agreement (Figure 2(c)). These *false diversity* pairs show that similar decisions on shared states can produce dissimilar solution trajectories. PreferSim directly compares the action preferences of two algorithms under aligned decision states to capture similarity. Additional details and results are provided in Appendix D.1.

3 METHOD

Figure 1 illustrates the overall workflow of ReAL-AHD, a framework that turns the archive accumulated during LLM-based heuristic search into an explicit relational landscape. We first formulate LLM-based AHD in Section 3.1. Section 3.2 then introduces the derivation and similarity topologies, PreferSim, and dynamic algorithm families. Finally, Section 3.3 presents hierarchical graph planning over this relational state.

3.1 PRELIMINARIES

Let $\mathcal{I}$ denote a target distribution of problem instances and $\mathcal{H}$ the program space induced by a predefined heuristic interface. A heuristic is represented as $h = (d, c)$, where d is a concise natural-language description of its algorithmic idea and c is executable code. Given an instance $I \sim \mathcal{I}$, an evaluator executes c and returns a scalar score $f(h, I)$. The objective is

$$h^\star = \arg\max_{h \in \mathcal{H}} F(h), \qquad F(h) = \mathbb{E}_{I \sim \mathcal{I}}[f(h, I)]\,. \tag{1}$$

For a minimization problem, we define f as the negative solution cost, so larger fitness is always better. In practice, $F(h)$ is estimated on a fixed evaluation set, and every evaluation consumes one unit from a budget of B evaluator calls.

3.2 RELATIONAL ALGORITHM LANDSCAPE

Derivation Topology. The derivation graph $\mathcal{G}^{(t)}_{\mathrm{der}} = (V_t, E^t_{\mathrm{der}})$ records the lineage of all non-duplicate heuristics discovered by round t. Each node $v \in V_t$ denotes a heuristic h_v, and a directed operator edge $e = (\mathcal{S}_e, o_e, v_e)$ denotes that operator o_e generated child v_e from source nodes $\mathcal{S}_e$. Since an operator may read several parents, e is a multi-source edge, and $\mathcal{G}_{\mathrm{der}}$ is a directed acyclic graph. Besides its fitness $r_v = F(h_v)$, each node maintains a visit count n_v and a lineage value Q_v. For a leaf, $Q_v = r_v$; otherwise,

$$Q_v = \frac{r_v + \sum_{u \in \mathrm{Ch}_{\mathrm{der}}(v)} Q_u}{1 + |\mathrm{Ch}_{\mathrm{der}}(v)|}, \tag{2}$$

where $\mathrm{Ch}_{\mathrm{der}}(v)$ denotes the direct derivation children of v. After an evaluation, visits and values are propagated through every derivation ancestor.

Similarity Topology. We maintain an undirected weighted similarity graph $\mathcal{G}^{(t)}_{\mathrm{sim}} = (V_t, E^t_{\mathrm{sim}}, W_t)$ with edge weights $w_{uv} = W_t(u, v) \in [0, 1]$. An undirected edge is retained when $w_{uv} \geq \tau$ or when either endpoint lies among the other's top-k_{nn} most similar nodes: $E^t_{\mathrm{sim}} = \big\{\{u, v\} : w_{uv} \geq \tau\big\} \cup E_{k_{\mathrm{nn}}}$. The threshold keeps strong relations, while the k_{nn} repair avoids isolated nodes and preserves a sparse local neighborhood for clustering.

Decision Preference Similarity. The edge weight w_{uv} is the similarity score assigned to heuristics h_u and h_v. Hence, the chosen similarity measure directly determines the topology of $\mathcal{G}_{\mathrm{sim}}$. Existing choices mainly compare implementations, as in CodeBLEU (Ren et al., 2020), or compare the trajectories produced by the heuristics, as in BehaveSim (Zhang & Lu, 2026). We introduce *PreferSim*, which compares what two heuristics prefer to do in aligned decision states. A small problem set $\mathcal{D}_p = \{I_m\}_{m=1}^M$ supplies shared decision states generated independently of h_u and h_v by task-specific controllers. At the ℓ-th state of I_m, $A_{m,\ell}(u, v)$ quantifies their action agreement. Let L^h_m be the number of valid preference records produced by h on I_m. We then define the edge weight as

$$w_{uv} = \mathrm{PreferSim}(h_u, h_v) = \frac{1}{M} \sum_{m=1}^{M} \frac{\sum_{\ell=1}^{\min(L^u_m, L^v_m)} A_{m,\ell}(u, v)}{\max(L^u_m, L^v_m)}. \tag{3}$$

Thus, PreferSim measures agreement on this reference state distribution, with unmatched preference records contributing zero. More details are provided in Appendix D.3.

Dynamic Algorithm Families. We extract families $\mathcal{F}_t = \{f_1, \ldots, f_{K_t}\}$ by spectral clustering (Shi & Malik, 2000; Ng et al., 2001) on the sparse affinity matrix, choosing $K_t \in [K_{\min}, K_{\max}]$ with the eigengap criterion (Von Luxburg, 2007; Zelnik-Manor & Perona, 2004). Let V_f denote the current members of family f. A new node is immediately assigned to the family with the largest mean similarity, and the full clustering is recomputed after graph growth or pruning. Families therefore act as dynamic planning units rather than permanent semantic classes, summarizing the mechanism regions currently visible in $\mathcal{G}_{\text{sim}}$, while $\mathcal{G}_{\text{der}}$ preserves how those regions were reached.

3.3 Hierarchical Graph Planning

Family Planning. Every generation attempt assigned to family f receives a graded outcome reward: 1.0 for a new global best, 0.8 for an elite admission, 0.2 for an ordinary valid offspring, and 0.0 for a poor or failed attempt. Let $\bar{R}_f$ and n_f denote the mean reward and attempt count in a sliding reward window (Fialho et al., 2010), and $N_t = \sum_f n_f$. With b_t evaluations already consumed, we define the remaining budget factor $\rho_t = \max\big(1 - \frac{b_t}{B}, \rho_{\min}\big)$ and score each family by

$$U_f = \bar{R}_f + c\rho_t \sqrt{\frac{\ln(N_t + 1)}{n_f + 1}}. \tag{4}$$

Here c is the shared exploration coefficient. The exploration bonus follows the upper-confidence-bound principle (Auer et al., 2002), with its strength scaled by the remaining budget. Family slots are distributed by a *softmax* over U_f; this reallocates effort without changing the evaluation budget of the round, and allows previously starved families to regain exploration pressure.

Node–Operator Planning. The next planning unit is a coupled node–operator action rather than a node choice followed by a separate operator choice. Different operators read different relations, so the appropriate value of a node depends on the operator that will expand it. For each family f and candidate operator o, we define

$$V_o(v) = \begin{cases} r_v, & o \in \{m1, m2, e2\}, \\ Q_v, & o = s1, \\ \max\limits_{\substack{u \in \mathcal{N}_{\text{sim}}(v) \\ f(u) \neq f(v)}} r_u, & o = s2. \end{cases} \tag{5}$$

Here $\mathcal{N}_{\text{sim}}(v)$ denotes the retained similarity neighbors of v. Thus, mutation and elite-inspiration operators value a node by its own quality, lineage synthesis favors a derivation region with consistently strong descendants, and similarity bridging values the best foreign mechanism directly reachable from the node. With $N_f = \sum_{u \in V_f} n_u$, the operator-conditioned node score is

$$U_o(v \mid f) = \text{norm}(V_o(v)) + c\rho_t \sqrt{\frac{\ln(N_f + 1)}{n_v + 1}}. \tag{6}$$

For every o, we sample an operator-conditioned node v_o from a *softmax* over $U_o(v \mid f)$. The sampled node is immediately paired with the operator, producing a candidate action (v_o, o). Consequently, mutation and elite inspiration $(m1, m2, e2)$, lineage synthesis $(s1)$, and cross-family bridging $(s2)$ all select their nodes within the given family. The global action $(e1)$ is the sole exception to family selection, with candidates drawn from the elites of all families. Operator $e2$ combines the selected node with the strongest elite from another family. Operator $s1$ receives the greedy high-Q descendant path that starts at the selected node and repeatedly follows the child with the largest lineage value until reaching a leaf. Operator $s2$ targets the other-family elite and adds intermediate nodes.

Graph-Aware Memory. Retrieving insights from past experience can guide subsequent LLM decisions (Zhao et al., 2024). The derivation and similarity relations determine which past evolutionary outcomes are relevant to a new generation. For every informative parent–offspring event, ReAL-AHD stores a compact edge-memory record. Before expanding v, it retrieves records incident to v and $\mathcal{N}_{\text{sim}}(v)$, ranks them by relational proximity and utility, and injects top-k_{mem} positive lessons and negative pitfalls into the prompt. The resulting offspring supplies closed-loop credit to every retrieved record. Implementation details are provided in Appendix D.5.

Algorithm 1 summarizes the resulting closed loop. Each round first uses family planning to distribute the available expansion slots. Node–operator planning converts these slots into executable (f, v, o) actions, while graph-aware memory supplies relevant guidance. Then the relational landscape and statistics are updated before the next round.

Algorithm 1 ReAL-AHD

Require: instance distribution $\mathcal{I}$, evaluator, LLM, budget B, operator library $\mathcal{O}$

```
Initialize the relational landscape G_der^(0), G_sim^(0)
while the budget B remains do
    Allocate family slots by Eq. 4
    Form a batch A of (f, v, o) actions by Eq. 6
    for all (f, v, o) ∈ A do
        Construct sources S, retrieve memories x
        Generate and evaluate h' ~ p_LLM( · | o, S, x)
    end for
    Update G_der^(t) and G_sim^(t), families F_t, and planning statistics
end while
return the best discovered heuristic
```

4 Experiments

4.1 Experimental Setup

Benchmarks and Datasets. We adopt standard evaluation protocols with distinct settings for evolution and testing to ensure robustness. Detailed configurations are provided in Appendix A.

- *Routing Problems*: Traveling Salesman Problem (TSP) (Matai et al., 2010) minimizes the total tour length, while Capacitated Vehicle Routing Problem (CVRP) (Toth & Vigo, 2002) adds vehicle-capacity constraints across multiple routes. Following Liu et al. (2024), we sample node coordinates from $[0, 1]^2$ (Kool et al., 2019) and generate demands and capacities using fixed task-specific generators. We evaluate on synthetic Euclidean instances of varying scales, and additionally use TSPLIB (Reinelt, 1991) and CVRPLIB (Uchoa et al., 2017) for out-of-distribution evaluation. Fitness is the negative solution cost.
- *Knapsack Problems*: 0–1 Knapsack Problem (KP) (Martello & Toth, 1990a) maximizes the total value of N items with values v_i and weights w_i under capacity W, while the Multidimensional Knapsack Problem (MKP) extends KP to multiple resource constraints. We generate KP instances of sizes $N \in \{50, 100, 200, 500\}$ following Zheng et al. (2025), and adopt the MKP setting of Ye et al. (2024) for ACO evaluation. Fitness is the total value subject to all capacity constraints.
- *Bin Packing Problem (BPP)* (Seiden, 2002): In the online setting, a stream of $L \in \{5\text{k}, 10\text{k}, 100\text{k}\}$ items must be assigned irrevocably to the fewest bins of capacity $C \in \{100, 300\}$. In the offline ACO setting, all $N \in \{500, 1000, 1500\}$ items are observed in advance; integer sizes are sampled from $[20, 100]$ and the bin capacity is 150. We report the excess-bin ratio relative to the volume lower bound (Martello & Toth, 1990b).

Heuristic Settings. We apply ReAL-AHD to evolve essential components within three standard frameworks. For constructive heuristics on TSP, CVRP, KP, and Online BPP, the system evolves the decision rule that selects or prioritizes a feasible action at each construction step. For Ant Colony Optimization (ACO) (Dorigo et al., 2006) on TSP, CVRP, MKP, and Offline BPP, it evolves the heuristic-information matrix or vector η from task features, while pheromone updates and solution sampling remain fixed. For Guided Local Search (GLS) (Voudouris & Tsang, 1999) on TSP, it evolves the edge-guidance logic used by the fixed perturbation and local-search procedure. The exact problem descriptions and function signatures are given in Appendix B.

Baselines and Implementation. We compare ReAL-AHD against three categories: (a) handcrafted heuristics (e.g., Greedy (Rosenkrantz et al., 1977), Best Fit (Johnson et al., 1974), vanilla

Table 1: Comparison of constructive heuristics on TSP and KP. Each objective averages 64 test instances and the reference values are obtained with LKH-3 (Helsgaun, 2017) for TSP and OR-Tools (Perron & Furnon, 2025) for KP. Bold indicates the best result in each column.

