INTERACTION-AWARE CO-EVOLUTION FOR AUTOMATED ALGORITHM DESIGN

Anonymous authors
Paper under double-blind review

ABSTRACT

Large language models (LLMs) enable automated algorithm design (AAD) by generating and refining executable heuristics. In solver–instance co-evolution, heuristics are improved alongside instance generators that expose their weaknesses. However, performance feedback in LLM-based AAD does not directly explain why a solver struggles: a high error may arise from limited overall solver capability, general instance difficulty, or a specific solver–instance interaction. Distinguishing these factors is therefore important for guiding AAD toward targeted improvements. We propose Interaction-Aware Co-Evolution (IACE), an evidence-guided framework for AAD that connects interaction diagnosis with controlled validation and evidence reuse. IACE extracts interaction residuals from performance feedback after accounting for average solver and instance effects. These residuals identify candidate weaknesses and guide hypotheses about the instance conditions under which a heuristic struggles. A bidirectional controlled validation procedure compares parent–child heuristics across paired original and modified instances to assess whether a proposed change addresses the targeted weakness. The resulting records are organized in a structured evidence memory, separating validated findings from unverified hypotheses and unsuccessful attempts. By retrieving relevant evidence for both heuristic modification and instance generation, IACE supports informed search across AAD iterations. Throughout this process, LLM parameters and the solver backbone remain fixed, while a designated heuristic component evolves. Experiments demonstrate state-of-the-art performance on widely recognized combinatorial optimization datasets. IACE brings diagnosis, validation, and evidence reuse into a unified process for AAD.

1 INTRODUCTION

Designing effective heuristics for combinatorial optimization requires understanding not only which algorithmic rules work, but also the conditions under which they fail. Large language models (LLMs) enable automated algorithm design (AAD) by generating and refining executable heuristics from task descriptions and performance feedback (Liu et al., 2024a; Ye et al., 2024; Van Stein & Bäck, 2024; Liu et al., 2024b; Zheng et al., 2025). Solver–instance co-evolution extends this process by jointly evolving heuristics and instance generators: heuristics adapt to challenging instances, while generators seek instances that expose their weaknesses (Hillis, 1990; Potter & De Jong, 2000). However, the value of this feedback depends on how failures are interpreted. A central challenge is therefore to turn observed failures into actionable evidence for subsequent algorithmic improvement.

Poor performance alone does not identify a specific algorithmic weakness. A large error may reflect limited overall solver capability, general instance difficulty, or a particular mismatch between a heuristic's decision rules and an instance's structure. These explanations suggest different directions for improvement, yet aggregate performance does not distinguish them. An instance that challenges most solvers offers limited evidence about the distinctive weakness of a particular heuristic, while strong average performance can conceal failures under specific instance conditions (Rice, 1976; Xu et al., 2008; Smith-Miles & Muñoz, 2023). Performance feedback can thus reveal where a solver struggles without explaining what should change.

We argue that targeted improvement requires connecting interaction diagnosis, controlled validation, and evidence reuse. Diagnosis identifies potentially informative failures, but an observed pattern

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.

does not establish its explanation. An LLM-generated account of that pattern remains a hypothesis (Shinn et al., 2023; Madaan et al., 2023), and an improvement in a modified heuristic does not by itself validate the account: the modification may help broadly without addressing the hypothesized weakness. Moreover, reusing explanations without distinguishing their evidential status risks propagating unsupported assumptions. Co-evolution therefore requires a process that tests proposed explanations and preserves the distinction between observed patterns, supported findings, and unresolved hypotheses.

To this end, we propose *Interaction-Aware Co-Evolution* (IACE), an evidence-guided framework for AAD. IACE accounts for average solver and instance effects and extracts interaction residuals that highlight unexpectedly poor solver–instance outcomes. These residuals serve as diagnostic signals for identifying candidate weaknesses and formulating hypotheses about the instance conditions responsible for them. IACE then applies bidirectional controlled validation, evaluating parent and child heuristics on paired original and modified instances. Comparing the heuristics on the same instances measures the effect of a heuristic modification; comparing this effect across paired instances assesses its dependence on the targeted condition. Together, these comparisons test whether the observed improvement supports the proposed explanation within the evaluated conditions.

IACE organizes the resulting records in a structured evidence memory that separates validated findings, unverified hypotheses, and unsuccessful attempts. A meta-strategy generator retrieves relevant evidence to guide both heuristic modification and instance generation, allowing subsequent iterations to build on supported findings and reconsider unsuccessful directions. This connects the two generation processes through accumulated evidence about solver–instance interactions, alongside performance-based selection. Throughout the process, LLM parameters and the solver backbone remain fixed, while a designated heuristic component evolves.

Contributions. Our main contributions are as follows: **(i)** Across diverse combinatorial optimization domains, such as TSP, OP, CVRP, MKP, and BPP, we introduce interaction-aware diagnosis for solver–instance co-evolution, identifying candidate heuristic weaknesses after accounting for average solver capability and instance difficulty. **(ii)** We develop bidirectional controlled validation across matched instance conditions to assess whether heuristic modifications address the hypothesized weaknesses. **(iii)** We integrate diagnosis and validation with structured evidence memory, enabling heuristic and instance generation to reuse findings according to evidential status. **(iv)** Experiments on established public TSPLIB instances demonstrate state-of-the-art performance, supporting the effectiveness of connecting diagnosis, validation, and evidence reuse for automated algorithm design.

2 Related Work

LLM-Based Automated Algorithm Design. Large language models (LLMs) have recently emerged as a powerful paradigm for automated algorithm design (AAD), enabling executable heuristics to be generated and improved through iterative search. Representative approaches include evolutionary heuristic generation (Liu et al., 2024a), reflective evolution (Ye et al., 2024), evolutionary metaheuristic synthesis (Van Stein & Bäck, 2024), Monte Carlo tree search (Zheng et al., 2025), neighborhood-based refinement (Xie et al., 2025), and analyses of evolutionary search mechanisms (Zhang et al., 2024). LLM4AD provides a unified platform for this emerging paradigm (Liu et al., 2024b), while EoH-S extends individual heuristic evolution toward heuristic sets (Liu et al., 2026a); a recent survey further summarizes the broader development of LLM-based algorithm design (Liu et al., 2026b). Beyond heuristic optimization, LLM-guided search has enabled mathematical and algorithmic discoveries (Romera-Paredes et al., 2024), evolutionary coding agents for scientific discovery (Novikov et al., 2025), direct optimization with LLMs (Yang et al., 2024; Liu et al., 2024c), code-level neural architecture search (Chen et al., 2023), and automatic reward design (Ma et al., 2024). These developments build on the broader foundations of hyper-heuristics (Burke et al., 2013; 2010) and Programming by Optimization (Hoos, 2012). However, most existing LLM-based AAD methods primarily rely on aggregate performance or textual feedback, which reveals whether a candidate performs poorly but provides limited evidence about why a particular solver fails under a particular instance condition.

Instance-Aware Automated Optimization. Automated algorithm design predates LLM-based methods and includes grammar-based heuristic generation (Mascia et al., 2014), automatic discovery of local-search procedures (Fukunaga, 2008), component-based solver construction (KhudaBukhsh et al., 2016), automatic ant-colony design (Lopez-Ibanez & Stutzle, 2012), and genetic-programming-based heuristic evolution (Burke et al., 2006). Algorithm configuration systems such as ParamILS (Hutter et al., 2009), sequential model-based optimization (Hutter et al., 2011), automatic MIP solver configuration (Hutter et al., 2010), and IRACE (López-Ibáñez et al., 2016) further established systematic search over algorithmic design spaces. In parallel, algorithm selection research has long emphasized instance-dependent performance, from the classical formulation of Rice (1976) to SATzilla (Xu et al., 2008), Hydra (Xu et al., 2010), AutoFolio (Lindauer et al., 2015), ISAC (Kadioglu et al., 2010), runtime prediction (Hutter et al., 2014), broader algorithm-selection studies (Kotthoff, 2016), and Instance Space Analysis (Smith-Miles & Muñoz, 2023). Co-evolution offers a complementary perspective by jointly adapting interacting populations (Hillis, 1990; Potter & De Jong, 2000; Ma et al., 2018), where solvers and challenging instances can be improved against each other. Classical studies of hard knapsack (Pisinger, 2005) and CVRP instances (Uchoa et al., 2017) further demonstrate that instance structure can expose algorithmic weaknesses. Nevertheless, instance hardness alone cannot distinguish global solver weakness or general instance difficulty from a specific solver–instance interaction. IACE addresses this distinction by removing marginal solver and instance effects and using residual interactions to identify condition-specific failures for subsequent algorithmic improvement.

Evidence-Guided Co-Evolution. Reflection and persistent experience have become important mechanisms for iterative LLM systems. Reflexion uses verbal reinforcement from previous trials (Shinn et al., 2023), Self-Refine performs iterative refinement through self-generated feedback (Madaan et al., 2023), Generative Agents retrieve accumulated memories to guide future behavior (Park et al., 2023), and Voyager maintains reusable executable skills during open-ended exploration (Wang et al., 2023). However, a plausible explanation or a successful modification does not necessarily establish that the hypothesized failure mechanism is correct. Reliable empirical comparison therefore requires careful statistical treatment across multiple evaluations (Demšar, 2006; Garcia & Herrera, 2008) and attention to variability across experimental runs (Henderson et al., 2018). IACE builds on these directions by treating solver–instance interactions as diagnostic signals, validating proposed explanations through matched controlled comparisons, and maintaining validated findings, unresolved hypotheses, and unsuccessful attempts with distinct evidential status. We evaluate this process across established combinatorial optimization settings, including TSP and TSPLIB (Reinelt, 1991; Applegate et al., 2011), Ant Colony Optimization (Dorigo et al., 1996; Dorigo & Gambardella, 1997), the Orienteering Problem and its variants (Golden et al., 1987; I-Ming et al., 1996; Vansteenwegen et al., 2011), and the 0–1 knapsack problem (Martello & Toth, 1988). In contrast to approaches that primarily optimize aggregate performance or generate increasingly difficult instances, IACE connects interaction diagnosis, controlled validation, and structured evidence reuse to convert observed solver–instance failures into actionable evidence for subsequent AAD.

3 METHOD

Interaction-Aware Co-Evolution (IACE) separates interaction-based hypothesis generation from confirmatory validation. Cross-evaluation first identifies solver–instance interactions that cannot be explained by average solver or instance difficulty. The interaction miner then proposes a targeted condition and a parent–child heuristic modification. Fresh matched evaluations subsequently test whether the proposed modification specifically repairs the diagnosed condition, and only statistically supported records are promoted to reusable evidence memory.

3.1 INTERACTION DIAGNOSIS

For a minimization instance $x \in \mathcal{X}$, a fixed solver backbone B executes heuristic h under budget β and random seed ω. With a positive reference value $c_{\mathrm{ref}}(x)$, define

$$C(h, x; \omega) = f_x(B[h](x; \omega, \beta)),$$
$$\mathrm{gap}(h, x; \omega) = \frac{C(h, x; \omega) - c_{\mathrm{ref}}(x)}{c_{\mathrm{ref}}(x)}. \tag{1}$$

The reference is shared across heuristics, is not selected to favor a candidate, and need not be optimal.

For $H \geq 2$ heuristics and $N \geq 2$ diagnostic instances, average m evaluation replicates:

$$\begin{aligned} Y_{hx} &= \frac{1}{m}\sum_{\ell=1}^{m} \mathrm{gap}(h, x; \omega_\ell), \\ r_{hx} &= Y_{hx} - \bar{Y}_{h\cdot} - \bar{Y}_{\cdot x} + \bar{Y}_{\cdot\cdot}. \end{aligned} \tag{2}$$

Positive r_{hx} indicates a gap larger than predicted by additive solver and instance effects. These residuals are panel-relative screening signals rather than confirmatory evidence.

Let

$$P_H = I_H - \frac{1}{H}\mathbf{1}\mathbf{1}^\top, \qquad P_N = I_N - \frac{1}{N}\mathbf{1}\mathbf{1}^\top, \tag{3}$$

and write $M = \mathbb{E}[Y]$. The population and empirical interaction matrices are

$$\Gamma = P_H M P_N, \qquad R = P_H Y P_N. \tag{4}$$

Double centering removes additive solver and instance effects, and each population interaction can equivalently be written as

$$\Gamma_{hx} = \frac{1}{HN}\sum_{i=1}^{H}\sum_{j=1}^{N} \left(M_{hx} - M_{hj} - M_{ix} + M_{ij}\right). \tag{5}$$

Thus global diagnosis averages local rectangular interaction contrasts.

Suppose the panel is fixed before evaluation and each centered replicate cell is σ^2-sub-Gaussian. Let

$$\rho = \left(1 - \frac{1}{H}\right)\left(1 - \frac{1}{N}\right).$$

With probability at least $1 - \eta$,

$$\|R - \Gamma\|_{\max} \leq \begin{cases} 4\rho\sigma\sqrt{\dfrac{2}{m}\log\dfrac{2HN}{\eta}}, & \text{arbitrary within-replicate dependence,} \\ \sigma\sqrt{\dfrac{2\rho}{m}\log\dfrac{2HN}{\eta}}, & \text{independent cells.} \end{cases} \tag{6}$$

For either applicable radius b, flagging $r_{hx} > \lambda + b$ excludes cells with $\Gamma_{hx} \leq \lambda$ on the same event and detects every cell with $\Gamma_{hx} > \lambda + 2b$. Detailed recovery proofs, diagnostic sample complexity, and limitations of diagnosis are given in Appendix B.

3.2 Matched Controlled Validation

Let $p(h)$ denote the parent of candidate h, and let $o \in \mathcal{V}$ denote the original condition. For each fresh matched group u, latent factors z_u generate

$$x_v^{(u)} = g_v(z_u; \theta_v), \qquad \bar{C}_{hv}^{(u)} = \frac{1}{s}\sum_{\ell=1}^{s} C\Big(h, x_v^{(u)}; \omega_{u\ell}\Big), \quad v \in \mathcal{V}. \tag{7}$$

Programs, generators, tested comparisons, descriptor checks, and seed protocols are fixed before observing fresh validation groups. Groups are independent, whereas outcomes within a group may be dependent.

Let

$$c_v^{(u)} = c_{\mathrm{ref}}(x_v^{(u)}), \qquad q_u = c_o^{(u)} > 0.$$

Using the same scale q_u for every condition, define

$$\begin{aligned} A_{hv}^{(u)} &= \frac{[\bar{C}_{p(h)v}^{(u)} - c_v^{(u)}] - [\bar{C}_{p(h)o}^{(u)} - c_o^{(u)}]}{q_u}, \\ G_{hv}^{(u)} &= \frac{\bar{C}_{p(h)v}^{(u)} - \bar{C}_{hv}^{(u)}}{q_u}, \\ D_{hv}^{(u)} &= G_{hv}^{(u)} - G_{ho}^{(u)}. \end{aligned} \tag{8}$$

Here A measures increased reference-adjusted difficulty for the parent, G measures candidate improvement under the target condition, and D measures the additional improvement relative to the original condition.

Equivalently, with

$$L_{hv}^{(u)} = \frac{\bar{C}_{hv}^{(u)} - c_v^{(u)}}{q_u}, \tag{9}$$

we have

$$A = L_{pv} - L_{po}, \qquad G_v = L_{pv} - L_{hv}, \qquad D = L_{pv} - L_{hv} - L_{po} + L_{ho}. \tag{10}$$

Hence D is exactly the local invariant interaction contrast associated with Eq. (5). For independent matched groups,

$$\mathbb{E}[D] = \mathbb{E}[G_v] - \mathbb{E}[G_o], \qquad \mathrm{Var}(\bar{D}) = \frac{\mathrm{Var}(G_v) + \mathrm{Var}(G_o) - 2\,\mathrm{Cov}(G_v, G_o)}{n}. \tag{11}$$

For a candidate–condition pair, write $Z_A = A$, $Z_G = G_v$, and $Z_D = D$, with $\mu_k = \mathbb{E}[Z_k]$. The pair is population-supported when

$$\mu_A > \tau_A, \qquad \mu_G > \tau_G, \qquad \mu_D > \tau_D, \tag{12}$$

for prespecified nonnegative thresholds.

3.3 Finite-Sample Evidence Promotion

Condition on the complete history $\mathcal{F}$ immediately before a validation batch. For pair j and $k \in \{A, G, D\}$, assume that independent matched groups satisfy

$$\mathbb{E}\left[\exp\left\{\lambda\left(Z_{jk}^{(u)} - \mu_{jk}\right)\right\}\middle|\mathcal{F}\right] \le \exp\left(\frac{\lambda^2 \nu_{jk}^2}{2}\right), \qquad \lambda \in \mathbb{R}, \tag{13}$$

for known conservative scales $\nu_{jk} > 0$.

Allocate $a_j \in (0, 1)$ before evaluation such that $\sum_j a_j \le \alpha$. At a predetermined sample size n_j, use

$$L_{jk} = \bar{Z}_{jk} - \nu_{jk}\sqrt{\frac{2}{n_j}\log\frac{1}{a_j}}, \qquad \mathrm{promote}(j) \iff L_{jk} > \tau_k \quad \forall k \in \{A, G, D\}. \tag{14}$$

A pair is population-unsupported when at least one inequality in Eq. (12) fails. The joint rule above is an intersection–union test and satisfies

$$\Pr(\exists j : j \text{ is population-unsupported and promoted}) \le \alpha. \tag{15}$$

For a supported pair j, define its coordinate-wise margins

$$\Delta_{jk} = \mu_{jk} - \tau_k > 0.$$

Its statistical gate passes with probability at least $1 - \zeta$ whenever

$$n_j > \max_{k \in \{A,G,D\}} \frac{2\nu_{jk}^2}{\Delta_{jk}^2}\left(\sqrt{\log\frac{1}{a_j}} + \sqrt{\log\frac{3}{\zeta}}\right)^2. \tag{16}$$

This inverse-quadratic dependence on the weakest validation margin is unavoidable in general. For any $\nu > 0$, $0 < \Delta \le \nu/2$, and $\alpha + \zeta < 1$, there exist bounded nonnegative four-outcome validation distributions with identical diagnostic information such that any procedure having false promotion at most α under the unsupported distribution and power at least $1 - \zeta$ under the supported distribution must satisfy

$$n \ge \frac{\nu^2}{\Delta^2}\,\mathrm{kl}(1 - \zeta, \alpha), \qquad \mathrm{kl}(p, q) = p\log\frac{p}{q} + (1 - p)\log\frac{1 - p}{1 - q}. \tag{17}$$

Thus weak condition-specific effects intrinsically require $\Omega(\nu^2/\Delta^2)$ fresh matched groups. Previous diagnostic and memory records may guide future proposals, whereas newly defined numerical claims require fresh validation. Complete proofs, simultaneous-coverage variants, conservative scale construction, adaptive stopping, and evidence transfer are given in Appendices D–F.

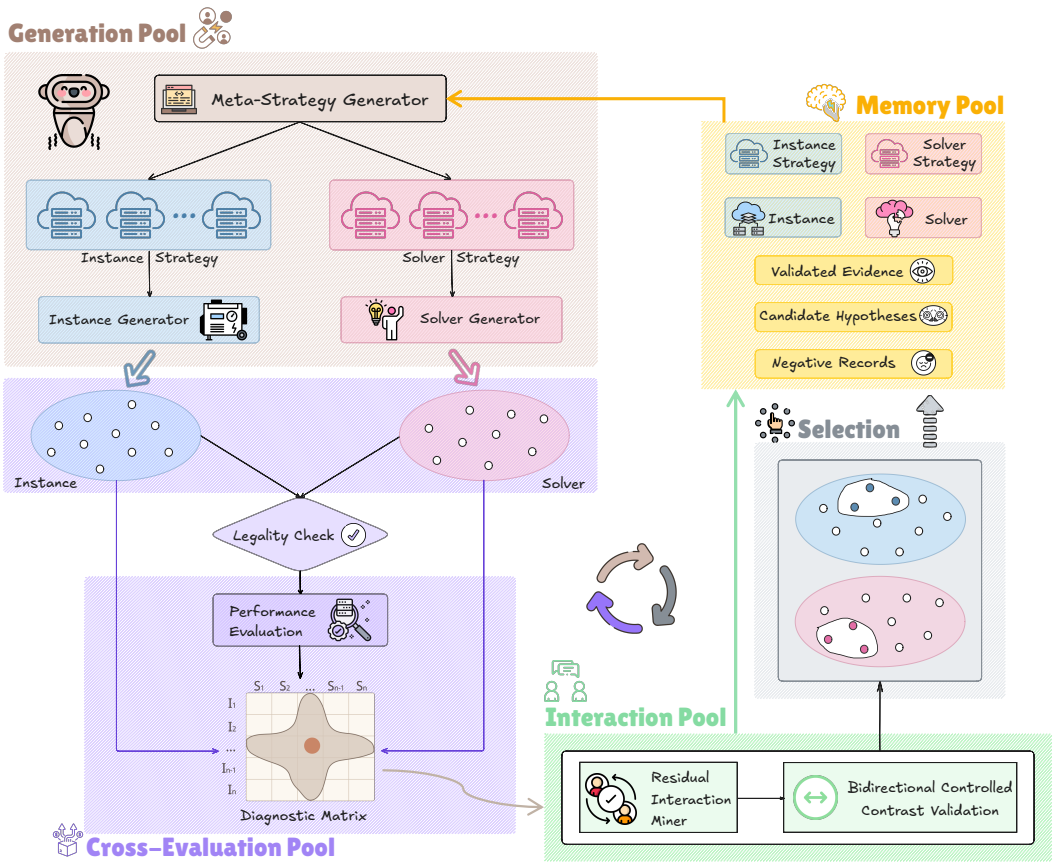

Figure 1: Architecture of the proposed IACE framework.

3.4 Interaction-Aware Co-Evolution

IACE integrates residual-based interaction diagnosis and matched controlled validation into a closed-loop co-evolutionary process. As illustrated in Figure 1, the framework is organized into four functional pools: the *generation pool*, *cross-evaluation pool*, *interaction pool*, and *memory pool*. Together, these pools establish an evidence pathway between solver and instance evolution: candidate populations generate interaction observations, informative observations are converted into testable hypotheses, and validation outcomes are retained to guide subsequent search.

Generation Pool. The generation pool produces candidate solvers and instances while keeping the solver backbone and LLM parameters fixed. Rather than generating candidates directly from raw performance feedback, IACE first invokes a meta-strategy generator to construct solver-side and instance-side search strategies from the current populations and retrieved historical evidence. Conditioned on these strategies, the solver generator modifies the designated heuristic component, whereas the instance generator produces candidate instances intended to probe informative solver behaviors. This intermediate strategy layer separates *what to explore* from *how to instantiate a candidate*, allowing accumulated interaction evidence to influence both sides of the co-evolutionary search. Generated programs and instances are subjected to legality and validity checks before entering evaluation.

Cross-Evaluation Pool. Valid solver and instance candidates are jointly evaluated over a shared panel, forming a solver–instance performance matrix. This matrix provides both the performance signal for population selection and the observations required for interaction diagnosis. Specifically, IACE removes the average solver and instance effects by double centering the matrix and uses the resulting interaction residuals to identify solver–instance pairs whose performance is unexpectedly poor relative to their marginal effects. Large positive residuals therefore prioritize interactions that

are more informative than instances that are uniformly difficult or heuristics that are uniformly weak. Importantly, these residuals are treated as diagnostic signals rather than verified explanations of failure.

Interaction Pool. The interaction pool converts diagnostic signals into testable hypotheses. For selected high-residual interactions, the residual interaction miner examines the corresponding heuristic, instance characteristics, and performance records to formulate a candidate weakness together with the instance condition under which it is expected to arise. The resulting hypothesis induces both a solver-side modification and an instance-side controlled variation. These interventions are evaluated using the generalized controlled validation procedure on fresh groups, which compares each candidate heuristic with its parent across matched original and modified instance conditions. This comparison distinguishes a general solver improvement from an improvement that is specifically associated with the targeted condition. A hypothesis is promoted to validated evidence only when the prescribed validity checks and confidence-bound criteria are satisfied; otherwise, it remains unresolved or is recorded as unsupported.

Memory Pool. The memory pool preserves the outcomes of diagnosis and validation together with their evidential status. It stores validated evidence separately from unverified hypotheses and unsuccessful attempts, together with the associated solver and instance strategies. This separation prevents an LLM-generated explanation from being reused as an established finding solely because it is plausible or accompanies an improved candidate. At subsequent iterations, the meta-strategy generator retrieves records relevant to the current solver and instance populations. Validated findings support exploitation of previously identified weaknesses, whereas unresolved hypotheses and negative records provide complementary signals for continued exploration and for avoiding repeatedly ineffective directions. The memory pool therefore acts as a structured interface through which interaction evidence influences future generation.

4 Experiments

4.1 Amplifying Adversarial Hardness under Distributional Shift with IACE

Performance under Standard and Hard Instance Distributions. We further evaluate IACE under both standard and hard TSP instance distributions across six problem sizes. As detailed in Appendix J.2, IACE achieves the lowest gap in all 12 settings. Under the standard distribution, IACE improves over the best-performing baseline by 0.14–0.22 percentage points, while under the hard distribution, the improvement increases to 2.33–3.85 percentage points, corresponding to relative gap reductions of 13.9–31.7%. These results demonstrate that the advantage of IACE persists across the evaluated instance distributions and becomes more pronounced in absolute terms on hard instances.

Deterioration in Solution Quality under Adversarial Instance Distributions. We further evaluate whether IACE can generate challenging instance distributions across diverse combinatorial optimization problems. As detailed in Appendix J.1, instances generated by IACE consistently yield less favorable outcomes for ACO-based solvers across TSP, OP, CVRP, MKP, and BPP. Specifically, the objective values increase for the minimization problems (TSP, CVRP, and BPP) and decrease for the maximization problems (OP and MKP), with the separation from the standard distributions generally becoming more pronounced as the problem size increases. These results demonstrate that IACE can generate challenging instances across heterogeneous problem domains and objective directions.

Interaction Diagnosis during Co-Evolution. Figure 2 shows the interaction residuals r_{hx} that IACE extracts from the cross-evaluation panel at iterations 1, 10, and 20 (10 heuristics $\times$ 10 diagnostic instances). Most cells lie close to zero, indicating that the additive heuristic and instance effects already account for most of the observed gap. Only a few pairs are flagged at each iteration (two, three, and two, respectively), so the diagnostic signal is sparse and localized rather than spread across the panel. The flagged pairs also change over the course of evolution: at iteration 10, heuristic S_7 shows large positive residuals on two different instances (I_1 and I_9, with $r_{hx} = 2.6$ and 4.1), whereas the flagged pairs at iterations 1 and 20 involve other heuristics and instances. These

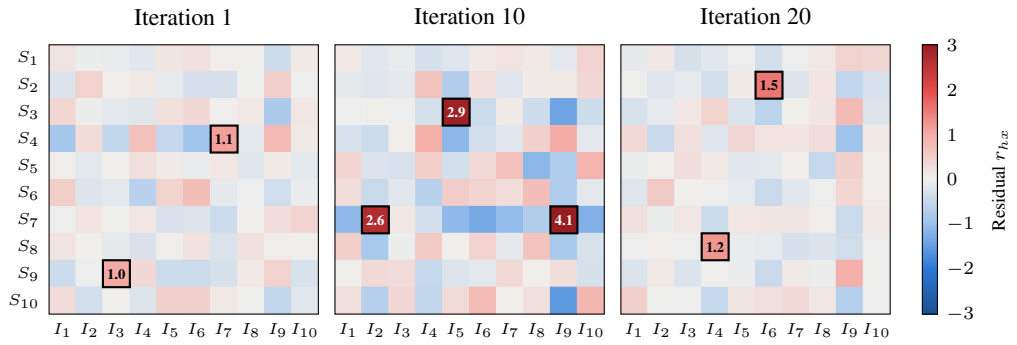

Figure 2: Interaction residuals $r_{hx} = Y_{hx} - \bar{Y}_{h\cdot} - \bar{Y}_{\cdot x} + \bar{Y}_{\cdot\cdot}$ under the hard instance distribution at iterations 1, 10, and 20. Rows correspond to heuristics S_h, and columns correspond to diagnostic instances I_x. Red cells indicate a larger gap than predicted by the additive heuristic and instance effects, whereas blue cells indicate a smaller gap. Outlined cells denote flagged solver–instance pairs. The colour scale is shared across all three panels.

residuals expose specific solver–instance conditions that raw gap alone would not single out, and provide the interaction pool with targeted inputs for hypothesis formation.

