ReEvo: Large Language Models as Hyper-Heuristics with Reflective Evolution

Haoran Ye [1 2] **Jiarui Wang** [3 2] **Zhiguang Cao** [4] **Guojie Song** [1 *]

Abstract

The omnipresence of NP-hard combinatorial optimization problems (COPs) compels domain experts to engage in trial-and-error heuristic design process. The long-standing endeavor of design automation has gained new momentum with the rise of large language models (LLMs). This paper introduces Language Hyper-Heuristics (LHHs), an emerging variant of Hyper-Heuristics that leverages LLMs for heuristic generation, featured by minimal manual intervention and open-ended heuristic spaces. To empower LHHs, we present Reflective Evolution (ReEvo), a generic searching framework that emulates the reflective design approach of human experts while much surpassing human capabilities with its scalable LLM inference, Internet-scale domain knowledge, and powerful evolutionary search. Evaluations across 12 COP settings show that 1) verbal reflections for evolution lead to smoother fitness landscapes, explicit inference of black-box COP settings, and better search results; 2) heuristics generated by ReEvo in minutes can outperform state-of-the-art human designs and neural solvers; 3) LHHs enable efficient algorithm design automation even when challenged with black-box COPs, demonstrating its potential for complex and novel real-world applications. Our code is available: https://github.com/ai4co/LLM-as-HH.

1. Introduction

NP-hard combinatorial optimization problems (COPs) pervade numerous real-world systems, each characterized by distinct constraints and objectives. The intrinsic complexity and heterogeneity of these problems consistently compel domain experts to develop heuristics through extensive trial and error for their approximate solutions (Hromkovič, 2013). Automation of heuristic designs represents a longstanding pursuit.

Prior endeavor in this direction includes Hyper-Heuristics (HHs) (Pillay & Qu, 2018) and Neural Combinatorial Optimization (NCO) (Berto et al., 2023). HHs build a heuristic from a set of predefined heuristics or heuristic components, and search for the best heuristic (combination). NCO solvers exploit deep representation learning to build parameterized heuristics from data. However, HHs are limited by heuristic spaces predefined by human experts (Pillay & Qu, 2018), while NCO is limited by the need for effective inductive bias (Drakulic et al., 2023), high computational demand (Joshi et al., 2020), and lack of interpretability and generalizability (Liu et al., 2023d).

The rise of large language models (LLMs) opens up new possibilities for heuristic design automation. Beyond their foundational understanding of programming logic and structure (Chen et al., 2023c; Liventsev et al., 2023; Madaan et al., 2023a), LLMs exhibit the ability to generate novel programs that lead to mathematical discoveries (Romera-Paredes et al., 2023). Preliminary success has also been observed in LLM-assisted heuristic designs for the Traveling Salesman Problem (TSP) (Liu et al., 2023b; 2024) and the online Bin Packing Problem (BPP) (Romera-Paredes et al., 2023). However, these two problems have been extensively studied, leaving a large amount of corpus pairing problem descriptions with their heuristic designs. The frequent exposure of these problem-heuristic pairs during LLM training may lead to their disproportionate representation in the heuristic space. We still lack a generic framework and reliable evaluations that support the use of LLMs as HHs for a wide range of COPs, especially the complex and black-box ones that permeate industrial scenarios without clear problem formulations.

This paper introduces the general concept of *Language Hyper-Heuristics (LHH)* to advance beyond preliminary attempts in individual COP settings. LHH constitutes an emerging variant of HH that utilizes LLMs for heuristic generations. It features minimal human intervention and open-ended heuristic spaces, showing promise to comprehensively shift the HH research paradigm. We elicit the power of LHH by presenting *Reflective Evolution (ReEvo)*. ReEvo couples humanoid reflections to boost the reasoning

[1] National Key Laboratory of General Artificial Intelligence, School of Intelligence Science and Technology, Peking University [2] AI4CO [3] Soochow University [4] Singapore Management University. Correspondence to: Guojie Song <gjsong@pku.edu.cn>.