Task	TSP						KP							
Test sets	N=50		N=100		N=200		N=50, W=12.5		N=100, W=25		N=200, W=25		N=500, W=25	
Methods	Obj.↓	Gap	Obj.↓	Gap	Obj.↓	Gap	Obj.↑	Gap	Obj.↑	Gap	Obj.↑	Gap	Obj.↑	Gap
Reference	5.695	–	7.754	–	10.677	–	20.171	–	40.175	–	58.038	–	91.622	–
Greedy	6.956	22.15%	9.684	24.93%	13.487	26.34%	20.113	0.282%	40.129	0.115%	57.980	0.099%	91.590	0.034%
POMO	**5.715**	**0.35%**	**7.781**	**0.35%**	13.899	30.18%	20.160	0.053%	40.165	0.024%	57.908	0.225%	91.054	0.619%
Population-based AHD														
EoH	6.454	13.35%	8.951	15.44%	12.536	17.41%	20.160	0.055%	40.163	0.030%	58.023	0.026%	91.612	0.011%
ReEvo	6.296	10.57%	8.732	12.52%	12.228	14.53%	20.159	0.054%	40.163	0.030%	57.985	0.093%	91.525	0.106%
HSEvo	6.369	11.87%	8.824	13.80%	12.320	15.38%	20.166	0.021%	40.169	0.014%	58.031	0.012%	91.617	0.005%
HiFo-Prompt	6.223	9.28%	8.627	11.27%	12.024	12.62%	20.164	0.039%	40.169	0.014%	58.026	0.019%	91.596	0.028%
Structure-based AHD														
MCTS-AHD	6.306	10.76%	8.714	12.39%	12.147	13.78%	20.158	0.061%	40.164	0.028%	58.022	0.027%	91.609	0.013%
PartEvo	6.308	10.78%	8.704	12.26%	12.181	14.09%	20.163	0.038%	40.168	0.017%	58.028	0.017%	91.614	0.009%
OpenEvolve	6.300	10.64%	8.773	13.16%	12.339	15.57%	20.165	0.029%	40.168	0.017%	58.010	0.047%	91.590	0.035%
Ours	6.124	7.55%	8.481	9.40%	**11.882**	**11.30%**	**20.171**	**0.000%**	**40.172**	**0.006%**	**58.034**	**0.006%**	**91.619**	**0.003%**

ACO/GLS); (b) Neural Combinatorial Optimization (NCO) methods (e.g., POMO (Kwon et al., 2020), DeepACO (Ye et al., 2023)); and (c) LLM-based AHD frameworks (FunSearch (Romera-Paredes et al., 2024), EoH (Liu et al., 2024), ReEvo (Ye et al., 2024), HSEvo (Dat et al., 2025), MCTS-AHD (Zheng et al., 2025), PartEvo (Hu & Zhang, 2025), and HiFo-Prompt (Chen et al., 2026)). We additionally include OpenEvolve (Novikov et al., 2025; Sharma, 2025) as a strong open-source baseline. For each method, we perform three independent runs, each comprising $B = 500$ attempted evaluations. The LLM backbone is DeepSeek-V4-Flash (DeepSeek-AI, 2026b), with temperature 1.0. To test robustness across LLM backbones, we additionally evaluate methods with GPT-5-nano (Singh et al., 2025) in Appendix E.5. Appendix C provides hyperparameters and more details.

4.2 Main Results

Constructive Heuristics. The constructive paradigm builds a feasible solution through a sequence of local decisions. We evolve the decision rule at a training scale and apply the discovered program to every test size without additional search.

TSP and KP. Table 1 reports the performance on synthetic instances. On TSP50 and TSP100, the NCO method POMO remains strongest, consistent with the advantage of a learned policy near its training distribution. Among LLM-based AHD methods, ReAL-AHD achieves the best objective at all three sizes and also yields the best overall result on TSP200. At this scale, ReAL-AHD attains an objective of 11.882, improving upon the closest AHD baseline, HiFo-Prompt, and reducing the reference gap from 12.62% to 11.30%. The same cross-size stability is even clearer on KP, where ReAL-AHD attains the best objective at every reported size, with reference gaps between 0.000% and 0.006%. Although these absolute KP margins are necessarily small near the reference solution, their persistence from $N = 50$ to $N = 500$ indicates that the discovered selection rule does not overfit to the evolution scale. Results on TSPLIB, CVRPLIB, and supplementary constructive CVRP are provided in Appendix E.3.

Online BPP. The Online BPP tests whether a priority rule remains reliable over a long sequence of irrevocable decisions. As shown in Table 2, ReAL-AHD achieves the lowest excess-bin ratio in five of the six settings. At $C = 100$, its excess-bin ratio drops from 0.70% to 0.35% and 0.02% as the stream grows from 5k to 10k and 100k items, remaining ahead of the strongest alternative in every column. The larger capacity enlarges the feasible-bin set and makes early decisions more ambiguous: at 5k_C300, HiFo-Prompt is marginally better. ReAL-AHD nevertheless becomes best at 10k and 100k, reaching 1.06% and 0.12%, respectively. This trend suggests that relational planning is particularly useful when small local placement biases can compound over a long horizon.

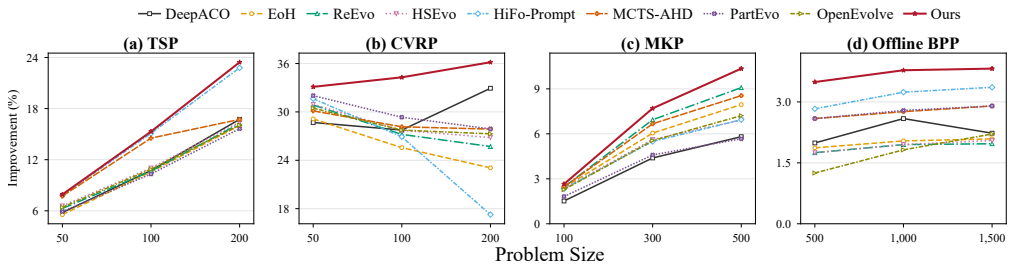

Figure 3: Relative objective improvement (%) over vanilla ACO across problem sizes (higher is better). For minimization tasks, improvement is the relative cost reduction, whereas for MKP it is the relative objective increase.

Ant Colony Optimization. Replacing only the handcrafted heuristic-information component in a fixed ACO shell (Figure 3), ReAL-AHD improves vanilla ACO most in every panel and scale: TSP gains rise from 7.92% to 23.44%, above HiFo-Prompt; CVRP stays above 33% and reaches 36.16% at $N = 200$; MKP rises from 2.65% to 10.35%, widening the margin over the closest baseline from 0.18 to 1.27 percentage points; Offline BPP remains stable at 3.49%, 3.78%, and 3.82% for $N = 500$, 1000, and 1500. Thus, the benefit extends beyond direct construction to heuristic information inside a stochastic solver, especially as TSP and MKP scale. Supplementary TSP-GLS evaluations are reported in Appendix E.4.

Table 2: Online BPP excess-bin ratio relative to the lower bound (lower is better). Bold indicates the best result in each column.

Methods	C=100			C=300		
	5k	10k	100k	5k	10k	100k
First Fit	4.31%	4.36%	4.11%	4.89%	4.45%	4.25%
Best Fit	3.95%	4.05%	3.86%	4.43%	4.02%	4.01%
Population-based AHD						
FunSearch	0.89%	0.50%	0.05%	2.91%	2.08%	0.20%
EoH	3.82%	3.87%	3.68%	4.19%	3.90%	3.78%
ReEvo	1.49%	0.75%	0.37%	3.03%	1.93%	0.58%
HSEvo	3.32%	3.17%	3.09%	3.91%	3.48%	3.22%
HiFo-Prompt	0.82%	0.42%	0.03%	**2.20%**	1.14%	0.14%
Structure-based AHD						
MCTS-AHD	0.71%	0.43%	0.13%	3.18%	2.35%	1.84%
PartEvo	1.30%	1.20%	1.03%	3.04%	2.50%	2.33%
OpenEvolve	0.79%	0.43%	0.04%	2.27%	1.27%	0.18%
Ours	**0.70%**	**0.35%**	**0.02%**	2.21%	**1.06%**	**0.12%**

4.3 Ablation Studies

We evaluate the contributions of the relational landscape, hierarchical planning, and the similarity measure on TSP and Online BPP, with results averaged over three independent runs. For the relation ablations, removing the derivation graph also reduces $s1$ to choosing a random path, whereas removing the similarity graph additionally reduces $s2$ to bridging with random candidates. For the planning and memory ablations, uniform allocation replaces family planning, full operator expansion after random node selection replaces node–operator planning, and the final variant disables the retrieval of memories. More results of ablation experiments are presented in Appendix E.6.

Relational Landscape and Hierarchical Planning. Table 3 shows that the two relational graphs provide complementary benefits. Removing only the derivation relation increases the TSP100 gap from 9.40% to 9.96% and the BPP ratios from 0.70% to 0.83%. Removing the similarity relation leads to a larger degradation on TSP, with the gaps increasing to 11.26% and 12.37%. This result suggests that algorithm neighborhoods and their induced families capture information that cannot be obtained from ancestry alone. These results show that derivation and similarity provide distinct views of the algorithm space, and that jointly retaining both relations is important for achieving the full performance gain. Both planning levels and graph-aware memory contribute to the performance of ReAL-AHD, with node–operator planning producing the largest improvement. This result supports the design principle that the value of a parent node depends on the operator that expands it. Replacing family planning with uniform allocation causes a smaller but consistent degradation, indicating that adaptive budget allocation remains beneficial after the family structure has been established. Removing memory also increases both TSP gaps and degrades the BPP ratio at 10k_C100.

Algorithm Similarity. The final block replaces PreferSim with CodeBLEU or BehaveSim, while keeping graph sparsification and the planning procedure unchanged. Compared with CodeBLEU,

Table 3: Ablations on relational algorithm landscape, hierarchical graph planning and PreferSim. TSP columns report the reference gap and BPP columns report the excess-bin ratio (lower is better).

Variants	TSP		Online BPP	
	N=100	N=200	$5k_C100$	$10k_C100$
ReAL-AHD	9.40%	11.30%	0.70%	0.35%
w/o derivation	9.96%	11.37%	0.83%	0.45%
w/o similarity	11.26%	12.37%	0.83%	0.52%
w/o family planning	9.77%	11.41%	0.81%	0.43%
w/o node–operator planning	12.24%	13.49%	0.97%	0.67%
w/o graph-aware memory	10.83%	11.99%	0.70%	0.62%
CodeBLEU	11.59%	13.40%	0.98%	0.49%
BehaveSim	11.39%	12.68%	0.83%	0.50%

PreferSim reduces the TSP gaps and the BPP ratios. BehaveSim outperforms CodeBLEU in three of the four columns, but remains behind PreferSim. These results indicate that similarity based on aligned action preferences induces a search topology that is more informative for algorithm discovery than those based on code or solution trajectory.

4.4 Efficiency Analysis

We compare token cost, final heuristic performance, and the progress made under the same evaluation budget on constructive TSP100. As shown in Figure 4(a), ReAL-AHD obtains the lowest test objective, consuming fewer tokens than competitive baselines. This cost is only 0.4% higher than that of MCTS-AHD, while reducing final objective significantly. Consequently, ReAL-AHD lies on the Pareto front in the trade-off between performance and token usage. Figure 4(b) further shows that ReAL-AHD converts evaluations into improvement more rapidly.

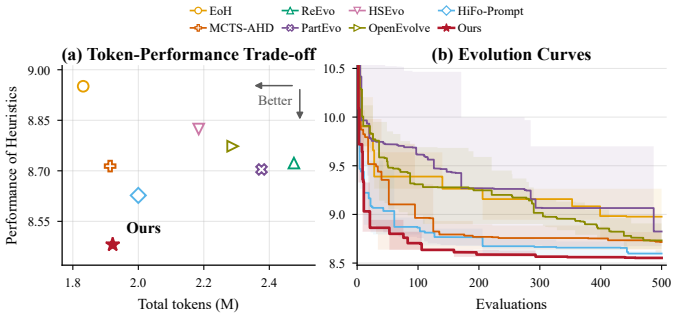

Figure 4: Efficiency analysis of constructive TSP. (a) Trade-off between final performance and token usage. (b) Evolution of the objective over evaluations.

5 Conclusion

We presented ReAL-AHD, an LLM-based AHD framework that represents the discovered algorithm space as a relational algorithm landscape, in which a derivation graph records lineages and credit assignment while a similarity graph, built on the proposed PreferSim measure, links algorithms that act alike under aligned decision states. Hierarchical graph planning turns this accumulated history into an explicit planning substrate, allocating the budget across dynamic algorithm families and selecting node–operator expansions within them, while graph-aware memory reuses relevant past outcomes to guide each generation. Across routing, knapsack, and bin packing problems and three heuristic paradigms, ReAL-AHD consistently outperformed existing LLM-based AHD methods and converted the evaluation budget into improvement more efficiently. One limitation is that the similarity computation depends on the chosen problem instances, and learning distribution-robust similarity measures remains future work.

AI USE STATEMENT

We used generative AI tools to assist with polishing the manuscript. As part of the evaluated AHD methods, LLMs generated and evolved executable heuristic programs, as described in Section 3. The reported performance values were obtained by executing candidate programs with task-specific evaluators on the specified benchmark instances. We have reviewed the AI-assisted manuscript and take responsibility for the research design, validation of generated code, interpretation of results, and final content.

ETHICS STATEMENT

This work studies automatic heuristic design for routing, knapsack, and bin packing problems using synthetic instances and established benchmark collections. It does not involve human participants or personal data. The intended benefit is to reduce manual algorithm-design effort and improve optimization quality. Generated heuristics should be checked for correctness and constraint satisfaction before use in real applications.

REPRODUCIBILITY STATEMENT

Section 3 describes the search procedure, with implementation details in Appendix D. Appendices A and B specify the problem definitions, data-generation procedures, evaluation and test splits, and heuristic interfaces. The settings and hyperparameters are reported in Section 4.1 and Appendix C.

REFERENCES

Florian Arnold and Kenneth Sörensen. Knowledge-guided local search for the vehicle routing problem. *Computers & Operations Research*, 105:32–46, 2019.

Peter Auer, Nicolò Cesa-Bianchi, and Paul Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine Learning*, 47(2–3):235–256, 2002. doi: 10.1023/A:1013689704352.

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.

Ignacio Castiñeiras, Milan De Cauwer, and Barry O'Sullivan. Weibull-based benchmarks for bin packing. In *International Conference on Principles and Practice of Constraint Programming*, pp. 207–222. Springer, 2012.

Chentong Chen, Mengyuan Zhong, Ye Fan, Jialong Shi, and Jianyong Sun. HiFo-Prompt: Prompting with hindsight and foresight for LLM-based automatic heuristic design. In *International Conference on Learning Representations*, 2026.

Wendi Cui, Jiaxin Zhang, Zhuohang Li, Hao Sun, Damien Lopez, Kamalika Das, Bradley A Malin, and Sricharan Kumar. Heuristic-based search algorithm in automatic instruction-focused prompt optimization: A survey. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 22093–22111, 2025.

Pham Vu Tuan Dat, Long Doan, and Huynh Thi Thanh Binh. HSEvo: elevating automatic heuristic design with diversity-driven harmony search and genetic algorithm using llms. In *Proceedings of the Thirty-Ninth AAAI Conference on Artificial Intelligence*. AAAI, 2025.