Hard Instance Generation across Different Solvers. We further evaluate whether IACE-generated hard instances remain challenging across different TSP solvers. As detailed in Appendix J.5, ReEvo, EoH, MCTS, LLM-LNS, and KGLS all exhibit substantially larger optimality gaps on the hard distribution than on the standard distribution as the problem size increases. Moreover, the separation between the two distributions generally widens at larger problem sizes. This pattern is observed consistently across solvers with different search mechanisms, suggesting that the generated instances capture difficulty beyond the behavior of any individual optimization procedure. The persistent degradation at larger scales further indicates that the generated hardness is not limited to small problem sizes. These results indicate that the hardness induced by IACE is not specific to a single solver, but transfers across multiple solution approaches.

Generalization to Real-World Instances. We further evaluate IACE on TSPLIB instances to assess its generalization beyond the synthetic instance distributions considered in our main experiments. As detailed in Appendix J.3, IACE achieves the lowest overall average optimality gap among the evaluated methods and attains or matches the best performance on the large majority of the evaluated instances. Its advantage is observed across a broad range of problem sizes, including both relatively small instances and large-scale instances with thousands of nodes. Moreover, the strongest competing method varies across instances, whereas IACE maintains consistently competitive performance across different scales. This suggests that the improvements of IACE are not tied to a particular instance size or to the characteristics of the synthetic distributions used in our evaluation. Together with the results on standard and hard instance distributions, these findings provide further evidence that IACE generalizes effectively to established TSP instances with diverse structural characteristics.

4.2 Ablation Study

Table 1 reports the ablation results for the main components of IACE. Full IACE obtains a Solver Gap of 8.26% and an Instance Gap of 12.10%. Removing RIM leads to the largest performance drop, with the Solver Gap increasing to 9.63% and the Instance Gap decreasing to 10.38%. This suggests that the interaction information captured by RIM is important for locating unresolved solver weaknesses and directing the following search. Controlled validation and memory also contribute to both metrics: removing either component weakens solver performance and reduces the hardness of the generated instances. The meta-strategy module has a smaller but still clear effect, indicating that higher-level search guidance helps coordinate the evolution process.

Table 1: Ablation study with mean ± standard deviation values. Lower Solver Gap and higher Instance Gap are better.

Variant	Solver Gap (%) ↓	Instance Gap (%) ↑
Full IACE	$\mathbf{8.26 \pm 0.62}$	$\mathbf{12.10 \pm 0.58}$
w/o Meta-Strategy	8.71 ± 0.31	11.54 ± 0.42
w/o RIM	9.63 ± 0.46	10.38 ± 0.57
w/o Controlled Validation	9.18 ± 0.61	11.27 ± 0.68
w/o Memory	9.04 ± 0.40	10.92 ± 0.49

4.3 Iterative Co-Evolution Dynamics

IACE couples instance generation with solver improvement through an iterative feedback process. As shown in Figure 3, the hardness gap of the generated instances increases over the iterations, while the solver gap gradually decreases. This means that the instance generator continues to produce more difficult cases as the solver evolves, instead of remaining within a fixed difficulty range. At the same time, the solver improves despite being exposed to increasingly challenging instances. The two trajectories therefore capture the interaction between the two sides of IACE: stronger solver behavior pushes the instance generator toward harder cases, and these cases provide new challenges for subsequent solver updates. This feedback process allows the search to evolve with the current solver state rather than relying on a static set of instances.

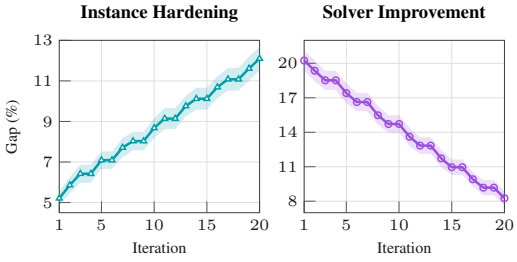

Figure 3: Co-evolution of instance hardness and solver adaptation across IACE iterations, aggregated over 5 independent runs.

Table 2: Sensitivity analysis across different LLM backbones.

LLM Backbone	Solver Gap (%) ↓	Instance Gap (%) ↑
GPT-5.6 Sol	$\mathbf{8.26 \pm 0.62}$	$\mathbf{12.10 \pm 0.58}$
Claude Opus 5	8.39 ± 0.49	11.96 ± 0.52
GPT-5.6 Terra	8.55 ± 0.54	11.78 ± 0.55
Claude Sonnet 5	8.68 ± 0.51	11.63 ± 0.47
DeepSeek-V4-Pro	8.84 ± 0.58	11.39 ± 0.61
GPT-5.4 mini	9.17 ± 0.66	10.91 ± 0.69

4.4 LLM Backbone Sensitivity

Table 2 examines the sensitivity of IACE to the choice of LLM backbone while keeping the remaining search and evaluation settings unchanged. GPT-5.6 Sol obtains a Solver Gap of 8.26% and an Instance Gap of 12.10%, with Claude Opus 5 and GPT-5.6 Terra producing similar results. Performance decreases moderately with Claude Sonnet 5 and DeepSeek-V4-Pro, while GPT-5.4 mini shows the largest change, reaching a Solver Gap of 9.17% and an Instance Gap of 10.91%. The limited variation across model families shows that IACE remains effective with different LLM backbones, while stronger models provide better solver improvement and harder instance generation.

5 Conclusion

We introduced *Interaction-Aware Co-Evolution* (IACE), an evidence-guided framework that unifies solver–instance interaction diagnosis, controlled validation, and structured evidence reuse, enabling automated algorithm design to distinguish observed failures from validated explanations and target solver weaknesses beyond average solver capability and instance difficulty. Across diverse combinatorial optimization domains, such as TSP, OP, CVRP, BPP, and MKP, IACE yields consistently challenging instance distributions and provides useful signals for heuristic improvement, achieves state-of-the-art performance on established public instances, and is further supported by ablations demonstrating the contribution of its core components.

AI USE STATEMENT

Generative AI tools were used as computational components of the proposed IACE framework. Specifically, large language models were used to generate candidate heuristic modifications, solver- and instance-side search strategies, and candidate explanations of observed solver–instance interactions. All LLM-generated outputs were treated as candidate artifacts rather than established findings. Generated programs were subject to predefined legality and validity checks, and candidate interaction explanations were evaluated using the controlled validation procedure described in the paper before any result was retained as validated evidence. All AI-assisted outputs were reviewed by the authors, who take responsibility for the final content of this work.

REFERENCES

David L Applegate, Robert E Bixby, Vašek Chvátal, and William J Cook. The traveling salesman problem: a computational study. In *The traveling salesman problem*. Princeton university press, 2011.

Edmund K Burke, Matthew R Hyde, and Graham Kendall. Evolving bin packing heuristics with genetic programming. In *International Conference on Parallel Problem Solving from Nature*, pp. 860–869. Springer, 2006.

Edmund K Burke, Matthew Hyde, Graham Kendall, Gabriela Ochoa, Ender Özcan, and John R Woodward. A classification of hyper-heuristic approaches. In *Handbook of metaheuristics*, pp. 449–468. Springer, 2010.

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.

Angelica Chen, David Dohan, and David So. Evoprompting: Language models for code-level neural architecture search. *Advances in neural information processing systems*, 36:7787–7817, 2023.

Janez Demšar. Statistical comparisons of classifiers over multiple data sets. *Journal of Machine learning research*, 7(Jan):1–30, 2006.

Marco Dorigo and Luca Maria Gambardella. Ant colony system: a cooperative learning approach to the traveling salesman problem. *IEEE Transactions on evolutionary computation*, 1(1):53–66, 1997.

Marco Dorigo, Vittorio Maniezzo, and Alberto Colorni. Ant system: optimization by a colony of cooperating agents. *IEEE transactions on systems, man, and cybernetics, part b (cybernetics)*, 26 (1):29–41, 1996.

Alex S Fukunaga. Automated discovery of local search heuristics for satisfiability testing. *Evolutionary computation*, 16(1):31–61, 2008.

Salvador Garcia and Francisco Herrera. An extension on" statistical comparisons of classifiers over multiple data sets" for all pairwise comparisons. *Journal of machine learning research*, 9(12), 2008.

Bruce L Golden, Larry Levy, and Rakesh Vohra. The orienteering problem. *Naval research logistics (nrl)*, 34(3):307–318, 1987.

Peter Henderson, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup, and David Meger. Deep reinforcement learning that matters. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.

W Daniel Hillis. Co-evolving parasites improve simulated evolution as an optimization procedure. *Physica D: Nonlinear Phenomena*, 42(1-3):228–234, 1990.

Holger H Hoos. Programming by optimization. *Communications of the ACM*, 55(2):70–80, 2012.

Frank Hutter, Holger H Hoos, Kevin Leyton-Brown, and Thomas Stützle. Paramils: an automatic algorithm configuration framework. *Journal of artificial intelligence research*, 36:267–306, 2009.

Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. Automated configuration of mixed integer programming solvers. In *International conference on integration of artificial intelligence (AI) and operations research (OR) techniques in constraint programming*, pp. 186–202. Springer, 2010.

Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In *International conference on learning and intelligent optimization*, pp. 507–523. Springer, 2011.

Frank Hutter, Lin Xu, Holger H Hoos, and Kevin Leyton-Brown. Algorithm runtime prediction: Methods & evaluation. *Artificial Intelligence*, 206:79–111, 2014.

C I-Ming, BL Golden, and EA Wasil. The team orienteering problem. *European Journal of Operational Research*, 88(3):464–474, 1996.

Serdar Kadioglu, Yuri Malitsky, Meinolf Sellmann, Kevin Tierney, et al. Isac-instance-specific algorithm configuration. In *ECAI*, volume 215, pp. 751–756, 2010.

Ashiqur R KhudaBukhsh, Lin Xu, Holger H Hoos, and Kevin Leyton-Brown. Satenstein: Automatically building local search sat solvers from components. *Artificial Intelligence*, 232:20–42, 2016.

Lars Kotthoff. Algorithm selection for combinatorial search problems: A survey. In *Data mining and constraint programming: Foundations of a cross-disciplinary approach*, pp. 149–190. Springer, 2016.

Marius Lindauer, Holger H Hoos, Frank Hutter, and Torsten Schaub. Autofolio: An automatically configured algorithm selector. *Journal of Artificial Intelligence Research*, 53:745–778, 2015.

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

Fei Liu, Rui Zhang, Zhuoliang Xie, Rui Sun, Kai Li, Qinglong Hu, Ping Guo, Xi Lin, Xialiang Tong, Mingxuan Yuan, et al. Llm4ad: A platform for algorithm design with large language model. *arXiv preprint arXiv:2412.17287*, 2024b.

Fei Liu, Yilu Liu, Qingfu Zhang, Tong Xialiang, and Mingxuan Yuan. Eoh-s: Evolution of heuristic set using llms for automated heuristic design. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pp. 37090–37098, 2026a.

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, 2026b.

Shengcai Liu, Caishun Chen, Xinghua Qu, Ke Tang, and Yew-Soon Ong. Large language models as evolutionary optimizers. In *2024 IEEE Congress on Evolutionary Computation (CEC)*, pp. 1–8. IEEE, 2024c.

Manuel Lopez-Ibanez and Thomas Stutzle. The automatic design of multiobjective ant colony optimization algorithms. *IEEE transactions on evolutionary computation*, 16(6):861–875, 2012.

Manuel López-Ibáñez, Jérémie Dubois-Lacoste, Leslie Pérez Cáceres, Mauro Birattari, and Thomas Stützle. The irace package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives*, 3:43–58, 2016.

Xiaoliang Ma, Xiaodong Li, Qingfu Zhang, Ke Tang, Zhengping Liang, Weixin Xie, and Zexuan Zhu. A survey on cooperative co-evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, 23(3):421–441, 2018.

Yecheng Jason Ma, William Liang, Guanzhi Wang, De-An Huang, Osbert Bastani, Dinesh Jayaraman, Yuke Zhu, Jim Fan, et al. Eureka: Human-level reward design via coding large language models. In *International conference on learning Representations*, volume 2024, pp. 26516–26560, 2024.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. Self-refine: Iterative refinement with self-feedback. *Advances in neural information processing systems*, 36:46534–46594, 2023.

Silvano Martello and Paolo Toth. A new algorithm for the 0-1 knapsack problem. *Management Science*, 34(5):633–644, 1988.

Franco Mascia, Manuel López-Ibáñez, Jérémie Dubois-Lacoste, and Thomas Stützle. Grammar-based generation of stochastic local search heuristics through automatic algorithm configuration tools. *Computers & operations research*, 51:190–199, 2014.

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.

Joon Sung Park, Joseph O'Brien, Carrie Jun Cai, Meredith Ringel Morris, Percy Liang, and Michael S Bernstein. Generative agents: Interactive simulacra of human behavior. In *Proceedings of the 36th annual acm symposium on user interface software and technology*, pp. 1–22, 2023.

David Pisinger. Where are the hard knapsack problems? *Computers & Operations Research*, 32(9): 2271–2284, 2005.

Mitchell A Potter and Kenneth A De Jong. Cooperative coevolution: An architecture for evolving coadapted subcomponents. *Evolutionary computation*, 8(1):1–29, 2000.

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

John R Rice. The algorithm selection problem. In *Advances in computers*, volume 15, pp. 65–118. Elsevier, 1976.

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.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *Advances in neural information processing systems*, 36:8634–8652, 2023.

Kate Smith-Miles and Mario Andrés Muñoz. Instance space analysis for algorithm testing: Methodology and software tools. *ACM Computing Surveys*, 55(12):1–31, 2023.

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.

Niki Van Stein and Thomas Bäck. Llamea: A large language model evolutionary algorithm for automatically generating metaheuristics. *IEEE Transactions on Evolutionary Computation*, 29 (2):331–345, 2024.

Pieter Vansteenwegen, Wouter Souffriau, and Dirk Van Oudheusden. The orienteering problem: A survey. *European journal of operational research*, 209(1):1–10, 2011.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *arXiv preprint arXiv:2305.16291*, 2023.

Zhuoliang Xie, Fei Liu, Zhenkun Wang, and Qingfu Zhang. Llm-driven neighborhood search for efficient heuristic design. In *2025 IEEE Congress on Evolutionary Computation (CEC)*, pp. 1–8. IEEE, 2025.

Lin Xu, Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. Satzilla: portfolio-based algorithm selection for sat. *Journal of artificial intelligence research*, 32:565–606, 2008.

Lin Xu, Holger Hoos, and Kevin Leyton-Brown. Hydra: Automatically configuring algorithms for portfolio-based selection. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 24, pp. 210–216, 2010.

Chengrun Yang, Xuezhi Wang, Yifeng Lu, Hanxiao Liu, Quoc V Le, Denny Zhou, and Xinyun Chen. Large language models as optimizers. In *International Conference on Learning Representations*, volume 2024, pp. 12028–12068, 2024.

Haoran Ye, Jiarui Wang, Zhiguang Cao, Federico Berto, Chuanbo Hua, Haeyeon Kim, Jinkyoo Park, and Guojie Song. Reevo: Large language models as hyper-heuristics with reflective evolution. *Advances in neural information processing systems*, 37:43571–43608, 2024.

Rui Zhang, Fei Liu, Xi Lin, Zhenkun Wang, Zhichao Lu, and Qingfu Zhang. Understanding the importance of evolutionary search in automated heuristic design with large language models. In *International Conference on Parallel Problem Solving from Nature*, pp. 185–202. Springer, 2024.

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

APPENDIX CONTENTS

A Problem Definitions **16**

A.1 Common Notation and Orientation of the Objective . . . 16

A.2 Traveling Salesman Problem . . . 17

A.3 Orienteering Problem . . . 17

A.4 Capacitated Vehicle Routing Problem . . . 18

A.5 Bin Packing Problem . . . 18

A.6 Multidimensional Knapsack Problem . . . 18

B Proofs for Interaction Diagnosis **19**

B.1 Projection, Additive Decomposition, and Rectangular Comparisons . . . 19

B.2 Uniform Recovery with Shared or Independent Evaluation Randomness . . . 20

B.3 Diagnostic Sample Complexity . . . 21

B.4 Why Diagnosis Alone Cannot Certify a New Repair . . . 22

C Proofs and Interpretation of Matched Validation **23**

C.1 Expected Effect, Matching, and Reference Adjustment . . . 23

D Proofs for the Finite-Sample Validation Rule **25**

D.1 A Conditional Concentration Lemma . . . 25

D.2 Joint Promotion as an Intersection–Union Test . . . 25

D.3 Information Required by a Four-Outcome Validator . . . 27

E Adaptive Proposals and Optional-Stopping Validity **28**

E.1 Proposal-Agnostic Memory Safety . . . 29

F Evidence Transfer Under Quantified Change **30**

G Operational Protocol and Reproducibility Details **31**

G.1 One Round of the Four-Pool Framework . . . 31

G.2 Fields of a Reusable Evidence Record . . . 31

G.3 Execution Failures and Validity Checks . . . 32

G.4 Evaluation Cost . . . 32

H Pseudocode **33**

H.1 Main Loop. . . . 33

H.2 Generation Pool . . . 33

H.3 Cross-Evaluation Pool and Interaction Diagnosis . . . 34

H.4 Interaction Pool . . . 34

H.5 Selection and Memory Pool . . . 34

H.6 Computational Cost . . . 35

I Prompt Templates **40**

I.1 Overview . . . 40

I.2 Problem-Specific Fields . . . 40

I.3 Prompts of the Co-Evolution Loop . . . 41

I.4 Auxiliary Prompts . . . 48

J Additional Experimental Analysis **50**
J.1 Hardness of IACE-Generated Instance Distributions . . . 50
J.2 Performance under Standard and Hard Instance Distributions . . . 50
J.3 Generalization to TSPLIB Instances . . . 51
J.4 ACO Optimization Dynamics on Generated Hard Instances . . . 54
J.5 Hard Instance Generation across Different Solvers . . . 58
K Experimental Details **58**
K.1 Computational Resources and Execution . . . 58
K.2 Common Co-Evolutionary Configuration . . . 58
K.3 LLM Generation and Program Constraints . . . 59
K.4 Evaluation Protocol and Reference Objectives . . . 60
K.5 Interaction Diagnosis and Controlled Validation . . . 60
K.6 Problem Domains and Experimental Regimes . . . 61
K.7 Baselines, Ablations, and Sensitivity Analysis . . . 61
K.8 Repeated Runs and Reporting . . . 62
L Discussion of Design Choices **62**
M Evolutionary Processes of Instance Generators and Heuristics **63**

A Problem Definitions

This appendix formalizes TSP, OP, CVRP, BPP, and MKP, the five problems of our experiments, in Appendices A.2–A.6. Interaction diagnosis is informative only if heuristics and instances can interact in more than one way, so the problems were selected for the variety of their solution structures rather than for a range of sizes of a single structure. Table 3 compares the problems by the form of a solution, the constraint that limits it, the direction of optimization, and the source of the reference value $c_{\mathrm{ref}}(x)$ of Equation (1).

Table 3: Overview of the five problems. The last column gives the reference value $c_{\mathrm{ref}}(x)$ used in the gap.

Problem	Solution	Constraint	Sense	Reference value $c_{\mathrm{ref}}(x)$
TSP	cyclic order of all nodes	none beyond the tour structure	min	known optimum on TSPLIB; otherwise LKH-3, plus an exact solver where tractable
OP	route through some of the customers	route length at most $L_{\max}$	max	exact model with a time limit (best solution found)
CVRP	routes that split the customers	load at most Q per route	min	strong CVRP heuristic with a large budget
BPP	assignment of items to bins	load at most W per bin	min	largest proven lower bound on the bin count
MKP	unordered subset of items	m capacities at once	max	exact model with a time limit (best solution found)

The problems stress a heuristic through different mechanisms. In TSP, every cyclic order is a feasible tour, so an instance can be difficult only through its distances. The length limit of OP, the vehicle capacity of CVRP, and the bin capacity of BPP each pull against the objective, and difficulty arises where the constraint and the objective disagree. In MKP, m capacities are active at the same time, and difficulty arises from how the weights are spread over them. The problems also differ in when a poor decision reveals its cost. In the routing problems, an edge that looks cheap may force an expensive connection when the route is closed many steps afterwards. In the packing problems, an early placement consumes capacity that is then missing for a better placement. An instance property that exposes a heuristic on one problem therefore says little about the others, and the interaction residuals are expected to point to different kinds of conditions: spatial patterns for the routing problems, and patterns of load and remaining capacity for the packing problems. Finally, combining minimization and maximization problems tests whether the orientation convention of Appendix A.1 suffices for the diagnosis and validation steps of Section 3 to be applied without change.

A.1 Common Notation and Orientation of the Objective

The problem size n counts nodes, customers, or items. For the routing problems, the entry $d_{ij} \geq 0$ gives the travel cost of the move from i to j, and $d_{ii} = 0$. In OP and CVRP, node 0 is the depot and the customers form $N = \{1, \ldots, n\}$. A *route* is a sequence $r = (r_0, r_1, \ldots, r_k, r_{k+1})$ with $r_0 = r_{k+1} = 0$ and pairwise distinct customers $r_1, \ldots, r_k \in N$. We write $N(r) = \{r_1, \ldots, r_k\}$ for the customers visited by r and

$$\ell(r) = \sum_{t=0}^{k} d_{r_t r_{t+1}} \tag{18}$$

for its length; the empty route $(0, 0)$ visits no customer and has length zero. For each problem, the function f_x in Equation (1) is the objective stated in Appendices A.2–A.6.

Equation (1) defines the gap of a minimization problem, which applies to TSP, CVRP, and BPP. In OP and MKP, larger objective values are better, and the gap measures how far the result falls short of the reference:

$$\mathrm{gap}(h, x; \omega) = \frac{c_{\mathrm{ref}}(x) - C(h, x; \omega)}{c_{\mathrm{ref}}(x)}. \tag{19}$$

With Equations (1) and (19), a larger gap indicates a worse result on all five problems. The averages Y_{hx}, the residuals r_{hx} of Equation (2), and the weakness threshold are therefore used unchanged. In

the validation contrasts of Equation (8), a maximization problem uses the shortfall $c_{\mathrm{ref}} - \bar{C}$ in place of the excess $\bar{C} - c_{\mathrm{ref}}$, and the gain $\bar{C}^{(u)}_{hv} - \bar{C}^{(u)}_{p(h)v}$ in place of $\bar{C}^{(u)}_{p(h)v} - \bar{C}^{(u)}_{hv}$. The criteria of Equation (12) then keep their meaning: under condition v, the parent falls further behind the reference, the candidate improves on its parent, and it improves more than under the original condition o.

Each instance receives its reference value $c_{\mathrm{ref}}(x)$ a single time, outside the co-evolution loop, and the value stays fixed during a run. Neither heuristics nor instance generators can influence it, so every change in a gap is caused by a change of the heuristic or of the instance. When $c_{\mathrm{ref}}(x)$ is the best solution found by a solver rather than a proven optimum or bound, a heuristic may surpass it; we report the resulting negative gaps as they are instead of setting them to zero. On TSPLIB, $c_{\mathrm{ref}}(x)$ equals the known optimum, and on BPP it bounds the optimum from below, so the gaps on these instances are never negative. Gaps are reported in percent.

A.2 Traveling Salesman Problem

A TSP instance is given by n nodes $N = \{1, \dots, n\}$ together with a nonnegative matrix $d \in \mathbb{R}^{n \times n}_{\geq 0}$ of travel costs. For Euclidean instances, every node is a point in the plane and d_{ij} equals the distance between the points of i and j. We describe a tour by its successor function $\sigma : N \to N$, where $\sigma(i)$ is the node visited directly after i. The function σ must be a bijection, and following it from any node must pass through all n nodes before returning to the start. Writing $\mathcal{T}_n$ for the set of such functions, the problem is

$$\min_{\sigma \in \mathcal{T}_n} \; L_{\mathrm{TSP}}(\sigma) = \sum_{i \in N} d_{i\,\sigma(i)}. \tag{20}$$

This form is equivalent to writing the tour as a permutation of N, and it turns the edge that closes the tour into an ordinary term of the sum. Every element of $\mathcal{T}_n$ is feasible, so TSP has no constraint beyond the tour structure itself.

The evolved heuristic receives only d, not the coordinates. An instance generator may therefore return a cost matrix directly. Such a matrix need not be derived from points and need not be metric, which gives the instance side access to structures that moving points around cannot produce.

Hard instances. Because no tour is infeasible, all difficulty must be placed in the distances. Edge scores that depend mainly on the length of an edge, such as $1/d_{ij}$, are easy to mislead. Tightly packed groups of nodes make the edges inside a group look much better than the few edges between groups, although every tour has to use some of the latter, and groups nested inside larger groups repeat this effect on each scale. A long edge that every good tour needs looks unattractive from both of its endpoints. Controlled violations of the triangle inequality change which neighbor appears nearest, while the instance remains well defined.

A.3 Orienteering Problem

An OP instance specifies a depot 0, customers $N = \{1, \dots, n\}$ with prizes $p_i \geq 0$, costs d_{ij} for all pairs in $\{0\} \cup N$, and a length limit $L_{\max} > 0$; the depot has no prize. We look for one route r in the sense of Appendix A.1, and only the customers in $N(r)$ earn their prize:

$$\max_{r} \; P_{\mathrm{OP}}(r) = \sum_{i \in N(r)} p_i \qquad \text{subject to} \qquad \ell(r) \leq L_{\max}. \tag{21}$$

The empty route is always feasible and earns nothing, and each customer is visited at most once. OP thus combines two decisions: which customers to include, and in which order to visit them. Unlike in TSP, the order matters for feasibility, since a set of customers may fit into the length limit in one visiting order but exceed it in another.

Hard instances. OP instances are hard when high prizes and short distances do not coincide. Examples are a remote cluster of valuable customers that the length limit does not allow to be visited completely, sequences of short edges through customers without prize, prizes that grow with the depot distance up to some radius and shrink beyond it, and length limits at which a rule favoring the highest prize per unit of distance moves into a region from which the remaining length no longer permits a good return.

A.4 Capacitated Vehicle Routing Problem

A CVRP instance has a depot 0, customers $N = \{1, \dots, n\}$ with demands $\delta_i > 0$, identical vehicles of capacity Q, and costs d_{ij} on $\{0\} \cup N$ that are either computed from coordinates or given directly. Unless a benchmark prescribes the fleet size, the number of routes K is part of the solution. We look for routes $r^1, \dots, r^K$ that jointly serve each customer once and never load a vehicle beyond Q:

$$\begin{aligned} \min_{K,\, r^1,\dots,r^K} \quad & L_{\mathrm{CVRP}} = \sum_{k=1}^{K} \ell(r^k) \\ \text{subject to} \quad & \sum_{i \in N(r^k)} \delta_i \le Q, \quad k = 1, \dots, K, \\ & N(r^k) \cap N(r^l) = \emptyset \ \ (k \neq l), \qquad \textstyle\bigcup_{k=1}^{K} N(r^k) = N. \end{aligned} \tag{22}$$

The capacity constraint and the requirement that the routes partition N must hold together. The order in which the customers are cheapest to visit may produce routes whose loads exceed Q, so an ordering that is good in terms of distance can still be infeasible.

Hard instances. Hard CVRP instances target the coupling between routing and loading. Customers that are close to each other may have demands that together exceed the capacity, so a grouping suggested by the geometry cannot be served by one vehicle. A remote customer with a large demand may need a vehicle of its own. A depot at the edge of the service area, instead of in its center, makes every trip to the far side expensive. Demands can also be chosen so that the order that is cheapest to drive is not an order in which the vehicles can be filled.

