ACEvo: Adversarial Co-Evolution of Problem Distributions and Solvers for Combinatorial Optimization

Ruibo Duan[1], **Yuxin Liu**[1], **Haoran Ye**[2], **Xinyao Dong**[1], **Zhiqiang Xu**[3], **Chenglin Fan**[1,†]
[1]Department of Computer Science and Engineering, Seoul National University
[2]School of Intelligence Science and Technology, Peking University
[3]Mohamed bin Zayed University of Artificial Intelligence (MBZUAI)

Abstract

Large language models (LLMs) are increasingly used to synthesize heuristic programs, yet most existing pipelines optimize solvers against fixed benchmark distributions. This static setup can obscure solver weaknesses and limit understanding of how LLM-designed algorithms adapt under distribution shift. We present Adversarial Co-Evolution(`ACEvo`), a closed-loop framework in which LLMs iteratively co-evolve two types of executable programs: heuristic solvers and problem generators. The generator proposes increasingly challenging instances, while the solver is refined to improve performance on the evolving distribution, forming an automated adversarial curriculum for program design and evaluation. We instantiate `ACEvo` on routing problems, including TSP, OP, and CVRP. Across these domains, the framework produces instance distributions that consistently induce larger optimality gaps than standard benchmarks and yields solver programs that outperform those obtained from static-training baselines under distribution shift. Beyond final performance, `ACEvo` provides a testbed for studying LLM-based algorithm design under evolving distributions, including how reflective mutation, adversarial feedback, and co-adaptation shape the evolution of both generators and solvers. These results suggest that closed-loop co-evolution is a promising paradigm for using language models not only to generate algorithms, but also to construct adaptive evaluation environments.

1 Introduction

Combinatorial optimization (CO) problems are central to Web-scale systems, underpinning tasks such as large-scale scheduling, content delivery, traffic routing, and personalized recommendation. The exponential growth of solution spaces and the heterogeneity of Web environments make the design of robust and scalable solvers a persistent challenge Garey & Johnson (1979); Gendreau & Potvin (2005). Traditional approaches rely on handcrafted heuristics, which are costly to design and difficult to adapt across diverse Web applications. Recently, large language models (LLMs) have demonstrated strong capabilities in code generation and complex reasoning, opening a promising direction for automating solver design tailored to Web-scale optimization Chen et al. (2021); Li et al. (2022).

Existing studies typically rely on fixed datasets or uniform instance generation protocols, resulting in limited diversity across problem settings. Moreover, most approaches focus on incremental enhancements to existing solvers rather than introducing fundamentally novel frameworks Ye et al. (2024); Bömer et al. (2025); Sartori & Blum (2025); Mu et al. (2025). This static evaluation paradigm restricts our ability to assess solver generalization and robustness. As a result, two critical challenges remain underexplored: (1) how to generate increasingly challenging problem instances, and (2) how to design solvers that can co-evolve alongside rising problem complexity. Addressing these challenges requires a fully

[†]Corresponding author: `fanchenglin@snu.ac.kr`

automated, mutually reinforcing framework that can simultaneously synthesize harder instances and adapt solver strategies in response.

To address these challenges, we introduce ACEvo (Adversarial Co-Evolution of Problem Distributions and Solvers), a novel framework that leverages large language models to co-evolve optimization problem instances and their corresponding heuristic solvers. ACEvo iteratively mutates instance-generation programs to produce increasingly complex and difficult problems, while concurrently synthesizing more adaptive solver code. This adversarial co-evolution loop enables the solver and problem to continuously challenge and improve each other.

Experimental results show that ACEvo can generate problem instances that are significantly more challenging than existing evaluation instance distributions, while simultaneously synthesizing heuristic solvers with strong performance and generalization capabilities—all achieved without manual intervention. This framework paves a new path for LLM-driven instance distribution generation and solver co-evolution in combinatorial optimization, enabling a dynamic evaluation paradigm that more faithfully assesses solver robustness and adaptability under increasing problem complexity. The proposed framework admits a more general formulation, as discussed in Appendix B.

We summarize our key contributions as follows:

- We introduce **ACEvo**, a novel framework that enables fully automated co-evolution of optimization instances and solver algorithms via large language models.
- We develop a mutation-based adversarial mechanism that adaptively modifies instance generation procedures to yield increasingly challenging optimization problems.
- We propose a structured generation process that leverages large language model to automatically synthesize adaptive, high-quality heuristic algorithms without human intervention.
- We empirically validate **ACEvo** across multiple combinatorial optimization tasks, demonstrating its broad effectiveness and robustness on classical TSPLIB Reinelt (1991) instances, and its capability to generate more challenging problem instances and synthesize solvers that achieve state-of-the-art performance.

2 Related Work

In this section, we summarize related works, including classical heuristics, instance distribution construction, adversarial optimization, and LLM-based methods.

Classical and Heuristic Methods for Combinatorial Optimization. Classical approaches to combinatorial optimization can be broadly categorized into exact algorithms and heuristic methods. Exact algorithms such as branch-and-bound Padberg & Rinaldi (1991) guarantee optimal solutions but suffer from poor scalability as problem size increases. To address this, heuristic approaches like greedy algorithms and local search are often employed to provide efficient, though suboptimal, solutions. Metaheuristics—such as simulated annealing Skiscim & Golden (1983) and ant colony optimization Dorigo & Gambardella (1997)—offer improved scalability on TSP and VRP Blum & Roli (2003); Liu et al. (2023a), while graph-based learning supports structured planning and correspondence tasks Ma et al. (2023); Zhang et al. (2024).

Instance Distribution Design and Difficulty Modeling. Beyond designing effective solvers, constructing instance distributions with high discriminatory power is critical for evaluating and comparing algorithm performance Smith-Miles & van Hemert (2011). While the widely-used benchmark TSPLIB Reinelt (1991) has been a standard in the literature Sun & Yang (2023); Luo et al. (2023); Li et al. (2023), it does not support instance difficulty adjustment based on solver capabilities. Other datasets generated by fixed-range random sampling often yield uniform or easy instances that lack diversity and fail to challenge solvers across a range of difficulty levels Smith-Miles & van Hemert (2011). On such datasets, solvers frequently converge to similar optima, making it difficult to assess their relative strengths Liu et al. (2024a); Yao et al. (2025); Romera-Paredes et al. (2024). Data-centric studies also emphasize long-tailed difficulty and sample value Xiao et al. (2026b;a), while larger instances involve complex structural patterns Applegate et al. (2011). In contrast, our pro-

posed ACEvo framework actively generates challenging problem instances that significantly amplify performance differences across solvers.

Adversarial Training in Combinatorial Optimization. Adversarial training, originally developed in the context of image generation via Generative Adversarial Networks (GANs) Goodfellow et al. (2014), has shown promise in synthesizing high-quality, diverse data Zhu et al. (2017); Ledig et al. (2017). Inspired by this success, recent research has applied adversarial ideas to combinatorial optimization Xin et al. (2022); Wang et al. (2023); He et al. (2021); Guo et al. (2020), primarily to improve solution quality or enhance solver generalization. These approaches typically focus on optimizing solutions via adversarial learning objectives. In contrast, our work introduces a co-evolutionary adversarial framework that simultaneously evolves both instance generators and solvers.

Large Language Models for Optimization. Recent advances have shown the potential of large language models (LLMs) in tackling a variety of optimization and structured prediction tasks by leveraging their strong generalization and program synthesis capabilities Zhao et al. (2025a); Jiang et al. (2024); Wu et al. (2023); Zhao et al. (2023); Iklassov et al. (2024); Zheng et al. (2024). Common strategies include few-shot prompting Brown et al. (2020), code completion Chen et al. (2021), and feedback-guided iterative refinement Liu et al. (2024a); Ye et al. (2024); Zelikman et al. (2022); Lin et al. (2026). Within combinatorial optimization, LLMs are primarily used as solvers Yang et al. (2024); Liu et al. (2024b); Zhao et al. (2025b); Tang et al. (2025), generating and refining candidate solutions through code synthesis. They are often integrated with evolutionary techniques such as genetic algorithms to iteratively improve solution quality Liu et al. (2023b; 2024a); Ye et al. (2024), demonstrating growing potential as hybrid learning and search agents.

3 Method

In this section, we present ACEvo, our evolutionary adversarial framework that jointly generates hard combinatorial optimization instances and synthesizes adaptive heuristic solvers. The method consists of four main components: (i) an LLM-based hard instance generator that mutates procedural programs to create increasingly challenging problems, (ii) an LLM-based heuristic generator that produces solver algorithms guided by symbolic prompts, (iii) an interaction protocol that formulates their competition as a symbolic adversarial game, and (iv) a co-evolutionary mechanism that couples solver improvement with instance difficulty escalation. The following subsections provide detailed explanations of these components and illustrate how ACEvo enables automated, feedback-driven co-evolution of instances and solvers.

3.1 LLM-Based Generators for Instances and Heuristics

ACEvo represents both instance generation and heuristic synthesis as LLM-generated executable programs. The following paragraphs formalize these two generators and the performance signals used to guide their evolution.

LLM-Based Hard Instance Generator. Let $\mathcal{G}$ denote the space of instance generators $g : \mathcal{Z} \to \mathcal{I}$, where $\mathcal{Z}$ is a latent seed space and $\mathcal{I}$ represents the space of problem instances (e.g., TSP instances). Each generator $g \in \mathcal{G}$ is an executable program synthesized by a language model $\mathcal{L}_{\text{inst}}$, conditioned on a natural language prompt $\mathcal{P}_{\text{inst}}$. Formally,

$$g \sim \mathcal{L}_{\text{inst}}(\mathcal{P}_{\text{inst}}), \quad I = g(z),\ z \sim \mathcal{Z}. \tag{1}$$

The resulting generator defines a programmatic distribution over instances through algorithmic logic, spatial sampling patterns, and geometric transformations.

To quantify instance difficulty in a normalized and scale-invariant manner, we measure the *relative optimality gap* with respect to a solver h:

$$\text{Gap}(I; h) = \frac{\mathbb{E}_{i=1}^{n}\left[f(h(I_i))\right]}{\mathbb{E}_{i=1}^{n}\left[f^*(I_i)\right]} - 1, \tag{2}$$

where $\{I_1, \ldots, I_n\}$ are instances sampled from the generator g, $f(h(I_i))$ denotes the heuristic solution cost, and $f^*(I_i)$ is the optimal or best-known reference cost. Larger gap values indicate greater instance hardness relative to solver h.

LLM-Based Heuristic Generator. Let $\mathcal{H}$ denote the space of heuristic solvers $h : \mathcal{I} \to \mathcal{S}$, where $\mathcal{S}$ is the solution space (e.g., feasible tours). Each heuristic $h \in \mathcal{H}$ is synthesized as an executable decision procedure by a language model $\mathcal{L}_{\text{heur}}$, conditioned on a natural language prompt $\mathcal{P}_{\text{heur}}$:

$$h \sim \mathcal{L}_{\text{heur}}(\mathcal{P}_{\text{heur}}). \tag{3}$$

The prompt encodes algorithmic priors such as greedy strategies, local refinement rules, or metaheuristic principles, together with task-specific constraints.