DeepSeek-AI. Deepseek-v4.1-flash: Pushing the limits of kv cache compression, 2026a.

DeepSeek-AI. Deepseek-v4: Towards highly efficient million-token context intelligence, 2026b.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 conference of the North American chapter of the association for computational linguistics: human language technologies, volume 1 (long and short papers)*, pp. 4171–4186, 2019.

Marco Dorigo, Mauro Birattari, and Thomas Stützle. Ant colony optimization. *IEEE computational intelligence magazine*, 1(4):28–39, 2006.

Zhangyin Feng, Daya Guo, Duyu Tang, Nan Duan, Xiaocheng Feng, Ming Gong, Linjun Shou, Bing Qin, Ting Liu, Daxin Jiang, et al. Codebert: A pre-trained model for programming and natural languages. In *Findings of the association for computational linguistics: EMNLP 2020*, pp. 1536–1547, 2020.

Álvaro Fialho, Luis Da Costa, Marc Schoenauer, and Michèle Sebag. Analyzing bandit-based adaptive operator selection mechanisms. *Annals of Mathematics and Artificial Intelligence*, 60(1–2): 25–64, 2010. doi: 10.1007/s10472-010-9213-y.

Google DeepMind. Gemini 3.5 Flash-Lite model card. `https://deepmind.google/models/model-cards/gemini-3-5-flash-lite/`, 2026. Accessed 2026-09-23.

Keld Helsgaun. An extension of the lin-kernighan-helsgaun tsp solver for constrained traveling salesman and vehicle routing problems. Technical report, Roskilde University, 2017. URL `http://www.akira.ruc.dk/~keld/research/LKH-3/LKH-3_REPORT.pdf`.

Qinglong Hu and Qingfu Zhang. Partition to evolve: Niching-enhanced evolution with LLMs for automated algorithm discovery. In D. Belgrave, C. Zhang, H. Lin, R. Pascanu, P. Koniusz, M. Ghassemi, and N. Chen (eds.), *Advances in Neural Information Processing Systems*, volume 38, Main Conference, pp. 154771–154817. Curran Associates, Inc., 2025. doi: 10.52202/085713-5177.

Ziyao Huang, Weiwei Wu, Kui Wu, Wei-Bin Lee, and Jianping Wang. Calm: Co-evolution of algorithms and language model for automatic heuristic design. In *International Conference on Learning Representations*, volume 2026, pp. 72468–72510, 2026.

Q. Huangfu and J. A. J. Hall. Parallelizing the dual revised simplex method. *Mathematical Programming Computation*, 10(1):119–142, 2018. doi: 10.1007/s12532-017-0130-5.

David S. Johnson, Alan Demers, Jeffrey D. Ullman, Michael R Garey, and Ronald L. Graham. Worst-case performance bounds for simple one-dimensional packing algorithms. *SIAM Journal on computing*, 3(4):299–325, 1974.

Wouter Kool, Herke van Hoof, and Max Welling. Attention, learn to solve routing problems! In *International Conference on Learning Representations*, 2019.

H. W. Kuhn. The hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2(1–2):83–97, 1955. doi: 10.1002/nav.3800020109.

Yeong-Dae Kwon, Jinho Choo, Byoungjip Kim, Iljoo Yoon, Youngjune Gwon, and Seungjai Min. Pomo: Policy optimization with multiple optima for reinforcement learning. *Advances in Neural Information Processing Systems*, 33:21188–21198, 2020.

Joel Lehman, Jonathan Gordon, Shawn Jain, Kamal Ndousse, Cathy Yeh, and Kenneth O. Stanley. Evolution through large models. *arXiv preprint arXiv:2206.08896*, 2022. URL `https://arxiv.org/abs/2206.08896`.

Fei Liu, Xialiang Tong, Mingxuan Yuan, Xi Lin, Fu Luo, Zhenkun Wang, Zhichao Lu, and Qingfu Zhang. Evolution of heuristics: towards efficient automatic algorithm design using large language model. In *Proceedings of the 41st International Conference on Machine Learning*, ICML'24, 2024.

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

Silvano Martello and Paolo Toth. *Knapsack problems: algorithms and computer implementations*. John Wiley & Sons, Inc., 1990a.

Silvano Martello and Paolo Toth. Lower bounds and reduction procedures for the bin packing problem. *Discrete applied mathematics*, 28(1):59–70, 1990b.

Rajesh Matai, Surya Prakash Singh, and Murari Lal Mittal. Traveling salesman problem: An overview of applications, formulations, and solution approaches. In Donald Davendra (ed.), *Traveling Salesman Problem, Theory and Applications*, pp. 1–24. InTech, 2010. doi: 10.5772/12909.

Moonshot AI. Kimi K2.7 Code. `https://huggingface.co/moonshotai/Kimi-K2.7-Code`, 2026. Model card. Accessed 2026-09-23.

Andrew Ng, Michael Jordan, and Yair Weiss. On spectral clustering: Analysis and an algorithm. *Advances in neural information processing systems*, 14, 2001.

Alexander Novikov, Ngân Vũ, Marvin Eisenberger, Emilien Dupont, Po-Sen Huang, Adam Zsolt Wagner, Sergey Shirobokov, Borislav Kozlovskii, Francisco JR Ruiz, Abbas Mehrabian, et al. Alphaevolve: A coding agent for scientific and algorithmic discovery. *arXiv preprint arXiv:2506.13131*, 2025.

OpenAI. GPT-5.6 system card. `https://deploymentsafety.openai.com/gpt-5-6`, 2026. Accessed 2026-09-23.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*, pp. 311–318, 2002.

Laurent Perron and Vincent Furnon. OR-Tools. Google, 2025. URL `https://developers.google.com/optimization/`.

Qwen Team. Qwen3.8-27B. `https://huggingface.co/Qwen/Qwen3.8-27B`, 2026. Model card. Accessed 2026-09-23.

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

Shuo Ren, Daya Guo, Shuai Lu, Long Zhou, Shujie Liu, Duyu Tang, Neel Sundaresan, Ming Zhou, Ambrosio Blanco, and Shuai Ma. Codebleu: a method for automatic evaluation of code synthesis. *arXiv preprint arXiv:2009.10297*, 2020.

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

Daniel J Rosenkrantz, Richard E Stearns, and Philip M Lewis, II. An analysis of several heuristics for the traveling salesman problem. *SIAM journal on computing*, 6(3):563–581, 1977.

Steven S Seiden. On the online bin packing problem. *Journal of the ACM (JACM)*, 49(5):640–671, 2002.

Asankhaya Sharma. OpenEvolve: an open-source evolutionary coding agent, 2025. URL `https://github.com/algorithmicsuperintelligence/openevolve`.

Jianbo Shi and J. Malik. Normalized cuts and image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(8):888–905, 2000. doi: 10.1109/34.868688.

Aaditya Singh, Adam Fry, Adam Perelman, Adam Tart, Adi Ganesh, Ahmed El-Kishky, Aidan McLaughlin, Aiden Low, AJ Ostrow, Akhila Ananthram, et al. Openai gpt-5 system card. *arXiv preprint arXiv:2601.03267*, 2025.

Thomas Stützle and Manuel López-Ibáñez. Automated design of metaheuristic algorithms. In *Handbook of metaheuristics*, pp. 541–579. Springer, 2018.

Paolo Toth and Daniele Vigo (eds.). *The Vehicle Routing Problem*. SIAM, 2002.

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.

Ulrike Von Luxburg. A tutorial on spectral clustering. *Statistics and computing*, 17(4):395–416, 2007.

Christos Voudouris and Edward Tsang. Guided local search and its application to the traveling salesman problem. *European journal of operational research*, 113(2):469–499, 1999.

Xingyu Wu, Sheng-hao Wu, Jibin Wu, Liang Feng, and Kay Chen Tan. Evolutionary computation in the era of large language model: Survey and roadmap. *IEEE Transactions on Evolutionary Computation*, 29(2):534–554, 2025. doi: 10.1109/TEVC.2024.3506731.

Haoran Ye, Jiarui Wang, Zhiguang Cao, Helan Liang, and Yong Li. DeepACO: Neural-enhanced ant systems for combinatorial optimization. *Advances in neural information processing systems*, 36:43706–43728, 2023.

Haoran Ye, Jiarui Wang, Zhiguang Cao, Federico Berto, Chuanbo Hua, Haeyeon Kim, Jinkyoo Park, and Guojie Song. ReEvo: Large language models as hyper-heuristics with reflective evolution. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024.

Lihi Zelnik-Manor and Pietro Perona. Self-tuning spectral clustering. *Advances in neural information processing systems*, 17, 2004.

Rui Zhang and Zhichao Lu. Rethinking code similarity for automated algorithm design with LLMs. In C. Vondrick, B. Hariharan, C. Raffel, L. Pinto, D. Yang, and A. Faust (eds.), *International Conference on Learning Representations*, pp. 88723–88745, 2026.

Andrew Zhao, Daniel Huang, Quentin 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.

Zhi Zheng, Zhuoliang Xie, Zhenkun Wang, and Bryan Hooi. Monte Carlo tree search for comprehensive exploration in LLM-based automatic heuristic design. In *Forty-second International Conference on Machine Learning*, 2025.

A Problem Definitions and Datasets

Traveling Salesman Problem (TSP). An instance consists of N cities with coordinates $x_i \in \mathbb{R}^2$ and a symmetric distance matrix D, where $D_{ij} = \|x_i - x_j\|_2$. A feasible solution is a permutation π of all cities, interpreted as a Hamiltonian cycle with $\pi_{N+1} = \pi_1$. Its objective is

$$\min_{\pi} \; z(\pi) = \sum_{i=1}^{N} D_{\pi_i, \pi_{i+1}}. \tag{7}$$

Synthetic coordinates are sampled independently and uniformly from $[0, 1]^2$.

Capacitated Vehicle Routing Problem (CVRP). An instance contains a depot indexed by 0, N customers, nonnegative demands q_i, and a common vehicle capacity C_{veh}. A solution is a collection of routes $\mathcal{R}$, each starting and ending at the depot, such that every customer is visited exactly once and $\sum_{i \in R \setminus \{0\}} q_i \le C_{\text{veh}}$ for every route $R \in \mathcal{R}$. We minimize the sum of Euclidean edge lengths over all routes. The synthetic setting permits up to N vehicles, so the number of routes is determined by the heuristic without a separate fleet-size penalty.

Bin Packing Problem (BPP). Given item sizes $s_1, \ldots, s_N \in [0, C]$ and identical bins of capacity C, BPP seeks a partition $\{B_1, \ldots, B_z\}$ of the items that minimizes z subject to $\sum_{i \in B_j} s_i \le C$ for every bin. Offline BPP exposes all item sizes before construction, whereas online BPP reveals one item at a time and requires an irrevocable assignment before the next arrival. For offline ACO, sizes are independent integers sampled uniformly from $\{20, \ldots, 100\}$ and $C = 150$. The online generator draws $X \sim \text{Weibull}(3, 45)$ and uses $s = \text{round}(\min\{X, 100\})$, following the Weibull benchmark construction (Castiñeiras et al., 2012). The reference is the volume lower bound $L_1 = \lceil \sum_i s_i / C \rceil$, which need not be attainable by an integral packing.

0–1 Knapsack Problem (KP). Each item i has weight $w_i > 0$ and value $v_i > 0$. With capacity W and binary selection variables x_i, the problem is

$$\max_{x \in \{0,1\}^N} \sum_{i=1}^{N} v_i x_i \quad \text{subject to} \quad \sum_{i=1}^{N} w_i x_i \le W. \tag{8}$$

Items may be omitted, and an item can be selected at most once. Evolution weights and values are sampled independently from the uniform distribution on $[0, 1]$. Capacity is $W = 12.5$ for $N = 50$ and $W = 25$ for $N \in \{100, 200, 500\}$.

Multidimensional Knapsack Problem (MKP). One item-selection vector must satisfy several resource constraints simultaneously, as specified in Section 4.1. For item values v_i, resource consumptions w_{ij}, and capacities b_j, it is defined by

$$\max_{x \in \{0,1\}^N} \sum_{i=1}^{N} v_i x_i \quad \text{subject to} \quad \sum_{i=1}^{N} w_{ij} x_i \le b_j, \qquad j = 1, \ldots, J. \tag{9}$$

Selecting item i consumes all components of its resource vector, and $x_i = 0$ leaves it unpacked; there is no assignment of items to separate containers. We use $J = 5$ constraints and normalized capacities $b_j = 1$.

Datasets. Table 4 specifies the evolution and test splits for combinations. TSP uses LKH-3 (Helsgaun, 2017) tour lengths as heuristic references, and KP uses the exact OR-Tools (Perron & Furnon, 2025) branch-and-bound solution of the integer-unit formulation. Constructive CVRP uses $C_{\text{veh}} = 40$, with evolution coordinates sampled independently and uniformly from $[0, 1]^2$ and integer customer demands uniformly from $\{1, \ldots, 9\}$. CVRP-ACO uses a time-limited OR-Tools routing reference. For MKP, the reference is the upper bound obtained by relaxing $x_i \in \{0, 1\}$ to $x_i \in [0, 1]$ and solving the resulting linear program with HiGHS (Huangfu & Hall, 2018). Both BPP settings use L_1. For the external routing benchmarks, the denominators are the published instance references from TSPLIB (Reinelt, 1991) and CVRPLIB (Uchoa et al., 2017).

Table 4: Dataset splits. $q \times n$ denotes q instances of size n.