A.5 Bin Packing Problem

A BPP instance lists n items $I = \{1, \dots, n\}$ with sizes $s_i \in (0, W]$; any number of bins of capacity W may be opened. A packing places each item in one bin, $\phi : I \to \{1, \dots, n\}$; since every item fits into a bin on its own, n bins always suffice. With the load $\lambda_k(\phi) = \sum_{i:\, \phi(i)=k} s_i$ of bin k, the number of bins in use is the number of distinct values of ϕ:

$$\min_{\phi} \ N_{\mathrm{BPP}}(\phi) = \big|\{\phi(i) : i \in I\}\big| \qquad \text{subject to} \qquad \lambda_k(\phi) \le W, \quad k = 1, \dots, n. \tag{23}$$

The objective is an integer, so for small n the gap can take only a few distinct values, and BPP results are averaged over sets of instances.

Hard instances. BPP has no geometry; its difficulty lies in the space left over in partially filled bins. Two large items that almost fill a bin together appear to be a good match, but both may be needed to complete other bins. Some sets of items can be packed optimally only in groups of three, which rules that first look for good pairs break apart. Some size distributions leave a gap in every bin that is too small for any remaining item.

A.6 Multidimensional Knapsack Problem

An MKP instance has n items $I = \{1, \dots, n\}$ and m resources. Item i has a profit $p_i > 0$ and a weight vector $w_i = (w_{i1}, \dots, w_{im}) \in \mathbb{R}^m_{\ge 0}$, and resource j has a capacity $b_j > 0$. We divide the weights of resource j by b_j, so that every capacity equals one. This rescaling does not change the set of feasible solutions, and it makes weights comparable across resources and instances. The decision is which set $S \subseteq I$ of items to pack:

$$\max_{S \subseteq I} \ P_{\mathrm{MKP}}(S) = \sum_{i \in S} p_i \qquad \text{subject to} \qquad \sum_{i \in S} w_{ij} \le 1, \quad j = 1, \dots, m. \tag{24}$$

MKP is the only one of the five problems whose solution is both unordered and allowed to leave items out.

Hard instances. Many MKP heuristics rank items by a single efficiency value, for example the profit divided by the total or average weight. Such a value hides how the weight of an item is distributed over the m resources, and hard instances take advantage of this. An item that uses a little of every resource can receive a high efficiency value while adding little profit, a valuable item that uses one or two resources heavily can receive a low one, and correlations between profits and particular resources can lead a greedy selection to exhaust one resource while the others still have room.

B Proofs for Interaction Diagnosis

This appendix proves the algebraic and statistical statements underlying the interaction diagnosis in Section 3.1. Throughout this section, the diagnostic panel is treated as fixed before the observations used to evaluate it. Equivalently, within an adaptive co-evolutionary round, all expectations and probability statements may be interpreted conditionally on the information available before evaluation of that panel. The conditional formulation requires the panel to have been fixed before the observations to which the corresponding bound is applied.

B.1 Projection, Additive Decomposition, and Rectangular Comparisons

Proof of the additive-invariance and rectangular-contrast result. Let the Frobenius inner product be

$$\langle U, V \rangle_F = \sum_{h,x} U_{hx} V_{hx}.$$

Define

$$\mathcal{S} = \left\{ U \in \mathbb{R}^{H \times N} : U\mathbf{1}_N = 0,\ \mathbf{1}_H^\top U = 0 \right\},$$

and

$$\mathcal{A} = \left\{ a\mathbf{1}_N^\top + \mathbf{1}_H b^\top : a \in \mathbb{R}^H,\ b \in \mathbb{R}^N \right\}.$$

Recall

$$P_H = I_H - \frac{1}{H}\mathbf{1}\mathbf{1}^\top, \qquad P_N = I_N - \frac{1}{N}\mathbf{1}\mathbf{1}^\top.$$

Both centering matrices are symmetric and idempotent. Consequently, the map

$$T(U) = P_H U P_N$$

is self-adjoint and idempotent under $\langle \cdot, \cdot \rangle_F$. Its range is exactly $\mathcal{S}$: every output has zero row and column sums, and $T(U) = U$ whenever $U \in \mathcal{S}$. Thus T is the orthogonal projection onto the interaction subspace.

Every additive matrix is orthogonal to every matrix in $\mathcal{S}$. Conversely, expanding the projection gives

$$M - P_H M P_N = \frac{1}{H}\mathbf{1}\mathbf{1}^\top M + \frac{1}{N} M \mathbf{1}\mathbf{1}^\top - \frac{1}{HN}\mathbf{1}\mathbf{1}^\top M \mathbf{1}\mathbf{1}^\top,$$

which belongs to $\mathcal{A}$. Hence

$$\mathcal{A} = \mathcal{S}^\perp,$$

and $M - \Gamma$ is the unique closest additive matrix to M.

Although the representation $a\mathbf{1}^\top + \mathbf{1}b^\top$ is not unique, an identifiable representation is

$$M_{hx} = \mu + a_h + b_x + \Gamma_{hx}, \qquad \mu = \bar{M}_{\cdot\cdot}, \quad a_h = \bar{M}_{h\cdot} - \mu, \quad b_x = \bar{M}_{\cdot x} - \mu,$$

with

$$\sum_h a_h = 0, \qquad \sum_x b_x = 0.$$

Applying P_H and P_N annihilates arbitrary additive perturbations:

$$P_H \left(M + a\mathbf{1}^\top + \mathbf{1}b^\top \right) P_N = P_H M P_N = \Gamma.$$

Thus a solver that is uniformly worse on all instances, or an instance that is uniformly harder for all solvers, does not by itself induce a solver–instance interaction.

For the rectangular identity, expand and average:

$$\frac{1}{HN}\sum_{i=1}^{H}\sum_{j=1}^{N}(M_{hx}-M_{hj}-M_{ix}+M_{ij}) = M_{hx}-\bar{M}_{h\cdot}-\bar{M}_{\cdot x}+\bar{M}_{\cdot\cdot}$$
$$= \Gamma_{hx}.$$

This proves the rectangular representation in Eq. (5).

Finally, consider a 2×2 table with rows (p,h) and columns (o,v), and let a linear functional on the table have coefficient matrix W. Invariance to adding a constant to either row requires the row sums of W to vanish. Invariance to adding a constant to either column requires the column sums to vanish. These constraints imply

$$W = c\begin{pmatrix}-1 & 1\\ 1 & -1\end{pmatrix}$$

for a scalar c. The corresponding functional is

$$c\left(L_{pv}-L_{hv}-L_{po}+L_{ho}\right).$$

Conversely, this contrast annihilates every additive row or column shift. Therefore the parent–child difference-in-differences is unique up to scale and sign among linear contrasts invariant to additive solver and condition effects. □

B.2 Uniform Recovery with Shared or Independent Evaluation Randomness

A centered random variable X is σ^2-sub-Gaussian when

$$\mathbb{E}\left[e^{\lambda X}\right] \le e^{\lambda^2\sigma^2/2}$$

for every real λ. The assumptions in the main text require independence across replicate matrices, but not necessarily among the cells within one replicate.

Proof of the uniform diagnostic-recovery bounds. Write

$$E = Y - M.$$

Because Y averages m independent replicate matrices, the average noise in every cell is (σ^2/m)-sub-Gaussian even when cells inside a replicate are dependent. A scalar Chernoff bound in each cell followed by a union bound gives

$$\Pr\left(\|E\|_{\max} > \sigma\sqrt{\frac{2}{m}\log\frac{2HN}{\eta}}\right) \le \eta.$$

For every row of P_H, the sum of the absolute coefficients is

$$1-\frac{1}{H}+\frac{H-1}{H} = 2\left(1-\frac{1}{H}\right).$$

The corresponding quantity for P_N is

$$2\left(1-\frac{1}{N}\right).$$

Therefore, with

$$\rho = \left(1-\frac{1}{H}\right)\left(1-\frac{1}{N}\right),$$

we have

$$\|P_H E P_N\|_{\max} \le 4\left(1-\frac{1}{H}\right)\left(1-\frac{1}{N}\right)\|E\|_{\max} = 4\rho\|E\|_{\max}.$$

Combining the two displays proves the dependence-robust line of Eq. (6). No independence across cells was used.

For the sharper result, suppose that the cells are also independent within every replicate. For a fixed pair (h, x),

$$(P_H E P_N)_{hx} = \sum_{i,j} (P_H)_{hi} (P_N)_{jx} E_{ij}.$$

The squared coefficient norm is

$$\begin{aligned} \sum_{i,j} (P_H)_{hi}^2 (P_N)_{jx}^2 &= (P_H^2)_{hh} (P_N^2)_{xx} \\ &= \left(1 - \frac{1}{H}\right)\left(1 - \frac{1}{N}\right) \\ &= \rho. \end{aligned}$$

The displayed weighted sum is therefore $(\sigma^2 \rho / m)$-sub-Gaussian. Its two-sided tail satisfies

$$\Pr\left(|(P_H E P_N)_{hx}| > t\right) \le 2 \exp\left(-\frac{m t^2}{2\sigma^2 \rho}\right).$$

A union bound over all HN cells proves the independent-cell line of Eq. (6).

For either applicable radius b, work on the event

$$\|R - \Gamma\|_{\max} \le b.$$

If

$$r_{hx} > \lambda + b,$$

then

$$\Gamma_{hx} \ge r_{hx} - b > \lambda.$$

Conversely, if

$$\Gamma_{hx} > \lambda + 2b,$$

then

$$r_{hx} \ge \Gamma_{hx} - b > \lambda + b.$$

Because the max-norm event is simultaneous over the entire panel, these implications continue to hold for cells selected after inspection of the residual matrix. □

What the theorem does not cover. A common seed across the cells of one replicate can induce dependence even when different replicate matrices are independent. The dependence-robust line of Eq. (6) covers this setting; the sharper independent-cell line should not be used solely because a common seed protocol is repeated across replicates.

Neither line automatically covers selecting the diagnostic panel using the same noisy observations later used in the bound. Conditioning on such a selection may change the relevant noise distribution. A panel selected using previous information and then evaluated on fresh seeds is covered by the conditional argument above. An adaptively reused panel without fresh evaluation should instead be treated as exploratory. The fresh-validation guarantee below does not require the diagnostic screening stage itself to be confirmatory.

B.3 Diagnostic Sample Complexity

Suppose a pair is called δ-separated at screening threshold λ when either

$$\Gamma_{hx} \le \lambda \qquad \text{or} \qquad \Gamma_{hx} \ge \lambda + \delta.$$

Uniform recovery within $\delta/2$ then recovers the corresponding above-threshold classification simultaneously for every separated pair.

Proof of the diagnostic sample-complexity bound. For the dependence-robust radius, requiring

$$4\rho\sigma \sqrt{\frac{2}{m} \log \frac{2HN}{\eta}} \le \frac{\delta}{2}$$

is equivalent to

$$m \geq \frac{128\rho^2\sigma^2}{\delta^2} \log \frac{2HN}{\eta}.$$

For independent cells, requiring

$$\sigma\sqrt{\frac{2\rho}{m} \log \frac{2HN}{\eta}} \leq \frac{\delta}{2}$$

is equivalent to

$$m \geq \frac{8\rho\sigma^2}{\delta^2} \log \frac{2HN}{\eta}.$$

Combining the two cases gives the compact statement

$$m \geq \begin{cases} \dfrac{128\rho^2\sigma^2}{\delta^2} \log \dfrac{2HN}{\eta}, & \text{dependence-robust bound,} \\ \dfrac{8\rho\sigma^2}{\delta^2} \log \dfrac{2HN}{\eta}, & \text{independent cells.} \end{cases} \tag{25}$$

On the resulting uniform event, every cell satisfying $\Gamma_{hx} \leq \lambda$ obeys

$$r_{hx} \leq \lambda + \frac{\delta}{2},$$

whereas every cell satisfying $\Gamma_{hx} \geq \lambda + \delta$ obeys

$$r_{hx} \geq \lambda + \frac{\delta}{2}.$$

Thus the common empirical threshold $\lambda + \delta/2$ correctly classifies all δ-separated pairs. Boundary ties may be handled conservatively without affecting the probability statement. □

The diagnostic replicate requirement therefore scales as

$$m = O\left(\frac{\sigma^2}{\delta^2} \log \frac{HN}{\eta}\right).$$

The two lines of Eq. (25) should not be interpreted as a universal penalty for using common random numbers. Shared randomness may reduce the variance of a particular contrast when it induces favorable covariance. The first line is instead a worst-case guarantee that remains valid under arbitrary within-replicate dependence.

B.4 Why Diagnosis Alone Cannot Certify a New Repair

Raw loss does not identify interaction. For any $B > 0$, the constant matrix

$$M = B\mathbf{1}\mathbf{1}^\top$$

can have arbitrarily large entries while

$$P_H M P_N = 0.$$

Thus large raw loss does not imply a large interaction.

Conversely, consider

$$\Gamma = \begin{pmatrix} B & -B \\ -B & B \end{pmatrix}, \qquad A = \begin{pmatrix} B & B+K \\ B+K & B+2K \end{pmatrix}, \qquad K > 2B.$$

The matrix A is additive and has zero interaction. In

$$M = \Gamma + A,$$

every entry is nonnegative, the $(1,1)$ entry is the smallest, yet

$$\Gamma_{11} = B > 0.$$

Hence an arbitrarily strong interaction may coexist with a low raw-loss rank.

Proof of diagnostic non-identification of an unevaluated repair. Fix an arbitrary diagnostic history that does not evaluate the new child on its original and target validation conditions. Keep every previously observed program outcome identical in two possible extensions of this history.

In both extensions, let the parent have normalized losses

$$(L_{po}, L_{pv}) = (2, 5).$$

In the first model, let the child have

$$(L_{ho}, L_{hv}) = (1, 4).$$

Then

$$(A, G_v, D) = (3, 1, 0),$$

so at zero thresholds the pair is unsupported.

In the second model, let

$$(L_{ho}, L_{hv}) = (2, 3),$$

giving

$$(A, G_v, D) = (3, 2, 2),$$

which is supported at zero thresholds.

These tables may be realized with reference value 1, common scale 1, and nonnegative costs equal to the displayed normalized losses plus 1. The two models agree on all quantities available at proposal time. Therefore any rule depending only on the diagnostic history, including any independent randomization, has the same distribution in both models.

Equivalently, the same diagnostic distribution admits one extension with

$$\mathbb{E}[D_{hv}] > 0$$

and another with

$$\mathbb{E}[D_{hv}] \leq 0.$$

New outcome information, or an additional structural assumption linking the child and target condition to the observed panel, is therefore necessary. □

C Proofs and Interpretation of Matched Validation

C.1 Expected Effect, Matching, and Reference Adjustment

Recall that the common normalization gives

$$D = G_v - G_o$$

pointwise for every matched group.

Proof of the matched-effect result. Linearity of expectation gives

$$\mathbb{E}[D] = \mathbb{E}[G_v] - \mathbb{E}[G_o].$$

For n independent and identically distributed matched groups,

$$\mathrm{Var}(\bar{D}) = \frac{\mathrm{Var}(D)}{n}.$$

Expanding the variance of the difference gives

$$\mathrm{Var}(D) = \mathrm{Var}(G_v) + \mathrm{Var}(G_o) - 2\,\mathrm{Cov}(G_v, G_o),$$

and therefore

$$\mathrm{Var}(\bar{D}) = \frac{\mathrm{Var}(G_v) + \mathrm{Var}(G_o) - 2\,\mathrm{Cov}(G_v, G_o)}{n},$$

which is the matched-variance identity used in the main text.

An unmatched estimator based on independent draws from the same two gain marginals has variance

$$\mathrm{Var}_{\mathrm{unmatched}} = \frac{\mathrm{Var}(G_v) + \mathrm{Var}(G_o)}{n}.$$

The difference between unmatched and matched variance is therefore

$$\mathrm{Var}_{\mathrm{unmatched}} - \mathrm{Var}_{\mathrm{matched}} = \frac{2\,\mathrm{Cov}(G_v, G_o)}{n}.$$

Positive covariance improves precision, zero covariance gives equality, and negative covariance makes the matched estimator less efficient.

For a fixed group, adding a condition-specific offset to both L_{pv} and L_{hv} leaves G_v unchanged. Likewise, an additive solver effect shared across the two conditions changes the two gains by the same amount and hence cancels from D. This is the 2×2 specialization of the additive invariance proved in Appendix B.1. □

Reference sensitivity. Keep all objective values and the common scale q fixed and replace

$$c'_v = c_v + e_v, \qquad c'_o = c_o + e_o.$$

The normalized losses shift by

$$L'_v = L_v - \frac{e_v}{q}, \qquad L'_o = L_o - \frac{e_o}{q}.$$

Consequently,

$$A' = A - \frac{e_v - e_o}{q}, \qquad G'_v = G_v, \qquad D' = D.$$

Thus additive reference error affects A but not the parent–child gain contrasts when the common scale is unchanged. If q itself changes, these equalities do not imply invariance to the resulting rescaling.

Matching is not a causal assumption. The expectation identity concerns the declared paired-group distribution. It requires no parallel-trends assumption because all four outcomes in the matched table are evaluated directly rather than imputing an unobserved counterfactual. However, the identity also does not show that a named instance feature is the unique causal explanation for the effect. A condition generator may modify several correlated properties simultaneously. Descriptor checks can document intended changes but cannot by themselves establish that every relevant nuisance property was held fixed. The population target is therefore the difference in solver improvement between the specified generator conditions.

Common versus condition-specific scales. Suppose the parent–child objective gain equals 1 in both conditions. Under a common scale

$$q = 1,$$

the two normalized gains are equal and

$$D = 0.$$

If the conditions are instead normalized by two different references, say 1 and 100, the normalized gains become

$$1 \qquad \text{and} \qquad 0.01,$$

which produces a nonzero difference. That contrast may be meaningful for another scientific question, but it is not the common-scale quantity used by IACE. The normalization rule is therefore part of the definition of the validated target.

Groups, not executions, are the sampling units. Parent, child, condition, and seed outcomes within one matched group may be dependent. The concentration assumption is imposed on the complete group-level statistic. Averaging additional solver seeds can reduce group-level variability, but it does not automatically multiply the number of independent latent groups. Likewise, any descriptive resampling procedure should resample complete matched groups and preserve their full covariance structure.

D PROOFS FOR THE FINITE-SAMPLE VALIDATION RULE

D.1 A CONDITIONAL CONCENTRATION LEMMA

Lemma. Conditional on a history $\mathcal{F}$, suppose $X_1, \dots, X_n$ are independent with common mean μ and satisfy

$$\mathbb{E}\left[e^{\lambda(X_u-\mu)} \mid \mathcal{F}\right] \le e^{\lambda^2\nu^2/2}$$

for every real λ. Then, for any $b > 0$,

$$\Pr\left(\bar{X} - \mu \ge b \mid \mathcal{F}\right) \le e^{-nb^2/(2\nu^2)},$$

and

$$\Pr\left(\bar{X} - \mu \le -b \mid \mathcal{F}\right) \le e^{-nb^2/(2\nu^2)}.$$

Proof. Conditional independence gives

$$\mathbb{E}\left[\exp\left\{\lambda \sum_{u=1}^{n}(X_u - \mu)\right\}\middle| \mathcal{F}\right] \le e^{n\lambda^2\nu^2/2}.$$

Markov's inequality therefore implies

$$\Pr\left(\bar{X} - \mu \ge b \mid \mathcal{F}\right) \le \exp\left(-\lambda n b + \frac{n\lambda^2\nu^2}{2}\right).$$

Choosing

$$\lambda = \frac{b}{\nu^2}$$

gives the upper-tail bound. Applying the same argument to $-X_u$ gives the lower-tail bound. □

D.2 JOINT PROMOTION AS AN INTERSECTION–UNION TEST

Proof of false-promotion control. Fix the complete pre-validation history. For every population-unsupported pair j, choose one coordinate $k(j)$ for which

$$\mu_{j,k(j)} \le \tau_{k(j)}.$$

The choice can be made using a deterministic tie-breaking rule and depends only on the conditional population means, not on the validation outcomes.

If pair j is nevertheless promoted by Eq. (14), then

$$\bar{Z}_{j,k(j)} - \mu_{j,k(j)} > \nu_{j,k(j)}\sqrt{\frac{2}{n_j}\log\frac{1}{a_j}}.$$

The conditional concentration lemma bounds this event by a_j. A union bound over unsupported pairs gives

$$\Pr\left(\exists j : j \text{ is unsupported and promoted} \mid \mathcal{F}\right) \le \sum_j a_j \le \alpha.$$

Taking expectation over the pre-validation history gives the unconditional bound in Eq. (15).

No independence between candidate–condition pairs, or between A, G, and D, was used. Additional legality or descriptor checks that can only refuse promotion cannot increase the false-promotion probability. □

The reason a factor of three is not required in the joint promotion decision is that the null is a union of three failures, whereas admission requires all three population inequalities to hold. Any false conjunction necessarily contains at least one fixed false constituent. This is the intersection–union principle.

Proof of the validation-power bound. Fix a supported pair and suppress its index j. Write

$$b_k = \nu_k \sqrt{\frac{2}{n} \log \frac{1}{a}}, \qquad s_k = \nu_k \sqrt{\frac{2}{n} \log \frac{3}{\zeta}}.$$

The sample-size requirement in Eq. (16) implies

$$b_k + s_k < \Delta_k \qquad \text{for every } k \in \{A, G, D\}.$$

By the conditional concentration lemma and a union bound over the three coordinates,

$$\bar{Z}_k \geq \mu_k - s_k \qquad \text{for all } k$$

with probability at least $1 - \zeta$. On this event,

$$L_k = \bar{Z}_k - b_k \geq \mu_k - s_k - b_k > \mu_k - \Delta_k = \tau_k.$$

Hence all three statistical gates pass simultaneously. □

As with any validation rule, the power statement concerns the statistical gate. The separate deterministic execution and instance-validity checks must also pass before a record is promoted.

Simultaneous numerical coverage is a different guarantee. The rule in the main text certifies the conjunction of three population inequalities. It does not assert that, with probability $1 - a_j$, all three reported lower confidence bounds lie below their corresponding means simultaneously. To obtain such a simultaneous numerical confidence statement, replace

$$\log \frac{1}{a_j}$$

by

$$\log \frac{3}{a_j}.$$

Each coordinate then fails with probability at most $a_j/3$, and a union bound over all pairs and coordinates is at most $\sum_j a_j$. The compound promotion decision can avoid this extra factor because any false conjunction has at least one fixed false constituent.

Known bounds and clipping. The scales ν_{jk} must be conservative. If every normalized loss lies in a known interval of width B, then

$$A, G \in [-B, B], \qquad D \in [-2B, 2B].$$

Hoeffding's lemma therefore allows

$$(\nu_A, \nu_G, \nu_D) = (B, B, 2B).$$

More generally, if a statistic Z_k lies in an interval of known width W_k, then

$$\nu_k = \frac{W_k}{2}.$$

If costs and references lie in $[0, K]$ and the common positive scale satisfies

$$q \geq q_{\min} > 0,$$

then every normalized loss lies in

$$\left[-\frac{K}{q_{\min}}, \frac{K}{q_{\min}} \right],$$

which yields finite conservative scales for A, G, and D.

An observed empirical standard deviation is not automatically a valid upper bound on a sub-Gaussian scale. Clipping may be used to obtain bounded statistics, but then the theorem concerns the clipped population target and the corresponding estimand must be stated explicitly.

D.3 Information Required by a Four-Outcome Validator

Proof of the validation sample lower bound. Use zero thresholds and let one fresh matched group's normalized losses be

$$\begin{array}{c|cc} & o & v \\ \hline p & 3\nu & 4\nu \\ h & 2\nu + X & 3\nu \end{array}, \qquad X \in \{-\nu, \nu\}.$$

All four losses are bounded and nonnegative. Direct substitution gives

$$A = \nu, \qquad G_v = \nu, \qquad G_o = \nu - X, \qquad D = X.$$

Under P_0, let

$$P_0(X = \nu) = P_0(X = -\nu) = \frac{1}{2}.$$

Under P_1, let

$$P_1(X = \nu) = \frac{1 + \Delta/\nu}{2}, \qquad P_1(X = -\nu) = \frac{1 - \Delta/\nu}{2}.$$

Then

$$\mu_D = 0 \quad \text{under } P_0, \qquad \mu_D = \Delta \quad \text{under } P_1,$$

while A and G_v are strictly positive in both distributions.

The statistic D has range width 2ν, so its centered version is ν^2-sub-Gaussian. The complete loss table contains exactly the information in X. All diagnostic records may be kept identical across the two models because the new child–condition outcomes need not occur in the diagnostic history.

Let

$$d = \frac{\Delta}{\nu}.$$

The relative entropy of one matched table under P_1 versus P_0 is

$$K_1 = \mathrm{kl}\left(\frac{1+d}{2}, \frac{1}{2}\right) \le d^2.$$

For completeness,

$$\log y \le y - 1$$

implies

$$\mathrm{KL}(P \| Q) \le \sum_x \frac{(P(x) - Q(x))^2}{Q(x)}.$$

Applying this inequality to the two Bernoulli probabilities gives the preceding d^2 upper bound.

For n independent groups,

$$\mathrm{KL}\left(P_1^n \| P_0^n\right) = nK_1.$$

Let $\mathcal{E}$ denote the event that the validator promotes the pair. Including an independent internal random seed of the validation algorithm does not change the input relative entropy. The log-sum inequality, applied to $\mathcal{E}$ and its complement, gives

$$nK_1 \ge \mathrm{kl}\left(P_1(\mathcal{E}), P_0(\mathcal{E})\right).$$

If

$$P_1(\mathcal{E}) \ge 1 - \zeta > \alpha \ge P_0(\mathcal{E}),$$

then binary relative entropy is increasing in its first argument and decreasing in its second over this ordered region. Hence

$$n\frac{\Delta^2}{\nu^2} \ge nK_1 \ge \mathrm{kl}(1 - \zeta, \alpha),$$

which proves Eq. (17). □

Scope of the lower bound. This result concerns fixed-budget validation in the four-outcome loss-table model. It shows that the ν^2/Δ^2 dependence required to distinguish a weak condition-specific effect cannot generally be removed. It does not establish an optimal cost for constructing the condition generator, discovering the candidate, or sequentially validating many hypotheses. In particular, it does not show that the multiple-comparison allocation penalty or the logarithmic term in the elementary anytime radius below is information-theoretically necessary.

E Adaptive Proposals and Optional-Stopping Validity

Repeated validation introduces two distinct forms of adaptivity. Previous outcomes may shape which hypotheses are proposed next, and the current validation outcomes may determine when evaluation stops. Conditioning on the complete pre-validation history handles the former, whereas the latter requires a time-uniform bound.

Let $\mathcal{F}_t^0$ denote the complete history immediately before validation in round t. This history includes all earlier rounds as well as current-round generation, screening, population selection, retrieval, and proposal randomness. Conditional on $\mathcal{F}_t^0$, the finite hypothesis family, its size J_t, the program and generator versions, thresholds, conservative scales, and allocated levels are fixed.

Each round receives a fresh stream of complete matched-group outcomes satisfying the stated conditional concentration assumption. Streams from different rounds may have different laws; the requirement is conditional freshness rather than a single stationary distribution throughout co-evolution.

Allocate levels according to a pathwise error ledger satisfying

$$\sum_{t,j} a_{tj} \le \alpha.$$

For example,

$$a_{tj} = \frac{6\alpha}{\pi^2 t^2 J_t}, \qquad J_t \ge 1,$$

is a valid summable allocation.