Once generated, the heuristic is evaluated behaviorally by applying it to instances $I \in \mathcal{I}$ and measuring its objective value $f(h(I))$. Importantly, heuristics are treated as black-box symbolic programs: no structural assumptions or manual interventions are imposed. Selection and refinement are driven solely by performance feedback, enabling the emergence of non-trivial and potentially unconventional algorithmic behaviors.

3.2 Co-Evolution Through Symbolic Adversarial Interaction

We formalize the learning process as a symbolic co-evolutionary game between an instance constructor $g \in \mathcal{G}$ and a heuristic solver $h \in \mathcal{H}$. This interaction is guided by a minimax formulation:

$$\min_{h \sim \mathcal{L}_{\text{heur}}} \max_{g \sim \mathcal{L}_{\text{inst}}} \mathbb{E}_{z \sim \mathcal{Z}} \left[\mathcal{H}(g(z); h) \right], \tag{4}$$

where $\mathcal{H}(g(z); h)$ denotes the relative hardness of the instance $I = g(z)$ with respect to solver h. This framework captures the adversarial dynamics between instance generation and solution strategies: the generator evolves to reveal vulnerabilities in current heuristics, while the solver adapts to counter increasingly complex instance structures.

Interaction Protocol. The co-evolution proceeds through iterative interaction in the algorithmic function space:

1. **Heuristic synthesis**: Sample a solver $h \sim \mathcal{L}_{\text{heur}}(\mathcal{P}_{\text{heur}})$ from the heuristic model via symbolic prompt programming;
2. **Hard instance constructor synthesis**: Sample an instance generator $g \sim \mathcal{L}_{\text{inst}}(\mathcal{P}_{\text{inst}})$, which defines a distribution over problem instances via structured algorithmic composition under adversarial evolutionary pressure throughout co-evolution;
3. **Evaluation**: Apply h to a batch of instances $\{I_i = g(z_i)\}_{i=1}^{n}$, compute the relative hardness $\mathcal{H}(I_i; h)$, and identify instances exhibiting maximal challenge;

This adversarial loop induces an emergent curriculum over the symbolic policy space: instance constructors synthesize increasingly diverse and non-trivial spatial structures, while solvers develop greater resilience and generalization capabilities. Notably, this process operates entirely without gradient-based optimization, relying instead on behavioral evaluation over executable algorithmic trajectories.

3.3 Architecture and Evolutionary Process of `ACEvo`

We propose `ACEvo`, a dual-loop adversarial co-evolution framework that simultaneously evolves hard combinatorial optimization instances and their corresponding solver heuristics. As illustrated in Figure 1, the framework integrates two tightly coupled processes instance evolution and solver evolution both orchestrated by large language models (LLMs) and driven by adversarial feedback. In each iteration, solvers are generated, evaluated, and refined through evolutionary reflection, while instance generators are mutated in parallel to exploit solver weaknesses and escalate difficulty (see Appendix D.7 for full pseudocode). This dynamic interaction forms a continual adversarial loop, where solvers progressively acquire stronger strategies and instances evolve into increasingly complex structures. Figure 1 provides an overview of this co-evolutionary cycle, highlighting how mutation, reflection, and selection jointly sustain the arms race between solvers and instances. A detailed analysis

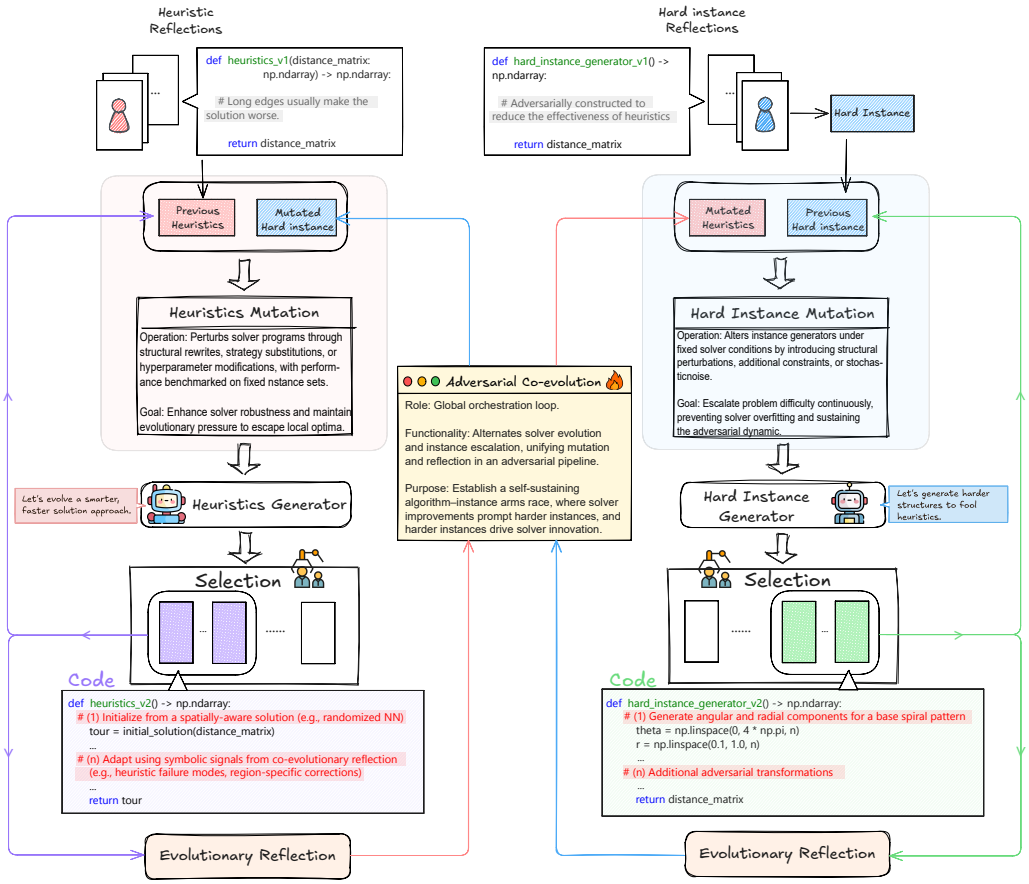

Figure 1: Illustration of the ACEvo framework and generated instance examples.

of the evolutionary process, including representative generator and heuristic variants across iterations, is provided in Appendix D.2.

Heuristics generator. The heuristics generator in ACEvo is an LLM-based component that synthesizes candidate solver programs as executable code fragments. These solvers are conditioned on problem descriptions and prior performance reflections, and are sampled from an open-ended algorithmic space without relying on predefined templates. This allows for the emergence of novel search strategies and scoring mechanisms throughout evolution.

Hard instance generator. The hard instance generator plays a symmetric role in ACEvo by producing instance-generation programs. Each generator defines a procedural method to construct challenging combinatorial problem instances, such as spatially clustered or structurally irregular TSP variants. These generators are optimized to exploit solver weaknesses, gradually shaping a landscape of increasing difficulty. This, in turn, causes solvers to exhibit significantly different performance on the generated hard instances, providing valuable guidance for subsequent optimization by highlighting robustness gaps and informing targeted improvement strategies.

Selection. ACEvo employs a selection mechanism to determine which solvers and instance generators proceed to the next evolutionary round. For solver evolution, candidates with superior performance on current hard instances are preferentially retained, while a small fraction of diverse solvers is preserved to maintain exploration. For instance evolution, generators that induce higher relative hardness are selected, subject to structural diversity constraints. This bidirectional selection sustains adversarial pressure and prevents premature convergence, enabling a continual escalation between solvers and instances.

Heuristics mutation. Given a fixed hard instance, ACEvo selects a previously generated solver and mutates it via the heuristics generator. Mutations may include algorithmic rewrites, strategy switching, or hyperparameter changes. The resulting solver is then evaluated on the current instance and compared to its predecessor to determine its relative effectiveness.

Hard instance mutation. Conversely, given a fixed solver, ACEvo mutates an existing instance generator using the hard instance generator. The objective is to synthesize instances that are more difficult for the solver, thereby increasing selection pressure and encouraging solver robustness.

Evolutionary reflection. ACEvo adopts evolutionary reflection, a symbolic diagnostic mechanism wherein LLMs evaluate behavioral shifts between original and mutated programs to generate high-level feedback—such as failure modes, inefficiencies, or refinement cues. These insights are embedded into future prompts, enabling implicit memory and long-term adaptation. Through this mechanism, ACEvo selectively produces mutated heuristics or hard instances based on task-specific objectives, allowing the co-evolutionary process to dynamically align with either solver optimization or instance difficulty escalation.

Adversarial co-evolution process. The two evolutionary paths in ACEvo are tightly coupled through adversarial interaction. As solvers improve, instance generators are pressured to synthesize harder instances; in turn, more difficult instances expose limitations in solver logic. This continuous feedback loop creates an arms race between the two components, driving the emergence of increasingly generalizable solvers and increasingly sophisticated instance distributions.

4 Experiments

We conduct experiments to evaluate the effectiveness of ACEvo across combinatorial optimization tasks and solver paradigms. Our goal is to show that ACEvo not only generates challenging instances that expose weaknesses of existing methods, but also evolves solvers that adapt to these harder distributions. We consider three representative routing problems: the Traveling Salesman Problem with Guided Local Search (TSP_GLS), the Traveling Salesman Problem with Ant Colony Optimization (TSP_ACO), and the Orienteering Problem with Ant Colony Optimization (OP_ACO), which together provide a broad testbed for assessing generality. Performance is mainly measured by relative optimality gap, with supplementary analyses including anytime curves, objective values, and instance visualizations. To ensure fairness, we compare against strong baselines such as FunSearch Romera-Paredes et al. (2024), EoH Liu et al. (2024a), LLM-LNS Ye et al. (2025), KGLS Arnold & Sörensen (2019) and ReEvo Ye et al. (2024) under both standard and ACEvo-generated datasets. Across all tasks, ACEvo consistently induces larger performance gaps for baselines while yielding solvers that close these gaps. Overall, the results confirm that ACEvo produces significantly harder instance distributions while synthesizing solvers with stronger adaptability and generalization. Statistical robustness and sensitivity analysis is provided in Appendix D.3.

4.1 Hardness Amplification and Solver Adaptation under ACEvo

Baseline vulnerability under ACEvo-generated instances. Figure 2 shows that ACEvo-generated instances significantly increase the performance gap across all TSP-GLS solvers and problem sizes. While FunSearch Romera-Paredes et al. (2024), EoH Liu et al. (2024a), ReEvo Ye et al. (2024), and KGLS Arnold & Sörensen (2019) perform well on standard data, their performance drops markedly on ACEvo instances, with the largest gap difference reaching 9%. These results underscore the limited generalization of existing solvers and validate ACEvo's ability to expose structural weaknesses through harder instance distributions.