Paradigm	Task	Evaluation set	Test set
Constructive	TSP	64×100	$64 \times \{50, 100, 200\}$
	KP	64×100	$64 \times \{50, 100, 200, 500\}$
	Online BPP	fixed L=5k streams	L=5k/10k/100k, $C \in \{100, 300\}$
	CVRP	64×100	$64 \times \{50, 100, 200\}$
ACO	TSP	10×50	$64 \times \{50, 100, 200\}$
	CVRP	10×50	$64 \times \{50, 100, 200\}$
	MKP	10×100	$64 \times \{100, 300, 500\}$
	Offline BPP	10×500	$64 \times \{500, 1000, 1500\}$
GLS	TSP	10×100	$64 \times \{50, 100, 200\}$

B Heuristic Paradigms

Constructive Heuristics. A constructive heuristic builds a feasible solution incrementally. Starting from an empty state, the algorithm repeatedly selects a component to add to the partial solution until the task is completed. Let S_t be the partial solution at step t, and $\mathcal{A}(S_t)$ be the set of available candidate actions. The decision is typically made by a scoring function $H(S_t, a)$:

$$a^* = \underset{a \in \mathcal{A}(S_t)}{\operatorname{argmax}} \; H(S_t, a). \tag{10}$$

On TSP, the rule chooses an unvisited city, and the evaluator closes the tour after all cities have been selected. On CVRP, the state additionally includes customer demands and residual vehicle capacity; returning to the depot starts a new route and restores capacity. On KP, the rule selects an index in the current compact item arrays, after which the evaluator removes that item and updates the remaining capacity. Online BPP exposes the arriving item and the residual capacities of feasible bins; the evolved function returns one priority per bin and the evaluator takes the highest-priority placement. LLM-based methods often focus on learning or evolving the mathematical form of $H(\cdot)$ to outperform standard human-designed greedy rules. The interfaces below specify the editable component.

Constructive interfaces (TSP, CVRP, KP, and online BPP)

```
def select_next_node(current_node, destination_node,
  unvisited_nodes, distance_matrix) -> int:
def select_next_node(current_node, depot, unvisited_nodes,
  rest_capacity, demands, distance_matrix) -> int:
def select_next_item(remaining_capacity, weights, values) -> int:
def priority(item, bins_remain_cap) -> np.ndarray:
```

Ant Colony Optimization. ACO constructs solutions by combining learned heuristic information η with a pheromone state τ^{ph} maintained by the solver (Dorigo et al., 2006). For feasible actions $a \in \mathcal{C}(s)$, the sampling probability has the form

$$p(a \mid s) = \frac{(\tau^{\mathrm{ph}}_{s,a})^{\alpha} \eta^{\beta}_{s,a}}{\sum_{b \in \mathcal{C}(s)} (\tau^{\mathrm{ph}}_{s,b})^{\alpha} \eta^{\beta}_{s,b}}, \tag{11}$$

where α and β control the pheromone and heuristic contributions. LLM-based methods evolve the function producing η from instance features. TSP and CVRP use edge-desirability matrices, MKP uses an item-desirability vector, and offline BPP uses an item-pair matrix that the solver aggregates over items already in the current bin. Table 5 gives the ant counts and iteration budgets used for evolution and test.

ACO interfaces (TSP, CVRP, MKP, and offline BPP)

```
def heuristics(distance_matrix) -> np.ndarray:
def heuristics(distance_matrix, coordinates, demands,
  capacity) -> np.ndarray:
def heuristics(prize, weight) -> np.ndarray:
def heuristics(demand, capacity) -> np.ndarray:
```

Guided Local Search. Guided Local Search (GLS) (Voudouris & Tsang, 1999; Arnold & Sörensen, 2019) alternates local improvement with feature penalties to escape local optima. For a TSP tour s, it augments the original objective $f(s)$ as

$$g(s) = f(s) + \lambda \sum_e p_e I_e(s), \tag{12}$$

where $I_e(s)$ indicates whether edge e belongs to the tour, p_e is its penalty counter, and λ controls the penalty strength. At a local optimum, standard GLS increments the penalties of edges maximizing

$$\mathrm{util}_e(s) = I_e(s)\frac{c_e}{1+p_e}, \tag{13}$$

where c_e is the edge length, and resumes local search under the augmented objective. In our AHD setting, the edge-length term in the utility calculation is replaced by evolved edge-guidance scores, while the local-search and penalty-update procedures remain fixed.

TSP-GLS interface

```
def heuristics(distance_matrix) -> np.ndarray:
```

C Experimental Details

Evaluation and Metrics. For a raw solution objective z and reference z_{ref}, we use $(z - z_{\mathrm{ref}})/z_{\mathrm{ref}}$ for minimization and $(z_{\mathrm{ref}} - z)/z_{\mathrm{ref}}$ for maximization. A gap is called an *optimality gap* only when the reference is certified optimal; otherwise we report a reference or bound gap. For BPP, the excess-bin ratio is $(z - L_1)/L_1$, where $L_1 = \lceil \sum_i s_i/C \rceil$ is the volume lower bound. The ACO improvement in Figure 3 is $(z_{\mathrm{ACO}} - z)/z_{\mathrm{ACO}}$ for minimization and $(z - z_{\mathrm{ACO}})/z_{\mathrm{ACO}}$ for maximization, so a positive number always indicates an improvement over vanilla ACO.

Aggregation across independent runs. The LLM-based AHD entries in Tables 1 and 2 are arithmetic means over three independent search runs. Within each run, the returned heuristic is selected by its evolution-set objective and then evaluated on the independent test set. For constructive TSP, let h_r denote the heuristic returned by run r, and let $\{I_{N,i}\}_{i=1}^{n_N}$ be the common test set at size N, with $n_N = 64$. The run-level test objective and reference gap are

$$\begin{aligned} z_{r,N} &= \frac{1}{n_N}\sum_{i=1}^{n_N} z(h_r, I_{N,i}), \\ g_{r,N} &= \frac{1}{n_N}\sum_{i=1}^{n_N} \frac{z(h_r, I_{N,i}) - z_{\mathrm{ref}}(I_{N,i})}{z_{\mathrm{ref}}(I_{N,i})}. \end{aligned} \tag{14}$$

Thus, each instance's gap is computed against its own reference before averaging. For either metric $x \in \{z, g\}$, the mean and sample standard deviation across searches are

$$\bar{x}_N = \frac{1}{R}\sum_{r=1}^{R} x_{r,N}, \qquad s_{x,N} = \sqrt{\frac{1}{R-1}\sum_{r=1}^{R}(x_{r,N} - \bar{x}_N)^2}. \tag{15}$$

For the constructive-TSP search curves in Figure 4(b), the line is the pointwise mean best-so-far evolution-set objective across the three runs, and the shaded band spans their minimum and maximum. Curves are restricted to the first 500 attempted evaluations, carrying the last recorded value forward where needed. The curve bands describe the spread of search trajectories and $s_{x,N}$ measures the variability of the final independent test scores.

Table 5: ACO hyperparameters for heuristic evaluation and test.

Setting	Evolution	Test
TSP	30 ants, 100 iterations	30 ants, 200 iterations
CVRP	30 ants, 100 iterations	30 ants, 200 iterations
MKP	10 ants, 50 iterations	10 ants, 100 iterations
Offline BPP	30 ants, 15 iterations	30 ants, 150 iterations

Hyperparameters. Table 6 lists the parameters of ReAL-AHD. For all tasks, each LLM-based AHD baseline is allocated an evaluation budget of 500 on the evaluation set. The heuristic time limit is 120 seconds per candidate evaluation. To mitigate statistical variance, we perform three independent runs per method for each application setting.

Table 6: ReAL-AHD hyperparameters.

Component	Value
initial size	4
heuristic time limit	120
PreferSim	$M = 8$ problems; top-$k = 5$
similarity graph	edge threshold $\tau = 0.8$; $k_{\mathrm{nn}} = 3$
dynamic families	$K_{\min} = 3$, $K_{\max} = 5$
UCB	exploration $c = 0.3$; $\rho_{\min} = 0.2$
family planning	reward window $= 20$
sampling	softmax temperature $= 0.8$
graph-aware memory	$k_{\mathrm{mem}} = 3$
memory effectiveness	EMA coefficient $\alpha_{\mathrm{mem}} = 0.3$

D Detailed Methodology

D.1 Details of the Empirical Observations

For a pair (u, v), the derivation distance $d_D(u, v)$ is the shortest path length after replacing each recorded parent–offspring link with an undirected unit-length edge. Multi-source derivations contribute one link per source. Our code comparator uses the AST and data-flow components of CodeBLEU (Ren et al., 2020), with weights $(0, 0, 0.5, 0.5)$. For BehaveSim (Zhang & Lu, 2026), let $\mathcal{T}(h, I, z)$ denote the sequence of intermediate solutions generated by heuristic h on instance I from starting point z. For constructive TSP, intermediate solutions are partial tours, compared using normalized edit distance:

$$d_I(x, y) = \frac{d_{\mathrm{edit}}(x, y)}{d_{\max}(I)}, \tag{16}$$

where $d_{\max}(I)$ is the maximum possible edit distance for the instance. For trajectories X and Y, we compute

$$\mathrm{DTW}_I(X, Y) = \min_{\pi \in \mathcal{A}(|X|,|Y|)} \sum_{(i,j)\in\pi} d_I(x_i, y_j), \tag{17}$$

$$\mathrm{Sim}_{\mathrm{PSTraj}}(X, Y) = 1 - \frac{\mathrm{DTW}_I(X, Y)}{\min\{|X|, |Y|\}}, \tag{18}$$

where $\mathcal{A}$ contains the admissible monotone alignment paths. The algorithm-level similarity is

$$s_B(u, v) = \mathbb{E}_{I\sim P^I}\mathbb{E}_{z\sim S_I}\left[\mathrm{Sim}_{\mathrm{PSTraj}}\big(\mathcal{T}(u, I, z), \mathcal{T}(v, I, z)\big)\right]. \tag{19}$$

We estimate these expectations using $M = 8$ shared problem instances and starting points.

Median-threshold groups. For median thresholds d_{50} and s_{50}, we call pairs *near* when $d_D \le d_{50}$ and *far* when $d_D > d_{50}$. A pair is *divergent* when it is near and has $s \le s_{50}$, and *convergent* when it is far and has $s > s_{50}$. These labels describe positions relative to the archive medians.

All values equal to a threshold enter its lower group. Because derivation distances are discrete, this convention assigns 75.36% of EoH pairs and 80.36% of MCTS-AHD pairs to the near group. Table 7 reports these group sizes and the quadrant shares, together with the conditional low-similarity rates $q_{\text{near}} = P(s \le s_{50} \mid d_D \le d_{50})$ and $q_{\text{far}} = P(s \le s_{50} \mid d_D > d_{50})$.

Table 7: Median-threshold comparisons on constructive TSP. Near, Div., and Conv. report percentages of all pairs. The conditional rates q_{near} and q_{far} report low-similarity percentages within the respective distance groups.

Archive	Similarity	d_{50}	s_{50}	Near	Div.	Conv.	q_{near}	q_{far}
EoH	s_C	3	0.6617	75.36	38.90	12.96	51.62	47.42
EoH	s_B	3	0.1125	75.36	39.00	13.33	51.76	45.90
MCTS-AHD	s_C	4	0.6740	80.36	39.45	8.78	49.09	55.27
MCTS-AHD	s_B	4	0.0725	80.36	39.17	8.81	48.74	55.14

The quadrant shares depend on these group sizes. Writing $p_D = P(d_D \le d_{50})$ and $p_S = P(s \le s_{50})$, independence would yield divergent and convergent shares of $p_D p_S$ and $(1 - p_D)(1 - p_S)$, respectively; their sum is 50% when both margins are balanced. For EoH with CodeBLEU, the observed margins give an independence reference of 38.12% divergent and 12.18% convergent pairs. The conditional rates account for the unequal distance-group sizes: low-similarity pairs constitute 48.74–51.76% of near pairs and 45.90–55.27% of far pairs across the four comparisons. Their similar prevalence in both groups is consistent with the weak rank associations reported above.

Outer-quartile groups. We also report the quadrant composition using outer-quartile thresholds. Specifically, near/far derivation means $d_D \le d_{25}$/$d_D > d_{75}$, and low/high similarity means $s \le s_{25}$/$s > s_{75}$. Pairs between either pair of thresholds are excluded from this check. Table 8 gives the thresholds and the combined share of near/low-similarity and far/high-similarity pairs among the retained pairs. Discrete derivation distances can again yield unequal group sizes.

Table 8: Descriptive quadrant shares under outer-quartile thresholds. Off-diagonal pairs satisfy either near derivation/low similarity or far derivation/high similarity.

Archive	Similarity	(d_{25}, d_{75})	(s_{25}, s_{75})	Off-diagonal
EoH	s_C	$(2, 3)$	$(0.5599, 0.7213)$	48.33%
EoH	s_B	$(2, 3)$	$(0.0625, 0.2275)$	53.29%
MCTS-AHD	s_C	$(3, 4)$	$(0.5281, 0.7422)$	51.09%
MCTS-AHD	s_B	$(3, 4)$	$(0.0400, 0.1763)$	41.46%

Action agreement on shared states. To isolate decisions from the trajectories they generate, we evaluate every archived heuristic on a common bank of $K_{\text{state}} = 1{,}000$ TSP states. Let $a_u(s)$ denote the city selected by heuristic u at state s. Exact action agreement is

$$A(u, v) = \frac{1}{K_{\text{state}}} \sum_{s \in \mathcal{S}} \mathbb{I}[a_u(s) = a_v(s)]. \tag{20}$$

D.2 Derivation Graph

In the notation of the main text, r_v is always oriented so that larger is better: it equals negative mean cost for minimization and mean value for maximization. Raw rewards enter Eq. 2, and normalization is applied when planning actions. Duplicate rejection compares exact source-code strings against the current elite set. A generation failure, timeout, or rejected duplicate creates no derivation node or operator edge. It instead produces an attempt outcome and increments visits for the selected node and its distinct ancestors.

An accepted offspring is attached to all source nodes through one multi-source operator edge. Sources precede the child in insertion order, which guarantees acyclicity. Direct children are deduplicated by node identifier in Eq. 2, and a shared ancestor receives one visit increment per accepted offspring, regardless of the number of paths reaching it. New offspring start with $Q_v = r_v$ and zero expansion visits. Algorithm 2 specifies the evaluation of the lineage recurrence.