When the validation sample size may depend on the observed outcomes, define

$$L_{tjk}^{\mathrm{any}}(n) = \bar{Z}_{tjk}(n) - \nu_{tjk}\sqrt{\frac{2}{n}\log\frac{\pi^2 n^2}{6a_{tj}}}. \tag{26}$$

Proof of adaptive and optional-stopping validity. Fix t, j, and k, and condition on $\mathcal{F}_t^0$. For every integer $n \ge 1$, the conditional concentration lemma and Eq. (26) give

$$\Pr\left(L_{tjk}^{\mathrm{any}}(n) > \mu_{tjk} \mid \mathcal{F}_t^0\right) \le \frac{6a_{tj}}{\pi^2 n^2}.$$

Since

$$\sum_{n=1}^{\infty} \frac{6}{\pi^2 n^2} = 1,$$

summing over all possible sample sizes yields

$$\Pr\left(\exists n \ge 1 : L_{tjk}^{\mathrm{any}}(n) > \mu_{tjk} \mid \mathcal{F}_t^0\right) \le a_{tj}. \tag{27}$$

This bound already holds over the entire hypothetical validation stream. Inspecting only a subsequence of sample sizes, or stopping at a data-dependent time, can only restrict the crossing event.

If pair j is unsupported at round t, choose one false coordinate $k(t,j)$ deterministically from its conditional means:

$$\mu_{tj,k(t,j)} \le \tau_{k(t,j)}.$$

Promotion at any sample size requires

$$L_{tj,k(t,j)}^{\mathrm{any}}(n) > \tau_{k(t,j)} \ge \mu_{tj,k(t,j)}.$$

Equation (27) therefore bounds the conditional probability of ever falsely promoting pair j by a_{tj}.

Let $\mathcal{E}_t$ be the event that some unsupported pair is promoted during round t. A finite-family union bound gives

$$\Pr\left(\mathcal{E}_t \mid \mathcal{F}_t^0\right) \le \sum_{j=1}^{J_t} a_{tj}.$$

The tower property, a union bound over rounds, and monotone convergence then yield

$$\begin{aligned}\Pr\left(\bigcup_{t\ge 1}\mathcal{E}_t\right) &\le \sum_{t\ge 1}\Pr(\mathcal{E}_t)\\ &\le \mathbb{E}\left[\sum_{t\ge 1}\sum_{j=1}^{J_t} a_{tj}\right]\\ &\le \alpha.\end{aligned}$$

Thus

$$\Pr\left(\exists\, t, j : \text{a population-unsupported pair enters validated memory}\right) \le \alpha. \tag{28}$$

The final inequality uses only the pathwise budget constraint. Allocated levels may themselves depend on earlier observations, provided that earlier allocations are not retroactively changed and the ledger never exceeds α. Additional deterministic validity checks can only reduce the promotion event. □

E.1 Proposal-Agnostic Memory Safety

Proof of proposal-agnostic memory safety. The adaptive-validity argument conditions on the complete pre-validation history $\mathcal{F}_t^0$ and places no restriction on how the frozen hypothesis family was generated from that history. Replacing the proposal or retrieval mechanism therefore changes only the conditional law of the frozen targets; it does not change the conditional crossing guarantee for their fresh validation streams.

The meta-strategy generator, residual interaction miner, population selection rule, and LLM may therefore use arbitrary combinations of previous residuals, validated findings, unsupported hypotheses, execution failures, and natural-language memory records. Provided every new numerical claim receives fresh validation observations and is charged to the same pathwise error ledger, Eq. (28) remains valid.

Events occurring after a record has been promoted may influence future proposals, but they cannot retroactively alter whether a false promotion occurred in the past. Applying the same argument to all later claims proves the result. □

Why adaptive proposal generation is allowed. All information used to propose a candidate is included in $\mathcal{F}_t^0$. After conditioning on this history, the new numerical targets are fixed before their validation data are observed. Nothing in the proof requires the residual interaction miner to identify the correct explanation or the candidate generator to be well calibrated. What is not allowed is modifying a candidate after seeing its new validation observations and then treating those same observations as fresh evidence for the modified candidate.

Why a fixed-sample bound cannot simply be monitored. At one predetermined sample size, the fixed-sample bound uses the full allocated level a_{tj}. Repeatedly inspecting that same radius creates additional rejection opportunities that are not charged to the original level. The anytime construction instead assigns the nth opportunity the error budget

$$\frac{6a_{tj}}{\pi^2 n^2},$$

whose sum over all positive integers n equals a_{tj}. Sharper confidence sequences may replace this elementary construction; the summable allocation is used because it makes the optional-stopping guarantee explicit.

What happens to old data. After a batch is closed, its observations may enter the history used for future proposal generation. They may motivate a new mutation, suggest a different generator, or reveal that a previous hypothesis should be reconsidered. They cannot be counted again as fresh validation groups for a newly defined claim under the analysis above.

A validated record therefore retains its original program versions, generator law, evaluation budget, reference and normalization rules, thresholds, and numerical contrasts. Statistical certification attaches to this recorded numerical target, not to every later natural-language paraphrase or generalization derived from it.

F Evidence Transfer Under Quantified Change

A validated record is tied to a particular program version, generator, normalization rule, and matched-group distribution. The following result identifies one quantitative setting in which support can be transferred to a nearby target.

Suppose every one of the four normalized losses changes pointwise by at most ϵ, and the paired-group law changes from P to Q with

$$\mathrm{TV}(P, Q) \le d.$$

Let the original statistic Z_k have interval width W_k on the common group space.

Because A, G, and D are respectively linear combinations of two, two, and four normalized losses, the sums of the absolute coefficients are

$$(c_A, c_G, c_D) = (2, 2, 4).$$

Proof of the evidence-transfer bound. For every coordinate k, the pointwise loss perturbation implies

$$|Z'_k(w) - Z_k(w)| \le c_k \epsilon \qquad \text{for every group } w.$$

Suppose

$$Z_k(w) \in [a_k, a_k + W_k].$$

If $W_k > 0$, define

$$f(w) = \frac{Z_k(w) - a_k}{W_k}, \qquad f(w) \in [0, 1].$$

Using

$$\mathrm{TV}(P, Q) = \sup_E |P(E) - Q(E)|,$$

the layer-cake identity gives

$$\begin{aligned} |\mathbb{E}_Q f - \mathbb{E}_P f| &= \left| \int_0^1 [Q(f > u) - P(f > u)] \, du \right| \\ &\le \mathrm{TV}(P, Q) \\ &\le d. \end{aligned}$$

When $W_k = 0$, the same conclusion is immediate.

Combining pointwise and distributional changes,

$$\begin{aligned} |\mathbb{E}_Q Z'_k - \mathbb{E}_P Z_k| &\le \mathbb{E}_Q |Z'_k - Z_k| + |\mathbb{E}_Q Z_k - \mathbb{E}_P Z_k| \\ &\le c_k \epsilon + W_k d. \end{aligned}$$

Therefore

$$|\mathbb{E}_Q[Z'_k] - \mathbb{E}_P[Z_k]| \le c_k \epsilon + W_k d, \qquad (c_A, c_G, c_D) = (2, 2, 4). \tag{29}$$

If

$$\mathbb{E}_P Z_k > \tau_k + c_k \epsilon + W_k d,$$

then Eq. (29) implies

$$\mathbb{E}_Q Z'_k > \tau_k.$$

Applying this implication to all three coordinates proves transfer of the supported conjunction. □

The error-controlled validation rule should therefore be applied to the original target using the inflated thresholds

$$\tau_k + c_k\epsilon + W_k d.$$

The drift bounds and inflated thresholds must be fixed before the validation observations used for the certificate are collected.

A practical restriction. A small syntactic change to a solver program is not by itself a bound on the resulting normalized loss. Likewise, similarity in a small collection of observed instance descriptors does not imply a small total-variation distance between the complete paired-group laws. Changes in the reference procedure, normalization rule, or seed distribution may also alter the target. Without externally justified bounds of the type assumed above, a changed program or condition defines a new target and requires fresh validation.

G Operational Protocol and Reproducibility Details

G.1 One Round of the Four-Pool Framework

The theory does not prescribe how the LLM implements meta-strategy generation, residual interpretation, or code mutation. These proposal-side components may remain flexible. The statistical addition is the explicit boundary between proposal generation and confirmation: the tested objects are frozen before fresh validation observations are collected.

Selection and validation have different roles. A solver may remain in the active search population because of favorable performance even when no new numerical claim about it is validated. Conversely, a validated record retains its evidential status even if its solver is later removed from the active population.

Algorithm 1 One round of IACE with evidence-controlled memory

Input: Solver and instance populations; typed memory $\mathcal{M}$; total error budget α.

1: **for** rounds $t = 1, 2, \ldots$ **do**
2: Retrieve relevant validated, candidate, and negative records with their status and scope.
3: Generate solver and instance strategies and instantiate legal heuristic and generator programs.
4: Cross-evaluate a shared diagnostic panel and compute Y and interaction residuals R.
5: Flag at most $m_{\max}$ pairs with $r_{hx} > \tau_r$ and $Y_{hx} > \tau_w$.
6: Apply the residual interaction miner to the flagged pairs and instantiate the proposed child heuristics and matched condition generators.
7: Freeze a finite hypothesis family, references, normalization rules, thresholds, and conservative scales.
8: Allocate levels a_{tj} from the remaining error budget before observing validation outcomes.
9: Draw fresh matched groups and evaluate the required program–condition combinations.
10: Compute A, G, and D and update the declared fixed-sample or anytime lower bounds.
11: Promote only pairs whose three coordinates exceed their thresholds and whose prespecified validity checks pass.
12: Store remaining outcomes as inconclusive, unsupported, or execution/control failures.
13: Select solver and instance populations by performance and close the validation batch.
14: **end for**

G.2 Fields of a Reusable Evidence Record

A reproducible evidence record should contain the parent and child program identifiers; the original and modified generator versions; the latent-variable and solver-seed laws; the solver backbone and evaluation budget; the reference and normalization rules; the three validation thresholds; conservative noise or range bounds; the assigned error level; the number of complete matched groups; the observed means and lower bounds for A, G, and D; the validity-check outcomes; and the exact population statement receiving support.

Solver strategies, instance strategies, and natural-language summaries may be associated with the record as provenance and search guidance, but they are not themselves statistically certified. The diagnostic panel and validation record also have different scopes. The diagnostic panel identifies the solver and instance programs used for screening and the observations used to form their scores. The evidence record identifies the fresh paired-group law and the frozen parent/child versions used for confirmation.

G.3 Execution Failures and Validity Checks

Timeouts, infeasible outputs, invalid generated instances, and failed descriptor checks must be handled according to a rule fixed before the corresponding observations are seen. Rejecting the entire candidate or refusing to issue a certificate preserves the error-control argument. Silently dropping difficult or unfavorable outcomes does not preserve the group law assumed in the proof.

A predetermined rejection sampler may instead define a conditional target distribution, but the resulting conditional law and its sampling cost must be reported. Likewise, the number of solver seeds used inside a matched group should either be fixed in advance or covered by an appropriate model for the resulting group-level statistic.

For maximization tasks, reverse the objective direction and use a declared positive common normalization scale. For example, if $R(h, x; \omega)$ denotes reward, one may set

$$C(h, x; \omega) = -R(h, x; \omega).$$

The theoretical development can then continue to use lower-is-better loss even when empirical plots retain the native direction of the original optimization problem.

G.4 Evaluation Cost

A complete diagnostic panel with H heuristics, N instances, and m evaluation replicates requires

$$mHN$$

solver executions. Once the matrix Y is available, double centering requires

$$O(HN)$$

arithmetic operations.

A full validation batch with n fresh matched groups, s solver seeds per group, H_{val} distinct parent/candidate programs, and V conditions requires

$$nsH_{\mathrm{val}}V$$

solver executions when evaluated fully. Shared parents need not be rerun independently for every child; the validation theory permits the resulting dependence among frozen candidate pairs.

The fresh-validation requirement is therefore an explicit computational cost rather than a free post-processing guarantee. The validation upper and lower sample-complexity results quantify the additional cost required to resolve a small condition-specific effect. They do not bound the computational cost of LLM proposal generation, the number of unsuccessful mutations, or the cost of discovering an informative condition.

H PSEUDOCODE

This appendix gives pseudocode for IACE that follows the structure of Figure 1. Algorithm 2 is the main loop over the four pools. Algorithms 3 and 4 cover the generation pool together with the legality and validity checks, Algorithm 5 the cross-evaluation pool with the interaction diagnosis of Section 3.1, Algorithms 6 and 7 the interaction pool with the controlled validation of Section 3.2, and Algorithm 8 selection and the memory pool. Steps annotated with a prompt number are single LLM calls that use the corresponding template of Appendix I; all other steps are computed without the LLM. The LLM parameters, the backbone B, and the budget β stay fixed throughout. The memory $\mathcal{M}$ and the fitness values F_H and F_G computed during selection are shared by all procedures. Table 4 lists the symbols that are not defined in the main text.

Table 4: Additional symbols used in the pseudocode.

Symbol	Meaning	Symbol	Meaning
$\mathcal{G}$, $g(z;\theta_g)$	generator population; generator with parameters θ_g	N_H, N_G	solver and generator population sizes
n_S, n_I	solver and instance strategies per iteration	n_x	panel instances drawn per generator
s_P	$+1$ for minimization, -1 for maximization	$p(\cdot)$, $s(\cdot)$	parent and strategy of a program
τ_r, τ_w	residual and weakness thresholds	$m_{\max}$	flagged pairs mined per iteration, at most
$\mathcal{F}$, Ψ	flagged pairs; hypotheses under test	h_ψ, g_{v_ψ}	candidate heuristic and varied generator for ψ
$\mathcal{C}$	prespecified family of tested comparisons	α, a_{tj}	total false-promotion budget; level assigned to hypothesis j in round t
ϵ, κ	control tolerance; required share of valid groups	$R_{\max}$	repair attempts per program, at most
$\mathcal{M}^{\mathrm{val}}$, $\mathcal{M}^{\mathrm{cand}}$, $\mathcal{M}^{\mathrm{neg}}$	validated evidence, candidate hypotheses, negative records	$\mathcal{M}^{\mathrm{arch}}$, k	archive of strategies and programs; records retrieved per status

H.1 MAIN LOOP

Algorithm 2 first builds the initial populations from a seed heuristic h_0 and a generator g_0 of the standard instance distribution. Each iteration then passes through the pools of Figure 1 in order. The generation pool proposes new heuristics and generators, the cross-evaluation pool evaluates them together with the current populations on a shared panel and flags solver–instance pairs, and the interaction pool turns the flagged pairs into hypotheses and tests them. Selection keeps the best heuristics and the hardest generators, and the memory pool records the outcomes so that the next call of the meta-strategy generator can use them. Heuristics whose runs fail on the panel are discarded before the interaction pool and selection. The total false-promotion budget α is shared by all validation batches of the whole run through a fixed ledger (Appendix E): each of the J_t hypotheses frozen in round t receives the level $a_{tj} = 6\alpha/(\pi^2 t^2 J_t)$, so that $\sum_{t\geq 1}\sum_{j\leq J_t} a_{tj} \leq \alpha$ and Eq. (28) holds for the complete run. Our experiments use $T = 20$ iterations.

H.2 GENERATION POOL

The generation pool separates *what to explore* from *how to instantiate a candidate* (Algorithm 3). The meta-strategy generator receives the current populations, the pairs flagged in the previous iteration, and the retrieved evidence, and returns n_S solver strategies and n_I instance strategies. Each strategy is then turned into a program by the solver generator or the instance generator, starting from a parent in the current population. Every generated program must pass the checks of Algorithm 4 before it is evaluated; a program that still fails after $R_{\max}$ repair attempts is discarded, and its strategy is stored as a negative record. Algorithm 4 also defines how a heuristic is evaluated on an instance, including the orientation of the gap for maximization problems.

H.3 Cross-Evaluation Pool and Interaction Diagnosis

The cross-evaluation pool evaluates all valid heuristics on a shared panel that contains n_x instances from each valid generator (Algorithm 5). A heuristic with a failed or timed-out run on any panel instance is recorded as a negative record and removed, together with its row of Y, before any statistic is computed; the filtered set of heuristics is returned and used by the remaining pools. The resulting matrix Y serves two purposes. Its row and column means give the fitness values used for selection, and its double-centered form gives the interaction residuals r_{hx} of Eq. (2). A pair is flagged when its residual exceeds τ_r and its gap exceeds the weakness threshold τ_w. This is the screening rule of Section 3.1 with the fixed threshold τ_r in place of $\lambda + b$. Because the panel is redrawn in every round and the seeds Ω are shared across the cells of a replicate, the flags are exploratory screening signals in the sense of Appendix B.2 and carry no confirmatory guarantee; confirmation is left to the fresh validation of Algorithm 7. At most $m_{\max}$ flagged pairs, those with the largest residuals, are passed on.

H.4 Interaction Pool

The interaction pool processes the flagged pairs in decreasing order of their residuals (Algorithm 6). For each pair, the residual interaction miner receives the code of the heuristic, the generator and descriptors of the instance, the performance record, and related memory records, and returns either a hypothesis or NULL. A hypothesis whose rule, descriptor, and parent heuristic coincide with those of a record in $\mathcal{M}^{\mathrm{val}} \cup \mathcal{M}^{\mathrm{neg}}$ is skipped; this check compares the structured fields of the records and does not use the LLM. A new hypothesis is stored as a candidate before it is tested, so an unverified explanation is never treated as a finding. It induces two interventions: a minimal solver-side modification h_ψ of the flagged heuristic, which becomes its parent $p(h_\psi)$, and a varied generator g_{v_ψ} that changes only the hypothesized property of the original generator g_o. If either intervention cannot be made legal or valid, the hypothesis is moved from $\mathcal{M}^{\mathrm{cand}}$ to $\mathcal{M}^{\mathrm{neg}}$, so that every record keeps exactly one evidential status. After mining, the $J_t = |\Psi|$ hypotheses of round t are frozen and each receives the level $a_{tj} = 6\alpha/(\pi^2 t^2 J_t)$ of the error ledger. Hypotheses that share g_o are validated together, so that one evaluation table covers all of their candidates and conditions.

Algorithm 7 implements the controlled validation of Sections 3.2 and 3.3. All programs, conditions, comparisons, thresholds, conservative scales ν_{jk}, levels a_j, the number of matched groups n, and the number of solver seeds s per group are fixed before the first validation sample is drawn. Each group u draws fresh latent factors z_u and fresh solver seeds Ω_u, as in Eq. (7), yields matched instances under all conditions, and evaluates them with every heuristic of the pool, the candidates together with their parents. The factor s_P orients the contrasts of Eq. (8) for maximization problems, as described in Appendix A.1. For each comparison j and coordinate $k \in \{A, G, D\}$, the lower bound $L_{jk} = \bar{Z}_{jk} - \nu_{jk}\sqrt{(2/n)\log(1/a_j)}$ of Eq. (14) is computed from the group-level statistics, which keeps all dependence between heuristics, conditions, and seeds inside a group. A comparison is validated when all three lower bounds exceed their thresholds, i.e., by the intersection–union rule of Eq. (14), so that the population criterion of Eq. (12) is certified with the false-promotion guarantee of Eqs. (15) and (28). It is marked unsupported when the symmetric upper bound of one coordinate does not exceed its threshold; this label guides later search but is not error-controlled. By a rule fixed in advance, a comparison is inconclusive whenever any run of the candidate or its parent fails, or when fewer than a fraction κ of the groups pass the descriptor check; such comparisons are never promoted, and their hypotheses stay in $\mathcal{M}^{\mathrm{cand}}$. Because n is fixed in advance, the fixed-sample bound suffices; if the number of groups is chosen adaptively, the anytime bound of Eq. (26) replaces it.

H.5 Selection and Memory Pool

Selection is performance-based (Algorithm 8). Programs that have no panel results yet, i.e., validated candidates and varied generators from the interaction pool, are first evaluated on the panel; a program whose panel evaluation fails is discarded. Heuristics are ranked by their mean gap on the panel, and generators by the mean gap their instances induce on the selected heuristics. The memory pool stores the outcome of every strategy: a strategy whose program improved on its parent is archived together with the program, and a strategy whose program did not is stored as a negative record. The archive $\mathcal{M}^{\mathrm{arch}}$ is not shown to the LLM; it stores the lineage $p(\cdot)$, $s(\cdot)$ of all programs,

which RETRIEVE uses to rank records by shared program lineage, and which makes every run reproducible. Retrieval ranks the records of each evidential status separately and returns the k most relevant ones per status, so that the prompts can present validated findings, candidate hypotheses, and negative records as separate lists.

H.6 COMPUTATIONAL COST

Per iteration, the LLM is called once by the meta-strategy generator, $n_S + n_I$ times by the solver and instance generators, $|\mathcal{F}| \le m_{\max}$ times by the residual interaction miner, and twice for every new hypothesis (Prompts 5 and 6), plus the calls of Prompt 9 for failed checks. The backbone is run at most $|\mathcal{H} \cup \mathcal{H}'| \cdot |\mathcal{X}_{\mathrm{diag}}| \cdot |\Omega|$ times for the panel and $\sum_b |\mathcal{H}_{\mathrm{pool},b}| \cdot |\mathcal{V}_b| \cdot n \cdot s$ times for the validation batches b. The reference solver is run once for every new panel instance and once for each of the $n\,|\mathcal{V}_b|$ fresh instances of every validation batch, and its values are cached. Validation dominates the cost when many hypotheses are tested; the cap $m_{\max}$ on flagged pairs bounds this part, and by Eq. (16) the required n grows as ν^2/Δ^2 in the weakest validation margin.

Algorithm 2 IACE: interaction-aware co-evolution

Input: objective f_x with sense s_P; backbone B; budget β; panel seeds Ω; seed heuristic h_0; standard generator g_0; sizes N_H, N_G; iterations T; total error budget α
Output: best heuristic $h^\star$; generator population $\mathcal{G}$; memory $\mathcal{M}$

$\mathcal{H} \leftarrow \{h_0\}$; $\mathcal{G} \leftarrow \{g_0\}$
for $j = 1, \dots, N_H - 1$ **do** ▷ initial solver population
  $h \leftarrow$ CHECKREPAIR(INITSOLVER($h_0, \mathcal{H}$), legal) ▷ Prompts 7, 9
  add h to $\mathcal{H}$ if $h \neq \perp$
end for
for $j = 1, \dots, N_G - 1$ **do** ▷ initial generator population
  $g \leftarrow$ CHECKREPAIR(INITGENERATOR($g_0, \mathcal{G}$), valid) ▷ Prompts 8, 9
  add g to $\mathcal{G}$ if $g \neq \perp$
end for
$\mathcal{M} = (\mathcal{M}^{\mathrm{val}}, \mathcal{M}^{\mathrm{cand}}, \mathcal{M}^{\mathrm{neg}}, \mathcal{M}^{\mathrm{arch}}) \leftarrow (\emptyset, \emptyset, \emptyset, \emptyset)$; $\mathcal{F} \leftarrow \emptyset$
for $t = 1, \dots, T$ **do**
  ▷ generation pool (Algorithm 3)
  $(\mathcal{H}', \mathcal{G}') \leftarrow$ GENERATIONPOOL($\mathcal{H}, \mathcal{G}, \mathcal{F}, \mathcal{M}$)
  ▷ cross-evaluation pool (Algorithm 5)
  $(\mathcal{H}_{\mathrm{ok}}, Y, r, \mathcal{F}, \mathcal{X}_{\mathrm{diag}}) \leftarrow$ CROSSEVALUATION($\mathcal{H} \cup \mathcal{H}',\ \mathcal{G} \cup \mathcal{G}'$)
  $\mathcal{H}' \leftarrow \mathcal{H}' \cap \mathcal{H}_{\mathrm{ok}}$ ▷ drop heuristics whose runs failed
  ▷ interaction pool (Algorithms 6 and 7)
  $(\mathcal{H}_{\mathrm{val}}, \mathcal{G}_{\mathrm{val}}) \leftarrow$ INTERACTIONPOOL($\mathcal{F}, Y, r, \mathcal{M}, t, \alpha$)
  ▷ selection and memory pool (Algorithm 8)
  $(\mathcal{H}, \mathcal{G}) \leftarrow$ SELECT($\mathcal{H}_{\mathrm{ok}} \cup \mathcal{H}_{\mathrm{val}},\ \mathcal{G} \cup \mathcal{G}' \cup \mathcal{G}_{\mathrm{val}},\ \mathcal{X}_{\mathrm{diag}}$)
  UPDATEMEMORY($\mathcal{M}, \mathcal{H}', \mathcal{G}', \mathcal{H}, \mathcal{G}$)
end for
return $h^\star = \arg\min_{h \in \mathcal{H}} F_H(h), \mathcal{G}, \mathcal{M}$ ▷ F_H from the last SELECT

Algorithm 3 Generation pool

procedure GENERATIONPOOL($\mathcal{H}, \mathcal{G}, \mathcal{F}, \mathcal{M}$)
  $E \leftarrow$ RETRIEVE($\mathcal{M}$; rules of $\mathcal{H}$, descriptors of $\mathcal{G}$) ▷ Algorithm 8
  $(\mathcal{S}^{\mathrm{sol}}, \mathcal{S}^{\mathrm{ins}}) \leftarrow$ METASTRATEGY($\mathcal{H}, \mathcal{G}, \mathcal{F}, E;\ n_S, n_I$) ▷ Prompt 1
  $\mathcal{H}' \leftarrow \emptyset$; $\mathcal{G}' \leftarrow \emptyset$
  for each solver strategy $s \in \mathcal{S}^{\mathrm{sol}}$ **do**
    $p \leftarrow$ SELECTPARENT($\mathcal{H}, s$) ▷ e.g., the heuristic whose weakness s targets
    $E_s \leftarrow$ RETRIEVE($\mathcal{M}$; target rule of s)
    $h \leftarrow$ CHECKREPAIR(SOLVERGEN(p, s, E_s), legal) ▷ Prompts 2, 9
    if $h = \perp$ **then**
      $\mathcal{M}^{\mathrm{neg}} \leftarrow \mathcal{M}^{\mathrm{neg}} \cup \{(s, \text{failed legality check})\}$
    else
      $p(h) \leftarrow p$; $s(h) \leftarrow s$; $\mathcal{H}' \leftarrow \mathcal{H}' \cup \{h\}$
    end if
  end for
  for each instance strategy $s \in \mathcal{S}^{\mathrm{ins}}$ **do**
    $g_p \leftarrow$ SELECTPARENT($\mathcal{G}, s$)
    $E_s \leftarrow$ RETRIEVE($\mathcal{M}$; target property of s)
    $g \leftarrow$ CHECKREPAIR(INSTANCEGEN(g_p, s, rules of $\mathcal{H}$, E_s), valid) ▷ Prompts 3, 9
    if $g = \perp$ **then**
      $\mathcal{M}^{\mathrm{neg}} \leftarrow \mathcal{M}^{\mathrm{neg}} \cup \{(s, \text{failed validity check})\}$
    else
      $p(g) \leftarrow g_p$; $s(g) \leftarrow s$; $\mathcal{G}' \leftarrow \mathcal{G}' \cup \{g\}$
    end if
  end for
  return $\mathcal{H}', \mathcal{G}'$
end procedure

Algorithm 4 Legality and validity checks, repair, and evaluation of a heuristic

function CHECKREPAIR(π, kind) ▷ π is a heuristic or a generator
  for $j = 0, 1, \ldots, R_{\max}$ **do**
    $(ok, err, input) \leftarrow$ TEST(π, kind)
    return π if ok
    $\pi \leftarrow$ REPAIR(π, err, $input$) if $j < R_{\max}$ ▷ Prompt 9
  end for
  return $\perp$
end function
function TEST(π, kind)
  if kind = legal **then** ▷ legality check
    run $B[\pi]$ on small test instances; check the signature, finite outputs of the documented shape, the time limit, and the feasibility of the returned solutions
  else ▷ validity check
    draw test factors z and check that $\pi(z; \theta)$ satisfies the validity constraints and the size n, returns its descriptors, and is reproduced by the same z
  end if
  return the outcome, the first error message, and the input that caused it
end function
function EVALUATE(h, x, Ω') ▷ Ω': the seed set used for this call
  $c \leftarrow c_{\mathrm{ref}}(x)$ ▷ computed once per instance and cached (Table 3)
  for each $\omega \in \Omega'$ **do**
    $C(h, x; \omega) \leftarrow f_x(B[h](x; \omega, \beta))$ ▷ Eq. (1)
    return $\perp$ if the run fails or exceeds the time limit
    $\mathrm{gap}(h, x; \omega) \leftarrow s_P\left(C(h, x; \omega) - c\right)/c$ ▷ Eqs. (1) and (19)
  end for
  return $\bar{C}_{hx} = \frac{1}{|\Omega'|}\sum_{\omega} C(h, x; \omega)$ and $Y_{hx} = \frac{1}{|\Omega'|}\sum_{\omega} \mathrm{gap}(h, x; \omega)$
end function