Solver adaptation on hard instance distributions. Table 1 compares performance gaps across standard and ACEvo-generated hard TSP datasets, evaluated over a wide range of instance sizes (TSP400–TSP1000). All baseline solvers experience a sharp degradation in performance on the hard instances, confirming the increased difficulty induced by ACEvo's

generator. In particular, while the gap values exceed 9% for the strongest baselines, the ACEvo-generated solver consistently achieves lower gaps, demonstrating its ability to co-adapt to the evolving distribution. These results highlight the dual effectiveness of ACEvo in synthesizing both harder problem instances and stronger, distribution-aware solvers.

Table 1: Performance comparison on standard and proposed hard TSP datasets (TSP400–TSP1000) in terms of optimality gap (%). Lower is better. **ACEvo (ours)** consistently generalizes to harder instances while baseline performance degrades.

Method	TSP400	TSP500	TSP600	TSP700	TSP800	TSP900	TSP1000
Standard Dataset							
FunSearch (std)	**0.738**	**0.993**	1.236	1.629	1.994	2.126	2.408
EoH (std)	1.583	2.144	2.561	2.911	3.129	3.298	3.483
LLM-LNS (std)	0.959	1.318	1.233	1.351	1.523	1.641	1.981
KGLS (std)	0.854	1.105	1.248	1.349	1.500	1.599	**1.649**
ReEvo (std)	0.806	1.034	1.249	1.377	1.518	1.623	1.724
ACEvo (ours)†	0.859	1.060	**1.223**	**1.319**	**1.490**	**1.565**	1.656
Proposed Hard Instance Dataset (Ours)							
FunSearch (hard)	9.997	13.263	8.391	9.286	10.491	9.694	9.052
EoH (hard)	10.497	13.617	8.753	9.551	10.666	10.129	9.579
LLM-LNS (hard)	10.052	13.142	8.421	9.244	10.395	9.608	8.952
KGLS (hard)	7.293	10.440	6.809	7.735	9.017	8.546	7.791
ReEvo (hard)	7.305	10.435	6.890	7.718	9.042	8.525	7.851
ACEvo (ours)†	**5.484**	**8.081**	**5.732**	**6.981**	**8.212**	**8.185**	**7.672**

† Our proposed method.

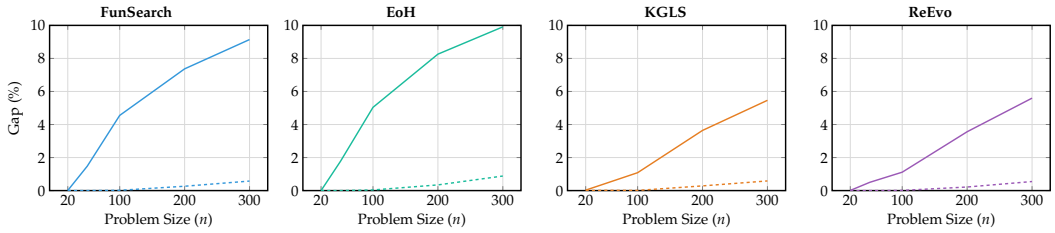

Figure 2: Gap (%) comparison across four methods on **standard (dashed)** and **hard (solid)** datasets. Lower values indicate better performance, while higher values reflect greater degradation under hard instance distributions.

Solver–instance synergy on the Orienteering Problem. Table 4 reports the performance of different heuristic–instance combinations for the Orienteering Problem solved via Ant Colony Optimization (OP_ACO). While heuristics generated by ReEvo Ye et al. (2024) remain reasonably effective on the hard instances constructed by ACEvo, the heuristics co-evolved within the ACEvo framework consistently outperform them across all evaluated problem sizes, indicating that jointly evolving heuristics and instances is more effective than relying on pre-trained or manually designed solvers. The heuristics synthesized by ACEvo require neither hand-crafted rule modifications nor hyperparameter tuning; instead, they are generated through symbolic interactions with large language models and refined via evolutionary reflection that integrates behavioral feedback into subsequent prompts. In contrast, static heuristics often generalize poorly under distribution shift, whereas the co-evolutionary dynamics of ACEvo enable solvers and instances to adapt to each other, allowing the automatic construction of challenging instance distributions and the discovery of robust optimization strategies for dynamic environments. Consistent solver–instance synergy is also observed for the Capacitated Vehicle Routing Problem solved via Ant Colony Optimization (CVRP_ACO); detailed results are reported in Appendix D.5.

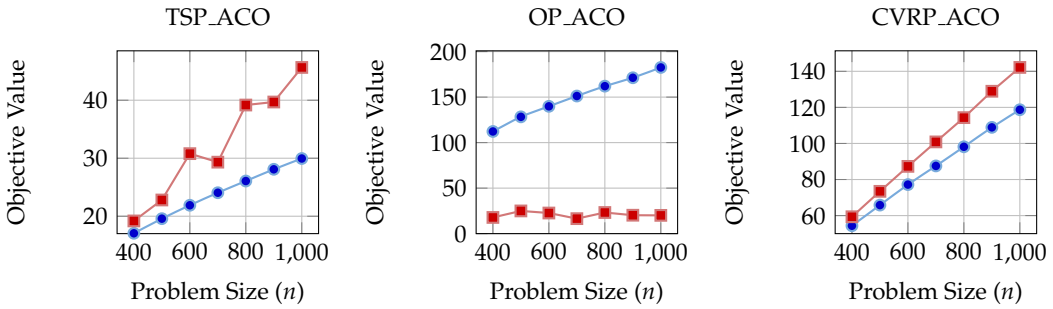

Figure 3: **Blue line:** performance on standard instances; **Red line:** performance on ACEvo-generated hard instances. Objective values using ReEvo heuristics on **TSP_ACO (left)**, **OP_ACO (middle)**, and **CVRP_ACO (right)**. In TSP and CVRP, higher objective values indicate worse solutions, whereas in OP, lower objective values indicate worse solutions. Across tasks, ACEvo-generated instances consistently induce more adverse outcomes, indicating increased optimization difficulty.

Instance-induced degradation under fixed solvers. To evaluate the intrinsic hardness of ACEvo-generated instances independently of solver adaptation, we assess the performance of ReEvo's Ye et al. (2024) heuristics on three representative combinatorial optimization tasks: the Traveling Salesman Problem solved via Ant Colony Optimization (TSP_ACO), the Orienteering Problem solved via Ant Colony Optimization (OP_ACO), and the Capacitated Vehicle Routing Problem solved via Ant Colony Optimization (CVRP_ACO). Figure 3 reports the objective values obtained on both standard and ACEvo-generated instance distributions across a range of problem sizes. In the TSP and CVRP settings, higher objective values correspond to worse solutions, whereas in OP, lower objective values indicate reduced reward collection. Across all three tasks, ACEvo-generated instances consistently lead to more unfavorable objective outcomes, demonstrating that the induced instance distributions effectively expose solver weaknesses under static, non-adaptive evaluation.

4.2 Generalization for the Traveling Salesman Problem

To further assess the generalization capability of ACEvo-generated solvers, we conduct evaluations on classical reference instances from the TSPLIB Reinelt (1991) dataset, a widely recognized benchmark for the Traveling Salesman Problem (TSP). These instances span diverse structural characteristics and problem scales, reflecting the complexity of real-world routing scenarios. By incorporating TSPLIB Reinelt (1991), we examine whether solvers evolved under ACEvo transfer effectively beyond the synthetic distributions used during training.

We compare ACEvo against three competitive LLM-based frameworks—FunSearch Romera-Paredes et al. (2024), ReEvo Ye et al. (2024), and LLM-LNS Ye et al. (2025). Across all evaluated TSPLIB instances with up to 2000 nodes, using Guided Local Search (GLS) as the underlying solver, ACEvo consistently achieves the lowest optimality gap, indicating superior adaptability and robustness.

These results suggest that the co-evolutionary process in ACEvo yields solvers that generalize beyond synthetic hard instances and remain effective on human-designed benchmarks. The consistent improvements over strong baselines further highlight the role of adversarial feedback in achieving robust generalization. The complete TSPLIB Reinelt (1991) results are reported in Table 2.

4.3 Ablation Study on Evolutionary Components

As detailed in Appendix D.6, we analyze the contribution of individual evolutionary operators in ACEvo by disabling adversarial feedback, reflection, or mutation, and evaluating the resulting solver gaps on hard instances. Removing *adversarial feedback* leads to the largest degradation in solver performance, with the solver gap increasing to 11.43% despite

Table 2: Comprehensive evaluation of ACEvo† and baseline frameworks on TSPLIB benchmark instances. Reported values indicate the optimality gap (%) relative to known optima. Bold numbers highlight results achieved by ACEvo, indicating cases where it attains or matches the best performance. † denotes our proposed method.

Instance	FunSearch	EoH	LLM–LNS	ReEvo	**ACEvo†**	**Instance**	FunSearch	EoH	LLM–LNS	ReEvo	**ACEvo†**	**Instance**	FunSearch	EoH	LLM–LNS	ReEvo	**ACEvo†**
kroE100	0.07	0.00	0.00	0.00	**0.00**	pr124	0.00	0.00	0.00	0.00	**0.00**	rd400	0.76	2.23	0.82	1.11	0.82
pr226	0.12	0.10	0.06	0.00	**0.00**	kroB150	0.03	0.08	0.01	0.00	**0.00**	u724	1.81	2.85	1.13	1.17	**1.02**
vm1748	3.88	4.33	4.33	1.57	**1.14**	ulysses22	0.12	0.00	0.00	0.00	**0.00**	kroB100	0.00	0.00	0.00	0.00	**0.00**
pr1002	1.89	3.27	1.16	0.75	**0.57**	bier127	0.12	0.26	0.01	0.16	**0.01**	kroD100	0.00	0.00	0.00	0.00	**0.00**
pr152	0.00	0.00	0.19	0.19	**0.00**	kroA100	0.02	0.02	0.02	0.02	**0.02**	u159	0.00	0.00	0.00	0.00	**0.00**
dsj1000	3.43	4.28	1.06	0.74	**0.73**	fl1577	4.79	5.03	5.03	2.34	**1.61**	st70	0.31	0.31	0.31	0.31	**0.31**
rat195	0.96	0.99	1.37	1.38	**0.96**	berlin52	0.03	0.03	0.03	0.03	**0.03**	kroA200	0.13	0.25	0.62	0.05	**0.05**
pr136	0.04	0.09	0.00	0.12	**0.00**	u1060	2.66	4.04	1.54	1.33	**1.14**	rat99	0.68	0.68	0.68	0.68	**0.68**
pr264	1.01	0.00	0.00	1.01	1.01	kroA150	0.01	0.00	0.00	0.00	**0.00**	lin318	0.98	1.46	1.09	0.70	**0.70**
d493	1.51	2.82	1.27	0.83	**0.72**	ch130	0.02	0.01	0.70	0.01	**0.01**	fl3795	6.55	4.38	4.38	2.69	**2.40**
pr76	0.00	0.00	0.00	0.00	**0.00**	pcb1173	3.91	5.07	2.91	1.13	**0.92**	u1432	4.37	4.84	3.02	1.18	**1.07**
a280	0.92	2.06	0.34	0.30	0.66	fl1400	4.88	7.66	2.28	3.07	**2.28**	ts225	0.00	0.00	0.00	0.00	**0.00**
kroB200	0.15	0.23	0.44	0.04	**0.04**	d198	0.46	0.40	0.29	0.31	0.40	u1817	4.78	4.62	4.62	2.34	**2.00**
nrw1379	3.48	3.82	2.99	1.79	**1.12**	d657	1.66	2.85	1.02	0.82	**0.72**	ch150	0.12	0.37	0.04	0.04	**0.04**
lin105	2.77	0.03	0.03	0.03	**0.03**	rl1889	4.86	4.08	4.08	1.50	**1.20**	d1655	5.04	5.79	5.79	2.42	**2.31**
tsp225	0.63	1.39	0.00	0.00	**0.00**	rat783	2.86	4.48	2.18	1.66	**1.22**	pr107	0.00	0.00	0.00	0.00	**0.00**
fl417	0.63	0.80	0.77	0.60	**0.58**	p654	0.22	0.75	0.05	0.16	**0.03**	vm1084	1.68	3.64	1.74	0.45	0.61
rd100	0.01	0.01	0.01	0.01	**0.01**	rl1323	3.29	4.35	1.93	2.00	**1.71**	linhp318	2.59	3.22	2.77	2.28	**1.99**
pcb442	0.84	1.15	0.96	1.06	**0.53**	eil101	1.94	2.59	2.08	1.78	**1.78**	rat575	2.33	3.11	1.88	1.28	**1.21**
gil262	0.53	0.59	0.48	0.40	**0.37**	eil76	1.23	1.53	1.18	1.18	**1.18**	pr299	0.25	0.61	0.11	0.21	0.16
pr439	1.56	2.80	1.97	0.13	0.60	kroC100	0.01	0.01	0.01	0.01	**0.01**	eil51	0.67	0.67	0.67	0.67	**0.67**