Preprint.

capabilities of LLMs, with evolutionary computation for efficient exploration of heuristic spaces. It emulates human experts by incorporating short-term reflections over relative performance of two heuristics and the reasons behind such discrepancy, alongside long-term reflections gathered across iterations. Our dual-level reflections lead to more effective heuristic search and facilitate inferring underlying problem structures when lacking understanding of complex and black-box COPs.

We extensively evaluate ReEvo across 12 COP settings, involving 3 heuristic variants, 5 types of COP, and both white-box and black-box views of each COP. To provide black-box views of COPs, we prompt using general forms without leaking COP-related information. Through fitness landscape analysis and comparisons with prior LHHs, we demonstrate that the integration of reflections results in smoother neighborhood structures, effective inference of black-box COP settings, and better search results. Within minutes of evolution, ReEvo can deliver state-of-the-art (SOTA) solvers compared to their expert-designed and NCO counterparts. Its robust performance on black-box COPs showcases its potential to revolutionize algorithm design automation for complex and novel real-world applications.

We summarize our contributions as follows.

- We introduce the concept of Language Hyper-Heuristics (LHHs), which bridges emerging attempts using LLMs for heuristic generation with a methodological group that enjoys decades of development. This integration allows for a theoretical basis for future research and deployment in LHHs.
- We present Reflective Evolution (ReEvo), coupling humanoid reflections with evolutionary computation to elicit the power of LHHs.
- We advocate and propose black-box prompting for reliable evaluations of LHHs. To some extent, it prevents "data leakage" in the context of LHHs and ensures their robust performance for complex and novel real-world applications.
- Evaluations across 12 COP settings demonstrate the SOTA performance of ReEvo and ReEvo-enhanced solvers. Our code is publicly available.

2. Related Work

Traditional Hyper-Heuristics (HHs). Traditional HHs function by selecting the best performing heuristic from a predefined set (Drake et al., 2020) or by generating new heuristics through the combination of simpler heuristic components (Duflo et al., 2019). With a theoretical basis established (Pillay & Qu, 2018), HHs offer a higher level of generality in solving various optimization problems (Zhu et al., 2023; Zambrano-Gutierrez et al., 2023; Guerriero & Saccomanno, 2023; Lim et al., 2023; Zhang et al., 2023; Mohammad Hasani Zade et al., 2023), but are limited by the heuristic space predefined by human experts.

Neural Combinatorial Optimization (NCO). Recent advances of NCO show promise in learning end-to-end solutions for COPs (Bengio et al., 2021; Yang & Whinston, 2023). NCO can be regarded as a variant of HH, wherein the heuristic space is defined by neural architectures, and solution pipelines and training algorithms serve to search within this heuristic space. Each trained neural network, under certain solution pipelines, represents a distinct heuristic. From this perspective, recent advancements in NCO HHs have led to better-aligned neural architectures (Jin et al., 2023; Fu Luo, 2023; Kim et al., 2023; Son et al., 2023) and advanced solution pipelines (Ma et al., 2023a; Li et al., 2023; Xiao et al., 2023; Ye et al., 2024) to define effective heuristic space, and improved training algorithms to explore the heuristic space (Kim et al., 2022; Jiang et al., 2023; Drakulic et al., 2023; Sun & Yang, 2023; Gao et al., 2023; Xiao et al., 2024; Wang et al., 2024), while targeting increasingly broader applications (Chen et al., 2023b; Zhou et al., 2023; Ma et al., 2023c). However, appropriate inductive bias, necessary for effective NCO solutions, entails manual tuning of model architectures and training algorithms. Also, NCO HHs are challenged by computationally expensive training, typically problem and scale-specific models, demanding hardware requirements for deployment, and a lack of interpretability and generalizability (Liu et al., 2023d).