Algorithm 5 Cross-evaluation pool and interaction diagnosis

procedure CROSSEVALUATION($\mathcal{H}_{\text{all}}, \mathcal{G}_{\text{all}}$)
  $\mathcal{X}_{\text{diag}} \leftarrow \emptyset$; $\mathcal{H}_{\text{fail}} \leftarrow \emptyset$
  for each $g \in \mathcal{G}_{\text{all}}$ and $j = 1, \ldots, n_x$ **do** ▷ shared panel
    draw shared factors z; $x \leftarrow g(z; \theta_g)$; $\text{gen}(x) \leftarrow g$; $\mathcal{X}_{\text{diag}} \leftarrow \mathcal{X}_{\text{diag}} \cup \{x\}$
  end for
  for each $h \in \mathcal{H}_{\text{all}}$ **do** ▷ performance evaluation
    for each $x \in \mathcal{X}_{\text{diag}}$ **do**
      $(\bar{C}_{hx}, Y_{hx}) \leftarrow$ EVALUATE(h, x, Ω) ▷ Algorithm 4
      if the evaluation returned $\perp$ **then**
        $\mathcal{H}_{\text{fail}} \leftarrow \mathcal{H}_{\text{fail}} \cup \{h\}$; **break** ▷ skip the remaining instances of h
      end if
    end for
  end for
  for each $h \in \mathcal{H}_{\text{fail}}$ **do**
    $\mathcal{M}^{\text{neg}} \leftarrow \mathcal{M}^{\text{neg}} \cup \{(h, \text{failed run})\}$
  end for
  $\mathcal{H}_{\text{all}} \leftarrow \mathcal{H}_{\text{all}} \setminus \mathcal{H}_{\text{fail}}$; delete the rows of $\mathcal{H}_{\text{fail}}$ from Y ▷ after the loop
  $\bar{Y}_{h\cdot} \leftarrow \frac{1}{|\mathcal{X}_{\text{diag}}|} \sum_x Y_{hx}$; $\bar{Y}_{\cdot x} \leftarrow \frac{1}{|\mathcal{H}_{\text{all}}|} \sum_h Y_{hx}$; $\bar{Y}_{\cdot\cdot} \leftarrow \frac{1}{|\mathcal{H}_{\text{all}}|\,|\mathcal{X}_{\text{diag}}|} \sum_{h,x} Y_{hx}$
  $r_{hx} \leftarrow Y_{hx} - \bar{Y}_{h\cdot} - \bar{Y}_{\cdot x} + \bar{Y}_{\cdot\cdot}$ for all $h \in \mathcal{H}_{\text{all}}$ and $x \in \mathcal{X}_{\text{diag}}$ ▷ Eq. (2)
  $\mathcal{F} \leftarrow \{(h, x) : r_{hx} > \tau_r \text{ and } Y_{hx} > \tau_w\}$ ▷ flagged pairs (exploratory)
  keep the $m_{\max}$ pairs of $\mathcal{F}$ with the largest residuals
  return $\mathcal{H}_{\text{all}}, Y, r, \mathcal{F}, \mathcal{X}_{\text{diag}}$
end procedure

Algorithm 6 Interaction pool: residual interaction mining and hypothesis testing

procedure INTERACTIONPOOL($\mathcal{F}, Y, r, \mathcal{M}, t, \alpha$)
  $\Psi \leftarrow \emptyset$; $\mathcal{H}_{\text{val}} \leftarrow \emptyset$; $\mathcal{G}_{\text{val}} \leftarrow \emptyset$
  for each $(h, x) \in \mathcal{F}$ in decreasing order of r_{hx} **do** ▷ residual interaction miner
    $g_o \leftarrow \text{gen}(x)$; $E_{hx} \leftarrow$ RETRIEVE($\mathcal{M}$; rules of h, descriptors of x)
    $\psi \leftarrow$ RIM(h, x, g_o, record of (h, x), E_{hx}) ▷ Prompt 4
    ▷ record: $Y_{hx}, r_{hx}, \bar{Y}_{h\cdot}, \bar{Y}_{\cdot x}, \bar{Y}_{\cdot\cdot}$, row of h, column of x
    if $\psi =$ NULL **or** (rule, descriptor, parent) of ψ match a record of $\mathcal{M}^{\text{val}} \cup \mathcal{M}^{\text{neg}}$ **then**
      continue
    end if
    $\mathcal{M}^{\text{cand}} \leftarrow \mathcal{M}^{\text{cand}} \cup \{\psi\}$ ▷ stored as unverified
    $h_\psi \leftarrow$ CHECKREPAIR(MODIFYSOLVER(h, ψ), legal) ▷ Prompt 5
    $g_{v_\psi} \leftarrow$ CHECKREPAIR(VARYINSTANCE(g_o, θ_o, ψ), valid) ▷ Prompt 6
    if $h_\psi = \perp$ **or** $g_{v_\psi} = \perp$ **then**
      move ψ from $\mathcal{M}^{\text{cand}}$ to $\mathcal{M}^{\text{neg}}$ with note "failed intervention"
      continue
    end if
    $p(h_\psi) \leftarrow h$; $\Psi \leftarrow \Psi \cup \{\psi\}$
  end for
  $J_t \leftarrow \max(|\Psi|, 1)$; $a_\psi \leftarrow 6\alpha/(\pi^2 t^2 J_t)$ for all $\psi \in \Psi$ ▷ error ledger, fixed before validation
  for each batch $\Psi_b \subseteq \Psi$ of hypotheses with the same original generator g_o **do**
    $\mathcal{H}_{\text{cand}} \leftarrow \{h_\psi\}_{\psi \in \Psi_b}$; $\mathcal{V} \leftarrow \{o\} \cup \{v_\psi\}_{\psi \in \Psi_b}$; $\mathcal{C} \leftarrow \{(h_\psi, v_\psi)\}_{\psi \in \Psi_b}$
    $res \leftarrow$ VALIDATE($\mathcal{H}_{\text{cand}}, \{g_v\}_{v \in \mathcal{V}}, \mathcal{C}, \{a_\psi\}_{\psi \in \Psi_b}$) ▷ Algorithm 7
    for each $\psi \in \Psi_b$ **do**
      if $res(h_\psi, v_\psi) =$ validated **then**
        move ψ to $\mathcal{M}^{\text{val}}$ with its programs, conditions, level a_ψ, and bounds
        $\mathcal{H}_{\text{val}} \leftarrow \mathcal{H}_{\text{val}} \cup \{h_\psi\}$; $\mathcal{G}_{\text{val}} \leftarrow \mathcal{G}_{\text{val}} \cup \{g_{v_\psi}\}$
      else if $res(h_\psi, v_\psi) =$ unsupported **then**
        move ψ to $\mathcal{M}^{\text{neg}}$ with the criterion that was not met
      else
        keep ψ in $\mathcal{M}^{\text{cand}}$ and attach its bounds ▷ inconclusive
      end if
    end for
  end for
  return $\mathcal{H}_{\text{val}}, \mathcal{G}_{\text{val}}$
end procedure

Algorithm 7 Bidirectional controlled contrast validation

function VALIDATE($\mathcal{H}_{\text{cand}}, \{g_v\}_{v\in\mathcal{V}}, \mathcal{C}, \{a_j\}_{j\in\mathcal{C}}$)
  $\mathcal{H}_{\text{pool}} \leftarrow \mathcal{H}_{\text{cand}} \cup \{p(h) : h \in \mathcal{H}_{\text{cand}}\}$ ▷ candidates and their parents
  fix $\mathcal{H}_{\text{pool}}$, $\{g_v\}$, $\mathcal{C}$, $\{a_j\}$, τ_A, τ_G, τ_D, ν_{jk}, n, and s before any evaluation ▷ Eqs. (13)–(14)
  $fail_h \leftarrow$ **false** for all $h \in \mathcal{H}_{\text{pool}}$
  for $u = 1, \dots, n$ **do**
    draw fresh shared factors z_u and fresh solver seeds $\Omega_u = \{\omega_{u1}, \dots, \omega_{us}\}$
    for each condition $v \in \mathcal{V}$ **do**
      $x_v^{(u)} \leftarrow g_v(z_u; \theta_v)$ ▷ Eq. (7)
      $ok_v^{(u)} \leftarrow$ descriptor check of $x_v^{(u)}$ against $x_o^{(u)}$: target range met, controls within ϵ
      for each $h \in \mathcal{H}_{\text{pool}}$ **do**
        $(\bar{C}_{hv}^{(u)}, \cdot) \leftarrow$ EVALUATE($h, x_v^{(u)}, \Omega_u$) ▷ Eq. (7)
        $fail_h \leftarrow$ **true** if the evaluation returned $\perp$
      end for
    end for
    $q_u \leftarrow c_{\text{ref}}(x_o^{(u)})$
    for each $(h, v) \in \mathcal{C}$ with $\neg fail_h \wedge \neg fail_{p(h)}$ **do** ▷ Eq. (8), oriented by s_P
      $A_{hv}^{(u)} \leftarrow s_P\big([\bar{C}_{p(h)v}^{(u)} - c_{\text{ref}}(x_v^{(u)})] - [\bar{C}_{p(h)o}^{(u)} - c_{\text{ref}}(x_o^{(u)})]\big)/q_u$
      $G_{hv}^{(u)} \leftarrow s_P\,(\bar{C}_{p(h)v}^{(u)} - \bar{C}_{hv}^{(u)})/q_u; \quad G_{ho}^{(u)} \leftarrow s_P\,(\bar{C}_{p(h)o}^{(u)} - \bar{C}_{ho}^{(u)})/q_u$
      $D_{hv}^{(u)} \leftarrow G_{hv}^{(u)} - G_{ho}^{(u)}$
    end for
  end for
  for each $j = (h, v) \in \mathcal{C}$ **do**
    if $fail_h$ **or** $fail_{p(h)}$ **or** $\frac{1}{n}\sum_u ok_v^{(u)} < \kappa$ **then**
      $res(j) \leftarrow$ inconclusive ▷ invalid execution or control
    else
      for each $k \in \{A, G, D\}$ **do** ▷ $Z_A = A_{hv}$, $Z_G = G_{hv}$, $Z_D = D_{hv}$
        $\bar{Z}_{jk} \leftarrow \frac{1}{n}\sum_{u=1}^{n} Z_{jk}^{(u)}; \quad b_{jk} \leftarrow \nu_{jk}\sqrt{(2/n)\log(1/a_j)}$
        $L_{jk} \leftarrow \bar{Z}_{jk} - b_{jk}; \quad U_{jk} \leftarrow \bar{Z}_{jk} + b_{jk}$
      end for
      if $L_{jk} > \tau_k$ for all $k \in \{A, G, D\}$ **then**
        $res(j) \leftarrow$ validated ▷ Eq. (14); certifies Eq. (12)
      else if $U_{jk} \le \tau_k$ for some $k \in \{A, G, D\}$ **then**
        $res(j) \leftarrow$ unsupported ▷ guides search; not error-controlled
      else
        $res(j) \leftarrow$ inconclusive
      end if
    end if
  end for
  return res together with $\bar{Z}_{jk}$, L_{jk}, and U_{jk}
end function

Algorithm 8 Selection and memory pool

procedure SELECT($\mathcal{H}_{\text{all}}, \mathcal{G}_{\text{all}}, \mathcal{X}_{\text{diag}}$)
  evaluate programs without panel results on $\mathcal{X}_{\text{diag}}$, adding n_x instances per new generator; discard programs whose evaluation returns $\perp$
  $F_H(h) \leftarrow Y_{h\cdot}$ for all $h \in \mathcal{H}_{\text{all}}$ ▷ lower is better
  $\mathcal{H} \leftarrow$ the N_H heuristics of $\mathcal{H}_{\text{all}}$ with the smallest F_H
  $F_G(g) \leftarrow$ mean of Y_{hx} over $h \in \mathcal{H}$ and the panel instances x of g ▷ higher is harder
  $\mathcal{G} \leftarrow$ the N_G generators of $\mathcal{G}_{\text{all}}$ with the largest F_G
  return $\mathcal{H}, \mathcal{G}$
end procedure
procedure UPDATEMEMORY($\mathcal{M}, \mathcal{H}', \mathcal{G}', \mathcal{H}, \mathcal{G}$)
  for each $h \in \mathcal{H}'$ **do** ▷ outcome of solver strategy $s(h)$
    if $F_H(h) < F_H(p(h))$ **then**
      $\mathcal{M}^{\text{arch}} \leftarrow \mathcal{M}^{\text{arch}} \cup \{(s(h), h, p(h), F_H(h))\}$
    else
      $\mathcal{M}^{\text{neg}} \leftarrow \mathcal{M}^{\text{neg}} \cup \{(s(h), \text{no improvement over } p(h))\}$
    end if
  end for
  for each $g \in \mathcal{G}'$ **do** ▷ outcome of instance strategy $s(g)$
    if $F_G(g) > F_G(p(g))$ **then**
      $\mathcal{M}^{\text{arch}} \leftarrow \mathcal{M}^{\text{arch}} \cup \{(s(g), g, p(g), F_G(g))\}$
    else
      $\mathcal{M}^{\text{neg}} \leftarrow \mathcal{M}^{\text{neg}} \cup \{(s(g), \text{no increase in hardness over } p(g))\}$
    end if
  end for
  add the selected heuristics $\mathcal{H}$ and generators $\mathcal{G}$ with their strategies and parents to $\mathcal{M}^{\text{arch}}$
end procedure
function RETRIEVE($\mathcal{M}$; query)
  for each status $\sigma \in \{\text{val}, \text{cand}, \text{neg}\}$ **do**
    rank the records of $\mathcal{M}^{\sigma}$ by relevance to the query, i.e., shared decision rules, shared descriptors, or a shared program lineage (read from $\mathcal{M}^{\text{arch}}$)
    $E^{\sigma} \leftarrow$ the k highest-ranked records of $\mathcal{M}^{\sigma}$
  end for
  return $(E^{\text{val}}, E^{\text{cand}}, E^{\text{neg}})$ ▷ statuses stay separate
end function

I PROMPT TEMPLATES

I.1 OVERVIEW

The LLM is called at four points of an IACE iteration (Figure 1). The meta-strategy generator (`prompt:meta`) reads the current populations and the retrieved evidence and proposes solver and instance strategies, which the solver generator (`prompt:solver`) and the instance generator (`prompt:instance`) implement as code. After cross-evaluation, the residual interaction miner (`prompt:rim`) turns each flagged solver–instance pair into a hypothesis. For every hypothesis, bidirectional controlled validation requires a solver-side modification (`prompt:mod`) and an instance-side variation (`prompt:var`). Legality checks, evaluation, residual extraction, the validation statistics, and the status of memory records are computed without the LLM. Table 5 summarizes all prompts, including the auxiliary prompts for initialization and code repair.

Table 5: Prompts used by IACE.

Prompt	Module	Called	Main inputs	Output
1	Meta-strategy generator	once per iteration	solver and instance populations, flagged pairs, retrieved evidence	solver and instance strategies (JSON)
2	Solver generator	per solver strategy	parent heuristic, strategy, related evidence	child heuristic (code)
3	Instance generator	per instance strategy	parent generator, strategy, solver rules, related evidence	instance generator (code)
4	Residual interaction miner	per flagged pair	heuristic code, instance descriptors, residuals, related memory	hypothesis (JSON)
5	Solver-side modification	per hypothesis	parent heuristic, hypothesis	candidate heuristic (code)
6	Instance-side variation	per hypothesis	original generator, hypothesis, control descriptors	varied generator (code)
7	Initial heuristic	initialization	reference heuristic, ideas in the population	heuristic (code)
8	Initial instance generator	initialization	standard generator, properties in the population	instance generator (code)
9	Code repair	after a failed check	code, error message, requirements	repaired code

Every prompt consists of a system message and a user message. Fields in braces, such as `{problem_description}`, are filled in at run time, and text in angle brackets describes what the model should write. JSON outputs are parsed before use, and code is extracted from the Python block of the response.

Evidence records. Records retrieved from the memory pool are inserted into Prompts 1–4 in the format below. The status of a record is set by the validation criteria of Section 3.3 and is never assigned by the LLM.

```
[<id>] status=<validated | candidate | negative>    iteration=<t>
rule: <decision rule of the heuristic>
condition: <descriptor> in <range under v>    (original condition: <range under o>)
programs: heuristic <parent id> -> <child id>; generator <o id> -> <v id>
statement: <rule R fails when condition C holds, because M>
validation: LCB(A)=<a>  LCB(G)=<g>  LCB(D)=<d>  over <k> groups
note: <for negative records: the criterion that was not met, or the failed check>
```

I.2 PROBLEM-SPECIFIC FIELDS

The fields `{problem_description}`, `{heuristic_signature}`, `{instance_format}`, and `{validity_constraints}` depend on the problem. Table 6 lists them for the five problems solved with the ACO backbone. For other backbones, `{heuristic_signature}` and `{component_role}` describe the evolved component of that backbone. The descriptors are examples of instance statistics used by the instance-side prompts.

Table 6: Problem-specific fields for the ACO backbone.

Problem	Instance fields	Evolved function and its output	Example descriptors
TSP min. tour length	n node coordinates	`heuristics(distance_matrix)`; scores of the $n \times n$ edges	nearest-neighbor distance CV, number of clusters
CVRP min. total route length	depot and customer coordinates, demands, capacity	`heuristics(distance_matrix, coordinates, demands, capacity)`; edge scores	demand-to-capacity ratio, depot offset
OP max. prize within a length budget	coordinates, prizes, maximum tour length	`heuristics(prize, distance, maxlen)`; edge scores	prize–distance correlation, budget ratio
MKP max. total prize	prizes, m-dimensional weights, capacities	`heuristics(prize, weight)`; item scores	prize–weight correlation, constraint tightness
BPP min. number of bins	item sizes, bin capacity	`heuristics(demand, capacity)`; scores of item pairs	mean size-to-capacity ratio, share of large items

I.3 Prompts of the Co-Evolution Loop

Prompts 1–3 belong to the generation pool and Prompts 4–6 to the interaction pool. The notation follows Section 3.1: Y_{hx} is the mean gap of heuristic h on instance x, r_{hx} the interaction residual, and A, G, D the validation contrasts of Equation (8).

Prompt 1: Meta-strategy generator. Called once per iteration; the solver strategies are passed to `prompt:solver` and the instance strategies to `prompt:instance`.

```
[System]
You are an expert in combinatorial optimization and automated algorithm design. You are
the strategy planner of a solver-instance co-evolution system. In each iteration you
decide which weaknesses of the current heuristics should be addressed on the solver side,
and which instance conditions should be explored on the instance side. You do not write
code; other agents implement your strategies. Base every decision on the evidence given
below, and keep validated findings, unverified hypotheses, and failed attempts clearly
apart.

[User]
## Problem
{problem_description}

## Fixed setting
- Solver backbone: {backbone_description}
- Evolvable component: {heuristic_signature}
- Role of the component in the backbone: {component_role}
- The backbone, the evaluation budget, the random seeds, and the LLM settings do not
  change during the search.
- Iteration: {t} of {T}

## Solver population (sorted by mean gap on the diagnostic panel)
{solver_population}
Format of each entry: id | parent id | one-line summary of decision rules | mean gap (%)

## Instance population (sorted by mean gap across solvers)
{instance_population}
Format of each entry: id | instance strategy | descriptor summary | mean gap (%)

## Interaction diagnosis
Solver-instance pairs flagged in the latest diagnostic matrix. A pair is flagged when its
interaction residual r_hx is large and positive and its gap exceeds the weakness
threshold. Hypotheses already derived from a pair are listed with it.
{flagged_pairs}
```

```
## Retrieved evidence
Validated findings. Each finding holds only under the conditions listed with it.
{validated_evidence}

Candidate hypotheses. Proposed from diagnostic signals but not yet validated.
{candidate_hypotheses}

Negative records. Hypotheses that were not supported, and attempts that did not improve
the heuristic.
{negative_records}

## Instructions
Propose {n_solver} solver strategies and {n_instance} instance strategies for the next
iteration. Start with a short analysis of the current state:
1. Which weaknesses are already validated, and under which instance conditions?
2. Which hypotheses are still open, and what evidence would settle them?
3. Which directions have already failed?

Each solver strategy must
- name the decision rule to change and the direction of the change,
- name the instance condition under which the current rule fails, and
- state where the gap should decrease and where it should stay unchanged.

Each instance strategy must
- name the instance property to vary and a descriptor that measures it,
- state how the generator should change this property, and
- state which solver behavior the new instances are expected to expose.

## Rules
- Use a validated finding only within the conditions under which it was validated.
- Treat candidate hypotheses as claims to be tested, not as facts.
- Do not repeat a direction from the negative records unless you explain what is
  different.
- Do not propose changes that only rescale constants or rename variables.
- Do not propose instance strategies that only change the problem size.
- No two strategies may target the same pair of rule and condition.
- At least one strategy must test an open hypothesis or explore a new condition.
- Do not write code.

## Output format
Return a JSON object and nothing else:
{
  "analysis": "<2-4 sentences on the current state>",
  "solver_strategies": [
    {
      "id": "S1",
      "target_rule": "<decision rule to change>",
      "failure_condition": "<instance condition under which the rule fails>",
      "direction": "<what to change, and why>",
      "expected_effect": "<where the gap should decrease or stay unchanged>",
      "evidence_ids": ["<record id>", ...],
      "basis": "validated" | "hypothesis" | "exploratory"
    },
    ...
  ],
  "instance_strategies": [
    {
      "id": "I1",
      "target_property": "<instance property to vary>",
      "descriptor": "<name and definition of the descriptor>",
      "direction": "<how the generator should change the property>",
      "probe": "<solver behavior the instances should expose>",
      "evidence_ids": ["<record id>", ...],
      "basis": "validated" | "hypothesis" | "exploratory"
    },
    ...
  ]
}
```

Prompt 2: Solver generator. Called once per solver strategy; the returned function replaces the designated heuristic component of the backbone.

```
[System]
You are an expert in heuristic design for combinatorial optimization and an experienced
Python programmer. You improve one component of a fixed solver by rewriting a single
Python function; the rest of the solver cannot be changed. Each new version is evaluated
against its parent on the same instances, so change the code in a targeted way and leave
everything else as it is. Respond only in the requested format.

[User]
## Problem
{problem_description}

## Solver backbone
{backbone_description}
The backbone calls the function below {call_site}. The evaluation budget is fixed:
{budget}.

## Function to modify
Signature: {heuristic_signature}
Inputs: {input_spec}
Output: {output_spec}

## Parent heuristic
Id: {parent_id}. Mean gap on the diagnostic panel: {parent_gap}%.
```python
{parent_code}
```

Strategy
Target rule: {target_rule}
Failure condition: {failure_condition}
Direction: {direction}
Expected effect: {expected_effect}

Related evidence
Validated findings about this rule:
{validated_findings}
Negative records about this rule (do not repeat them):
{negative_records}

Instructions
1. Find the part of the parent code that implements the target rule.
2. Consider how this rule behaves on instances that satisfy the failure condition, and why
   it makes poor decisions there.
3. Change the rule as described in the strategy. Keep the other parts of the parent
   unchanged, so that a difference in performance can be attributed to this change.
4. Check the code against the constraints below before answering.

Constraints
- Keep the function name, the argument names and their order, and the return type.
- For every valid input, return an array of the documented shape that contains only finite
  values. Handle edge cases such as {edge_cases}.
- Use only NumPy and the Python standard library. Do not read or write files, access the
  network, print, or use global variables.
- Do not use randomness unless it comes from {rng_argument}.
- The function is called many times in each run. Prefer vectorized NumPy operations and
  keep the time complexity within {complexity_bound}.
- Do not return the parent code unchanged, and do not reproduce a negative record.

Output format
Description: <one sentence on the modified rule and its expected effect>
```python
<complete implementation of the function, including imports>
```
```

Prompt 3: Instance generator. Called once per instance strategy; the returned generator is checked for validity before cross-evaluation.

```
[System]
You are an expert in benchmark design for combinatorial optimization. You write Python
generators that produce problem instances with a controllable structure. The goal is not
to make instances as hard as possible for every solver, but to create conditions under
which different solvers behave differently, so that their weaknesses can be located and
repaired. Every generated instance must be valid.

[User]
## Problem
{problem_description}

## Instance format
{instance_format}
Validity constraints: {validity_constraints}

## Parent generator
Id: {generator_id}. Mean gap across the current solvers: {parent_gap}%.
Descriptor summary of its instances: {parent_descriptors}
```python
{parent_generator_code}
```

Strategy
Target property: {target_property}
Descriptor: {descriptor}
Direction: {direction}
Solver behavior to probe: {probe}

Current solvers
Summaries of the decision rules of the current heuristics:
{solver_rules}

Related evidence
Instance conditions validated to expose a weakness:
{validated_conditions}
Instance conditions that did not separate the solvers:
{negative_records}

Instructions
1. Express the target property through one or more explicit parameters in theta. The
   effect of each parameter on the descriptor should be predictable.
2. Use the shared factors z (random seed and problem size n) for all random sampling.
3. Compute the descriptors of every generated instance and return them with the instance.
4. Make sure that every instance is valid for all admissible values of z and theta.

Constraints
- Signature: generate(z: dict, theta: dict) -> tuple[dict, dict], where z = {"seed": int,
  "n": int} and every key of theta has a default value.
- Return (instance, descriptors). The instance follows the format above; descriptors maps
  each descriptor name to a float.
- Draw all random numbers from rng = np.random.default_rng(z["seed"]). For a fixed z,
  changing theta should change only the targeted property.
- The descriptor should change monotonically with the main parameter in theta, while the
  problem size and the scale of the instance stay fixed.
- Avoid degenerate instances, such as duplicate points or zero demands, and extreme
  values.
- Use only NumPy and the Python standard library. Generating one instance of size {n_max}
  must take less than {time_limit} seconds.

Output format
Description: <one sentence on the targeted property and how theta controls it>
Descriptor: <name and formula of the descriptor>
```python
<complete implementation of generate(z, theta) and its helper functions>
```
```