the generated instances being less challenging (7.76%), indicating that reduced instance difficulty alone does not yield stronger solvers and that adaptive pressure from adversarial interaction is critical for effective solver evolution. Disabling *reflection* increases the solver gap to 10.21%, as the absence of structured diagnostic signals results in less targeted solver adaptation and weaker performance gains. When *mutation* is removed, the solver gap rises to 9.33%; although its effect is smaller than that of adversarial feedback or reflection, mutation remains important for maintaining population diversity and preventing evolutionary stagnation. With all components enabled, the full ACEvo achieves the lowest solver gap (8.10%) while generating the hardest instances (10.43%), showing that adversarial feedback, reflection, and mutation play complementary roles and must be jointly applied to support stable co-evolution and robust solver generalization.

4.4 Structural Visualization of Optimization Instances

To gain qualitative insights into the structural properties of ACEvo-generated instances, we visualize the spatial distributions of four representative problem types—TSP_GLS, TSP_ACO, OP_ACO ,and CVRP_ACO—using two nonlinear manifold learning techniques, t-SNE Van der Maaten & Hinton (2008) and UMAP McInnes et al. (2018), as well as standard coordinate plots (Appendix D.1). Each visualization is performed over a batch of 500 instances. Across all tasks, the standard (bottom row) instance distributions exhibit uniform dispersion and low geometric complexity, consistent with conventional Euclidean instance distributions. In contrast, ACEvo-generated hard instances (top and middle rows) reveal pronounced structural patterns: t-SNE projections expose fragmented clusters and boundary concentrations, while UMAP highlights manifold separation and high-density cores. These effects are particularly evident in OP_ACO, where diverse structural motifs emerge, and in CVRP_ACO, which exhibits anisotropic and curved manifolds reflecting capacity-induced coupling and depot-centered routing constraints. Taken together, the above observations indicate that ACEvo not only increases algorithmic difficulty but also fundamentally reshapes the instance distribution at the distributional level; these changes are clearly discernible in both visual and structural characteristics and further amplify the combinatorial complexity of the resulting optimization problems.

5 Conclusion

We introduce ACEvo, a combinatorial optimization framework that enables adversarial co-evolution of problem instances and solvers mediated by large language models. By replacing static instance distributions and manually designed heuristics with a feedback-driven evolutionary loop, ACEvo jointly amplifies instance hardness and solver capability, enabling the autonomous emergence of adaptive algorithms. Experiments on classical combinatorial

optimization tasks, including the Traveling Salesman Problem (TSP), Orienteering Problem (OP), and Capacitated Vehicle Routing Problem (CVRP), across widely used public datasets and TSPLIB instances demonstrate that ACEvo generates substantially harder instances while achieving state-of-the-art performance, with synthesized solvers outperforming leading LLM-based baselines. ACEvo opens a new frontier for research at the intersection of adversarial instance generation, solver synthesis, and combinatorial optimization. By coupling evolutionary co-evolution with language-model reasoning, it establishes a pathway toward sustained, self-driven algorithmic innovation.

Acknowledgments

This work was supported by the Institute of Information and Communications Technology Planning and Evaluation (IITP) grant funded by the Korean government (MSIT) (No. RS-2025-02214654), and by the New Faculty Startup Fund from Seoul National University.

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.

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

Christian Blum and Andrea Roli. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Computing Surveys*, 35(3):268–308, 2003. doi: 10.1145/937503.937505.

Thomas Bömer, Nico Koltermann, Max Disselnmeyer, Laura Dörr, and Anne Meyer. Leveraging large language models to develop heuristics for emerging optimization problems. *CoRR*, abs/2503.03350, 2025. doi: 10.48550/ARXIV.2503.03350. URL https://doi.org/10.48550/arXiv.2503.03350.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, and et al. Language models are few-shot learners. In *NeurIPS 2020*, 2020. URL https://proceedings.neurips.cc/paper/2020/hash/1457c0d6bfcb4967418bfb8ac142f64a-Abstract.html.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, and et al. Evaluating large language models trained on code. *CoRR*, abs/2107.03374, 2021. URL https://arxiv.org/abs/2107.03374.

M. Dorigo and L.M. Gambardella. Ant colony system: A cooperative learning approach to the traveling salesman problem. *IEEE Transactions on Evolutionary Computation*, 1(1):53–66, 1997. doi: 10.1109/4235.585892.

M. R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979. ISBN 0-7167-1044-7.

Michel Gendreau and Jean-Yves Potvin. Metaheuristics in combinatorial optimization. *Annals of Operations Research*, 140(1):189–213, 2005. doi: 10.1007/s10479-005-3971-7. URL https://doi.org/10.1007/s10479-005-3971-7.

Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *NeurIPS 2014*, pp. 2672–2680, 2014. URL https://proceedings.neurips.cc/paper/2014/hash/5ca3e9b122f61f8f06494c97b1afccf3-Abstract.html.

Yi-Nan Guo, Jianjiao Ji, Ying Tan, and Shi Cheng. Multi-objective combinatorial generative adversarial optimization and its application in crowdsensing. In *ICSI 2020*, volume 12145 of *Lecture Notes in Computer Science*, pp. 423–434. Springer, 2020. doi: 10.1007/978-3-030-53956-6_38.

Cheng He, Shihua Huang, Ran Cheng, Kay Chen Tan, and Yaochu Jin. Evolutionary multiobjective optimization driven by generative adversarial networks (gans). *IEEE Transactions on Cybernetics*, 51(6):3129–3142, 2021. doi: 10.1109/TCYB.2020.2985081.

Zangir Iklassov, Yali Du, Farkhad Akimov, and Martin Takac. Self-guiding exploration for combinatorial problems, 2024. URL https://arxiv.org/abs/2405.17950.

Shuo Jiang, Min Xie, and Jianxi Luo. Large language models for combinatorial optimization of design structure matrix. *CoRR*, abs/2411.12571, 2024. doi: 10.48550/ARXIV.2411.12571. URL https://doi.org/10.48550/arXiv.2411.12571.

Christian Ledig, Lucas Theis, Ferenc Huszar, Jose Caballero, Andrew Cunningham, Alejandro Acosta, Andrew P. Aitken, Alykhan Tejani, Johannes Totz, Zehan Wang, and Wenzhe Shi. Photo-realistic single image super-resolution using a generative adversarial network. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 105–114. IEEE Computer Society, 2017. doi: 10.1109/CVPR.2017.19. URL https://doi.org/10.1109/CVPR.2017.19.

Yang Li, Jinpei Guo, Runzhong Wang, and Junchi Yan. From distribution learning in training to gradient search in testing for combinatorial optimization. In *NeurIPS 2023*, 2023. URL https://papers.nips.cc/paper_files/paper/2023/hash/9c93b3cd3bc60c0fe7b0c2d74a2da966-Abstract-Conference.html.

Yujia Li, David H. Choi, Junyoung Chung, Nate Kushman, Julian Schrittwieser, Rémi Leblond, Tom Eccles, James Keeling, Felix Gimeno, Agustin Dal Lago, Thomas Hubert, Peter Choy, Cyprien de Masson d'Autume, Igor Babuschkin, Xinyun Chen, Po-Sen Huang, Johannes Welbl, Sven Gowal, Alexey Cherepanov, James Molloy, Daniel J. Mankowitz, Esme Sutherland Robson, Pushmeet Kohli, Nando de Freitas, Koray Kavukcuoglu, and Oriol Vinyals. Competition-level code generation with alphacode. *CoRR*, abs/2203.07814, 2022. doi: 10.48550/arXiv.2203.07814. URL https://doi.org/10.48550/arXiv.2203.07814.

Zhiming Lin, Kai Zhao, Sophie Zhang, Peilai Yu, and Canran Xiao. Cec-zero: Zero-supervision character error correction with self-generated rewards. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pp. 23612–23620, 2026.

Fei Liu, Chengyu Lu, Lin Gui, Qingfu Zhang, Xialiang Tong, and Mingxuan Yuan. Heuristics for vehicle routing problem: A survey and recent advances. *CoRR*, abs/2303.04147, 2023a. doi: 10.48550/ARXIV.2303.04147. URL https://doi.org/10.48550/arXiv.2303.04147.

Fei Liu, Xialiang Tong, Mingxuan Yuan, and Qingfu Zhang. Algorithm evolution using large language model. *CoRR*, abs/2311.15249, 2023b. doi: 10.48550/ARXIV.2311.15249. URL https://doi.org/10.48550/arXiv.2311.15249. arXiv:2311.15249.

Fei Liu, Xialiang Tong, Mingxuan Yuan, Xi Lin, Fu Luo, Zhenkun Wang, Zhichao Lu, and Qingfu Zhang. Evolution of heuristics: Towards efficient automatic algorithm design using large language model. In *ICML 2024*, 2024a. URL https://openreview.net/forum?id=BwAkaxqiLB.

Shengcai Liu, Caishun Chen, Xinghua Qu, Ke Tang, and Yew-Soon Ong. Large language models as evolutionary optimizers. In *IEEE Congress on Evolutionary Computation (CEC) 2024*, pp. 1–8, 2024b. doi: 10.1109/CEC60901.2024.10611913. URL https://doi.org/10.1109/CEC60901.2024.10611913.