LLMs for Code Generation and Optimization. The rise of LLMs introduces new prospects for diverse fields (Xi et al., 2023; Wang et al., 2023c; Zhao et al., 2023; Ji et al., 2023). Among others, code generation capabilities of LLMs are utilized for code debugging (Chen et al., 2023c; Liventsev et al., 2023), enhancing code performance (Madaan et al., 2023a), solving algorithmic competition challenges (Li et al., 2022; Shinn et al., 2023), robotics (Liang et al., 2023; Wang et al., 2023b), and general task solving (Yang et al., 2023b; Zhang et al., 2022). Interleaving LLM generations with evaluations yields powerful methods for prompt optimization (Zhou et al., 2022; Wang et al., 2023d; Guo et al., 2023), reinforcement learning (RL) reward design (Ma et al., 2023b), algorithmic (self-)improvement (Zelikman et al., 2023; Liu et al., 2023c;a), neural architecture search (Chen et al., 2023a), and general solution optimization (Yang et al., 2023a; Brooks et al., 2023; Wang et al., 2023a), with many under evolutionary frameworks (Wu et al., 2024; Hemberg et al., 2024; Chao et al., 2024). Most related to ReEvo, Liu et al. (2023b; 2024) and Romera-Paredes et al. (2023) leverage LLMs to develop heuristics for TSP and online BPP, respectively. We go beyond these individual attempts to propose generic LHH, along with broader applications, more reliable evaluations, and improved heuristics. In addition, ReEvo contributes to a smoother fitness landscape, showing potential to enhance other tasks involving LLMs for code generation and optimization.

Self-Reflections of LLMs. Shinn et al. (2023) propose to reinforce language agents via linguistic feedback, which is subsequently harnessed for various tasks (Madaan et al., 2023b; Wang et al., 2023e). While Shinn et al. (2023) leverage binary rewards indicating passing or failing test cases in programming tasks, ReEvo extends the scope of verbal RL feedback to comparative analysis of two heuristics, which can be analogous to verbal gradient information (Pryzant et al., 2023) within heuristic spaces. Moreover, ReEvo incorporates reflection within an evolutionary framework, presenting a powerful integration of self-reflection and evolutionary computation.

3. Language Hyper-Heuristics

HHs explore a search space of heuristic configurations to select or generate heuristics for the underlying optimization problem. This leads to a dual-level framework formally defined as follows.

Definition 3.1 (Hyper-Heuristic)**.** In the optimization problem P with solution space S and objective function $f : S \to \mathbb{R}$, a Hyper-Heuristic (HH) functions within a heuristic space H to identify the optimal heuristic h^*. This is achieved by optimizing a meta-objective function $F : H \to \mathbb{R}$, which evaluates the effectiveness of heuristics in producing high-quality solutions in S. Formally, HH aims to find h^* such that $F(h^*) = \min_{h \in H} F(h)$, indirectly optimizing f in P via heuristic generation and selection.

Depending on how the heuristic space H is defined, traditional HHs can be categorized into selection and generation HHs. Here, we introduce a novel variant of HHs, Language Hyper-Heuristics (LHH), where heuristics in space H are generated by LLMs. Compared with traditional HHs, LHHs do not require problem dependent low-level heuristics or heuristic components predefined by human experts. Instead, LHHs leverage the capabilities of LLMs to explore an open-ended heuristic space, showing potential to push the boundary of human knowledge. We formally define LHHs as follows.

Definition 3.2 (Language Hyper-Heuristic)**.** A Language Hyper-Heuristic (LHH) is an HH variant where heuristics in space H are generated by LLMs. It aims to optimize a meta-objective function F for a problem P.

In this work, we define the meta-objective function F as the expected performance of a heuristic h on problem instances following certain distributions, which is estimated by its average performance on a dataset of problem instances.

4. Reflective Evolution

ReEvo is schematically illustrated in Figure 1. Under an evolutionary framework, LLMs assume two types of roles: a *generator LLM* for generating individuals and a *reflector LLM* for guiding the generation with reflections. ReEvo, as an LHH, features a distinct individual encoding, where each individual is the code snippet of a heuristic. Its evolution begins with population initialization, followed by five iterative steps: selection, short-term reflection, crossover, long-term reflection, and elitist mutation. We interleave evaluations both after crossover and after mutation.