Prompt 4: Residual interaction miner. Called for every flagged solver–instance pair.

```
[System]
You are an expert in the analysis of heuristics for combinatorial optimization. You will
be shown a solver-instance pair on which the solver performs much worse than expected from
its average performance and from the average difficulty of the instance. Such a pair
points to a possible interaction between a decision rule of the heuristic and a property
of the instance. Your task is to formulate one precise, testable hypothesis about this
interaction. A hypothesis is not a finding: it will be tested in a controlled experiment
and may be rejected.

[User]
## Problem
{problem_description}

## Diagnostic setting
The diagnostic panel contains {n_heuristics} heuristics and {n_instances} instances. Y_hx
is the gap of heuristic h on instance x, averaged over {n_seeds} seeds. The interaction
residual is
    r_hx = Y_hx - mean_h - mean_x + mean_all,
where mean_h is the mean gap of h over all panel instances, mean_x is the mean gap of all
heuristics on x, and mean_all is the overall mean. A pair is flagged when r_hx is large
and positive and Y_hx exceeds the weakness threshold {tau_w}.

## Flagged pair
Heuristic h: {solver_id} (parent: {parent_id})
Summary: {solver_summary}
```python
{solver_code}
```
Instance x: {instance_id}, generated by {generator_id} with theta = {theta}
Descriptors of x: {descriptors}
Percentiles of these descriptors within the panel: {descriptor_percentiles}

Performance record
Y_hx = {pair_gap}   r_hx = {residual}
mean_h = {mean_h}   mean_x = {mean_x}   mean_all = {mean_all}
Row of h (gaps of h on the other panel instances): {row_gaps}
Column of x (gaps of the other heuristics on x): {column_gaps}

Related memory
Validated findings about similar rules or conditions: {validated_records}
Open candidate hypotheses: {candidate_records}
Negative records: {negative_records}

Instructions
1. Compare the row of h with the column of x. Which property distinguishes x from the
   instances on which h performs well? What do the heuristics that perform well on x do
   differently from h?
2. Find the decision rule in the code of h whose behavior depends on this property.
3. Explain step by step how the property leads the rule to poor decisions.
4. Propose a modification of the rule that should remove the failure.
5. Propose a controlled variation of the instance: starting from the original condition o,
   change only this property to obtain a varied condition v.
6. State what the controlled experiment should show if the hypothesis is correct.

Rules
- Do not explain the failure by the overall weakness of h or the overall difficulty of x;
  both effects are already removed from r_hx.
- Quote the lines of code that implement the rule.
- The property must be measured by a descriptor that can be computed for any instance.
- The variation v must change only this property. The shared factors z (seed and size)
  stay the same.
- If the hypothesis is correct, then under v (a) the excess gap of the parent over the
  reference increases (A > 0), (b) the modified heuristic improves on the parent (G > 0),
  and (c) this improvement is larger than under o (D > 0).
- Do not restate a validated finding as a new hypothesis. If a negative record covers the
  same rule and property, propose a different mechanism.
- If the data do not support any specific mechanism, return null.
```

```
## Output format
Return a JSON object (or null) and nothing else:
{
  "hypothesis": "<rule R of h fails when condition C holds, because M>",
  "rule": "<name of the decision rule>",
  "code_lines": "<quoted lines from the code of h>",
  "descriptor": {
    "name": "<descriptor name>",
    "definition": "<how to compute it>",
    "value_under_o": "<value or range under the original condition>",
    "target_under_v": "<target range under the varied condition>"
  },
  "mechanism": "<step-by-step explanation>",
  "solver_modification": "<change to the rule>",
  "instance_variation": "<change to theta, with everything else fixed>",
  "expected_outcome": {"A": "> 0", "G": "> 0", "D": "> 0"},
  "alternative_explanations": "<other explanations and why they are less likely>",
  "confidence": "low" | "medium" | "high"
}
```

Prompt 5: Solver-side modification for controlled validation. The returned function is the candidate h that is compared with its parent $p(h)$.

```
[System]
You are an expert in heuristic design for combinatorial optimization and an experienced
Python programmer. You implement a modification that was proposed to test a hypothesis
about a heuristic. The modified heuristic is compared with the original one on paired
instances, so the modification must be minimal and must correspond exactly to the
hypothesis. Respond only in the requested format.

[User]
## Problem
{problem_description}

## Function
Signature: {heuristic_signature}
Inputs: {input_spec}
Output: {output_spec}

## Parent heuristic
Id: {parent_id}
```python
{parent_code}
```

Hypothesis under test
{hypothesis}
Rule: {rule}
Relevant lines:
{code_lines}
Proposed modification: {solver_modification}

Instructions
1. Implement the proposed modification of the rule, and nothing else.
2. Do not add other improvements, even if you see opportunities for them. Additional
   changes would make the validation result impossible to interpret.
3. Keep all other lines of the parent identical, including names and constants.
4. Where the condition of the hypothesis does not hold, the modified heuristic should
   behave like the parent as far as possible.

Constraints
- Keep the function name, the arguments, and the return type unchanged.
- For every valid input, return an array of the documented shape with finite values only.
- Use only NumPy and the Python standard library. No file access, printing, or global
  state.
```

```
## Output format
Changed lines: <the lines that differ from the parent>
```python
<complete implementation of the modified function>
```
```

Prompt 6: Instance-side variation for controlled validation. The returned generator g_v defines the varied condition v; the original generator defines o.

```
[System]
You write instance generators for controlled experiments in combinatorial optimization.
Given the generator of an original condition and a hypothesis about one instance property,
you write a generator for a varied condition that changes only this property. A heuristic
and its modification are compared on paired instances from the two generators. Any
unintended difference between the two conditions would confound the comparison, so the
change must be as clean as possible.

[User]
## Problem
{problem_description}

## Instance format
{instance_format}
Validity constraints: {validity_constraints}

## Hypothesis under test
{hypothesis}
Targeted descriptor: {descriptor}
Value under the original condition o: {value_under_o}
Target under the varied condition v: {target_under_v}
Requested variation: {instance_variation}
Additional levels (optional): {levels}

## Original generator g_o
theta_o = {theta_o}
```python
{original_generator_code}
```

Control descriptors
These descriptors must stay within {tolerance} of their values under o:
{control_descriptors}

Instructions
1. Find the sampling steps of g_o that determine the targeted property.
2. Change only these steps, controlled by new parameters in theta_v, so that the targeted
   descriptor moves into the target range.
3. Keep all other steps identical. Draw random numbers in the same order as g_o, so that
   g_o and g_v produce matched instances for the same z.
4. If several levels are requested, use one parameter that moves the descriptor
   monotonically across the levels.
5. Check the targeted descriptor, the control descriptors, and validity on the seeds
   {check_seeds}.

Constraints
- Signature: generate_varied(z: dict, theta_v: dict) -> tuple[dict, dict]. The returned
  descriptors must include the targeted descriptor and all control descriptors.
- For at least {coverage} of the seeds, the targeted descriptor must lie in the target
  range and every control descriptor must stay within the tolerance.
- Every instance must satisfy the validity constraints. The problem size and scale must
  not change.
- Do not tune the variation to a particular heuristic. The generator is fixed before the
  validation run and is evaluated on fresh seeds.
```

```
## Output format
Description: <one sentence on what is varied and what is held fixed>
Check: <how the targeted and control descriptors are computed and compared>
```python
<complete implementation of generate_varied(z, theta_v) and its helpers>
```
```

I.4 Auxiliary Prompts

The following prompts build the initial populations and repair generated code that fails a legality or validity check.

Prompt 7: Initial heuristic generation, used to build the initial solver population.

```
[System]
You are an expert in heuristic design for combinatorial optimization and an experienced
Python programmer. You write a heuristic function that is plugged into a fixed solver.
Respond only in the requested format.

[User]
## Problem
{problem_description}

## Function to write
Signature: {heuristic_signature}
Inputs: {input_spec}
Output: {output_spec}
Called by the solver backbone: {backbone_description}

## Reference heuristic
```python
{seed_code}
```

Ideas already in the population
{existing_ideas}

Instructions
Write a new heuristic whose main idea differs from the reference and from the ideas listed
above. Use problem knowledge, for example distances, demands, prizes, or capacities, to
estimate how promising each decision is. Keep the function short and efficient.

Constraints
- Keep the signature unchanged, and return an array of the documented shape with finite
  values only.
- Use only NumPy and the Python standard library.

Output format
Description: <one sentence describing the main idea>
```python
<complete implementation of the function>
```
```

Prompt 8: Initial instance generator, used to build the initial instance population.

```
[System]
You are an expert in benchmark design for combinatorial optimization. You write Python
generators of valid problem instances with a controllable structure. Respond only in the
requested format.

[User]
## Problem
{problem_description}

## Instance format
{instance_format}
Validity constraints: {validity_constraints}
```

```
## Standard generator
```python
{standard_generator_code}
```

Properties already in the population
{existing_properties}

Instructions
Write a generator whose instances differ from the standard distribution in one structural
property that is not listed above, for example spatial clustering, correlations between
attributes, or the tightness of the constraints. Expose the property through parameters in
theta, and report a descriptor that measures it.

Constraints
- Signature: generate(z: dict, theta: dict) -> tuple[dict, dict], with z = {"seed": int,
  "n": int}.
- Draw all random numbers from rng = np.random.default_rng(z["seed"]).
- Every instance must satisfy the validity constraints.

Output format
Description: <one sentence on the structural property and how theta controls it>
Descriptor: <name and formula of the descriptor>
```python
<complete implementation of generate(z, theta) and its helper functions>
```
```

Prompt 9: Code repair, called when generated code fails a legality or validity check.

```
[System]
You fix Python code that failed an automatic check. Make the smallest change that fixes
the error, and do not change the idea of the code. Respond only in the requested format.

[User]
## Task
The following {code_kind} failed the {check_name} check.

## Code
```python
{failed_code}
```

Error
{error_message}
Input that triggered the error: {failing_input}

Requirements
{requirements}

Instructions
1. Identify the cause of the error.
2. Fix it with the smallest possible change; keep the signature and the algorithmic idea.
3. Make sure that the fixed code satisfies all requirements above.

Output format
Cause: <one sentence>
```python
<complete fixed code>
```
```

J Additional Experimental Analysis

J.1 Hardness of IACE-Generated Instance Distributions

We next ask whether the hard distributions produced by IACE remain difficult when the underlying problem changes. To this end, we run an ACO-based solver on both the standard and the IACE-generated distributions of TSP, OP, CVRP, MKP, and BPP. Figure 4 plots the resulting objective values for problem sizes between 50 and 400.

In every domain, the solver performs worse on the generated instances than on the standard ones. On TSP, CVRP, and BPP, which are minimization problems, the objective obtained on the generated instances lies above the objective obtained on the standard instances at every tested size. On OP and MKP, which are maximization problems, it lies below. The sign of the effect therefore follows the sense of each objective, but in all five cases the generated instances lower the quality of the solutions returned by ACO.

The separation between the two distributions also grows with problem size. On TSP, the two curves are close at the smallest size and far apart at the largest. CVRP follows the same pattern, and the difference keeps increasing as more customers are added. On BPP, the generated instances require more bins than the standard ones at every size, and this difference becomes larger as the number of items increases. The effect is therefore not confined to small instances, where a generator could rely on a few unusual configurations.

The maximization problems behave in much the same way. On OP, the objective under the standard distribution rises quickly with the number of nodes, while the objective under the generated distribution stays far lower and rises only slightly. On MKP, the standard objective also increases steadily with instance size, whereas the objective on the generated instances hardly changes over the same range. In both problems, the curves for the two distributions move further apart as the size increases.

Taken together, the degradation caused by the generated instances does not depend on the problem type or on whether the objective is minimized or maximized. It appears in all five domains and is easier to see at larger sizes, which suggests that IACE can generate challenging instances for problems with quite different structures.

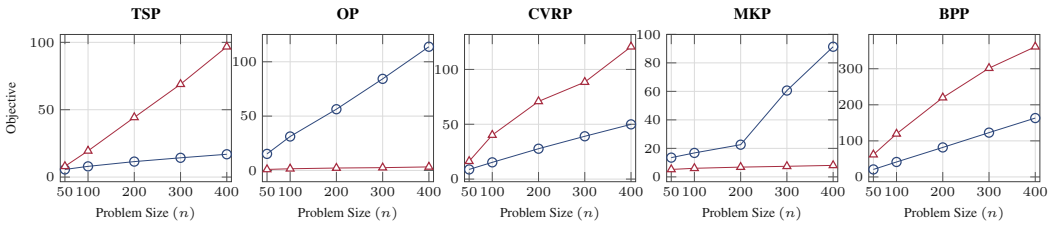

Figure 4: Solution quality of an ACO-based solver under standard and IACE-generated hard instance distributions across five combinatorial optimization problems. Each panel reports the solver objective as a function of problem size n. For the minimization problems (TSP, CVRP, and BPP), higher objective values indicate harder instances, whereas for the maximization problems (OP and MKP), lower objective values indicate harder instances. IACE-generated instances consistently produce less favorable solver outcomes across all five domains, with the separation from the standard distributions becoming more pronounced as the problem size increases.

J.2 Performance under Standard and Hard Instance Distributions

We further evaluate IACE under both standard and hard TSP instance distributions across six problem sizes ranging from 500 to 1000 nodes. The complete results are reported in Table 7. Across all 12 settings, IACE consistently achieves the lowest optimality gap among the evaluated methods.

Under the standard instance distribution, the differences among the strongest methods are relatively small, while IACE consistently outperforms the best-performing baseline at every problem size. Specifically, the absolute improvements over the strongest baseline range from 0.14 to 0.22 percentage points across the six problem sizes. This indicates that the improvement of IACE is maintained as the problem size increases from 500 to 1000 nodes.

Under the hard instance distribution, the advantage of IACE becomes substantially larger in absolute terms. Compared with the strongest baseline at each problem size, IACE reduces the gap by 3.85, 3.73, 3.27, 2.89, 2.61, and 2.33 percentage points for TSP sizes 500, 600, 700, 800, 900, and 1000, respectively. These improvements correspond to relative gap reductions of 31.7%, 28.3%, 23.2%, 19.4%, 16.6%, and 13.9%, respectively.

The hard instance distribution nevertheless remains challenging for all methods, including IACE. As the problem size increases from 500 to 1000 nodes, the gap of IACE increases from 8.31% to 14.38%, while its relative improvement over the strongest baseline decreases from 31.7% to 13.9%. Thus, although IACE retains the lowest gap across all evaluated hard-instance sizes, its proportional advantage becomes smaller as the problem size increases.

Table 7: Gap (%) under standard and hard instance distributions for TSP. Lower is better. The best result within each instance distribution and problem size is shown in bold.

	Standard Instance Distribution						*Hard Instance Distribution*					
Size	**ReEvo**	**EoH**	**MCTS**	**LNS**	**KGLS**	**IACE**	**ReEvo**	**EoH**	**MCTS**	**LNS**	**KGLS**	**IACE**
500	1.03	2.14	1.06	1.32	1.11	**0.88**	12.28	15.06	12.57	14.82	12.16	**8.31**
600	1.25	2.56	1.24	1.23	1.25	**1.03**	13.64	15.43	13.19	15.21	13.47	**9.46**
700	1.38	2.91	1.38	1.35	1.35	**1.17**	14.09	16.17	14.62	15.88	14.31	**10.82**
800	1.52	3.13	1.54	1.52	1.50	**1.34**	15.27	16.82	15.08	17.63	14.86	**11.97**
900	1.62	3.30	1.61	1.64	1.60	**1.38**	16.03	18.46	15.77	18.09	16.24	**13.16**
1000	1.72	3.48	1.72	1.98	1.65	**1.51**	16.71	20.13	17.32	18.74	16.93	**14.38**

J.3 Generalization to TSPLIB Instances

We further evaluate IACE on TSPLIB instances to examine its performance beyond the synthetic instance distributions used in the main experiments. The selected instances cover a broad range of problem sizes, from fewer than 100 nodes to several thousand nodes. We compare IACE with ReEvo, EoH, MCTS, and LNS, and report the optimality gap relative to the known optimum. Detailed instance-level results are provided in Table 8.

IACE obtains an average optimality gap of 0.84%, compared with 1.07% for ReEvo, 2.09% for EoH, 1.11% for MCTS, and 1.42% for LNS. Across the evaluated instances, IACE achieves or matches the lowest gap in the large majority of cases, while the strongest baseline varies across individual instances. In particular, the performance difference is not restricted to a specific range of instance sizes. Similar relative behavior can be observed for both compact and substantially larger problem instances.

The results also show similar behavior across different problem scales. On many small and medium-sized instances, IACE reaches the known optimum or closely matches it. On larger instances, IACE remains competitive with the compared methods. For example, the optimality gaps obtained by IACE are 1.69% on `rl1323`, 1.80% on `pcb1173`, 1.62% on `pcb3038`, and 2.43% on `fnl4461`, all of which are lower than the corresponding results of the compared baselines. These results are consistent with the observations obtained under the synthetic distributions, despite the differences in instance structure and scale. They also suggest that the learned search strategy does not rely exclusively on the characteristics of the training distribution.

There are also instances for which IACE does not obtain the lowest optimality gap. Examples include `pcb442`, `dsj1000`, and `rl1304`. This variation is expected because the compared methods exhibit different strengths across individual problem instances. Overall, the TSPLIB results provide additional evidence that the performance observed on the synthetic standard and hard distributions is also maintained across a diverse set of real-world TSP instances.

Table 8: Optimality gap (%) on TSPLIB instances. Lower values indicate better solution quality. IACE (Ours) is boldfaced when it achieves the best or tied-best result.

Instance	ReEvo	EoH	MCTS	LNS	IACE (Ours)
a280	0.30%	2.06%	0.30%	0.34%	**0.30%**
ch150	0.04%	0.37%	0.24%	0.04%	**0.04%**
d198	0.31%	0.40%	0.63%	0.29%	**0.27%**
eil101	1.78%	2.59%	1.78%	2.08%	**1.78%**
eil76	1.18%	1.53%	1.18%	1.18%	**1.18%**
gil262	0.57%	0.59%	0.36%	0.48%	**0.30%**
ts225	0.00%	0.00%	0.00%	0.00%	**0.00%**
rat99	0.68%	0.68%	0.68%	0.68%	**0.68%**
rl1889	1.50%	4.08%	1.41%	4.08%	**1.32%**
u1817	2.34%	4.62%	2.34%	4.62%	**2.10%**
d1655	2.42%	5.79%	2.86%	5.79%	**2.38%**
bier127	0.01%	0.26%	0.01%	0.01%	**0.01%**
lin318	0.66%	1.46%	1.02%	1.09%	**0.32%**
eil51	0.67%	0.67%	0.67%	0.67%	**0.67%**
d493	0.93%	2.82%	1.35%	1.27%	**0.93%**
kroB100	0.00%	0.00%	0.00%	0.00%	**0.00%**
kroC100	0.01%	0.01%	0.01%	0.01%	**0.01%**
ch130	0.01%	0.01%	0.01%	0.70%	**0.01%**
pr299	0.21%	0.61%	0.58%	0.11%	0.32%
fl417	1.35%	0.80%	0.66%	0.77%	**0.62%**
d657	0.82%	2.85%	1.28%	1.02%	**0.56%**
kroA150	0.00%	0.00%	0.00%	0.00%	**0.00%**
fl1577	2.34%	5.03%	2.75%	5.03%	2.48%
u724	1.17%	2.85%	1.08%	1.13%	**0.83%**
pr264	1.01%	0.00%	1.01%	0.00%	**0.00%**
pr226	0.01%	0.10%	0.01%	0.06%	**0.00%**
pr439	1.22%	2.80%	2.39%	1.97%	**1.17%**
d1291	4.28%	6.53%	4.18%	2.54%	**2.34%**
kroA100	0.02%	0.02%	0.02%	0.02%	**0.02%**
kroA200	0.05%	0.25%	0.21%	0.62%	**0.00%**
kroB150	0.01%	0.08%	0.01%	0.01%	**0.01%**
kroB200	0.04%	0.23%	0.07%	0.44%	**0.01%**
kroD100	0.00%	0.00%	0.00%	0.00%	**0.00%**
kroE100	0.00%	0.00%	0.00%	0.00%	**0.00%**
lin105	0.03%	0.03%	0.03%	0.03%	**0.03%**
p654	1.62%	0.75%	1.60%	0.05%	0.09%

Table 8: Optimality gap (%) on TSPLIB instances, continued. Lower values indicate better solution quality. IACE (Ours) is boldfaced when it achieves the best or tied-best result.

Instance	ReEvo	EoH	MCTS	LNS	IACE (Ours)
pcb442	1.06%	1.15%	0.36%	0.96%	1.12%
pr107	0.00%	0.00%	0.00%	0.00%	**0.00%**
pr124	0.00%	0.00%	0.00%	0.00%	**0.00%**
pr136	0.24%	0.09%	0.11%	0.00%	**0.00%**
pr144	0.00%	0.00%	0.00%	0.00%	**0.00%**
pr152	0.19%	0.00%	0.00%	0.19%	**0.00%**
pr76	0.00%	0.00%	0.00%	0.00%	**0.00%**
rat195	0.86%	0.99%	0.85%	1.37%	**0.78%**
rat783	2.13%	4.48%	2.10%	2.18%	2.15%
rd100	0.01%	0.01%	0.01%	0.01%	**0.01%**
rd400	0.83%	2.23%	0.80%	0.82%	0.90%
rl1304	4.81%	6.52%	4.00%	2.40%	3.14%
rl1323	2.00%	4.35%	2.05%	1.93%	**1.69%**
st70	0.31%	0.31%	0.31%	0.31%	**0.31%**
tsp225	0.00%	1.39%	0.00%	0.00%	**0.00%**
u1060	1.34%	4.04%	1.83%	1.54%	**1.25%**
u1432	1.18%	4.84%	1.08%	3.02%	1.32%
u159	0.00%	0.00%	0.00%	0.00%	**0.00%**
u574	1.94%	2.85%	0.94%	1.38%	**0.53%**
vm1084	0.91%	3.64%	0.73%	1.74%	**0.64%**
vm1748	1.57%	4.33%	1.54%	4.33%	**1.36%**
dsj1000	1.97%	4.28%	1.72%	1.06%	1.58%
fl1400	3.07%	7.66%	4.08%	2.28%	**2.10%**
nrw1379	1.78%	3.82%	2.19%	2.99%	**1.45%**
pcb1173	2.15%	5.07%	2.19%	2.91%	**1.80%**
pr1002	1.34%	3.27%	1.36%	1.16%	**1.05%**
u2319	0.74%	2.34%	0.62%	2.34%	**0.53%**
linhp318	2.33%	3.22%	2.69%	2.77%	**2.18%**
fl3795	2.69%	4.38%	3.45%	4.38%	**2.57%**
fnl4461	3.22%	4.63%	3.07%	4.63%	**2.43%**
pcb3038	2.47%	4.13%	2.08%	4.13%	**1.62%**
d2103	0.95%	1.88%	0.95%	1.88%	1.37%
u2152	2.89%	4.60%	2.92%	4.60%	**2.58%**
rat575	1.36%	3.11%	1.59%	1.88%	**1.23%**
pr2392	2.39%	4.19%	2.45%	4.19%	**1.47%**

J.4 ACO Optimization Dynamics on Generated Hard Instances

To examine whether the difficulty of the generated instances persists under increased optimization effort, we track the optimization trajectories of Ant Colony Optimization (ACO) on both the standard and the IACE-generated hard instances. For each problem size, we record the best objective value found after 1, 100, 500, 1000, 2000, and 3000 evaluations. Figures 5–9 show the resulting convergence curves. Note that the direction of improvement depends on the problem: lower objective values are better for TSP, CVRP, and BPP, whereas higher values are better for OP and MKP.

Hard instances remain difficult after convergence. Across all five problem domains, increasing the ACO search budget consistently improves the objective on both the standard and hard distributions. Nevertheless, the gap between the two distributions remains substantial even after the optimization trajectories have largely saturated. For the minimization problems, the hard instances converge to consistently higher objective values than their standard counterparts. For example, at $n = 400$, the final TSP objective is 96.738 on hard instances compared with 16.922 on standard instances, while CVRP converges to 120.692 versus 49.906, and BPP to 360.962 versus 163.234. The same phenomenon appears in the opposite direction for the maximization problems: at $n = 400$, OP reaches only 3.247 on hard instances compared with 113.832 on standard instances, while MKP reaches 8.063 compared with 91.314. Thus, the observed hardness cannot be explained solely by an insufficient optimization budget: the separation remains pronounced after thousands of evaluations.

The generated instances do not merely slow down early optimization. A second observation is that the hard instances exhibit meaningful optimization progress rather than pathological or completely stagnant search behavior. In TSP, for example, the objective on the $n = 400$ hard instances decreases from 130.077 after the first evaluation to 108.245 after 100 evaluations, 101.003 after 500, and eventually 96.738 after 3000. Similarly, the $n = 400$ CVRP objective decreases from 134.718 to 120.692, while the BPP objective decreases from 386.137 to 360.962. For the maximization tasks, ACO also improves substantially on the hard instances: the $n = 400$ MKP objective increases from 5.084 to 8.063, and the corresponding OP objective increases from 2.819 to 3.247. These trajectories indicate that ACO is able to exploit additional search effort on the generated instances; however, this additional effort does not eliminate the performance gap relative to the standard distribution.

Most gains occur early, followed by stable saturation. The convergence profiles also reveal a consistent diminishing-return pattern. Across tasks, the largest improvements typically occur within the first few hundred evaluations, after which the curves gradually flatten. This behavior is particularly clear for OP, where both standard and hard instances approach their final values by approximately 500–1000 evaluations. TSP and CVRP show a more gradual reduction, especially for larger instances, while BPP exhibits comparatively modest late-stage improvement. MKP follows the analogous increasing trend, with larger hard instances continuing to improve up to roughly 1000–2000 evaluations before stabilizing. Consequently, extending the search budget from 2000 to 3000 evaluations generally produces only marginal additional gains.

The hardness gap becomes especially pronounced at larger scales. For several domains, the separation between standard and generated instances becomes more visible as the problem size increases. This effect is particularly strong for TSP, where the difference between the final ACO objectives on the two distributions widens considerably from the smallest to the largest instances. A similar scale-dependent separation is visible for the maximization problems, where the standard objective grows rapidly with instance size while the objective attained on the generated hard instances remains substantially lower. These results suggest that IACE does not merely produce isolated difficult instances at small sizes, but generates instances whose difficulty persists, and in several cases becomes increasingly apparent, as the problem scale grows.

Implication. Taken together, these convergence results offer a complementary view of instance hardness beyond a single fixed-budget measurement. The generated instances remain challenging throughout the optimization process, retain a clear performance separation after ACO has nearly converged, and preserve this separation across multiple problem sizes and problem families. This suggests that the hardness induced by IACE reflects persistent structural difficulty for the ACO solver, rather than a transient artifact of early-stage search or an arbitrarily chosen evaluation budget.

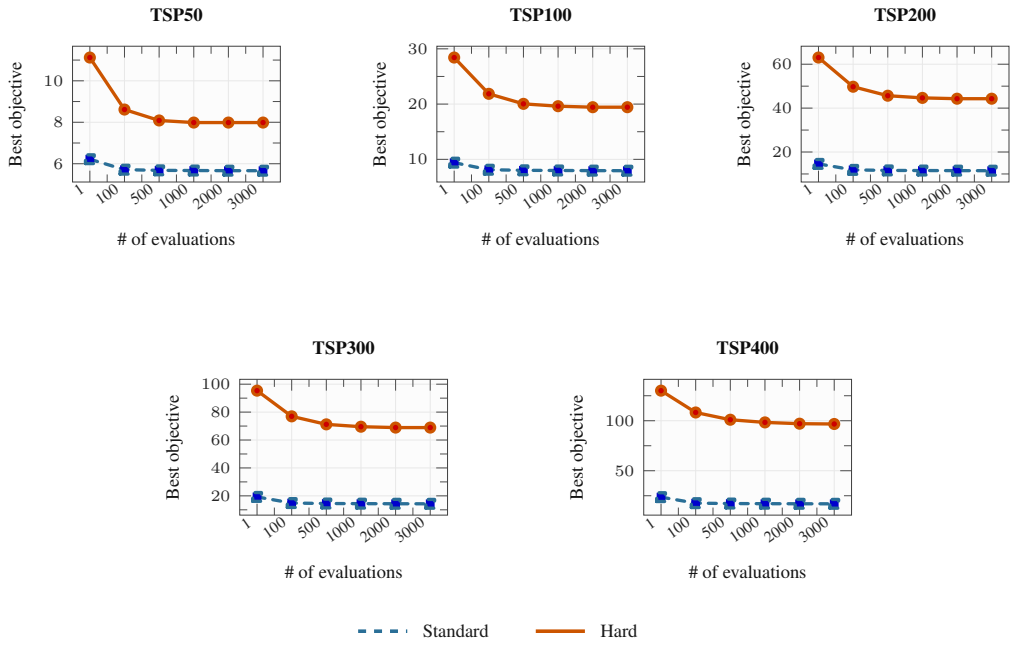

Figure 5: ACO evolution curves on standard and IACE-generated hard TSP instances. Lower objective values are better.