Algorithm 2 Derivation update

Require: selected node v, source set $\mathcal{S}$, operator o, result h', derivation DAG
if h' fails evaluation **then**
  $\mathcal{A} \leftarrow \{v\} \cup \mathrm{Anc}(v)$
  for all $a \in \mathcal{A}$ **do**
    $n_a \leftarrow n_a + 1$ ▷ Leave r_a and Q_a unchanged
  end for
else
  Create node u with reward r_u, value $Q_u \leftarrow r_u$, and $n_u \leftarrow 0$
  Insert the multi-source edge $(\mathrm{unique}(\mathcal{S}), o, u)$
  $\mathcal{A} \leftarrow \mathrm{Anc}(u)$
  for all $a \in \mathcal{A}$ in reverse topological order **do**
    $n_a \leftarrow n_a + 1$
    $Q_a \leftarrow \big(r_a + \sum_{c \in \mathrm{Ch}_{\mathrm{der}}(a)} Q_c\big)/(1 + |\mathrm{Ch}_{\mathrm{der}}(a)|)$
  end for
end if

D.3 Details of PreferSim

At the ℓ-th state of I_m, $\mathcal{C}^h_{m,\ell}$ denotes the feasible candidate identifiers and $\pi^h_{m,\ell}$ is the ordered list of the heuristic's top preferences. For heuristics h_u and h_v, we quantify action agreement by

$$\begin{aligned} R_{m,\ell}(u,v) &= \frac{|\operatorname{set}(\pi^u_{m,\ell}) \cap \operatorname{set}(\pi^v_{m,\ell})|}{\min(|\pi^u_{m,\ell}|, |\pi^v_{m,\ell}|)}, \quad & T_{m,\ell}(u,v) &= \mathbb{I}[\pi^u_{m,\ell,1} = \pi^v_{m,\ell,1}], \\ C_{m,\ell}(u,v) &= \frac{|\mathcal{C}^u_{m,\ell} \cap \mathcal{C}^v_{m,\ell}|}{|\mathcal{C}^u_{m,\ell} \cup \mathcal{C}^v_{m,\ell}|}, & A_{m,\ell}(u,v) &= \lambda_R R_{m,\ell} + \lambda_T T_{m,\ell} + \lambda_C C_{m,\ell}, \end{aligned} \tag{21}$$

$R_{m,\ell}$ measures agreement among highly preferred actions, $T_{m,\ell}$ rewards an identical first choice, and $C_{m,\ell}$ checks that both lists were drawn from the same candidate set. We use $(\lambda_R, \lambda_T, \lambda_C) = (0.7, 0.2, 0.1)$ for records with state contexts.

PreferSim uses a fixed problem set $\mathcal{D}_p = \{I_m\}_{m=1}^M$ drawn from the evolution split. Score-based interfaces are evaluated on heuristic-independent canonical states. Selector-based interfaces preserve the public evaluator signature and recover a ranking through counterfactual calls, as detailed below. Shared base states are generated independently of the compared heuristics, providing a common context for separating decision preferences from path-dependent differences.

Score-based adapters for ACO. For TSP and CVRP, the evolved heuristic returns a desirability matrix. The adapter applies the same finiteness, non-negativity, clipping, and normalization rules as the corresponding evaluator, masks infeasible actions, and sorts the feasible entries of the current row. A deterministic nearest-neighbor controller advances TSP tours. CVRP additionally tracks residual vehicle capacity and depot returns while using the same capacity mask as the solver. Hence, the current node, decision index, and resource state are identical for all heuristics on an instance. Offline BPP uses the evaluator's row-aggregation rule for items already placed in the current bin. A deterministic best-fit controller constructs aligned bin states, and rankings are recorded only after a bin contains at least one item. MKP returns one desirability value per item. At each canonical residual-capacity state, the adapter filters infeasible items and ranks the remaining entries of this vector.

Selector-based adapters. Constructive heuristics expose a selected action rather than raw action scores. To recover a top-k preference, the adapter queries the selector on the current state, removes the returned action from the candidate representation, and queries again until k valid actions have been obtained or no candidate remains. For KP, removal is performed on the compact weight and value arrays exactly as in the evaluator, with local indices mapped back to stable item identifiers. For CVRP, feasibility masks retain capacity and depot.

D.4 SIMILARITY GRAPH

For each newly inserted node, we compute and cache its similarities to the active nodes. The threshold rule retains every edge with weight at least τ, and the nearest-neighbor rule adds the new node's k_{nn} strongest connections. The implementation additionally repairs nodes with fewer than k_{nn} incident edges by consulting their highest-similarity neighbors.

Let A be the symmetric clustering affinity and D_A its diagonal degree matrix. We form the normalized Laplacian $L = I - D_A^{-1/2} A D_A^{-1/2}$ and order its eigenvalues as $\lambda_1 \le \cdots \le \lambda_n$. The number of families is selected by

$$K_t = \operatorname*{arg\,max}_{K \in \{K_{\min}, \ldots, \min(K_{\max}, n-1)\}} (\lambda_{K+1} - \lambda_K). \tag{22}$$

Spectral clustering uses the precomputed affinity, k-means label assignment, and a fixed random seed. Before $K_{\min} + 1$ active nodes are available, selection uses the global candidate pool. Between full clustering steps, a new node is assigned to the family maximizing its mean cached similarity to current members. Full clustering is triggered when families are first created, after pruning, or when at least $\max\{\lceil n/10 \rceil, K_{\min} + 1\}$ nodes have been added since the previous clustering. New cluster labels are matched to old family identifiers by solving a maximum-weight bipartite assignment problem (Kuhn, 1955), with weights given by shared membership counts. Clusters without a positive-overlap match receive fresh identifiers.

Periodic pruning removes low-quality and near-duplicate nodes from the similarity graph and drops their associated memory records. Current elites and the best node of every family are protected, and at least one node is retained per family with an adaptive retention rate. Pruning never deletes nodes or edges from the derivation graph; it only removes them from similarity-based clustering, planning, and memory retrieval.

D.5 HIERARCHICAL GRAPH PLANNING

UCB-inspired planning with softmax. In Eqs. 4 and 6, the count-dependent bonus raises the relative priority of less-visited families and nodes; its scale $c\rho_t$ adjusts this exploration pressure to the remaining evaluation budget. Softmax converts the resulting scores into weights, with its temperature controlling how strongly selection favors higher scores. At the family level, these weights are converted into integer budget allocations; at the node level, they define sampling probabilities for each operator's eligible candidates. This randomized node selection preserves opportunities to expand alternatives with competitive scores, while the exploration bonus specifically accounts for how often candidates have been visited. Together, they combine visit-aware exploration with smooth budget allocation and stochastic node selection over the evolving landscape. We therefore describe this mechanism as UCB-inspired planning, using the count-based exploration principle of UCB to guide heuristic search.

Family planning. Family rewards use the bounded outcome scale in Section 3.3. Outcome classification first checks for a new global best, then for an elite admission that is no worse than the selected parent. A valid offspring is classified as poor when its fitness falls below the family's mean by more than 20% of the absolute mean. A family with an empty window uses the mean reward of currently informed families. After reclustering, a matched family retaining fewer than half of its previous members restarts with an empty reward window. These rules distinguish a newly formed mechanism region from a persistent family with recently unproductive expansions. The allocation routine converts softmax weights into integer slots while keeping the total family pool fixed at $b_0 K_t$, where b_0 is the number of the non-global operators.

Node–operator planning. For cross-family bridging, define the foreign-neighbor set $\mathcal{B}(v) = \{u \in \mathcal{N}_{\mathrm{sim}}(v) : f(u) \neq f(v)\}$. The complete scoring rule, including the empty-neighborhood case in Eq. 5, is

$$V_{s2}(v) = \begin{cases} \max_{u \in \mathcal{B}(v)} r_u, & \mathcal{B}(v) \neq \varnothing, \\ Q_v, & \mathcal{B}(v) = \varnothing. \end{cases} \tag{23}$$

These values are refreshed before node–operator selection in each planning round using the current similarity neighbors and family assignments.

For an operator's eligible candidate set $\mathcal{V}_{f,o}$, let $a_o = \min_{v \in \mathcal{V}_{f,o}} V_o(v)$ and $b_o = \max_{v \in \mathcal{V}_{f,o}} V_o(v)$. The normalized exploitation term is

$$\mathrm{norm}(V_o(v)) = \begin{cases} (V_o(v) - a_o)/(b_o - a_o), & b_o - a_o \geq 10^{-10}, \\ 0.5, & \text{otherwise.} \end{cases} \tag{24}$$

For each non-global operator o, ReAL-AHD first samples its operator-conditioned node v_o within the selected family and immediately forms the pair (v_o, o). For $s1$, only non-leaf family members are eligible; if a family has no eligible node, its $s1$ entry is omitted and the remaining ranked pairs fill the allocated slots. If family planning supplies fewer slots than there are entries, pairs are ranked by their operator-conditioned scores and truncated; an excess budget cycles through the ranking again. Each retained pair is instantiated by the relational source rule associated with its operator. Operators $m1$ and $m2$ receive only the selected node. Operator $s2$ targets the other-family elite, denoted by $u^\star$, and adds intermediate nodes. Thus, the absence of foreign similarity neighbors changes the selection value to Q_v but does not by itself disable $s2$. The expansion routine skips $s1$ or $s2$ actions with fewer than two source programs, before LLM generation and evaluation.

Table 9: Evolutionary operators.

Op.	Source set	Search role
i1	Reference implementation and task interface	Initialization
m1	Selected node	Mechanism innovation
m2	Selected node	Local refinement
s1	Selected non-leaf node and its greedy high-Q descendant path	Lineage synthesis
e1	Best node from every family	Global diversification
e2	Selected node and the best other-family node	Elite inspiration
s2	Ordered bridge sequence to an other-family elite	Cross-family bridging

Graph-aware memory. An evolutionary event is stored as $e = (v_{\mathrm{par}}, v_{\mathrm{off}}, o, y, \Delta F, d_e, \eta)$, where y is the outcome status, d_e is the offspring's compact natural-language description, and $\eta \in [-1, 1]$ is the learned effectiveness, initialized to zero. The fitness change is $\Delta F = r_{v_{\mathrm{off}}} - r_{v_{\mathrm{par}}}$, so positive values indicate improvement. For multi-source operators, v_{par} is the selected expansion node, irrespective of the order of the source programs in the prompt. New global bests and elite admissions create positive records; clearly poor offspring and timeouts with a recoverable idea create negative records.

Before expanding node v with operator o, retrieval considers records incident to v and records incident to its strongest similarity neighbors. For an incident record, the base relevance is 2; for a record reached through neighbor u, it is $W_t(v, u)$. The retrieval score is the base relevance plus the learned effectiveness η. Records reached through multiple nodes are deduplicated by memory identifier, retaining the highest relevance score. We search at most five neighbors with similarity at least τ and inject at most k_{mem} records. With $k_{\mathrm{mem}} = 3$, selection first reserves one position for the highest-ranked positive record when available, then takes negative records in score order, and fills any remaining positions with the highest-ranked unused records. Let $g \in [-1, 1]$ denote the outcome credit of the new offspring. We use $g = 1.0$ for a new global best, 0.8 for an elite admission, 0 for an ordinary offspring, -0.5 for poor performance or timeout, and -0.3 for an invalid result. Whenever an attempt receives outcome credit, every memory record injected into its prompt is updated once by

$$\eta_e \leftarrow \mathrm{clip}_{[-1,1]}\big((1 - \alpha_{\mathrm{mem}})\eta_e + \alpha_{\mathrm{mem}} g\big) \tag{25}$$

All injected records receive the same outcome credit; records not retrieved for that attempt retain their previous effectiveness. This update makes retrieval responsive to the recent outcomes of generations that used each record.

E Additional Results

E.1 Statistical Analysis

To assess the stability of ReAL-AHD, we report the objectives mean $\bar{x}_N$ and standard deviation $s_{z,N}$ over three independent runs for constructive TSP in Table 10.

Table 10: TSP constructive heuristics: mean and std. over three independent runs.

	$N = 50$		$N = 100$		$N = 200$	
Method	mean	std.	mean	std.	mean	std.
EoH	6.454	0.150	8.951	0.259	12.536	0.424
ReEvo	6.296	0.119	8.732	0.141	12.228	0.238
HSEvo	6.369	0.065	8.824	0.137	12.320	0.231
HiFo-Prompt	6.223	0.070	8.627	0.087	12.024	0.128
MCTS-AHD	6.306	0.043	8.714	0.041	12.147	0.051
PartEvo	6.308	0.158	8.704	0.208	12.181	0.333
OpenEvolve	6.300	0.079	8.773	0.136	12.339	0.240
Ours	**6.124**	0.015	**8.481**	0.048	**11.882**	0.121

E.2 Comparison with the API Variant of CALM

CALM (Huang et al., 2026) occupies a distinct methodological setting because its full framework jointly evolves heuristics and the language model that generates them. It combines evolutionary prompt generation with reinforcement learning via group relative policy optimization (GRPO), using heuristic-evaluation feedback to update the LLM parameters during search. We therefore present a dedicated comparison with *CALM (API, w/o GRPO)*, the fixed-model API variant reported in the original paper. This variant retains CALM's evolutionary search components, including its mutation and crossover operators and population-collapse mechanism, while omitting the model-parameter updates. The comparison follows the main experimental setting in Section 4.1. Table 11 covers constructive TSP at $N \in \{50, 100, 200\}$ and online BPP at stream lengths $L \in \{5\text{k}, 10\text{k}, 100\text{k}\}$ with capacities $C \in \{100, 300\}$. ReAL-AHD outperforms CALM (API, w/o GRPO) in all nine settings.

Table 11: Comparison with CALM's API variant on constructive TSP and online BPP.