Individual Encoding. ReEvo optimizes toward best-performing heuristics via an evolutionary process, specifically a Genetic Algorithm (GA), but diverges from conventional evolutionary algorithms in two key aspects. First, the individuals in ReEvo are code snippets of heuristics designed to solve COPs, rather than being direct solutions to the COPs themselves. Second, these individuals are generated by LLMs, with their generation not constrained by any predefined encoding format, except for adhering to a specified function signature.

Population Initialization. ReEvo initializes a heuristic population by prompting a generator LLM with task specifications and a seed heuristic. The task specification contains COP descriptions (if available), heuristic designation, and heuristic functionality. Inclusion of a seed heuristic is optional. Seed heuristics, either trivial or expertly crafted to improve upon, can provide in-context examples that encourage valid heuristic generation and bias the search toward potentially more promising directions.

Selection. ReEvo selects parent pairs from successfully executed heuristics at random (Liu et al., 2023b), while avoiding pairing heuristics with an identical meta-objective value F. This random selection strategy encourages exploration, countering the tendency for premature convergence observed in LHHs. Alternative techniques can be adopted to preserve exploration, such as island-based evolution in FunSearch (Romera-Paredes et al., 2023) when a large number of heuristic generations (e.g., on the order of 10^6 in FunSearch) are feasible.

Short-Term Reflection. For each pair of heuristic parents, ReEvo provides the reflector LLM with their relative performance and prompts it to reflect upon such discrepancies and give hints accordingly. Shinn et al. (2023) discuss how reflections can provide LLMs with RL reward signals in a verbal format for code generation tasks. While Shinn et al. (2023) limit the scope to binary reward signals—either

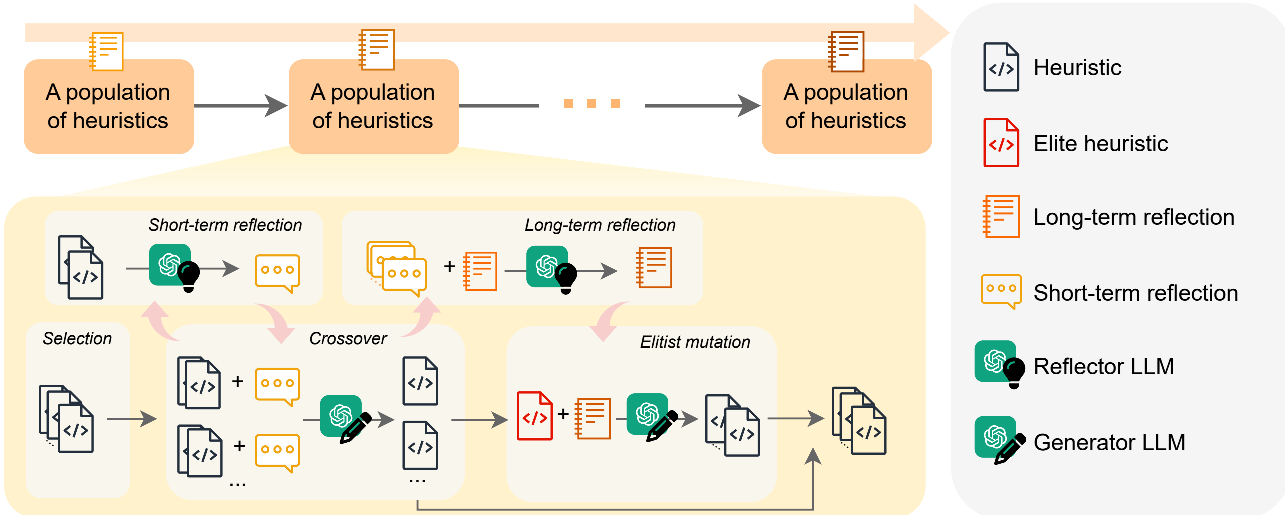

Figure 1. A schematic illustration of ReEvo. **Top**: ReEvo maintains and evolves a population of heuristics, where insights and knowledge are verbalized as long-term reflections and accumulated throughout iterations. **Bottom**: A ReEvo iteration contains five sequential steps: selection, short-term reflection, crossover, long-term reflection, and elitist mutation.

passing or failing test cases—ReEvo reflects by performing comparative analyses of two heuristics. This approach can be analogous to providing *"verbal gradient"* information within heuristic spaces.