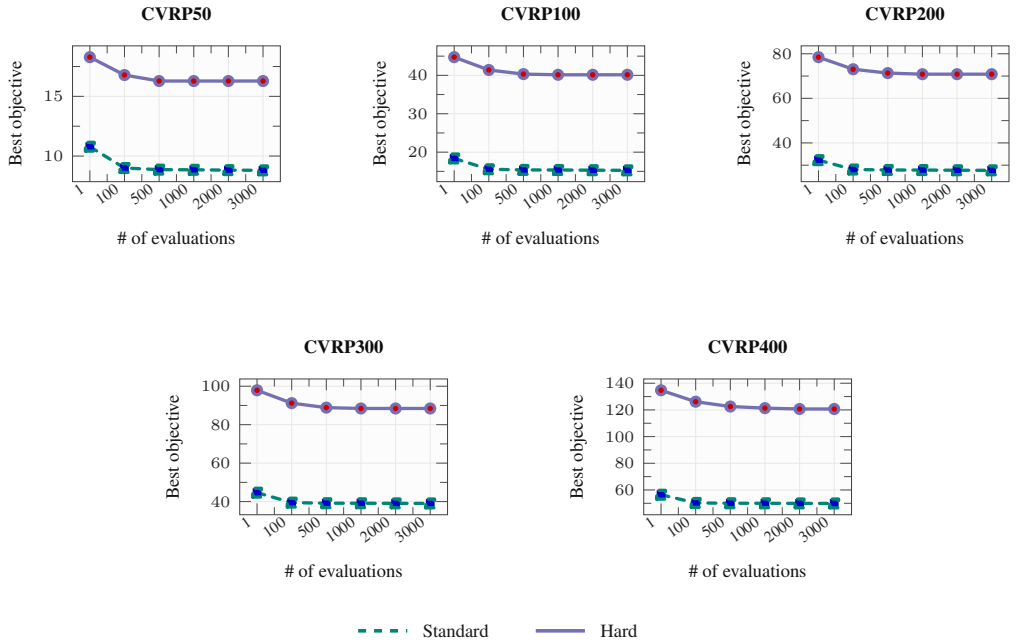

Figure 6: ACO evolution curves on standard and IACE-generated hard CVRP instances. Lower objective values are better.

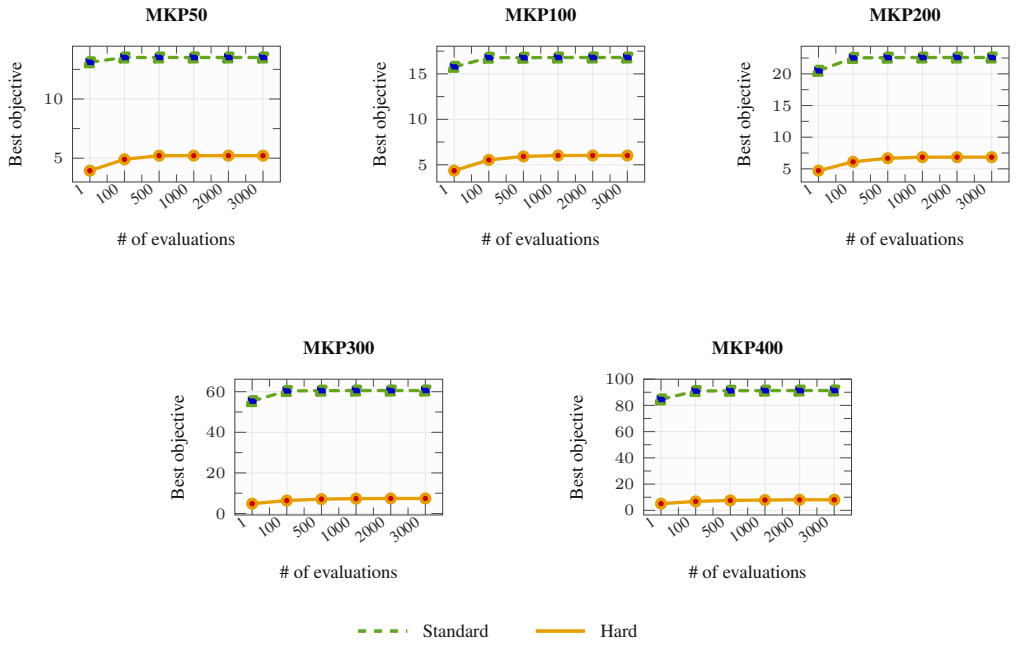

Figure 7: ACO evolution curves on standard and IACE-generated hard MKP instances. Higher objective values are better.

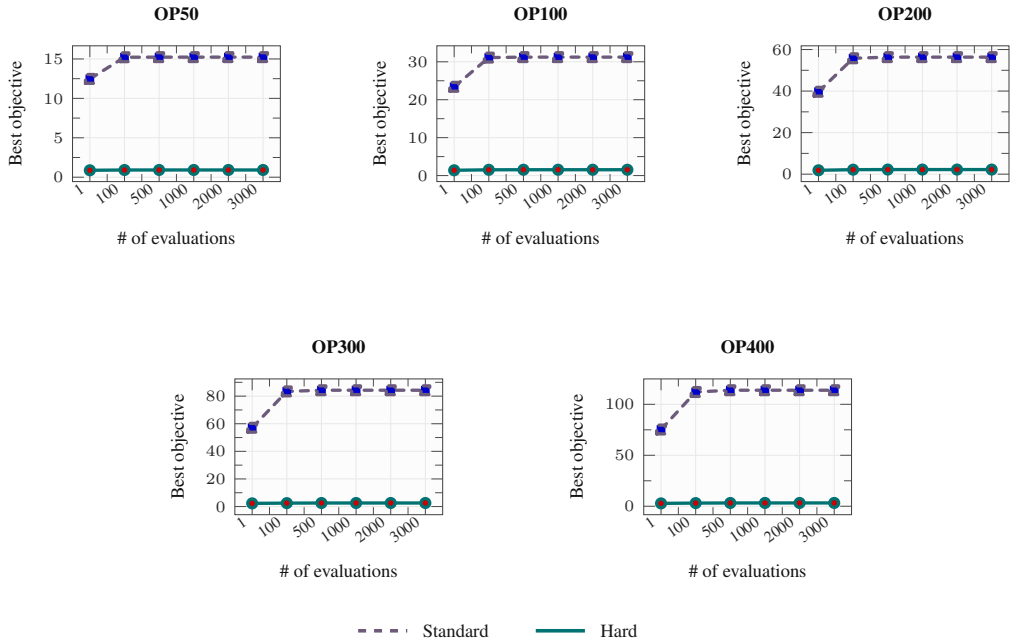

Figure 8: ACO evolution curves on standard and IACE-generated hard OP instances. Higher objective values are better.

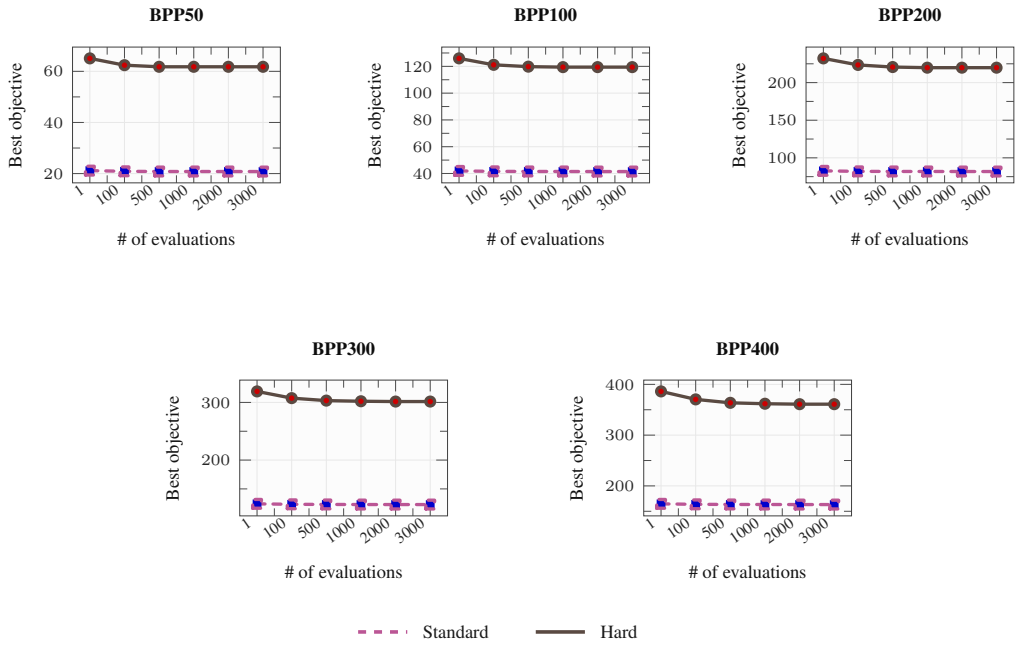

Figure 9: ACO evolution curves on standard and IACE-generated hard BPP instances. Lower objective values are better.

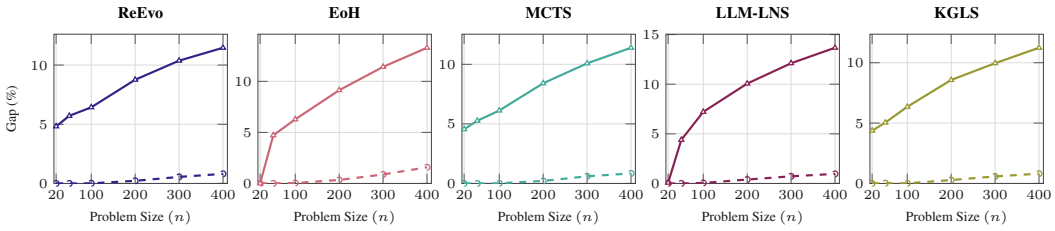

Figure 10: Performance gaps on standard and IACE-generated hard TSP instances. Dashed lines denote standard instances, while solid lines denote hard instances. Each color corresponds to one solver.

J.5 Hard Instance Generation across Different Solvers

We provide a detailed cross-solver evaluation of IACE-generated hard instances in Figure 10. Specifically, we evaluate ReEvo, EoH, MCTS, LLM-LNS, and KGLS on both standard and IACE-generated hard TSP instances across a wide range of problem sizes.

Across all solvers, optimality gaps remain small on the standard distribution and grow only mildly with problem size. In contrast, the IACE-generated hard distribution induces substantially larger gaps for every solver, and these gaps increase steadily as instances become larger. Even solvers that are near-optimal on standard instances exhibit clear degradation once evaluated on the hard distribution.

The separation between the standard and hard distributions becomes more pronounced as the problem size increases. For medium-sized instances, the hard distribution already yields gaps far exceeding the near-zero gaps observed on standard instances, and on the largest instances this difference widens further for every solver.

The consistent degradation across ReEvo, EoH, MCTS, LLM-LNS, and KGLS indicates that the generated instances are not challenging only for a particular optimization procedure. Instead, the difficulty transfers across multiple solver families, providing additional evidence that IACE captures broadly challenging instance characteristics.

K Experimental Details

This section provides additional implementation and evaluation details for the empirical study of IACE. Unless otherwise stated, the same co-evolutionary budget, population sizes, solver interfaces, and evaluation protocol are used across problem domains. Parameters that define the optimization backbone are fixed before evolution and are not modified by the LLM. The LLM is allowed to modify only the designated heuristic component and the executable instance generator exposed by the corresponding problem interface.

K.1 Computational Resources and Execution

All combinatorial-optimization experiments were executed on a server equipped with an AMD EPYC 7742 CPU. The computational workload of IACE consists primarily of repeated solver–instance evaluations, together with LLM-based program generation and the statistical processing required for interaction diagnosis and controlled validation.

The numerical processing performed by IACE includes construction of the cross-evaluation matrix, computation of heuristic and instance main effects, double centering of the performance matrix, extraction of interaction residuals, and computation of the matched validation contrasts. These operations are lightweight relative to repeated execution of the combinatorial optimization solvers, particularly for the larger problem sizes considered in the experiments.

For a given experimental configuration, the solver backbone, evaluation budget, random-seed protocol, and problem interface are held fixed throughout evolution. Thus, changes in solver performance or generated-instance hardness arise from changes in the evolving heuristic and instance-generator programs rather than from changes in the underlying optimization budget.

K.2 Common Co-Evolutionary Configuration

We use a solver population of

$$N_H = 10$$

and an instance-generator population of

$$N_G = 10$$

throughout the main experiments. Each co-evolutionary run contains

$$T = 20$$

iterations. The same population sizes and evolutionary horizon are used across TSP, OP, CVRP, MKP, and BPP unless an experiment explicitly varies the component under study.

Table 9: Common experimental settings used by IACE. These settings are fixed across problem domains unless an experiment explicitly states otherwise.

Setting	Value
CPU	AMD EPYC 7742
Solver population size N_H	10
Instance-generator population size N_G	10
Co-evolutionary iterations T	20
Diagnostic panel	10 heuristics $\times$ 10 instances
Solver backbone	fixed within each configuration
Evaluation budget	fixed within each configuration
Random-seed protocol	fixed within each configuration
Validation observations	fresh matched groups

The initial solver population is constructed from a seed heuristic, while the initial instance population is constructed from the standard generator of the corresponding problem. Additional initial population members are generated from these seeds subject to the same legality and validity checks used during subsequent iterations.

At each iteration, the generation pool produces candidate heuristic programs and candidate instance generators. The candidates are combined with the current populations for cross-evaluation, after which population selection retains the strongest solver candidates and the hardest valid instance generators for the next iteration.

The shared diagnostic panel contains 10 heuristics and 10 diagnostic instances. Therefore, each diagnostic matrix contains

$$10 \times 10$$

solver–instance cells before averaging over the prescribed evaluation seeds. The same matrix is used both for population-level performance statistics and for computing the double-centered interaction residuals described in the main method.

For clarity, the common experimental settings are summarized in Table 9.

K.3 LLM Generation and Program Constraints

IACE uses the LLM for semantic proposal generation and executable program modification. Numerical evaluation, population fitness, interaction residuals, validation contrasts, and statistical confidence calculations are performed outside the LLM.

At each iteration, the meta-strategy generator receives the current solver population, the current instance-generator population, previously identified interaction information, and retrieved memory records. It proposes solver-side and instance-side search strategies. A solver-side strategy is instantiated by modifying the designated heuristic component of a selected parent program, while an instance-side strategy is instantiated by modifying a parent generator.

Generated solver programs must preserve the required function signature and output format. Before a candidate is admitted to cross-evaluation, it is checked for successful execution, interface consistency, finite numerical outputs, and compatibility with the corresponding optimization backbone. Candidates that cannot be repaired into valid executable programs are discarded from the active candidate set.

Generated instance programs are checked separately. Each generator must produce an instance with the required problem size and satisfy the corresponding problem-domain constraints. For generators used in controlled validation, the required instance descriptors are also computed so that the intended condition modification can be checked before the corresponding validation result is admitted.

The LLM-backbone sensitivity experiment changes only the model used for semantic generation while retaining the remaining search and evaluation configuration. The evaluated backbones are GPT-5.6 Sol, Claude Opus 5, GPT-5.6 Terra, Claude Sonnet 5, DeepSeek-V4-Pro, and GPT-5.4 mini. Population sizes, co-evolutionary iterations, and the optimization-side evaluation protocol are unchanged in this comparison.

Table 10: Reference objectives used for evaluation. The reference value is shared across heuristics evaluated on the same instance and need not be globally optimal.

Problem	Objective sense	Reference objective $c_{\mathrm{ref}}(x)$
TSP	Min.	Known TSPLIB optimum when available; otherwise LKH-3, with an exact solver where tractable.
OP	Max.	Best solution found by a time-limited exact model.
CVRP	Min.	Strong CVRP heuristic evaluated with a larger computational budget.
BPP	Min.	Largest available proven lower bound on the bin count.
MKP	Max.	Best solution found by a time-limited exact model.

K.4 Evaluation Protocol and Reference Objectives

Each heuristic is evaluated through the same fixed optimization backbone for a given task. When the backbone is stochastic, the declared evaluation-seed protocol is shared across the heuristics being compared. The resulting observations are then aggregated to form the solver–instance performance matrix used for selection and interaction diagnosis.

For minimization problems, performance is measured relative to the reference objective through the normalized excess objective defined in the main method. For maximization problems, the orientation is reversed so that a larger gap consistently indicates worse solver performance across all five domains. This common orientation allows the same interaction residual and weakness-detection convention to be used without changing the direction of the statistical tests.

The reference objective $c_{\mathrm{ref}}(x)$ is obtained independently of the evolving candidate being evaluated and is kept fixed for that instance. The source of the reference differs across problems and is summarized in Table 10.

A reference value is used as a common external comparison scale rather than as an assumption of optimality. In particular, if a candidate achieves an objective better than a non-optimal reference, the resulting signed gap is retained rather than artificially truncated.

K.5 Interaction Diagnosis and Controlled Validation

The diagnostic and confirmatory stages use distinct evaluation roles. Cross-evaluation first produces the matrix from which heuristic effects, instance effects, and solver–instance interaction residuals are computed. A pair is considered diagnostically interesting when it exhibits both poor performance and a positive interaction residual beyond the specified screening criterion.

The diagnostic residual is used to determine where additional analysis should be concentrated. It is not itself treated as evidence that a particular explanation or solver modification is correct. Selected high-residual pairs are instead passed to the interaction pool, which proposes a candidate weakness, a minimal solver-side modification, and a corresponding controlled instance-side variation.

Before a controlled validation batch begins, the candidate heuristics, their parent heuristics, the original and modified conditions, the comparisons to be tested, and the associated statistical quantities are fixed. The validation stage then draws fresh matched groups. Within one matched group, the original and modified conditions share the declared latent factors, and all required parent–candidate evaluations are performed on the resulting instances.

The complete matched group is the independent sampling unit. Outcomes for different heuristics, conditions, and solver seeds within one group may therefore remain dependent. This dependence is preserved when the group-level statistics A, G, and D are constructed.

A candidate–condition pair is promoted only when it satisfies the three validation requirements defined in the main method: the modified condition must increase the parent's reference-adjusted difficulty, the candidate must improve on its parent under that condition, and the improvement under the target condition must exceed the corresponding improvement under the original condition. Comparisons that do not satisfy the required statistical criteria are not promoted as validated findings.

K.6 Problem Domains and Experimental Regimes

The experimental suite covers five combinatorial optimization problems: TSP, OP, CVRP, MKP, and BPP. These tasks were selected to cover different solution structures and different mechanisms by which an instance can expose a heuristic weakness. TSP, CVRP, and BPP are minimization problems, whereas OP and MKP are maximization problems.

The routing problems expose heuristic behavior through distance structure, route closure, depot dependence, reward accessibility, and capacity constraints. The packing and selection problems expose different interactions through item compatibility and the distribution of resource consumption across one or multiple capacity constraints.

Multi-domain hard-instance evaluation. To examine whether the generated hard distributions extend beyond a single optimization problem, an ACO-based solver is evaluated under both the standard and IACE-generated instance distributions for TSP, OP, CVRP, MKP, and BPP. The evaluated problem sizes are

$$n \in \{50, 100, 200, 300, 400\}.$$

The corresponding plots retain the native objective direction of each task: lower objective values are preferred for TSP, CVRP, and BPP, whereas higher values are preferred for OP and MKP.

Standard and hard TSP distributions. The large-scale TSP comparison evaluates the methods at

$$n \in \{500, 600, 700, 800, 900, 1000\}$$

under both the standard TSP distribution and the hard distribution generated by IACE. The same size grid and distribution definitions are used for all compared methods.

TSPLIB evaluation. Generalization beyond the synthetic distributions is evaluated on TSPLIB instances. The benchmark set covers a broad range of problem sizes, from instances with fewer than one hundred nodes to instances with several thousand nodes. For these instances, performance is reported relative to the corresponding known TSPLIB optimum.

Cross-solver hardness evaluation. To determine whether the generated hard TSP distribution is specific to one optimization mechanism, the same standard and hard distributions are evaluated with multiple solver families, including ReEvo, EoH, MCTS, LLM-LNS, and KGLS. This experiment evaluates whether the deterioration induced by the generated distribution is preserved when the solver itself is changed.

Optimization-budget sensitivity. The ACO convergence experiment evaluates whether additional search effort removes the difficulty of the generated instances. Solver performance is recorded at

$$1,\ 100,\ 500,\ 1000,\ 2000,\ \text{and } 3000$$

ACO iterations. The same evaluation checkpoints are used under the standard and hard distributions. This experiment distinguishes persistent structural hardness from poor performance caused only by an insufficient early search budget.

K.7 Baselines, Ablations, and Sensitivity Analysis

For the large-scale synthetic TSP comparison, IACE is compared with ReEvo, EoH, MCTS, LNS, and KGLS under the same standard and hard instance distributions. The TSPLIB evaluation compares IACE with ReEvo, EoH, MCTS, and LNS. The cross-solver hardness experiment evaluates the standard and generated-hard distributions using ReEvo, EoH, MCTS, LLM-LNS, and KGLS.

The ablation study evaluates the full IACE framework together with four variants:

w/o Meta-Strategy, w/o RIM, w/o Controlled Validation, w/o Memory.

Each ablation removes one major component while retaining the remaining co-evolutionary configuration. This isolates the roles of high-level search coordination, residual interaction mining, controlled validation, and evidence memory.

The LLM-backbone sensitivity experiment evaluates IACE with several different language models, including GPT-5.6 Sol, Claude Opus 5, GPT-5.6 Terra, Claude Sonnet 5, DeepSeek-V4-Pro, and GPT-5.4 mini. Only the LLM backbone is changed in this comparison, while the remaining search and evaluation settings are kept fixed. This isolates the sensitivity of IACE to the semantic generation component from changes in the underlying optimization configuration.

K.8 Repeated Runs and Reporting

The iterative co-evolution analysis reported in the main paper is aggregated over five independent runs. For the corresponding evolutionary curves, the central trajectory reports the mean across runs.

Where repeated-run tables are reported, results are summarized using mean $\pm$ standard deviation. For comparisons with an external reference objective, performance is reported using the corresponding normalized optimality gap. For the multi-domain hard-instance analysis, raw objective values are additionally reported because the purpose of that experiment is to compare solver behavior under the standard and generated instance distributions while preserving the native optimization direction of each problem.

Within each comparison, the problem-size grid, solver-side evaluation configuration, and distribution definition are held fixed across the methods being compared. The same generated hard distribution is used when evaluating different solver families in the cross-solver hardness experiment.

L Discussion of Design Choices

We discuss several design choices that affect how interaction signals are interpreted and how evidence is retained during co-evolution.

Interaction residuals rather than raw performance. Raw performance does not distinguish a weak heuristic from a difficult instance or from a specific mismatch between the two. IACE removes the marginal heuristic and instance effects before screening solver–instance pairs. The residual is thus used only as a relative interaction signal. A large residual indicates that a heuristic performs worse on an instance than would be expected from their respective marginal effects; it does not imply that the heuristic is globally poor or that the instance is globally hard. This distinction matters for targeted modification, since uniformly weak heuristics and uniformly difficult instances do not by themselves identify a condition-specific weakness.

Matched validation of a proposed repair. Improvement of a child heuristic is not sufficient to support the explanation that motivated the change. The same modification may improve performance on many instances, including instances that do not exhibit the diagnosed condition. IACE therefore evaluates the parent and child on both the original and modified conditions. The parent–child comparison measures the effect of the heuristic modification, while the difference between conditions measures whether that effect is stronger under the targeted condition. The four outcomes are evaluated within the same matched group so that the condition-specific contrast is formed directly from paired observations.

Evidence states in memory. The interaction miner produces hypotheses rather than verified explanations. For this reason, IACE does not store all generated explanations under a single status. Records that pass controlled validation are kept separately from unresolved hypotheses and unsuccessful attempts. Later generations may use validated records as supported evidence, while unresolved and negative records remain available as search history. In particular, a failed validation need not be discarded: it records that a proposed modification did not satisfy the specified validation criteria under the tested condition and can prevent the same direction from being repeatedly proposed.

Shared reference objectives. For a fixed instance, the reference objective is shared by all heuristics and is fixed before co-evolution. It is used to normalize performance rather than to participate in the search. Hence two heuristics evaluated on the same instance are compared against the same baseline. The reference value need not be a proven optimum; the requirement for the comparisons used in IACE is that it is fixed independently of the candidate heuristic. When a heuristic surpasses a nonoptimal reference, the resulting negative gap is retained rather than clipped.

Separation of screening and validation. Interaction diagnosis and controlled validation use related contrasts but have different roles. Residuals are computed on the cross-evaluation panel to screen for solver–instance pairs that merit further inspection. They are not treated as confirmation of a failure mechanism. Once a hypothesis and a candidate repair have been defined, fresh matched evaluations are used for validation. Information from earlier rounds may influence which hypothesis is tested next, but it is not reused as fresh validation evidence for the newly defined claim.

M Evolutionary Processes of Instance Generators and Heuristics

Figures 11–16 provide representative program-level trajectories produced during IACE co-evolution. Each figure places the successive hard-instance generators on the left and the corresponding heuristic functions on the right. The progression from V1 to V3 exposes how executable programs are progressively modified as the search accumulates information about solver–instance interactions. Rather than merely adjusting scalar parameters, later generations often introduce new structural patterns on the instance side and new structural signals on the heuristic side. The highlighted regions indicate the major program changes introduced across successive versions.

TSP–GLS. Figure 11 shows the evolutionary trajectory for TSP with Guided Local Search. The first hard-instance generator constructs normalized Euclidean distance matrices from independently sampled two-dimensional points, providing a relatively unstructured starting distribution. V2 reorganizes the points into multiple groups whose number grows with instance size and strongly contracts distances within the same group. This creates locally dense regions whose internal connections are much cheaper than transitions across regions. V3 further develops this structure through layered point generation, stronger within-layer contraction, and explicitly modified edges between selected layers, producing a more heterogeneous distance landscape.

The heuristic evolves from a simple row-relative distance measure in V1. V2 supplements this quantity with the normalized rank of each edge within its source row, allowing the guide to distinguish edges according to both relative magnitude and local ordering. V3 adds a short-edge backbone extracted from the lower tail of the distance distribution and a neighborhood-based risk term comparing an edge with nearby alternatives. Selected backbone edges are then protected from the ordinary guide penalty. Overall, the trajectory moves from generic distance normalization toward a guide that explicitly reasons about local ordering, persistent short-edge structure, and the relative risk of alternative transitions.

TSP–ACO. Figure 12 presents the corresponding process for TSP with Ant Colony Optimization. V1 samples dense symmetric edge-weight matrices from a narrow high-cost interval. V2 introduces a latent random route whose consecutive edges are assigned much lower costs and additionally inserts very-low-cost shortcut connections. The resulting instances contain attractive local edges whose relationship to the embedded route is more structured than in the initial random matrices. V3 adds a further partition of the latent route into alternating node groups, constructs a structured background over these groups, and introduces additional low-cost cross-group and skip connections.

The initial heuristic normalizes an edge by the aggregate cost of its destination column and then applies an inverse transformation. V2 adds two contextual signals: standardized deviation from the column-level cost distribution and normalized rank within that column. These quantities modulate the inverse-cost score so that edge quality is judged relative to competing alternatives. V3 additionally measures the contrast between an edge and the nearest available connection from the same source and combines this signal with standardized cost and rank. The resulting progression changes the heuristic from a purely relative inverse-distance rule into a context-sensitive score that evaluates how exceptional an edge is within its local competitive environment.

OP–ACO. Figure 13 illustrates the evolution for the Orienteering Problem with Ant Colony Optimization. The first generator constructs symmetric instances whose travel costs are drawn from a relatively high interval. V2 partitions the non-depot nodes into several groups, assigns a distinct within-group cost range, and inserts selected links between neighboring groups. V3 further couples the generated distance scale to the route budget, creates more tightly organized groups, introduces

a small collection of highly connected nodes, and explicitly varies the costs between the depot and the remaining nodes. The later generators therefore create route structures in which accessibility depends jointly on regional organization, depot position, and the available budget.

The V1 heuristic evaluates candidate moves mainly from destination prize and normalized travel cost. V2 introduces the cost of returning toward the depot and computes the remaining budget after accounting for both the current transition and this return requirement. The resulting feasibility factor suppresses moves that consume too much of the available route length. V3 further incorporates a destination-dependent isolation measure and an explicit depot-cost term, while transitions that cannot satisfy the route constraint are masked out. Hence the heuristic evolves from direct reward–cost scoring toward a rule that jointly considers reward, remaining budget, accessibility, and the structural position of a node within the instance.