Fu Luo, Xi Lin, Fei Liu, Qingfu Zhang, and Zhenkun Wang. Neural combinatorial optimization with heavy decoder: Toward large scale generalization. In *NeurIPS 2023*, 2023. URL https://papers.nips.cc/paper_files/paper/2023/hash/1c10d0c087c14689628124bbc8fa69f6-Abstract-Conference.html.

Lin Ma, Jiangtao Gong, Hao Xu, Hao Chen, Hao Zhao, Wenbing Huang, and Guyue Zhou. Planning assembly sequence with graph transformer. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 12395–12401. IEEE, 2023.

L. McInnes, J. Healy, and J. Melville. UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction. *ArXiv e-prints*, February 2018.

Chaoxu Mu, Xufeng Zhang, and Hui Wang. Planning of heuristics: Strategic planning on large language models with monte carlo tree search for automating heuristic optimization. *CoRR*, abs/2502.11422, 2025. doi: 10.48550/ARXIV.2502.11422. URL https://doi.org/10.48550/arXiv.2502.11422. arXiv:2502.11422.

Manfred Padberg and Giovanni Rinaldi. A branch-and-cut algorithm for the resolution of large-scale symmetric traveling salesman problems. *SIAM Review*, 33(1):60–100, 1991. doi: 10.1137/1033004.

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

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

Camilo Chacón Sartori and Christian Blum. Improving existing optimization algorithms with llms. *CoRR*, abs/2502.08298, 2025. doi: 10.48550/ARXIV.2502.08298. URL https://doi.org/10.48550/arXiv.2502.08298.

Christopher C. Skiscim and Bruce L. Golden. Optimization by simulated annealing: A preliminary computational study for the tsp. In *Winter Simulation Conference (WSC)*, pp. 523–535. ACM, 1983. URL http://dl.acm.org/citation.cfm?id=801546.

Kate Smith-Miles and Jano I. van Hemert. Discovering the suitability of optimisation algorithms by learning from evolved instances. *Annals of Mathematics and Artificial Intelligence*, 61(2):87–104, 2011. doi: 10.1007/s10472-011-9230-5.

Zhiqing Sun and Yiming Yang. Difusco: Graph-based diffusion solvers for combinatorial optimization. In *NeurIPS 2023*, 2023. URL https://papers.nips.cc/paper_files/paper/2023/hash/0ba520d93c3df592c83a611961314c98-Abstract-Conference.html.

Xinyu Tang, Xiaolei Wang, Wayne Xin Zhao, Siyuan Lu, Yaliang Li, and Ji-Rong Wen. Unleashing the potential of large language models as prompt optimizers: Analogical analysis with gradient-based model optimizers. In *AAAI 2025*, pp. 25264–25272, 2025. doi: 10.1609/AAAI.V39I24.34713. URL https://doi.org/10.1609/aaai.v39i24.34713.

Laurens Van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(11), 2008.

Qi Wang, Yongsheng Hao, and Jiawei Zhang. Generative inverse reinforcement learning for learning 2-opt heuristics without extrinsic rewards in routing problems. *Journal of King Saud University - Computer and Information Sciences*, 35(9):101787, 2023. doi: 10.1016/j.jksuci.2023.101787.

Xingyu Wu, Yan Zhong, Jibin Wu, and Kay Chen Tan. As-llm: When algorithm selection meets large language model. *CoRR*, abs/2311.13184, 2023. doi: 10.48550/ARXIV.2311.13184. URL https://doi.org/10.48550/arXiv.2311.13184.

Canran Xiao, Jiabao Dou, Zhiming Lin, Zong Ke, and Liwei Hou. From points to coalitions: Hierarchical contrastive shapley values for prioritizing data samples. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pp. 15995–16003, 2026a.

Canran Xiao, Chuangxin Zhao, Zong Ke, Fei Shen, and Li Shen. Curiosity-driven cooperation for long-tailed multi-label learning. *Neural Networks*, pp. 109075, 2026b.

Liang Xin, Wen Song, Zhiguang Cao, and Jie Zhang. Generative adversarial training for neural combinatorial optimization models, 2022. URL https://openreview.net/forum?id=9vsRT9mc7U.

Chengrun Yang, Xuezhi Wang, Yifeng Lu, Hanxiao Liu, Quoc V. Le, Denny Zhou, and Xinyun Chen. Large language models as optimizers. In *ICLR 2024*, 2024. URL https://openreview.net/forum?id=Bb4VGOWELI.

Shunyu Yao, Fei Liu, Xi Lin, Zhichao Lu, Zhenkun Wang, and Qingfu Zhang. Multi-objective evolution of heuristic using large language model. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI-25)*, pp. 27144–27152. AAAI Press, 2025. doi: 10.1609/AAAI.V39I25.34922. URL https://doi.org/10.1609/aaai.v39i25.34922.

Haoran Ye, Jiarui Wang, Zhiguang Cao, Federico Berto, Chuanbo Hua, Haeyeon Kim, Jinkyoo Park, and Guojie Song. Reevo: Large language models as hyper-heuristics with reflective evolution. In *NeurIPS 2024*, 2024. URL https://papers.nips.cc/paper_files/paper/2024/hash/4ced59d480e07d290b6f29fc8798f195-Abstract-Conference.html.

Huigen Ye, Hua Xu, An Yan, and Yaoyang Cheng. Large language model-driven large neighborhood search for large-scale milp problems. In *Forty-second International Conference on Machine Learning*, 2025.

Eric Zelikman, Yuhuai Wu, Jesse Mu, and Noah D. Goodman. Star: Bootstrapping reasoning with reasoning. In *NeurIPS 2022*, 2022. URL https://papers.nips.cc/paper_files/paper/2022/hash/639a9a172c044fbb64175b5fad42e9a5-Abstract-Conference.html.

Yifei Zhang, Hao Zhao, Hongyang Li, and Siheng Chen. Fastmac: Stochastic spectral sampling of correspondence graph. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 17857–17867, 2024.

Jie Zhao, Kang Hao Cheong, and Witold Pedrycz. Bridging visualization and optimization: Multimodal large language models on graph-structured combinatorial optimization. *CoRR*, abs/2501.11968, 2025a. doi: 10.48550/ARXIV.2501.11968. URL https://doi.org/10.48550/arXiv.2501.11968.

Jie Zhao, Tao Wen, and Kang Hao Cheong. Can large language models be trusted as black-box evolutionary optimizers for combinatorial problems? *CoRR*, abs/2501.15081, 2025b. doi: 10.48550/ARXIV.2501.15081. URL https://doi.org/10.48550/arXiv.2501.15081.

Zirui Zhao, Wee Sun Lee, and David Hsu. Large language models as commonsense knowledge for large-scale task planning. In *NeurIPS 2023*, 2023. URL https://papers.nips.cc/paper_files/paper/2023/hash/65a39213d7d0e1eb5d192aa77e77eeb7-Abstract-Conference.html.

Xiaoji Zheng, Lixiu Wu, Zhijie Yan, Yuanrong Tang, Hao Zhao, Chen Zhong, Bokui Chen, and Jiangtao Gong. Large language models powered context-aware motion prediction in autonomous driving. In *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 980–985. IEEE, 2024.

Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A. Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *ICCV 2017*, pp. 2242–2251, 2017. doi: 10.1109/ICCV.2017.244.

Appendix Overview

This appendix provides extended theoretical formulations, prompt specifications, experimental configurations, and supplementary empirical analyses supporting the main paper.

A. Extended Preliminaries . . . 15
A.1 Formal Problem Setting . . . 15
A.2 Solver Evaluation over Generated Instance Distributions . . . 15
A.3 Background on the Traveling Salesman Problem . . . 15
A.4 Background on the Orienteering Problem . . . 15
A.5 Background on the Capacitated Vehicle Routing Problem . . . 16
A.6 Heuristic Solvers for Routing Problems . . . 16
A.7 Instance Distribution Shift in Combinatorial Optimization . . . 16
A.8 Objective Direction and Hardness Signals . . . 16
A.9 Primary Prompts . . . 17
A.10 Task-Tailored Prompt Components . . . 18

B. Extended Applications and Generalization Scope . . . 20

C. Experimental Details . . . 21
C.1 Guided Local Search for the Traveling Salesman Problem . . . 21
C.2 Experimental Configuration . . . 21
C.3 Objective Value Definition in TSP ACO, OP ACO, and CVRP ACO . . . 21

D. Detailed Analysis . . . 22
D.1 Detailed Description of the Visualization Protocol . . . 22
D.2 Evolution Process . . . 24
D.3 Statistical Robustness and Sensitivity Analysis . . . 24
D.4 Solver–instance Synergy on the Orienteering Problem . . . 25
D.5 Solver–instance Synergy on the Capacitated Vehicle Routing Problem . . . 25
D.6 Ablation Study on Evolutionary Components . . . 26
D.7 Adversarial Co-Evolution Procedure of `ACEvo` . . . 26

A Extended Preliminaries

A.1 Formal Problem Setting

Let $\mathcal{I}$ denote the space of problem instances and $\mathcal{S}$ the space of feasible solutions. A combinatorial optimization problem is specified by an objective function

$$f : \mathcal{S} \to \mathbb{R}, \tag{5}$$

where lower values correspond to better solutions.

A heuristic solver is modeled as a mapping

$$h : \mathcal{I} \to \mathcal{S}, \quad h \in \mathcal{H}, \tag{6}$$

where $\mathcal{H}$ denotes the hypothesis space of admissible solvers.

An instance generator is defined as a mapping

$$g : \mathcal{Z} \to \mathcal{I}, \quad g \in \mathcal{G}, \tag{7}$$

where $\mathcal{Z}$ is a latent seed space equipped with a base distribution $z \sim \mathcal{Z}$.

The generator g induces a distribution over problem instances:

$$I \sim \mathcal{D}_g, \qquad \mathcal{D}_g := g_{\#}(\mathcal{Z}), \tag{8}$$

where $g_{\#}$ denotes the pushforward measure of g.

A.2 Solver Evaluation over Generated Instance Distributions

Given a solver $h \in \mathcal{H}$ and a generator $g \in \mathcal{G}$, solver performance is evaluated in expectation over the induced instance distribution:

$$\mathcal{L}(h; g) \;=\; \mathbb{E}_{I \sim \mathcal{D}_g}\left[f(h(I))\right]. \tag{9}$$

In practice, this expectation is approximated via Monte Carlo sampling:

$$\widehat{\mathcal{L}}(h; g) \;=\; \frac{1}{n}\sum_{i=1}^{n} f(h(g(z_i))), \qquad z_i \sim \mathcal{Z}. \tag{10}$$

This formulation treats the solver as a black-box mapping from instances to solutions and does not assume differentiability or access to internal solver states.

A.3 Background on the Traveling Salesman Problem

The Traveling Salesman Problem (TSP) is a classical routing problem defined on a complete weighted graph. Given a graph $G = (V, E, w)$, where $|V| = n$, the weight function $w : E \to \mathbb{R}_+$ assigns a non-negative travel cost to each edge.