Crossover. ReEvo prompts the generator LLM to generate an offspring heuristic, given task specifications, a pair of parent heuristics, explicit indications of their relative performance, short-term reflections over the pair, and generation instructions.

Long-Term Reflection. ReEvo accumulates expertise in improving heuristics via long-term reflections. The reflector LLM, given the previous long-term reflections and newly gained short-term reflections, is prompted to reflect on them and give hints for improved heuristic design. Long-term reflections can be initialized with manually defined hints or as an empty string.

Elitist Mutation. ReEvo employs an elitist mutation approach, wherein the generator LLM is prompted to improve the elite individual—the all-time best heuristic—based on long-term reflections. A mutation prompt consists of task specifications, an elite heuristic, long-term reflections, and generation instructions.

Evaluation. We evaluate F of an individual heuristic using the objective value averaged over a training dataset. In this work, evaluations are performed subsequent to both crossover and mutation; that is, all generated heuristics are evaluated, maximizing the utilization of LLM API calls. A validation dataset is used to select the best among the elite solutions of multiple ReEvo runs. Finally, the best heuristic is evaluated on a test dataset in our experiments.

5. Fitness Landscape Analysis

The fitness landscape of an HH is used to understand the structure and characteristics of its search space $F : H \to \mathbb{R}$ (Ochoa et al., 2009). This understanding is essential for designing more effective search algorithms.

Traditionally, the neighborhood of a solution is defined as a set of solutions that can be reached from the current solution by a single move of a certain heuristic. However, LHHs feature a probabilistic nature and open-ended search space. Therefore, we redefine its neighborhood as follows.

Definition 5.1 (Neighborhood)**.** Let LLM denote an LHH, x a specific prompt, and h_c the current heuristic. Given LLM and x, the neighborhood of h_c is defined as a set $\mathcal{N}$, where each element $h \in \mathcal{N}$ represents a heuristic that LLM can mutate h_c into, in response to x. Formally, the neighborhood is given by:

$$\mathcal{N}(h_c) = \{h \mid LLM(h|h_c, x) > \xi\}.$$

Here, $LLM(h|h_c, x)$ denotes the probability distribution of generating h after prompting with x and the heuristic h_c, and ξ is a small threshold value. In practice, the neighborhood can be approximated by sampling from the distribution $LLM(h|h_c, x)$ for a large number of times.

Among the metrics used for landscape analysis, autocorrelation is extensible to LHHs under our definition of neighborhood. Autocorrelation reflects the ruggedness of a land-

Table 1. Autocorrelation analysis of ReEvo.

	Correlation length ↑	Objective ↓
Random	0.35 ± 0.05	-
w/o reflection	0.28 ± 0.07	12.08 ± 7.15
w/ reflection	**1.28** ± 0.62	**6.53** ± 0.60

scape, which is related to the difficulty of an optimization problem (Ochoa et al., 2009; Hordijk, 1996). It is formally defined as follows.

Definition 5.2 (Autocorrelation)**.** Autocorrelation refers to the measure of correlation structure of a fitness landscape. It is derived from the autocorrelation function of a time series of fitness values, which are generated by a random walk on the landscape via neiboring points. It is given by:

$$r_i = \frac{\sum_{t=1}^{T-i}(f_t - \bar{f})(f_{t+i} - \bar{f})}{\sum_{t=1}^{T}(f_t - \bar{f})^2},$$

where $\bar{f}$ is the mean fitness of the points visited, T is the size of the random walk, and i is the time lag between points in the walk.

Based on the autocorrelation function, correlation length is defined below (Weinberger, 1990).

Definition 5.3 (Correlation Length)**.** Given an autocorrelation function, the correlation length l is formulated as $l = -1/\ln(|r_1|)$ for $r_1 \neq 0$. It reflects the ruggedness of a landscape, and smaller values indicate a more rugged landscape.