(a) Constructive TSP

	$N = 50$		$N = 100$		$N = 200$	
Method	Obj.	Gap	Obj.	Gap	Obj.	Gap
CALM (API, w/o GRPO)	6.354	11.58%	8.813	13.64%	12.273	14.95%
ReAL-AHD	6.124	7.55%	8.481	9.40%	11.882	11.30%

(b) Online BPP

	$C = 100$			$C = 300$		
Method	$L = 5\text{k}$	$L = 10\text{k}$	$L = 100\text{k}$	$L = 5\text{k}$	$L = 10\text{k}$	$L = 100\text{k}$
CALM (API, w/o GRPO)	1.33%	0.92%	0.45%	2.99%	1.94%	0.74%
ReAL-AHD	0.70%	0.35%	0.02%	2.21%	1.06%	0.12%

E.3 Results on Routing Problems

Table 12 evaluates constructive heuristics on CVRP defined in Appendix A. The selector evolved on two-dimensional instances is applied without further search, testing transfer across spatial and demand distributions as well as customer counts. ReAL-AHD obtains the lowest objective at all three test sizes, with reference gaps of 2.21%, 2.05%, and 1.47% for $N = 50$, 100, and 200, respectively.

Table 13 further evaluates the constructive TSP and CVRP heuristics on standard benchmark instances. For TSPLIB, we use 34 instances grouped by size into Easy ($N \leq 100$), Medium ($100 < N \leq 200$), and Hard ($N > 200$). The 11 Easy instances are eil51, berlin52, eil76, pr76, rat99, kroA100, kroB100, kroC100, kroD100, kroE100, and rd100; the 12 Medium instances are eil101, lin105, pr107, bier127, ch130, ch150, kroA150, kroB150, pr152, u159, kroA200, and kroB200; and the 11 Hard instances are ts225, tsp225, pr226, pr264, a280, pr299, lin318, rd400, fl417, pr439, and d493. For CVRPLIB, we use sets A, B, P, and X; the first three contain instances with at most 100 customers, whereas X spans 100 to 1000 customers. Gaps are measured against the optimal or best-known costs published with each benchmark, and AVG is the macro-average over the size groups or sets. ReAL-AHD achieves the best result of all methods on the two columns with the largest instances, 13.72% on TSPLIB Hard and 13.22% on CVRPLIB X. Among LLM-based AHD methods, ReAL-AHD also has the lowest average gap on both collections, 12.01% on TSPLIB and 18.11% on CVRPLIB.

Table 12: Comparison of constructive heuristics on CVRP. Bold indicates the best result in each column.

Test sets	N=50		N=100		N=200	
Methods	Obj.↓	Gap	Obj.↓	Gap	Obj.↓	Gap
Reference	15.471	–	30.013	–	58.412	–
Greedy	18.622	20.52%	37.030	23.29%	70.576	20.81%
POMO	18.421	19.45%	36.024	20.15%	68.764	17.76%
Population-based AHD						
EoH	17.985	16.40%	35.743	19.08%	69.880	19.65%
ReEvo	16.525	6.84%	33.350	11.11%	66.081	13.16%
HSEvo	16.318	5.52%	31.712	5.64%	61.320	4.97%
HiFo-Prompt	17.244	11.57%	33.780	12.57%	65.306	11.81%
Structure-based AHD						
MCTS-AHD	17.671	14.34%	35.131	17.02%	67.580	15.68%
PartEvo	15.954	3.14%	30.833	2.71%	59.452	1.78%
OpenEvolve	15.940	3.06%	30.923	3.02%	60.172	3.01%
Ours	**15.810**	**2.21%**	**30.628**	**2.05%**	**59.263**	**1.47%**

Table 13: Gaps on TSPLIB and CVRPLIB. TSPLIB instances are grouped by size into Easy ($N \leq 100$), Medium ($100 < N \leq 200$), and Hard ($N > 200$), and AVG is the macro-average over groups or sets. Bold indicates the best result in each column.

Task	TSPLIB				CVRPLIB				
Sets	Easy	Medium	Hard	AVG	A	B	P	X	AVG
Greedy	28.86%	25.66%	25.57%	26.70%	39.14%	41.78%	33.76%	25.47%	35.04%
POMO	**1.23%**	**1.88%**	15.47%	**6.19%**	**4.86%**	**5.16%**	**14.69%**	21.40%	**11.53%**
Population-based AHD									
EoH	15.66%	18.38%	18.51%	17.52%	37.27%	39.14%	33.26%	26.24%	33.98%
ReEvo	13.07%	14.48%	18.94%	15.49%	35.96%	34.27%	30.30%	27.62%	32.04%
HSEvo	14.98%	16.36%	17.78%	16.37%	22.39%	19.26%	17.53%	14.19%	18.34%
HiFo-Prompt	10.52%	11.73%	15.35%	12.53%	32.89%	34.55%	29.15%	21.85%	29.61%
Structure-based AHD									
MCTS-AHD	12.73%	12.74%	15.45%	13.64%	36.04%	36.01%	23.71%	30.82%	31.65%
PartEvo	16.40%	17.96%	22.54%	18.97%	23.11%	18.75%	18.91%	14.69%	18.87%
OpenEvolve	19.00%	18.27%	20.64%	19.30%	23.54%	21.13%	20.99%	16.16%	20.46%
Ours	11.00%	11.32%	**13.72%**	12.01%	22.68%	18.14%	18.39%	**13.22%**	18.11%

E.4 GUIDED LOCAL SEARCH

GLS repeatedly penalizes undesirable tour edges and re-optimizes the augmented objective to escape local minima. We evaluate both solution quality and runtime because a more elaborate guidance function is useful only if its additional computation is compensated by faster or more effective search. Table 14 provides the results. ReAL-AHD achieves the smallest reference gaps and improves both quality and runtime relative to vanilla GLS. These results show that the discovered guidance can improve an established local-search procedure across sizes.

Table 14: TSP-GLS quality and runtime per instance. Gap is measured against the LKH-3 reference.

	TSP50		TSP100		TSP200	
Method	Gap (%)	Time (s)	Gap (%)	Time (s)	Gap (%)	Time (s)
Vanilla GLS	0.067%	0.08	0.073%	4.09	0.586%	15.93
EoH	0.035%	0.13	0.111%	3.65	0.667%	16.16
ReEvo	0.034%	0.10	0.088%	4.16	0.566%	19.08
HSEvo	0.034%	0.11	0.084%	4.81	0.576%	21.21
HiFo-Prompt	0.027%	0.09	0.078%	5.48	0.607%	14.27
MCTS-AHD	0.069%	0.06	0.086%	3.29	0.801%	12.18
PartEvo	0.060%	0.06	0.081%	3.22	0.493%	12.45
OpenEvolve	0.036%	0.07	0.074%	3.54	0.468%	13.56
Ours	**0.018%**	0.08	**0.065%**	3.74	**0.407%**	14.51

E.5 REAL-AHD WITH OTHER LLMS

We repeat the comparison with GPT-5-nano on constructive TSP and MKP-ACO, covering direct next-action selection and heuristic information inside a stochastic solver. Table 15 shows that ReAL-AHD attains the best objective among the compared AHD methods in all settings. On TSP100, it reduces the gap from the closest baseline's 11.61% to 10.78%, and on TSP200 from 12.82% to 12.40%. For MKP, the gaps are 0.73%, 0.39%, and 0.56% across the three sizes, compared with the strongest baselines of 0.78%, 0.43%, and 0.67%. The ordering therefore persists across different LLM backbones and heuristic interfaces, supporting the contribution of relational search organization.

Table 15: AHD results with GPT-5-nano on constructive TSP and MKP-ACO. In the Reference row, TSP entries are LKH-3 reference values, whereas MKP-ACO entries are vanilla ACO results. Bold indicates the best result among the compared AHD methods.

Task	TSP-Constructive						MKP-ACO					
Test sets	N=50		N=100		N=200		N=100		N=300		N=500	
Methods	Obj.↓	Gap	Obj.↓	Gap	Obj.↓	Gap	Obj.↑	Gap	Obj.↑	Gap	Obj.↑	Gap
Reference	5.695	–	7.754	–	10.677	–	21.960	3.22%	60.178	7.68%	98.203	10.11%
Population-based AHD												
EoH	6.340	11.36%	8.790	13.36%	12.229	14.54%	22.290	1.69%	61.936	4.51%	101.754	5.99%
ReEvo	6.303	10.74%	8.712	12.36%	12.223	14.49%	22.381	1.33%	62.642	3.66%	102.799	5.35%
HSEvo	6.374	11.95%	8.901	14.82%	12.444	16.54%	22.377	1.36%	62.483	3.93%	102.136	6.01%
HiFo-Prompt	6.245	9.69%	8.696	12.16%	12.149	13.79%	22.522	0.86%	64.771	0.43%	107.466	0.67%
Structure-based AHD												
MCTS-AHD	6.311	10.84%	8.652	11.61%	12.044	12.82%	22.529	0.78%	64.663	0.65%	107.201	1.03%
PartEvo	6.362	11.71%	8.855	14.18%	12.471	16.81%	22.402	1.27%	63.326	2.53%	104.549	3.38%
OpenEvolve	6.340	11.40%	8.765	13.05%	12.290	15.11%	22.420	1.16%	62.969	3.05%	103.247	4.62%
Ours	**6.242**	**9.63%**	**8.589**	**10.78%**	**12.000**	**12.40%**	**22.542**	**0.73%**	**64.783**	**0.39%**	**107.515**	**0.56%**

Table 16 further measures the reduction in the best evolution-set objective from initialization to the end of search. ReAL-AHD improves the initial best by 10.21% with DeepSeek-V4-Flash and

16.74% with DeepSeek-V4.1-Flash. These results show that relational planning improves the candidates supplied by multiple LLM backbones, complementing the cross-method comparison with GPT-5-nano.

Table 16: Evolution improvement on constructive TSP (N=100) across LLM backbones.

Backbone	Init best	Evolved best	Improvement
DeepSeek-V4-Flash (DeepSeek-AI, 2026b)	9.555	8.579	10.21%
DeepSeek-V4.1-Flash (DeepSeek-AI, 2026a)	9.610	8.002	16.74%
GPT-5-nano (Singh et al., 2025)	8.944	8.600	3.85%
GPT-5.6-Luna (OpenAI, 2026)	10.718	9.860	8.00%
Gemini-3.5-Flash-Lite (Google DeepMind, 2026)	10.196	8.791	13.78%
Kimi-K2.7-Code (Moonshot AI, 2026)	9.753	8.616	11.66%
Qwen-3.8-27B (Qwen Team, 2026)	9.823	8.853	9.88%

E.6 Detailed Ablation Results

We extend Section 4.3 by examining how the two relations and hierarchical graph planning contribute to search, followed by a sensitivity analysis of graph construction.

Extended component ablations. Table 17 reports five additional variants. *w/o both relations* removes both relational graphs. *Static algorithm families* retains the partition obtained at the first clustering and assigns subsequent nodes incrementally, without reclustering existing members. The two *node selection* variants replace the exploitation value in Eq. 5 with Q_v or r_v for every non-global operator, retaining the exploration term and the remaining planning procedure. Finally, *random memory* retains memories but selects records randomly from the same pool, isolating the contribution of relational retrieval.

ReAL-AHD achieves the best result in every column of Table 17. Using both relations reduces the TSP100 and TSP200 gaps by 3.45 and 2.86 percentage points relative to removing both, respectively. Together with the single-relation ablations in Table 3, this supports the complementary roles of lineage and similarity in organizing search. Updating the family partition also matters: dynamic families reduce the TSP gaps from 11.19% and 12.47% to 9.40% and 11.30%, with corresponding improvements on both Online BPP settings. The family structure is useful when it adapts to the algorithms discovered during search. Operator-conditioned selection outperforms both shared-value alternatives in all four settings, supporting the use of distinct exploitation values for refinement, lineage synthesis, and similarity bridging. Relational memory retrieval yields a benefit on TSP: compared with random retrieval, it reduces the gaps from 10.49% to 9.40% on TSP100 and from 11.79% to 11.30% on TSP200. These results support planning with dynamic families, operator-conditioned values, and memories selected for the current relational context.

Table 17: Complete ablations on relational algorithm landscape and hierarchical graph planning. TSP reports the reference gap; Online BPP reports the excess-bin ratio (lower is better).

Variants	TSP		Online BPP	
	N=100	N=200	5k_C100	10k_C100
ReAL-AHD	9.40%	11.30%	0.70%	0.35%
w/o both relations	12.85%	14.16%	0.86%	0.57%
static algorithm families	11.19%	12.47%	0.80%	0.48%
node selection based on Q_v	11.63%	12.46%	0.78%	0.38%
node selection based on r_v	10.85%	12.17%	0.73%	0.44%
random memory	10.49%	11.79%	0.76%	0.56%

Sensitivity to graph construction. The similarity threshold τ retains strong relations, while k_{nn} controls the nearest-neighbor connections used to maintain local connectivity. For the similarity ablations in Table 3, we use quantile-matched thresholds of 0.8, 0.6, and 0.3 for PreferSim, CodeBLEU, and BehaveSim, respectively, with $k_{\mathrm{nn}} = 3$ fixed across measures. For PreferSim, Table 18

varies one parameter at a time around the default $(\tau, k_{\mathrm{nn}}) = (0.8, 3)$. This shared default yields the best TSP200 gap and Online BPP 10k_C100 ratio among the tested configurations, and is within 0.04 and 0.05 percentage points of the best TSP100 and 5k_C100 results, respectively. Moreover, every tested configuration improves all four metrics over both the CodeBLEU and BehaveSim replacements reported in Table 3, showing that the benefit of PreferSim persists across these graph-construction choices.

Table 18: Sensitivity to the similarity threshold τ and the neighbor k_{nn}. TSP reports the reference gap; Online BPP reports the excess-bin ratio (lower is better).