A feasible solution corresponds to a tour π over V. The objective is to minimize the total travel cost:

$$\min_{\pi} \sum_{(i,j)\in\pi} w(i,j) \quad \text{s.t.} \quad \pi = (v_{\pi_1}, \ldots, v_{\pi_n}),\ \{\pi_1, \ldots, \pi_n\} = V. \tag{11}$$

A.4 Background on the Orienteering Problem

The Orienteering Problem (OP) is a routing problem that generalizes the Traveling Salesman Problem by associating each node with a non-negative reward and introducing a global budget constraint. Given a complete weighted graph $G = (V, E, w)$, each node $v \in V$ is assigned a reward $r(v) \geq 0$, and each edge $(i, j) \in E$ has an associated travel cost $w(i, j)$.

A feasible solution corresponds to a tour $\pi \subseteq V$ that starts and ends at a designated depot. The objective is to maximize the total collected reward:

$$\max_{\pi \subseteq V} \sum_{v\in\pi} r(v) \quad \text{s.t.} \quad \sum_{(i,j)\in\pi} w(i,j) \leq B. \tag{12}$$

A.5 Background on the Capacitated Vehicle Routing Problem

The Capacitated Vehicle Routing Problem (CVRP) extends classical routing problems by introducing multiple routes and explicit capacity constraints. Given a set of customer nodes $V = \{1, \ldots, n\}$ and a depot node 0, each customer $i \in V$ is associated with a demand $d_i > 0$. A fleet of identical vehicles, each with capacity Q, departs from and returns to the depot to serve all customers.

A feasible solution consists of a set of routes $\{\pi_1, \ldots, \pi_K\}$. The objective is to minimize the total routing cost:

$$\min \sum_{k=1}^{K} \sum_{(i,j)\in\pi_k} w(i,j) \quad \text{s.t.} \quad \sum_{i\in\pi_k} d_i \leq Q, \quad \forall k = 1, \ldots, K. \tag{13}$$

A.6 Heuristic Solvers for Routing Problems

Heuristic solvers for routing problems can be abstracted as iterative procedures that refine an initial feasible solution

$$\pi^{(0)} \in \mathcal{S}(I) \tag{14}$$

through a sequence of solver-dependent transformations:

$$\pi^{(t+1)} = \mathcal{T}_\theta\left(\pi^{(t)}, I\right), \tag{15}$$

where $\mathcal{T}_\theta$ denotes a heuristic update operator parameterized by solver-specific logic θ, and I represents the problem instance.

This abstraction encompasses a wide range of classical heuristics, including local-search-based methods and population-based approaches. Different solvers instantiate $\mathcal{T}_\theta$ using distinct mechanisms, while sharing the common objective of iteratively improving solution quality under the constraints imposed by the instance structure.

A.7 Instance Distribution Shift in Combinatorial Optimization

Let $\mathcal{D}_{\text{std}}$ denote a reference instance distribution and $\mathcal{D}_g$ an instance distribution induced by a generator $g \in \mathcal{G}$. A distribution shift occurs when

$$\mathcal{D}_g \neq \mathcal{D}_{\text{std}}. \tag{16}$$

Solver behavior under distributional shift can be characterized by comparing expected performance across distributions:

$$\mathbb{E}_{I\sim\mathcal{D}_{\text{std}}}[f(h(I))] \quad \text{and} \quad \mathbb{E}_{I\sim\mathcal{D}_g}[f(h(I))], \tag{17}$$

where h denotes a solver and $f(\cdot)$ the associated objective function.

By iteratively mutating generator programs in $\mathcal{G}$, `ACEvo` induces a sequence of instance distributions

$$\mathcal{D}_{g_1}, \mathcal{D}_{g_2}, \ldots, \mathcal{D}_{g_T}, \tag{18}$$

which enables the systematic study of solver behavior under progressively shifting instance distributions.

A.8 Objective Direction and Hardness Signals

The routing problems considered in this work include both minimization and maximization objectives. Specifically, TSP and CVRP are formulated as minimization problems, whereas OP is formulated as a maximization problem. For notational consistency, solver performance is evaluated directly using the task-specific objective function $f(\cdot)$.

When assessing instance difficulty under a fixed solver, degradation in solver performance manifests differently depending on the objective direction. For minimization problems,

larger objective values indicate worse solutions, whereas for maximization problems, smaller objective values indicate worse solutions. Accordingly, instance hardness can be characterized by relative deterioration in solver performance with respect to the reference instance distribution.

A.9 Primary Prompts

You are a specialist in algorithmic optimization, with a focus on heuristic strategies. Your task is to construct effective heuristics that leverage underlying structural patterns within the problem to solve optimization tasks. The approach should demonstrate clear improvements in efficiency and solution quality compared to prior methods.

Prompt 1: System prompt for heuristics generator LLM.

You are a specialist in generating challenging input distributions for combinatorial optimization problems. Your task is to construct complex problem instances characterized by rich structural diversity. The resulting instances should surpass prior designs in both difficulty and diversity.

Prompt 2: System prompt for hard instance generator LLM.

Implement the {function_name} function to solve {problem_description}.

Prompt 3: Problem Statement.

{problem_statement}

{base_function}

Invent a creative variation named {func_name_v2}. Return only the code, enclosed in a properly formatted Python code string block like this:

```python ...```

Prompt 4: Initialization.
```

{problem_statement}

{base_function}

[Prior reflection]
{reflection}

[Previous result]
{result}

[Code]
{func_signature}

[Improved code]
Construct a mutated version of the original function titled {`func_name`}`_v2`, guided by the given feedback. Return only the function's implementation as a Python string enclosed within a properly formatted code block:
```python ...```

Prompt 5: Mutation (heuristics) or mutation (hard instance)

[Information]
{mutation}

[Adaptivity node]
{diagnostic anchor}

Prompt 6: User prompt for LLM.

A.10 Task-Tailored Prompt Components

#heuristics
The "heuristics" function receives a distance matrix as input and outputs a matrix of the same dimensions, where each entry reflects a preliminary estimate of the cost or undesirability of selecting the corresponding edge in a solution.

#hard_instance_generator
The "hard_instance_generator" function takes as input a solver, and returns a distance matrix that is adversarially constructed to maximize the solver's failure or suboptimality. The return is a valid TSP instance in the standard distance matrix format.

Prompt 7: Problem description (TSP_GLS).
```

#heuristics
The "heuristics" function processes a given distance matrix and outputs a matrix of the same shape, where each entry reflects a prior estimate of how beneficial it may be to include the corresponding edge in the final solution.

#hard_instance_generator
The "hard_instance_generator" function returns a distance matrix of fixed shape, intentionally constructed to be difficult for optimization algorithms to solve, often by embedding deceptive or irregular structural patterns.

Prompt 8: Problem description (TSP_ACO).

```
#TSP_GLS
def heuristics_v1(distance_matrix: np.ndarray
) -> np.ndarray:
    # Long-distance edges are typically
    # avoided in optimal TSP_GLS solutions.

    return distance_matrix

#TSP_ACO
def heuristics_v1(distance_matrix: np.ndarray
) -> np.ndarray:
    return 1 / distance_matrix
```

Prompt 9: Base_function (heuristics).

```
def hard_instance_generator_v1(
    distance_matrix: np.ndarray
    ) -> np.ndarray:

[task_constraint]

# The function must return all datasets directly
# as separate numpy arrays
# (not in a list or dictionary),
# following exactly this format:

# Generate each dataset here...

return hard_instance_20, hard_instance_50,
hard_instance_100, hard_instance_200, hard_instance_300,
hard_instance_400, hard_instance_500, hard_instance_600,
hard_instance_700, hard_instance_800, hard_instance_900,
hard_instance_1000

# Do not include any test code or example calls.
# Only output the function definition.
```

Prompt 10: Base_function (hard_instance_generator).

Write a hard_instance_generator function that generates complex and challenging [task] instances entirely in the form of adjacency matrices, to improve the performance of [task] algorithms.
The hard_instance_generator function must return multiple numpy arrays with the following exact shapes:
(64, 20, 20)
(64, 50, 50)
(64, 100, 100)
(64, 200, 200)
(64, 300, 300)
(64, 400, 400)
(64, 500, 500)
(64, 600, 600)
(64, 700, 700)
(64, 800, 800)
(64, 900, 900)
(64, 1000, 1000)
Each instance must be an "adjacency matrix" representing a [task] problem with Euclidean distances or generated structural complexity (e.g., fractal, clustered, or spiral distributions). "Do not return coordinates; only return the adjacency matrices".

Prompt 11: Task_constraint.

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

Prompt 12: Function signature.

Generate increasingly numerous and complex hard instances that specifically target the weaknesses of [task], and improve the corresponding code accordingly.

Prompt 13: Diagnostic anchor.

B Extended Applications and Generalization Scope

The proposed ACEvo framework is generally applicable to optimization settings in which both problem instances and solvers can be represented as executable symbolic programs, and where relative solver performance provides a meaningful behavioral signal for guiding evolutionary feedback. Under this formulation, the adversarial co-evolution mechanism in ACEvo operates over a broad class of program- or string-based optimization settings in which candidate solutions are evaluated through performance-based criteria.

This formulation is consistent with prior observations on reflective evolutionary search (e.g., ReEvo Ye et al. (2024)), which suggest that performance-driven symbolic evolution can be effective across diverse optimization settings without requiring access to internal solver representations. From a modeling standpoint, ACEvo treats solvers and instance generators uniformly as executable artifacts, enabling the co-evolutionary process to be driven solely by observed behavioral outcomes.

This positions ACEvo as a general mechanism for jointly amplifying instance difficulty and solver robustness under distributional shift, rather than as a method specialized for a particular task family.

C Experimental Details

Computational resources. Our framework performs co-evolutionary updates on both heuristics and problem instances, leveraging parallel execution whenever computational resources are sufficient. The total wall-clock time for each iteration of `ACEvo` is influenced by both the latency of model queries and the cost of instance evaluation. Claude 3.7 Sonnet was used as the language model backend, accessed exclusively through the Anthropic API. Across all Claude 3.7 Sonnet queries in `ACEvo`, the estimated average API cost was approximately $0.05 per call. All evaluations were conducted on a server powered by an AMD EPYC 7742 CPU.

Table 3: TSP_GLS settings across problem sizes.

Problem Size (n)	**Perturbation Moves**	**Iteration Limit**
20	5	73
50	30	175
100	40	800
200	40	800
300	40	800
400	40	800
500	40	800
600	40	800
700	40	800
800	40	800
900	40	800
1000	40	800

C.1 Guided Local Search for the Traveling Salesman Problem

To ensure consistent solver behavior across varying TSP instance sizes, we set the number of perturbation moves and iteration limits as a function of the problem scale. As summarized in Table 3, smaller instances are assigned fewer moves and iterations, while larger instances use fixed values to balance efficiency and effectiveness. These parameters were tuned to maintain a stable computational budget and reliable convergence behavior within our evaluation framework.