MKP–ACO. Figure 15 shows the evolutionary process for the Multidimensional Knapsack Problem with Ant Colony Optimization. V1 samples item prizes and five-dimensional weight vectors from broad independent distributions. V2 introduces heterogeneous item roles: one subset receives moderate prizes together with relatively small and similar loads across dimensions, whereas another subset receives larger prizes but is assigned a substantially heavier load in one randomly selected resource dimension. The item order is then permuted so that these latent roles are not encoded by position. V3 extends this construction by generating several item categories with size-dependent perturbations and stronger dimension-specific load patterns, further increasing heterogeneity across the resource dimensions.

The first heuristic uses normalized prize divided by aggregate item weight. V2 augments this scalar efficiency measure with the maximum load across resource dimensions and applies a balancing factor before normalization. V3 explicitly measures the ratio between peak and total load, counts the number of dimensions whose weights exceed the item-specific mean, and combines these quantities with the base efficiency score. This progression therefore changes the heuristic from an aggregate prize–weight criterion into a multidimensional assessment of whether an apparently efficient item concentrates its resource consumption in only a few constrained dimensions.

CVRP–ACO. Figure 14 depicts the evolution for Capacitated Vehicle Routing with Ant Colony Optimization. The initial generator samples customer demands and coordinates from relatively broad distributions. V2 replaces this diffuse geometry with customers concentrated around four spatial centers and simultaneously shifts the demand distribution toward larger values. The generated instances therefore combine geographically compact clusters with stronger vehicle-capacity pressure. V3 preserves the clustered geometry but further modifies the demand assignments, tightens the spatial concentration, introduces a small number of customers away from the dominant clusters, and randomizes customer ordering. These changes produce more heterogeneous interactions between route geometry and load structure.

The V1 heuristic combines inverse travel distance with a factor derived from destination demand. V2 adds a depot-aware route factor that penalizes transitions according to both the current travel cost and the destination's distance from the depot. V3 incorporates additional local-neighborhood information and computes pairwise demand compatibility relative to vehicle capacity, while retaining a stronger depot-sensitive route term. As a result, later heuristic versions evaluate a transition not only by proximity and demand but also by whether the paired customers are compatible with the capacity constraint and how the transition interacts with the surrounding route geometry.

BPP–ACO. Figure 16 presents the evolution for Bin Packing with Ant Colony Optimization. The initial generator samples item sizes from a single broad interval under a fixed bin capacity. V2 divides the items into two roles: connector-sized items in a moderate range and larger items in a higher range, after which their positions are randomly permuted. V3 introduces a richer mixture that includes connector items, several semi-large size levels with local perturbations, and even larger items that are difficult to combine with many alternatives. The final clipping and permutation steps preserve feasibility while hiding these latent item roles from the ordering.

The first heuristic scores feasible item pairs primarily according to the residual capacity left after packing them together. V2 additionally uses the pair's fill ratio and the normalized difference between the two item sizes, thereby distinguishing pairs that leave similar residual space. V3 measures

how many feasible partners remain available for each item and converts this quantity into a rarity bonus. This term is combined with fill quality and complementarity, so items with relatively few future pairing opportunities can receive additional priority. The trajectory therefore moves from immediate bin utilization toward guidance that also accounts for the future flexibility of the remaining packing problem.

Taken together, these examples show that the two evolving program populations can acquire increasingly structured representations on both sides of the optimization process. Hard-instance generators introduce geometric organization, latent routes, regional connectivity, demand structure, resource asymmetry, and packing compatibility, while the corresponding heuristics develop ranking, local-comparison, feasibility, capacity, and scarcity-aware signals. The trajectories are intended to visualize the executable changes produced during co-evolution rather than to assign causal credit to individual code edits. Condition-specific statistical support is determined separately by the matched validation and evidence-control mechanisms described in the main method.

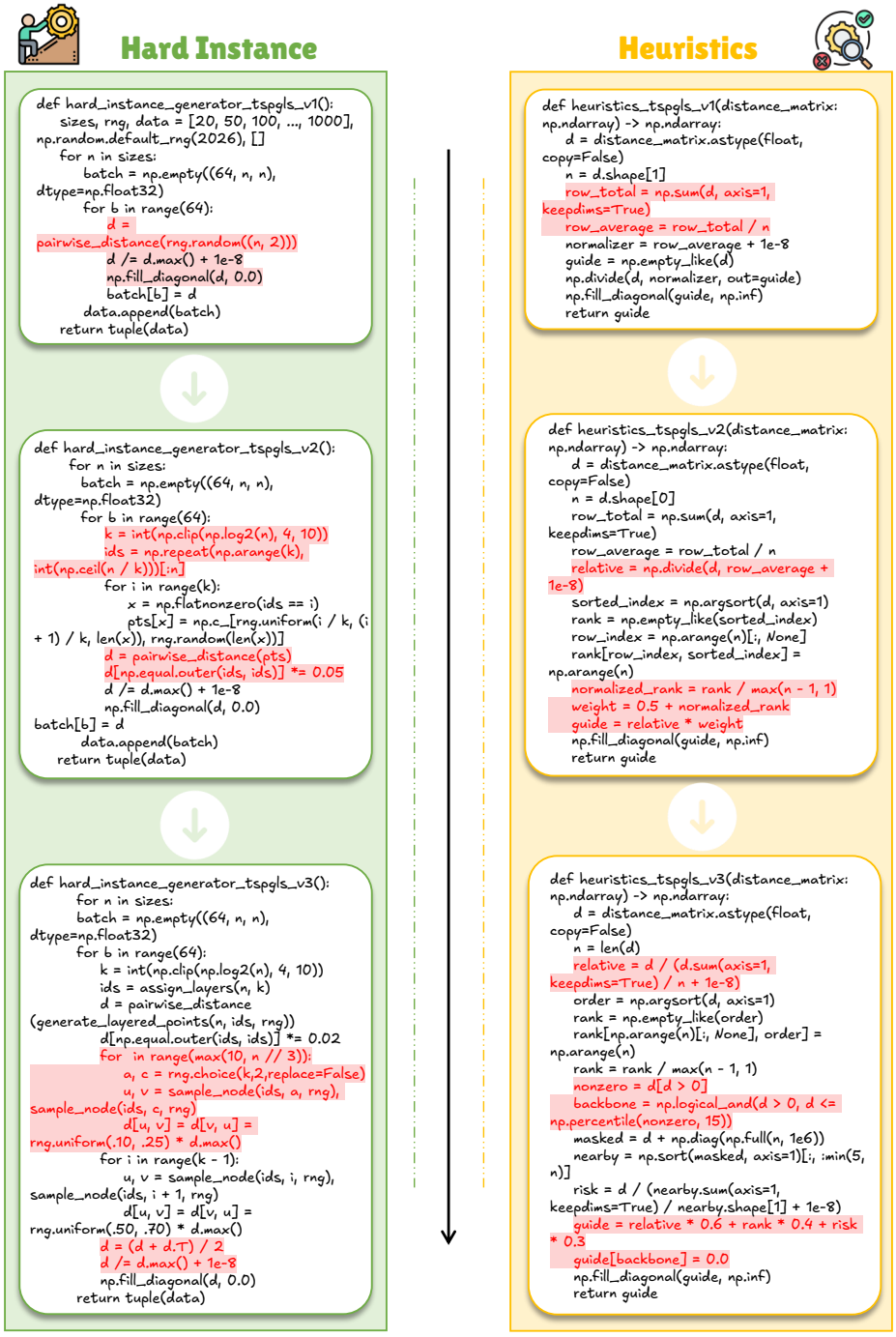

Figure 11: Evolutionary trajectory for TSP with Guided Local Search (TSP-GLS). The hard-instance generator evolves from generic Euclidean instances toward clustered and layered structures, while the heuristic progresses from row-normalized distance information toward rank-aware and local-structure-aware guidance. Highlighted code indicates the principal changes introduced in successive versions.

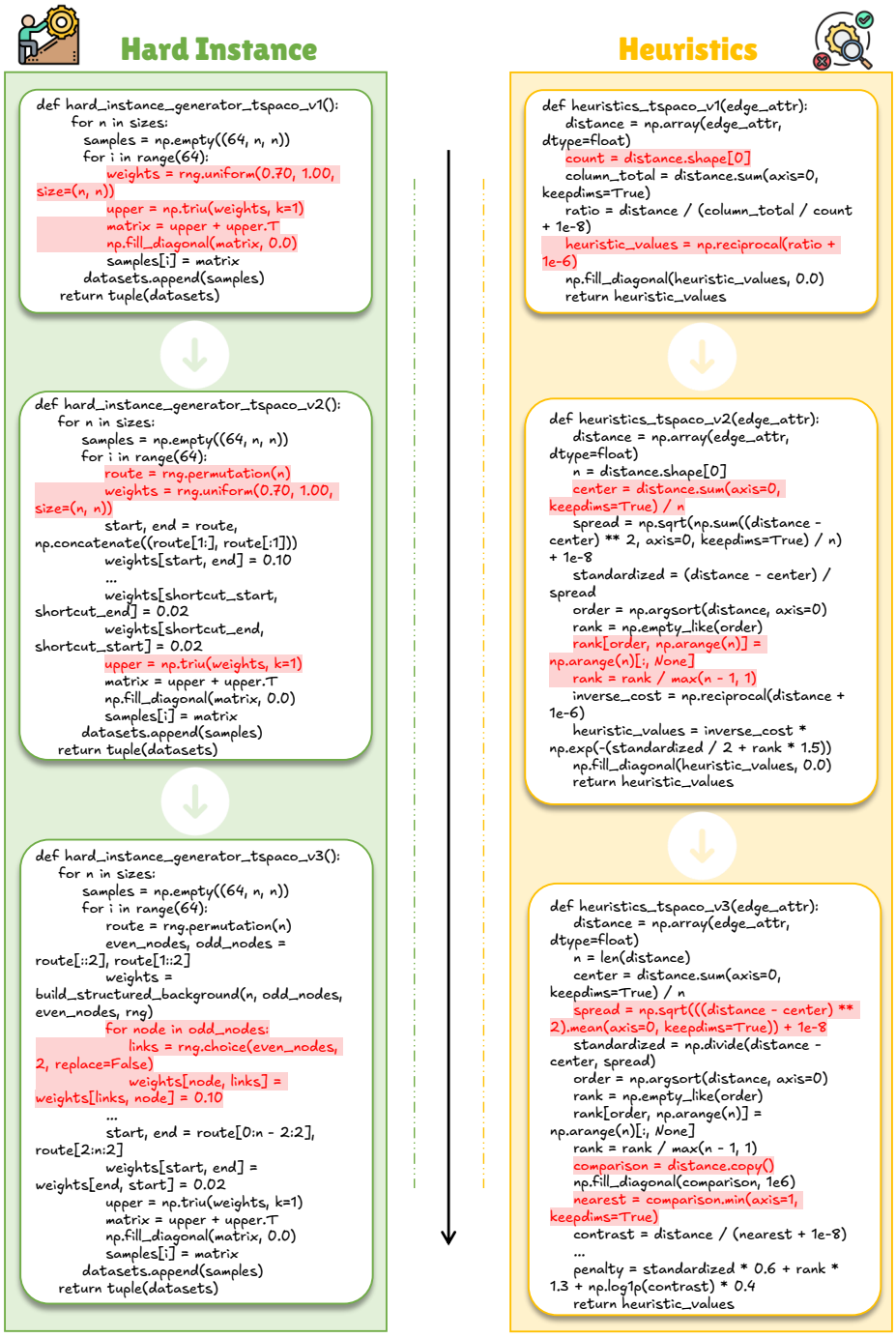

Figure 12: Evolutionary trajectory for TSP with Ant Colony Optimization (TSP-ACO). Later hard-instance generators introduce latent route structure, low-cost shortcuts, and structured connectivity, while the heuristic increasingly incorporates standardized distance, ranking, and local edge-contrast information.

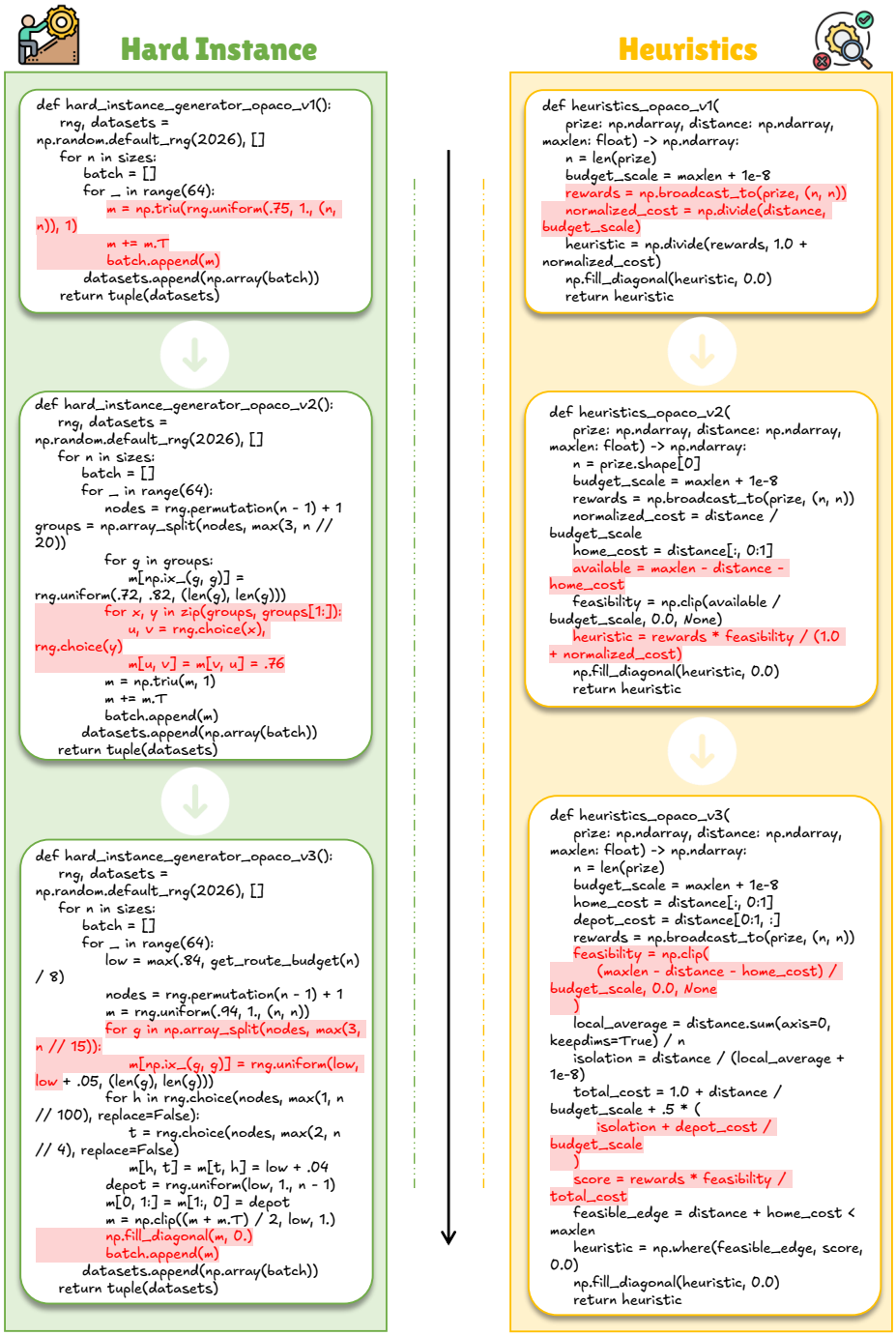

Figure 13: Evolutionary trajectory for the Orienteering Problem with Ant Colony Optimization (OP-ACO). The instance side develops grouped, hub-like, and depot-sensitive structures, while the heuristic side introduces remaining-budget feasibility, return-cost information, and local-isolation terms.

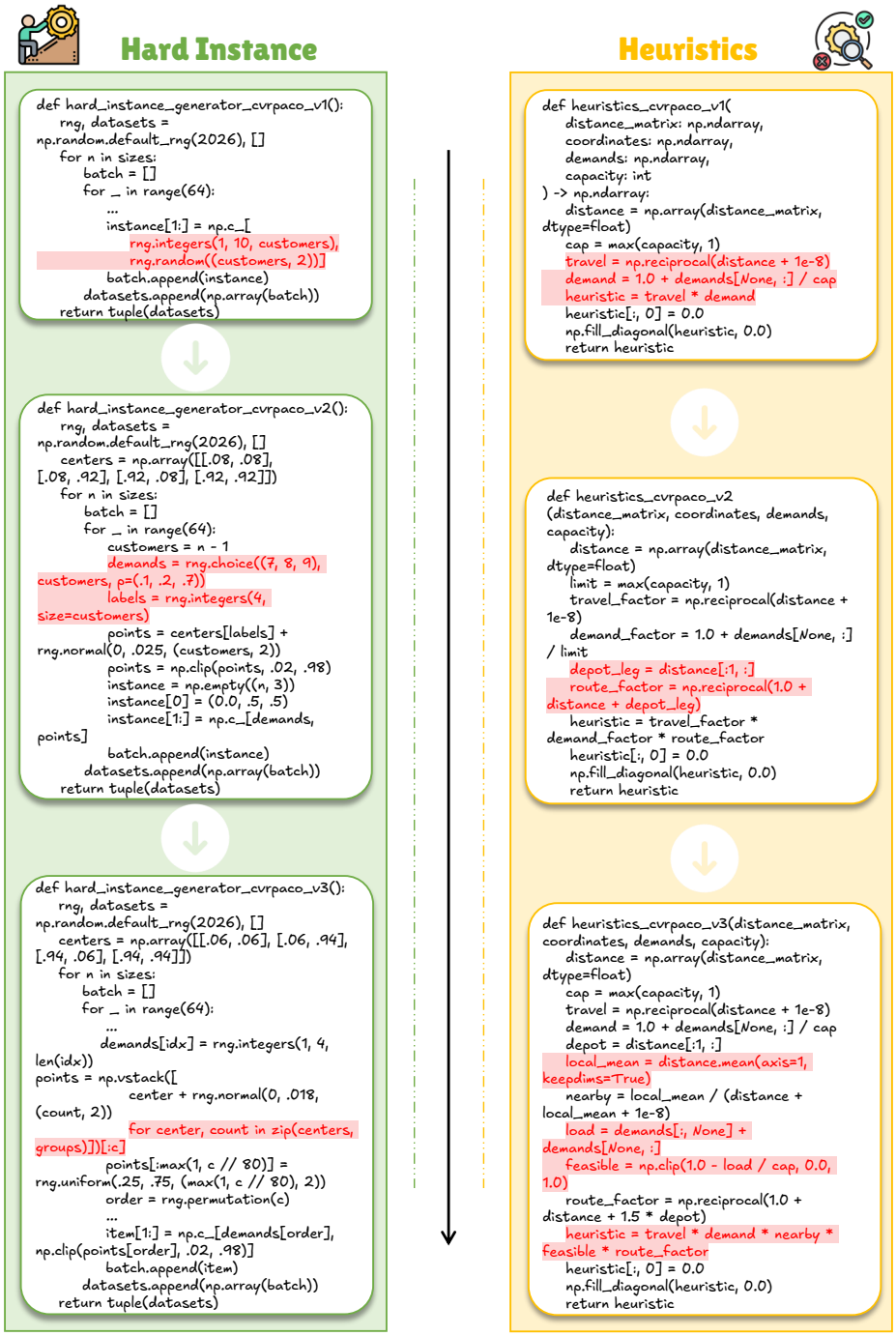

Figure 14: Evolutionary trajectory for Capacitated Vehicle Routing with Ant Colony Optimization (CVRP-ACO). The hard-instance side evolves toward clustered customer geometry and structured demand patterns, while the heuristic adds depot-aware routing, local-neighborhood information, and capacity-related feasibility.

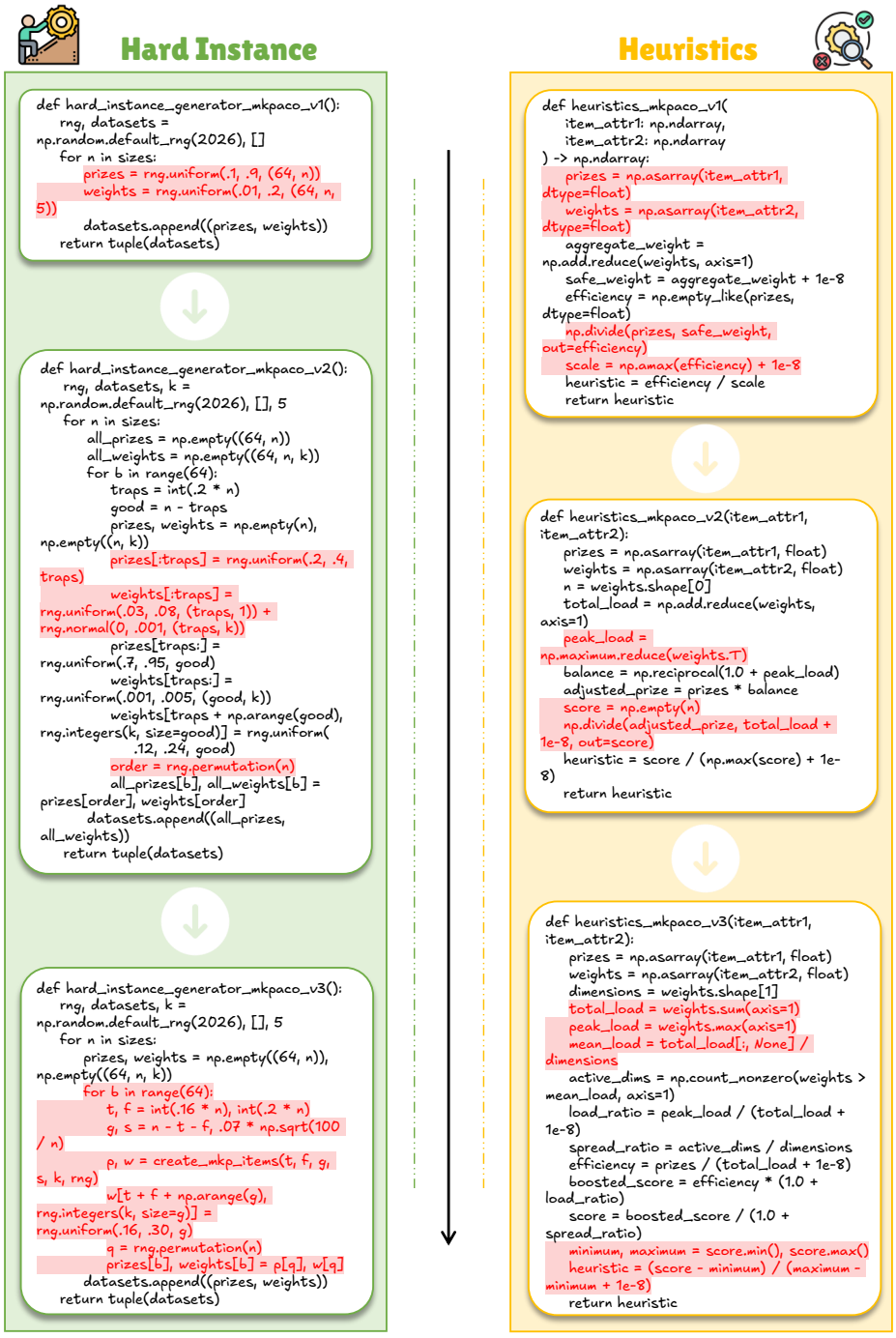

Figure 15: Evolutionary trajectory for the Multidimensional Knapsack Problem with Ant Colony Optimization (MKP-ACO). Instance evolution introduces increasingly heterogeneous and dimension-specific resource patterns, while heuristic evolution progresses from aggregate prize–weight efficiency toward peak-load and multidimensional load-distribution awareness.

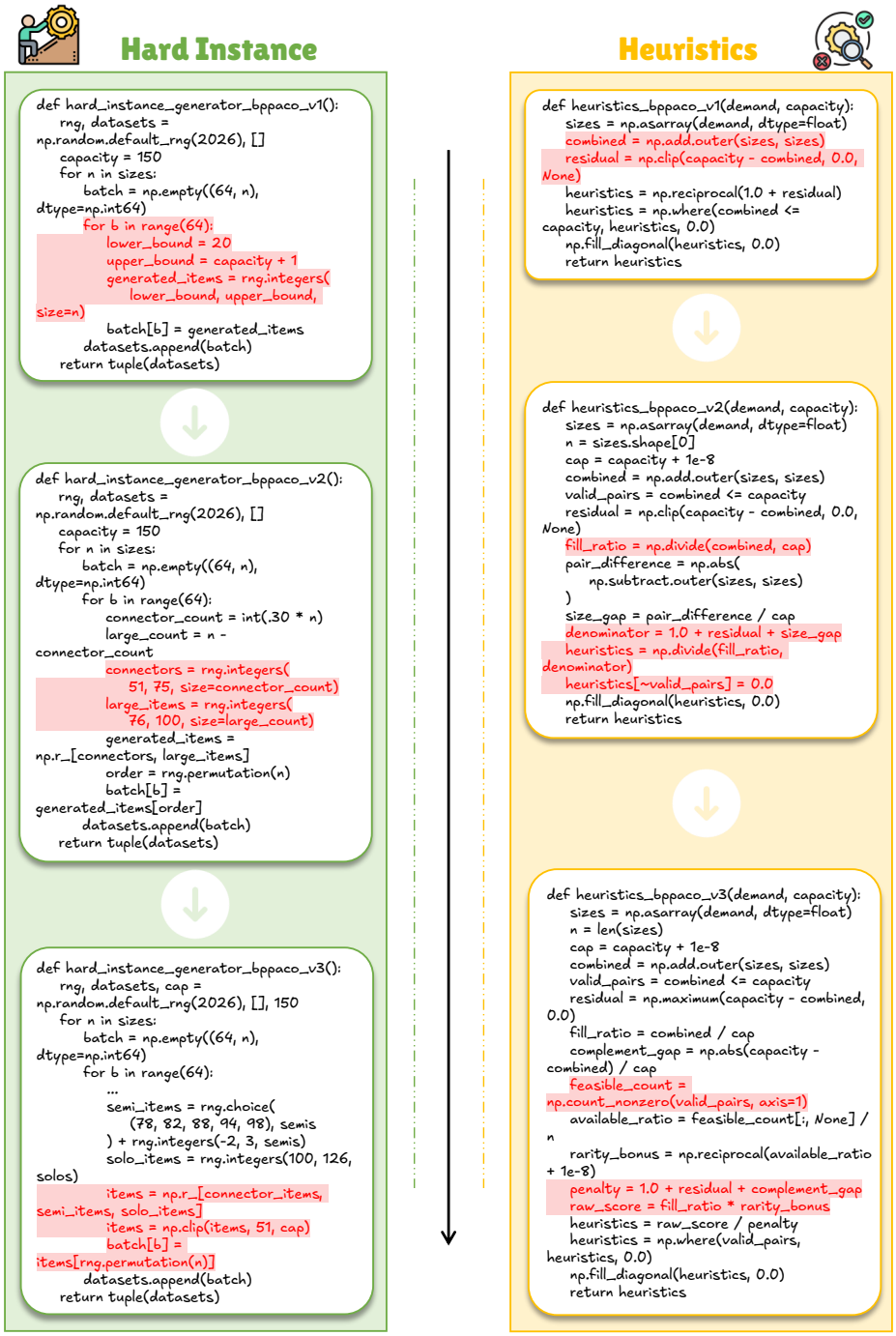

Figure 16: Evolutionary trajectory for the Bin Packing Problem with Ant Colony Optimization (BPP-ACO). Hard-instance generators evolve from broadly random item sizes toward heterogeneous connector, semi-large, and large-item roles, while the heuristic progressively incorporates fill quality, size complementarity, and feasible-partner scarcity.