Parameters		TSP		Online BPP	
τ	k_{nn}	$N = 100$	$N = 200$	5k_C100	10k_C100
0.8	3	9.40%	11.30%	0.70%	0.35%
0.8	1	10.85%	11.32%	0.76%	0.41%
0.8	5	9.36%	12.12%	0.82%	0.42%
0.7	3	10.59%	11.97%	0.78%	0.40%
0.9	3	10.18%	11.34%	0.65%	0.39%

E.7 Analysis of Similarity

Sensitivity to preference capture. We evaluate how consistently PreferSim recovers similarity rankings and local neighborhoods when the problem instances are resampled. The experiment uses fixed program archives from EoH, MCTS-AHD, and ReAL-AHD, with the same evaluable programs retained across all configurations within each archive. We vary $M \in \{2, 4, 8, 16\}$ with $k = 5$, and $k \in \{1, 3, 5, 10\}$ with $M = 8$. For each configuration, we conduct 20 trials, each drawing two disjoint subsets of M instances and constructing a separate pairwise similarity matrix from the cached preference records on each subset. Rank stability is the Spearman correlation between the upper-triangular entries of the two matrices, excluding the diagonal. Neighborhood stability is the mean Jaccard overlap between each program's two sets of top-5 neighbors. Self-neighbors are excluded and ties are resolved by a fixed program identifier; the neighborhood size remains five for every preference depth k. Figure 5 reports means and one-standard-deviation bands over these problem-resampling trials.

Increasing the problem count improves both forms of stability across all three archives. With $k = 5$, increasing M from 2 to 8 raises the mean rank correlations from 0.614–0.681 to 0.867–0.892, and the mean neighborhood overlaps from 0.318–0.373 to 0.479–0.527. Using $M = 16$ further raises rank correlations to 0.934–0.942 and neighborhood overlaps to 0.555–0.611. These trends show that aggregating preferences across instances produces increasingly reproducible similarity structure. The default $M = 8$ already provides strong rank agreement with a compact problem set, but the larger setting increases computational cost. Preference depth exhibits a different pattern. At $M = 8$, the largest gain occurs when moving from the first choice alone ($k = 1$) to a short list of alternatives ($k = 3$). Increasing k from 5 to 10 changes the mean rank correlation by 0.009–0.020 and the mean neighborhood overlap by less than 0.009 in each archive. The small changes at larger depths support $k = 5$, which records leading alternatives while bounding the number of counterfactual selector calls. The shared trends across EoH, MCTS-AHD, and ReAL-AHD support this capture configuration across different program collections.

Computational cost. We measure the cost of constructing and maintaining the similarity relation through offline and online settings. Timing experiments use AMD EPYC 7702 CPUs. The capture and graph parameters follow the default configuration: $M = 8$, $k = 5$, $\tau = 0.8$, and $k_{\mathrm{nn}} = 3$. Offline measurements use constructive TSP and MKP-ACO archives from EoH and MCTS-AHD, covering a selector interface and a direct score interface. For each task, the same 20 additional candidate programs are evaluated across archive sizes, with the existing programs' preference records, pairwise similarities, and graph state prepared in advance. Each insertion starts from the same pre-insertion state. Capture is measured once per candidate and reused across archive sizes; comparison measures the new program against the cached archive; graph update excludes similarity computation; and reclustering measures one complete invocation. Table 19 reports the mean and standard

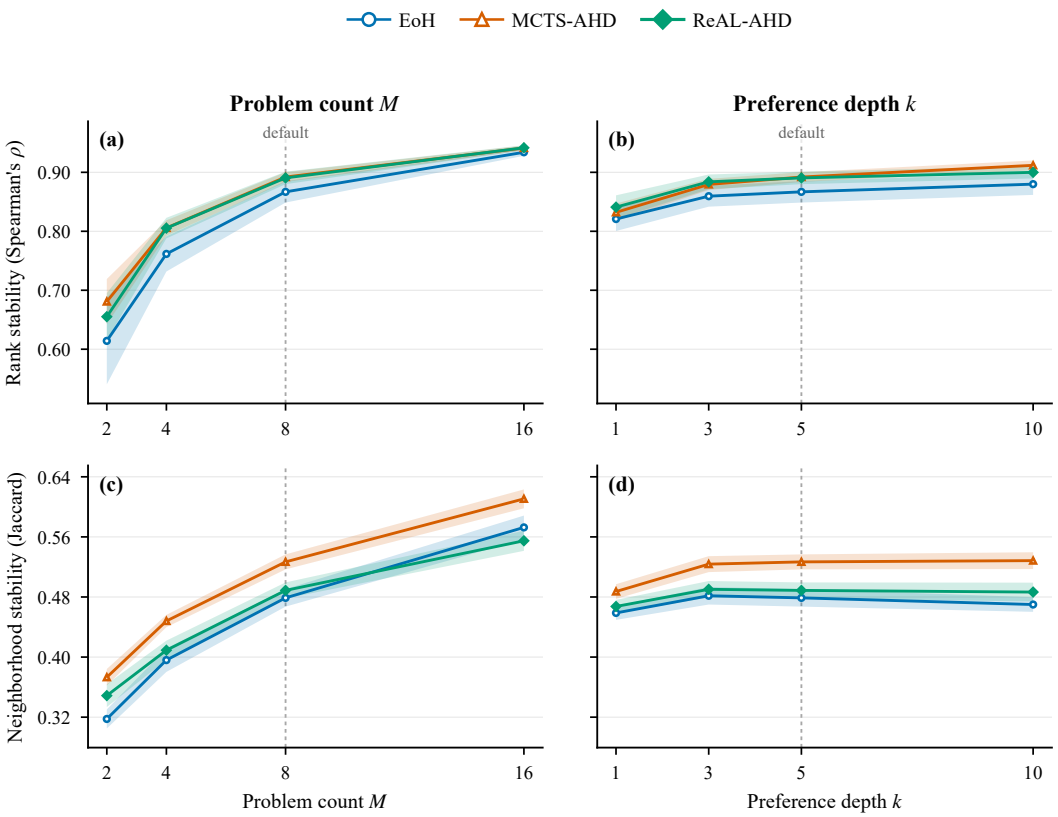

Figure 5: Sensitivity of PreferSim to preference capture on fixed heuristics from EoH, MCTS-AHD, and ReAL-AHD. (a, c) Varying the number of instances M with $k = 5$. (b, d) Varying the preference depth k with $M = 8$.

deviation across candidates. Cached preference comparison makes the growing archive inexpensive to reuse. At $H = 300$, PreferSim compares a new TSP program with the archive in 1.10–1.12 seconds, compared with 10.40–12.96 seconds for CodeBLEU. For MKP-ACO, the corresponding times are approximately 1.02 seconds and 19.49–23.25 seconds. The PreferSim comparison time grows approximately linearly with archive size, while capture is independent of that size. Mean capture time is 16.116 seconds per candidate in the TSP selector benchmark and 0.318 seconds in the MKP score benchmark, reflecting the different execution requirements of the two interfaces. Across the offline settings, mean graph-update time remains below 18 milliseconds and mean reclustering time below 0.37 seconds per invocation.

The online measurements use the LLM backbone, temperature, and evaluation budget in Section 4.1, and additionally include Online BPP. Total time is measured over the complete search, including generation, evaluation, and relation maintenance, and excludes the final test evaluation. We decompose this elapsed time as

$$T_{\text{search}} = T_{\text{LLM}} + T_{\text{evaluation}} + T_{\text{maintenance}} + T_{\text{other}}. \tag{26}$$

Here, T_{LLM} covers LLM generation; $T_{\text{evaluation}}$ covers candidate execution on the evolution instances; $T_{\text{maintenance}}$ includes preference capture, cached comparison, and graph and family updates; and T_{other} covers the remaining initialization, node selection, and I/O. Figure 6 presents total search time on constructive TSP100. Even with the same resources and evaluation budget, candidate heuristics can differ in execution cost, so $T_{\text{evaluation}}$ varies with the programs generated during search. A cheaper heuristic can be faster to evaluate while yielding a worse objective, whereas a more computationally intensive heuristic may improve solution quality. The figure therefore characterizes the observed time–quality trade-off; total time is interpreted jointly with the achieved objective.

Table 19: Offline computational cost on fixed program archives. Entries are mean ± standard deviation over 20 candidate programs. H is the archive size before insertion. CodeBLEU requires no preference capture.

Similarity	H	Capture (s)	Compare (s)	Graph update (ms)	Recluster (ms)
Constructive TSP / EoH					
PreferSim	100	16.116 ± 10.426	0.370 ± 0.003	0.593 ± 0.014	38.994 ± 0.508
CodeBLEU	100	–	4.381 ± 1.386	0.504 ± 0.007	29.333 ± 0.355
PreferSim	300	16.116 ± 10.426	1.102 ± 0.006	1.745 ± 0.024	118.804 ± 3.813
CodeBLEU	300	–	12.962 ± 3.057	2.231 ± 0.381	109.304 ± 22.538
PreferSim	500	16.116 ± 10.426	1.837 ± 0.011	4.033 ± 0.235	355.579 ± 3.212
CodeBLEU	500	–	22.849 ± 5.835	6.572 ± 0.193	272.069 ± 1.588
Constructive TSP / MCTS-AHD					
PreferSim	100	16.116 ± 10.426	0.376 ± 0.003	0.544 ± 0.012	32.957 ± 2.377
CodeBLEU	100	–	3.400 ± 1.044	0.574 ± 0.008	27.002 ± 0.429
PreferSim	300	16.116 ± 10.426	1.116 ± 0.014	1.647 ± 0.061	111.380 ± 0.757
CodeBLEU	300	–	10.397 ± 3.558	3.076 ± 0.091	110.948 ± 0.345
PreferSim	500	16.116 ± 10.426	1.831 ± 0.022	4.406 ± 2.405	277.082 ± 2.889
CodeBLEU	500	–	16.849 ± 5.128	9.017 ± 0.528	283.968 ± 5.122
MKP-ACO / EoH					
PreferSim	100	0.318 ± 0.291	0.343 ± 0.007	0.814 ± 0.035	30.849 ± 1.381
CodeBLEU	100	–	6.414 ± 2.998	0.556 ± 0.027	26.241 ± 0.461
PreferSim	300	0.318 ± 0.291	1.017 ± 0.005	6.026 ± 0.139	118.829 ± 0.618
CodeBLEU	300	–	19.487 ± 8.975	2.578 ± 0.065	133.052 ± 7.577
PreferSim	500	0.318 ± 0.291	1.647 ± 0.007	17.523 ± 0.472	368.101 ± 3.205
CodeBLEU	500	–	31.955 ± 14.481	5.121 ± 0.179	247.284 ± 1.281
MKP-ACO / MCTS-AHD					
PreferSim	100	0.318 ± 0.291	0.342 ± 0.002	0.620 ± 0.013	33.866 ± 0.157
CodeBLEU	100	–	7.654 ± 3.056	0.524 ± 0.013	29.934 ± 1.175
PreferSim	300	0.318 ± 0.291	1.022 ± 0.003	3.347 ± 0.147	127.526 ± 0.576
CodeBLEU	300	–	23.255 ± 8.966	2.299 ± 0.088	138.502 ± 0.606
PreferSim	500	0.318 ± 0.291	1.591 ± 0.008	7.978 ± 0.401	260.771 ± 2.427
CodeBLEU	500	–	36.815 ± 14.310	4.949 ± 0.201	345.659 ± 4.518

Table 20: Online runtime breakdown of ReAL-AHD.

Task	Total (s)	Capture (s)	Compare (s)	Similarity relation update (s)	Maintenance
Constructive TSP	12262.4	3477.1	352.1	28.8	31.5%
Online BPP	8086.8	5.3	2.8	6.6	0.18%
MKP-ACO	6667.9	1250.6	217.5	13.1	22.2%

Table 20 isolates the relation-maintenance component of ReAL-AHD. The update column combines graph insertion and edge repair, incremental family assignment, triggered reclustering, and similarity pruning. Here, $T_{\text{maintenance}} = T_{\text{capture}} + T_{\text{compare}} + T_{\text{update}}$, and the maintenance share is $T_{\text{maintenance}}/T_{\text{search}}$. Table 20 reports shares of 31.5% for constructive TSP, 22.2% for MKP-ACO, and 0.18% for Online BPP. Capture accounts for 90.1% and 84.4% of the measured relation cost on TSP and MKP-ACO, respectively. Graph and family updates account for at most 0.24% of total search time in all three tasks. This decomposition identifies preference execution as the main cost on TSP and MKP-ACO, while showing that the relational structure itself can be maintained with little additional computation.

BehaveSim and PreferSim. We examine two discovered heuristics, A and B, whose mean PreferSim and BehaveSim over $M{=}8$ TSP100 instances are 0.982 and 0.357, respectively. Figure 7 focuses on one instance, where the corresponding scores are 0.990 and 0.492.

The two functions share an isolation-aware nearest-neighbor rule. The excerpts below retain their scoring expressions and final decision branches; candidate loops and bookkeeping are omitted.

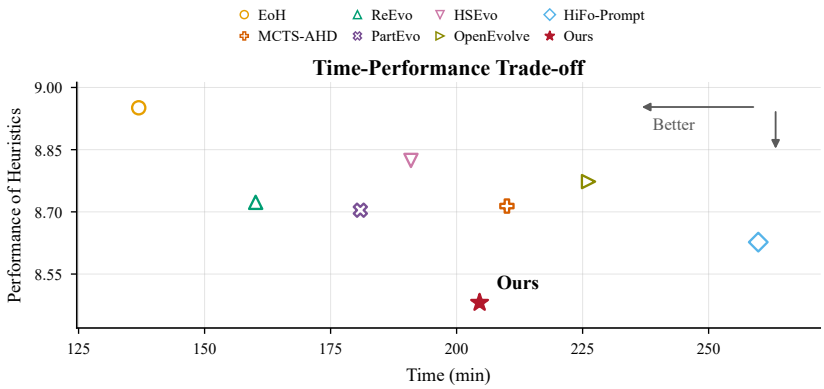

Figure 6: Total search time and final test objective on constructive TSP100.