C.2 Experimental Configuration

Each evolutionary run in `ACEvo` was conducted with a population size of 10 and 20 iterations of adversarial co-evolution. In each iteration, solvers and instance generators were evaluated and updated based on performance. The system achieved stable convergence within these iterations, demonstrating that this configuration effectively balances efficiency and evolutionary diversity. This setup ensures consistent evolutionary pressure while maintaining manageable computational cost, providing a reliable foundation for all reported results.

C.3 Objective Value Definition in TSP_ACO, OP_ACO, and CVRP_ACO

The objective functions are determined by the underlying combinatorial optimization problems (TSP, OP, and CVRP) and are invariant to the choice of solver (see Appendix A). In particular, TSP_ACO, OP_ACO, and CVRP_ACO differ only in their heuristic search mechanisms, while the evaluation objective $f(\cdot)$ is consistently defined across all methods.

D Detailed Analysis

D.1 Detailed Description of the Visualization Protocol

Figure 4 provides an overview of the visualization grid used to analyze instance structures. The first, second, and third rows correspond to t–SNE Van der Maaten & Hinton (2008), UMAP McInnes et al. (2018), and standard two–dimensional layouts, respectively. The t–SNE Van der Maaten & Hinton (2008) and UMAP McInnes et al. (2018) rows illustrate the distributions of `ACEvo`–generated hard instances, while the bottom (Std) row depicts standard datasets for reference. The columns represent TSP_GLS, TSP_ACO, OP_ACO, and CVRP_ACO from left to right, with CVRP_ACO additionally incorporating capacity constraints and depot–centered routing structures. Each scatter plot shows a two–dimensional embedding of instance representations obtained via nonlinear manifold projection, enabling qualitative comparison between `ACEvo`–generated hard instances and datasets such as ReEvo Ye et al. (2024). Compared with conventional datasets, the `ACEvo` instances exhibit denser local clusters, broader dispersion, and clearer structural boundaries, revealing adaptive dynamics of adversarial co–evolution—where the generator continually diversifies its outputs as the solver improves. This emergent diversity underscores `ACEvo`'s ability to break dataset homogeneity and foster coadaptation between instance generation and solver evolution.

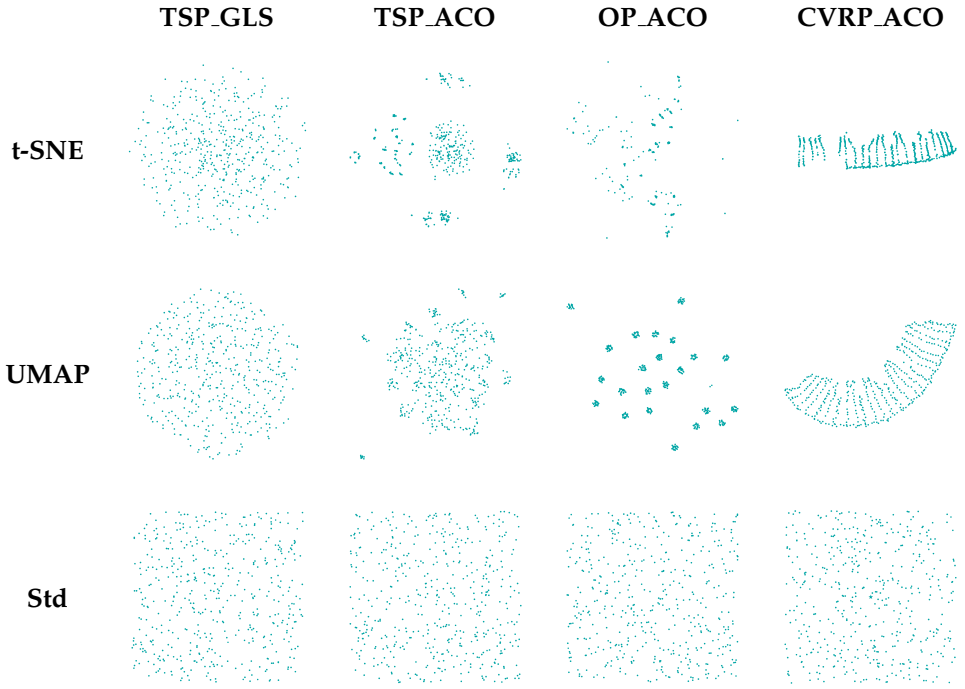

Figure 4: Visualization of `ACEvo`-generated instance distributions across TSP_GLS, TSP_ACO, OP_ACO, and CVRP_ACO using t-SNE, UMAP, and standard instance layouts. Top row shows optimization tasks; left column shows visualization methods. The standard instance layout consists of problem instances sampled from isotropic Gaussian distributions, without any adversarial modification or structural complexity enhancement.

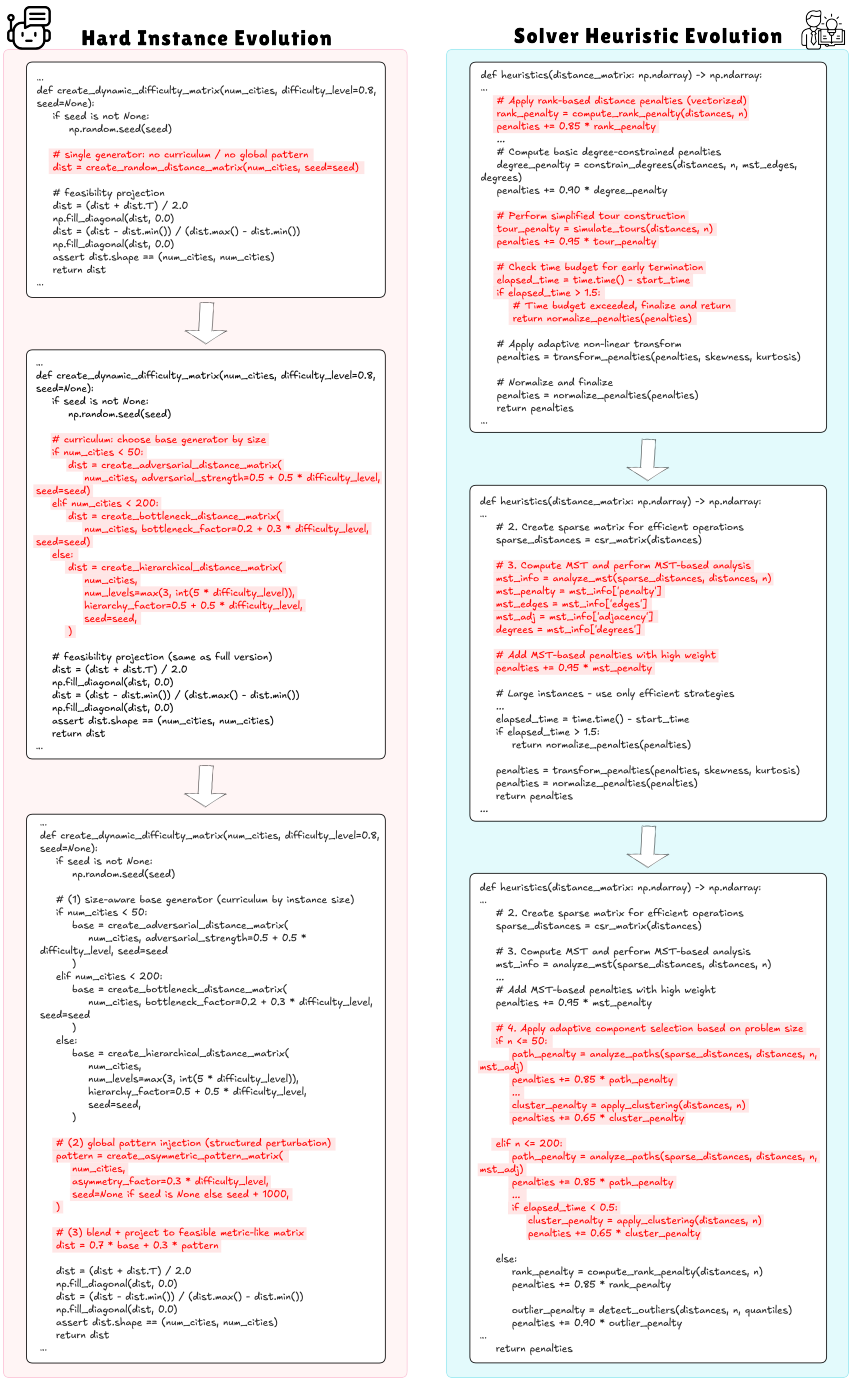

Figure 5: Representative program fragments from different stages of the ACEvo co-evolution process, illustrating hard instance evolution (left) and solver heuristic evolution (right).

D.2 Evolution Process

Figure 5 illustrates representative program fragments sampled at different stages of the ACEvo co-evolution process. The figure is intended to clarify how instance generators and heuristic scoring functions evolve structurally through mutation and selection, rather than to enumerate all implementation details. Each fragment corresponds to a concrete executable variant retained after selection in a specific evolutionary stage. Only representative snapshots are shown, as the full evolutionary trajectory consists of many intermediate mutations. The examples are chosen to highlight structural changes that persist across multiple iterations, rather than transient or degenerate variants.

Hard Instance Evolution (left). The initial generator constructs distance matrices using a single randomized procedure, without explicit global structure or scale awareness. In later variants, the generator adopts a size-conditioned design, where different base construction mechanisms are selected according to the number of cities. This enables distinct structural regimes to emerge at different problem scales, such as adversarial perturbations for small instances, bottlenecked structures for medium instances, and hierarchical layouts for larger instances. In the final variant, an explicit global pattern component is injected and blended with the base generator, introducing structured asymmetries while preserving metric feasibility through normalization and projection. Across all stages, feasibility constraints are enforced to ensure valid TSP instances.

Solver Heuristic Evolution (right). The heuristic function evolves from simple, uniform penalty assignments toward progressively more structured analyses of the distance matrix. Intermediate variants introduce rank-based penalties, degree-aware signals, and approximate tour simulations to capture local and mid-range structure. Later versions incorporate sparse representations and minimum spanning tree (MST)–based analysis to extract global connectivity information efficiently. In the final stage, the heuristic adaptively activates different analytical components—such as path-based penalties, clustering signals, or outlier detection—based on problem size and time constraints, allowing scale-dependent trade-offs between expressiveness and computational cost.

D.3 Statistical Robustness and Sensitivity Analysis

To assess the robustness of `ACEvo` under stochastic generation, we perform independent runs with different random seeds for the ablation experiments reported in Table 6. No new experimental settings are introduced; each run follows the same experimental protocol and differs only in random initialization and stochastic sampling. Each run involves independent sampling from the large language model, together with randomized mutation, reflection, and selection operations during the co-evolutionary process. This setup provides an empirical sensitivity analysis with respect to random seeds and stochastic LLM sampling. Table 6 reports the mean performance and one standard deviation across these runs. This repeated evaluation allows us to quantify the extent to which the observed performance depends on stochastic fluctuations rather than systematic properties of the proposed framework.

The full `ACEvo` configuration exhibits relatively small run-to-run variation in both hard-instance generation and solver performance. The corresponding standard deviations remain lower than those observed for most ablated variants, indicating that the complete framework maintains stable performance across different stochastic realizations. In contrast, removing adversarial feedback, reflection, or mutation generally increases performance variability, in addition to degrading the mean results. This pattern further indicates that the individual components contribute not only to higher average performance but also to reducing the sensitivity of the co-evolutionary process to random initialization and sampling noise.

These findings suggest that the gains of `ACEvo` are not driven by a single favorable random seed or sampling trajectory. Rather, the combination of adversarial feedback, evolutionary reflection, and mutation produces consistently strong outcomes across independent runs, demonstrating robustness to the inherent stochasticity of LLM-based generation and evolutionary search. The consistency of these results across independent repetitions therefore strengthens the statistical reliability of the conclusions drawn from the main ablation study.

Table 4: OP_ACO objective values under different heuristic–instance combinations. Higher is better.

Problem Size (n)	**heuristics (ReEvo)**Ye et al. (2024) $\oplus$ **hard instance (ACEvo)**	**heuristics (ACEvo)** $\oplus$ **hard instance (ACEvo)**
100	7.883	**7.923**
200	13.862	**14.157**
300	17.461	**17.635**
400	17.773	**18.662**
500	24.944	**25.388**
600	22.655	**23.532**
700	16.457	**17.516**
800	23.210	**24.752**
900	20.217	**20.760**
1000	20.061	**21.205**

Table 5: CVRP_ACO objective values under different heuristic–instance combinations. Lower is better.

Problem Size (n)	**heuristics (ReEvo)**Ye et al. (2024) $\oplus$ **hard instance (ACEvo)**	**heuristics (ACEvo)** $\oplus$ **hard instance (ACEvo)**
100	15.878	**15.616**
200	30.288	**29.853**
300	44.153	**43.915**
400	57.984	**57.128**
500	71.895	**70.932**
600	85.552	**84.695**
700	99.122	**97.920**
800	112.340	**110.835**
900	126.612	**125.317**
1000	139.767	**138.566**

D.4 Solver–instance synergy on the Orienteering Problem.

This appendix provides supplementary experimental results for solver–instance synergy on the Orienteering Problem. In particular, Table 4 reports the performance of different heuristic–instance combinations for OP_ACO. These results complement the discussion in the main text and further support the effectiveness of `ACEvo` in jointly evolving heuristics and hard instance distributions.

D.5 Solver–instance synergy on the Capacitated Vehicle Routing Problem.

Table 5 summarizes the objective values obtained by different heuristic–instance combinations on the Capacitated Vehicle Routing Problem solved via Ant Colony Optimization (CVRP_ACO). Across all evaluated problem sizes, heuristics co-evolved with `ACEvo` consistently yield lower routing costs on the same hard instances than heuristics generated by ReEvo Ye et al. (2024), indicating a systematic performance advantage under capacity-constrained routing scenarios.

This advantage is particularly notable given the additional feasibility constraints imposed by CVRP, where demand satisfaction and vehicle capacity jointly shape route construction

and cost accumulation. The results suggest that solvers evolved within the ACEvo framework better internalize these structural constraints under jointly constrained and highly nonconvex settings when operating on adversarial instance distributions. Together, these findings provide complementary evidence that the proposed co-evolutionary paradigm generalizes beyond unconstrained or reward-based routing problems, extending effectively to capacity-constrained combinatorial optimization tasks.

Beyond absolute performance differences, the relative ordering between heuristics remains stable as problem size increases. As the number of customers grows, both methods exhibit higher objective values due to increased routing complexity; however, the performance gap in favor of ACEvo-evolved heuristics persists across all evaluated scales. This consistent trend suggests that the observed solver–instance synergy is not driven by isolated problem sizes or favorable configurations, but reflects a systematic advantage that scales with the complexity of capacity-constrained routing tasks.

Table 6: Ablation study on the evolutionary components of ACEvo. We report the mean performance gap (%) $\pm$ one standard deviation across 10 independent runs, where each run averages the gap over all test instances. Lower is better.

Method	**Hard Instance Gap (%)**	**Solver Gap (%)**
Full ACEvo (all enabled)	$\mathbf{10.43 \pm 0.19}$	$\mathbf{8.10 \pm 0.16}$
w/o Adversarial Feedback	7.76 ± 0.44	11.43 ± 0.57
w/o Reflection	7.74 ± 0.26	10.21 ± 0.34
w/o Mutation	8.62 ± 0.31	9.33 ± 0.23

D.6 Ablation Study on Evolutionary Components

Table 6 presents an ablation study on the key evolutionary components of ACEvo, including adversarial feedback, reflection, and mutation. The results are reported as mean gap values with standard deviations over 10 independent runs. Overall, the full ACEvo framework achieves the strongest hard-instance generation performance while also maintaining the best solver-side performance, indicating that these components contribute jointly to the effectiveness and stability of the co-evolutionary process.

D.7 Adversarial Co-Evolution Procedure of ACEvo

Algorithm 1 summarizes the overall adversarial co-evolutionary procedure of ACEvo, as illustrated in Figure 1. The framework maintains two interacting populations: a population of heuristic solvers and a population of hard-instance generators. At each iteration, the current generators produce candidate instance batches on which the solver population is evaluated, thereby identifying both strong solvers and generators that induce substantial difficulty. Conditioned on the hardest instances, the solver branch mutates existing heuristics, evaluates their behavioral improvements, and selects candidates that preserve both effectiveness and diversity. Symmetrically, conditioned on the current solver population, the instance branch mutates generator programs to construct increasingly challenging instances that expose solver vulnerabilities more sharply.

To guide the evolutionary updates, the framework extracts structured feedback from the observed behavioral differences between parent and mutated programs. This feedback is incorporated into subsequent optimization steps, enabling both populations to progressively adapt toward more effective search strategies and more challenging instance distributions. By repeatedly alternating mutation, selection, and feedback-driven adaptation on both sides, ACEvo induces a closed-loop adversarial dynamic in which solver robustness and instance hardness are progressively strengthened together. The procedure ultimately returns the strongest evolved solver and the hardest evolved instance generator obtained during co-evolution.

Algorithm 1: ACEvo: Adversarial Co-Evolution of Heuristic Solvers and Hard Instance Generators

Input: problem specification $\mathcal{T}$; heuristic language model L_{heur}; instance language model L_{inst}; initial prompts $P_{\text{heur}}^{(0)}$ and $P_{\text{inst}}^{(0)}$; heuristic population size M_h; generator population size M_g; evaluation budget B; number of iterations T

Output: best heuristic solver $h^\star$ and best hard-instance generator $g^\star$

Init *sample initial heuristic population* $\mathcal{H}^{(0)} = \{h_1^{(0)}, \ldots, h_{M_h}^{(0)}\} \sim L_{\text{heur}}(P_{\text{heur}}^{(0)})$
sample initial generator population $\mathcal{G}^{(0)} = \{g_1^{(0)}, \ldots, g_{M_g}^{(0)}\} \sim L_{\text{inst}}(P_{\text{inst}}^{(0)})$
initialize reflection memories $\mathcal{R}_h^{(0)} \leftarrow \emptyset$ *and* $\mathcal{R}_g^{(0)} \leftarrow \emptyset$
for $t \leftarrow 1$ **to** T **do**
  `// Phase 1: Cross-evaluation between solver and generator populations`
  foreach $g \in \mathcal{G}^{(t-1)}$ **do**
    sample an instance batch $\mathcal{I}_g = \{I_1, \ldots, I_B\}$ using g
  end
  foreach $h \in \mathcal{H}^{(t-1)}$ **do**
    evaluate h on all batches $\mathcal{I}_g$ generated by $\mathcal{G}^{(t-1)}$
    compute performance scores $f(h(I))$ and relative hardness values
  end
  select the currently hardest generator $\hat{g}^{(t-1)}$ according to induced solver gap
  select the currently strongest solver $\hat{h}^{(t-1)}$ according to performance on hard instances
  `// Phase 2: Heuristic evolution under adversarial hard instances`
  construct hard batch $\hat{\mathcal{I}}^{(t-1)}$ using $\hat{g}^{(t-1)}$
  initialize mutated heuristic set $\tilde{\mathcal{H}}^{(t)} \leftarrow \emptyset$
  foreach $h \in \mathcal{H}^{(t-1)}$ **do**
    compose heuristic mutation prompt using $\mathcal{T}$, h, $\hat{\mathcal{I}}^{(t-1)}$, and $\mathcal{R}_h^{(t-1)}$
    generate mutated solver $\tilde{h} \sim L_{\text{heur}}(\tilde{P}_{\text{heur}})$
    evaluate both h and $\tilde{h}$ on $\hat{\mathcal{I}}^{(t-1)}$
    derive evolutionary reflection r_h from their behavioral difference, failure modes, and performance traces
    add $\tilde{h}$ to $\tilde{\mathcal{H}}^{(t)}$
    update heuristic reflection memory with r_h
  end
  select high-performing and diverse solvers from $\mathcal{H}^{(t-1)} \cup \tilde{\mathcal{H}}^{(t)}$ to form $\mathcal{H}^{(t)}$
  `// Phase 3: Hard-instance evolution against current solvers`
  choose anchor solver set $\bar{\mathcal{H}}^{(t)} \subseteq \mathcal{H}^{(t)}$
  initialize mutated generator set $\tilde{\mathcal{G}}^{(t)} \leftarrow \emptyset$
  foreach $g \in \mathcal{G}^{(t-1)}$ **do**
    compose generator mutation prompt using $\mathcal{T}$, g, $\bar{\mathcal{H}}^{(t)}$, and $\mathcal{R}_g^{(t-1)}$
    generate mutated generator $\tilde{g} \sim L_{\text{inst}}(\tilde{P}_{\text{inst}})$
    sample candidate hard-instance batch $\mathcal{I}_{\tilde{g}}$ using $\tilde{g}$
    evaluate hardness of $\mathcal{I}_{\tilde{g}}$ against $\bar{\mathcal{H}}^{(t)}$
    derive evolutionary reflection r_g from hardness increase, structural changes, and solver failures
    add $\tilde{g}$ to $\tilde{\mathcal{G}}^{(t)}$
    update generator reflection memory with r_g
  end
  select harder and structurally diverse generators from $\mathcal{G}^{(t-1)} \cup \tilde{\mathcal{G}}^{(t)}$ to form $\mathcal{G}^{(t)}$
  `// Phase 4: Adversarial co-evolution`
  recompute cross-play between $\mathcal{H}^{(t)}$ and $\mathcal{G}^{(t)}$
  preserve stronger solvers and harder generators to sustain adversarial pressure
end
select $h^\star = \arg\min_{h \in \mathcal{H}^{(T)}} \text{HardInstanceGap}(h)$
select $g^\star = \arg\max_{g \in \mathcal{G}^{(T)}} \text{Hardness}(g \mid \mathcal{H}^{(T)})$
return $h^\star, g^\star$