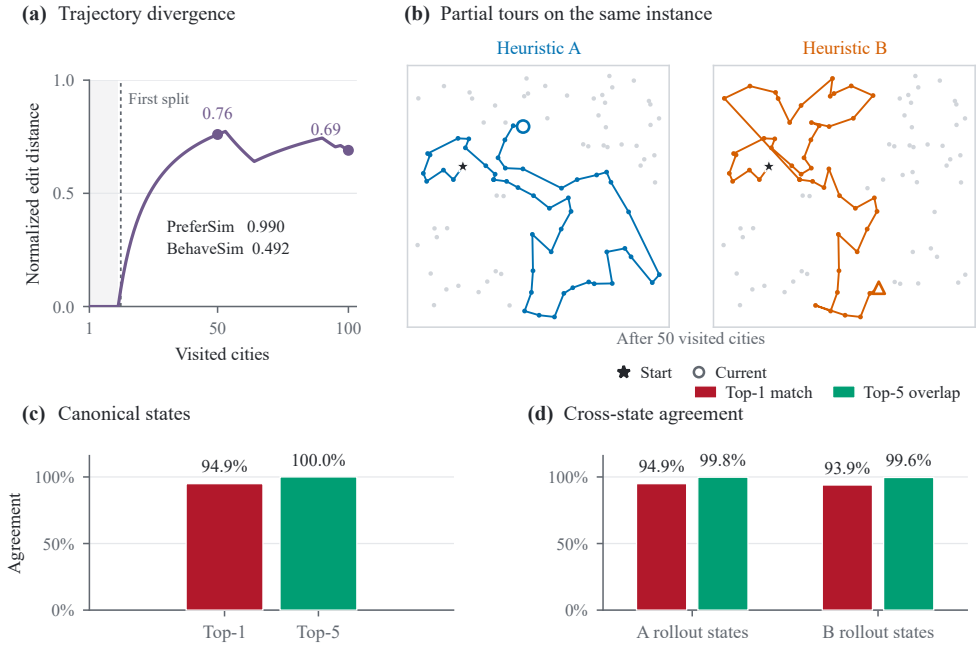

Figure 7: Two discovered heuristics measured with BehaveSim and PreferSim. (a) Edit distance between equally long rollout prefixes, normalized by the number of visited cities; the first differing choice appends the thirteenth city. (b) Open partial paths after 50 visited cities. (c) Agreement on canonical states generated by a deterministic controller independently of A and B. (d) Agreement on states visited by A and, separately, by B.

Here, `U` is the number of unvisited cities, and `best_node` minimizes the candidate score. In A, `min1` and `min2` are the two smallest distances from a candidate to other unvisited cities or the destination. The adaptive weights `w1` and `w2` combine `min1` with `two_step`, the minimum candidate–intermediate–destination distance over admissible future cities. The terms `-gamma * future_cost` and `-delta * gap` favor candidates with costly future connections or a large gap between their nearest alternatives. In B, `d_later` is the nearest-future distance; its coefficient `gamma = 0.7 * p`, together with `alpha = 1.0 - 0.4 * p`, gives isolation more weight relative to immediate travel cost when many cities remain. The final branches expose the shared decision rule: A (`best_cost_later`) and B (`best_d_later`) both measure the score-minimizing candidate's nearest-future distance. When this distance is smaller than the immediate travel distance, both functions return `nearest_node` on these instances. Although B first checks `best_saving`

> 0, its saving expression satisfies $d(c, z) - d(c, i) - d(i, z) \leq 0$ for Euclidean distances, where c, i, and z denote the current city, candidate, and destination. Thus, the positive-savings return is inactive, and both functions use the same nearest-neighbor fallback condition despite their different scoring terms.

Heuristic A

```
alpha = 1.0
beta = 1.0 / (U + 1)
gamma = 0.5 * U / (U + 1)
delta = 0.2
...
gap = (min2 - min1) if min2 != float('inf') else 0.0
future_cost = w1 * min1 + w2 * two_step
score = (alpha * d_cur + beta * d_dest
        - gamma * future_cost - delta * gap)
...
if best_cost_later < distance_matrix[current_node, best_node]:
    return nearest_node
return best_node
```

Heuristic B

```
p = U / (U + 1.0)
alpha = 1.0 - 0.4 * p
beta = 1.0 / (U + 1.0)
gamma = 0.7 * p
delta = 0.2 * (1.0 - p)
...
saving = (distance_matrix[current_node, destination_node]
         - (d_now + d_dest))
score = (alpha * d_now + beta * d_dest
        - gamma * d_later + delta * saving)
...
if best_d_later < distance_matrix[current_node, best_node]:
    if (best_saving_node is not None
        and best_saving_node != best_node and best_saving > 0):
        return best_saving_node
    return nearest_node
return best_node
```

Replaying the functions locates the first disagreement after 12 visited cities, at current city 65. The different choices change both the current city and the unvisited set, allowing subsequent trajectories to separate: normalized prefix edit distance reaches 0.76 after 50 visited cities (Figure 7(a,b)). Nevertheless, the functions agree on the next city in 94.9% of canonical states and have 100% top-5 set overlap (Figure 7(c)). This agreement also holds on states encountered during execution: top-1 agreement is 94.9% on A's states and 93.9% on B's states, with top-5 overlap of 99.8% and 99.6%, respectively (Figure 7(d)). The case illustrates how PreferSim captures a shared decision pattern while trajectory comparison reflects the state changes induced by occasional disagreements.

F Case Studies

We examine constructive TSP runs and present three situations observed during evolutionary search to demonstrate the effectiveness of ReAL-AHD. Objectives are tour lengths (lower is better); node and family identifiers are local to each run.

Mechanism discovery and refinement. Table 21 traces two instances of delayed mechanism reuse. Node 124 introduces shortlist-based greedy rollouts, which node 299 combines with adaptive regret scoring. Node 303 introduces exact dynamic programming for small remaining subproblems; node 381 retains this branch and restores the stronger rollout-based rule for larger states. These intermediate programs contribute reusable mechanisms to later global-best discoveries. The lineage culminates in node 386, which constructs a complete path and refines it with 2-opt.

Table 21: Selected derivations. Bold objectives mark global-best updates.

Node	Parents	Op.	Obj.↓	Mechanism
124	63	$m1$	11.582	Shortlist candidates, then simulate complete greedy rollouts.
299	124,180,264	$s2$	**8.722**	Combine adaptive regret scoring with the rollout framework.
303	299,264,300	$e1$	10.687	Use exact Held–Karp DP for at most 10 remaining cities.
381	299,303	$e2$	**8.713**	Retain the DP branch; restore scoring and rollouts for larger states.
386	381,330,264	$e1$	**8.587**	Construct a complete path and refine it with 2-opt.

Cross-family mechanism transfer. Global-best ownership follows $f_0 \to f_1 \to f_0$ (Table 22). Node 402 combines a distance-threshold shortlist from f_0 with greedy rollouts developed in f_1, reaching 8.6267. Node 449 then combines f_1's lookahead and geometric off-path terms with f_5's adaptive regret and bottleneck scoring, reaching 8.6150. The improvements accompany identifiable reuse of mechanisms across family boundaries.

Table 22: Late global-best succession.

Node	Family	Op.	Obj.↓	Mechanism origins
155	f_0	$s2$	8.6750	Interpolated second-best/mean regret, incorporating node 25 from f_5.
402	f_1	$s2$	8.6267	Distance-threshold shortlist (f_0) and greedy rollouts (f_1).
449	f_0	$e2$	8.6150	Lookahead and off-path terms (f_1); adaptive regret and bottleneck scoring (f_5).

Adaptive planning. Figure 8(a) shows how family budgets adapt to recent rewards and membership changes. The family f_7 receives 10 of 20 slots and f_0's allocation grows from 6 to 7 as its mean reward rises. Figure 8(b) tracks operator-specific node choices across consecutive rounds in f_2, showing both temporal switching and distinct selections across operators.

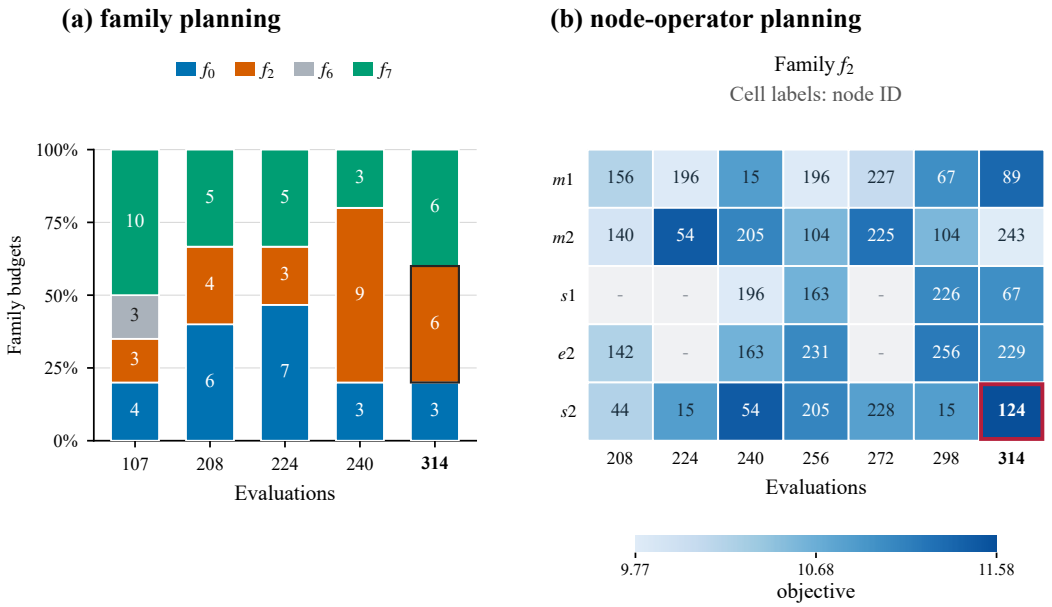

Figure 8: Adaptive planning. (a) Family budget shares at selected rounds. (b) Node selections in f_2 over consecutive planning rounds. Cell labels give node IDs and colors encode their objectives. The red-outlined selection yields a global-best node.

G PROMPT TEMPLATES

Each generation request consists of the shared system message and one operator-specific user message. When memory retrieval returns records, the formatted memory block is appended to the user message. The system message specifies the task through {`func_name`}, {`task`}, and {`inout_inf`}. For initialization, {`seed_func`} supplies the reference implementation and {`other_inf`} supplies additional domain knowledge. For single-source operators, {`algorithm`} and {`code`} contain the selected algorithm's description and implementation. For multi-source operators, {`num_indivs`} is the number of source programs and {`indivs_block`} concatenates their descriptions and code in the supplied order.

The same system message supplies the function interface and response format for initialization and all evolutionary operators. Initialization uses the reference implementation and domain knowledge. The six evolutionary operators use the source programs specified in Table 9 and Appendix D.5. Operators $m1$ and $m2$ receive the selected node; $s1$ receives the selected node followed by its greedy high-Q descendant path; and $e1$ receives the best node from each family. For $e2$, the elite from another family appears first and the selected node appears last, preserving the selected node's form while drawing inspiration from the elite. For $s2$, the supplied sequence connects the selected node to the other-family elite, with intermediate nodes.

Each retrieved memory record contributes its offspring description and observed outcome, labeled `EFFECTIVE` for positive records or `INEFFECTIVE` for negative records. The header is followed immediately by one line per retrieved record, in retrieval order. Each line uses the first record format when a numerical improvement is available for a non-root event, and the second otherwise. The resulting block is appended to the operator-specific user message.

Source Algorithm Entry

No.{i} algorithm's description and its corresponding code are:
{algorithm}
{code}

System Prompt

YOUR ROLE
Design heuristic algorithms for complex optimization problems.

PROBLEM
Function: {func_name}
Description: {task}
Spec: {inout_inf}

OUTPUT FORMAT
Output ONLY these two parts, nothing else:
1. A one-sentence algorithm description in curly braces: {description}
2. Complete runnable Python code in ```` ```python ... ``` ```` (include imports inside the function)

Prompt for Initialization ($i1$)

TASK: Generate Initial Algorithm
You are tasked with designing a novel heuristic algorithm from scratch.

REFERENCE IMPLEMENTATION
Below is a baseline implementation to illustrate the expected code structure:
{seed_func}

ADDITIONAL DOMAIN KNOWLEDGE
{other_inf}

Prompt for Mechanism Innovation ($m1$)

EXISTING ALGORITHM
Algorithm Description: {algorithm}
Code:
{code}

TASK: Innovate the Algorithm's Mechanisms
Innovate the given algorithm's core mechanisms.

Prompt for Local Refinement ($m2$)

EXISTING ALGORITHM
Algorithm Description: {algorithm}
Code:
{code}

TASK: Make a Local Improvement
Make a small, local improvement to this algorithm while preserving its core mechanisms.

Prompt for Global Diversification ($e1$)

EXISTING ALGORITHMS
I have {num_indivs} existing algorithms with their codes as follows:
{indivs_block}

TASK: Generate a Diverse Algorithm
Create a new algorithm that is **totally different** from the given algorithms — use different data structures, control flow, or an algorithmic paradigm they do not rely on.

Prompt for Elite Inspiration ($e2$)

EXISTING ALGORITHMS
I have {num_indivs} existing algorithms with their codes as follows:
{indivs_block}

TASK: Generate a Cross-Inspired Algorithm
Create a new algorithm that has a **similar form to algorithm No.**{last_idx} while being **inspired by algorithm No.1**.

Prompt for Lineage Synthesis ($s1$)

EXISTING ALGORITHMS
I have {num_indivs} existing algorithms with their codes:
{indivs_block}

TASK: Generate an Inspired Algorithm
Create a new algorithm that is inspired by all the above algorithms. Identify ideas in the provided algorithms that are clearly helpful toward a better algorithm.

Prompt for Cross-Family Bridging ($s2$)

EXISTING ALGORITHMS
I have a sequence of algorithms that gradually transition from one approach to another. They are ordered by similarity, showing how mechanisms evolve step by step:
{indivs_block}

TASK: Bridge Two Algorithm Families
Create a new algorithm that bridges the gap between them by combining the strengths of both ends.

Memory Block Template

PAST EXPERIENCE (this or similar algorithms)
Build on **EFFECTIVE** ideas; avoid repeating **INEFFECTIVE** ones.

```
[{label}] ({status}, improvement={improvement}) {idea}
[{label}] ({status}) {idea}
```

Here, {status} is the recorded outcome, such as `global_best`, `elite_admission`, `poor_performance`, or `timeout`; {idea} is the stored one-sentence offspring description. The displayed improvement is the fitness change.