We conduct random walks based on the neighborhood established with our crossover prompt either with or without short-term reflections. To generate random walks, we set the population size to 1 and skip invalid heuristics. The selection always picks the current and last heuristics for short-term reflection and crossover, and we do not implement mutation.

We perform an autocorrelation analysis using *TSP50_ACO* settings (detailed in Section 6), and additionally include a baseline of uniform and random data generation. Table 1 presents the correlation length and the average objective value of the random walks. The correlation length is averaged over 3 runs each with 40 random walk steps, while the objective value is averaged over all 3×40 heuristics. The results verify that implementing reflection leads to less rugged landscape and better search results. As discussed in Section 4, reflections can indeed function as verbal gradients that leads to better neighborhood structures.

Table 2. Parameters of ReEvo.

Parameter	Value
LLM (generator and reflector)	gpt-3.5-turbo-0613
LLM temperature (generator and reflector)	1
Population size	10
Maximum number of evaluations	100
Mutation rate	0.5

6. Comparative Evaluations

6.1. Experimental Setup

Benchmarks. We showcase ReEvo by evolving heuristics for solution improvement, stochastic solution sampling, and deterministic solution construction: (1) *TSP_GLS*: ReEvo evolves heuristics to enhance the perturbation phase of Guided Local Search (GLS) for TSP. (2) *COP_ACO*: ReEvo evolves heuristics for Ant Colony Optimization wherein solutions are stochastically sampled. We experiment with five COPs. (3) *TSP_constructive*: ReEvo evolves deterministic constructive heuristics for next-node selection in TSP. TSPLIB instances are used for the evaluations in Table 6. Besides that, all COP instances used are synthetic (Ye et al., 2023).

Prompts and Parameters. Our prompts are gathered in Appendix A. We use the parameters in Table 2 for all ReEvo runs. Problem-specific parameters are given in Appendix C.

Baselines. We compare ReEvo and its generated heuristics with a comprehensive set of baselines.

- Heuristics designed by human experts (Skinderowicz, 2022; Cai et al., 2022; Sohrabi et al., 2021; Fidanova, 2020; Levine & Ducatelle, 2004; Arnold & Sörensen, 2019).
- GHPP, a traditional HH based on genetic programming (Duflo et al., 2019).
- NCO solvers (Hudson et al., 2022; Ye et al., 2023; Sui et al., 2023; Ma et al., 2023a).
- LHHs (Liu et al., 2023b; 2024). We reproduce AEL and give its implementation details in Appendix B.

Evaluation Pipelines. Each ReEvo run "trains" a heuristic on a training dataset of 10 instances. The best heuristic for each run is evaluated on a validation dataset of 64 instances. We perform 3 ReEvo runs and pick the best-performing heuristic during validation for final tests using 64 instances. The best ReEvo-generated heuristics are collected in Appendix D.

Table 3. Evaluation results of different local search (LS) variants. We report optimality gaps and per-instance execution time.

Method	Type	TSP20		TSP50		TSP100		TSP200	
		Opt. gap (%)	Time (s)	Opt. gap (%)	Time (s)	Opt. gap (%)	Time (s)	Opt. gap (%)	Time (s)
NeuOpt * (Ma et al., 2023a)	LS+RL	0.000	0.124	0.000	1.32	0.027	2.67	0.403	4.81
GNNGLS (Hudson et al., 2022)	GLS+SL	0.000	0.116	0.052	3.83	0.705	6.78	3.522	9.92
NeuralGLS † (Sui et al., 2023)	GLS+SL	0.000	10.005	0.003	10.01	0.470	10.02	3.622	10.12
AEL-GLS † (Liu et al., 2024)	GLS+LHH	0.000	1.015	0.000	2.90	0.032	7.15	-	-
KGLS ‡ (Arnold & Sörensen, 2019)	GLS	0.004	0.001	0.017	0.03	0.002	1.55	0.284	2.52
KGLS-ReEvo ‡ (ours)	GLS+LHH	**0.000**	**0.001**	**0.000**	**0.03**	**0.000**	**1.55**	**0.216**	**2.52**

*: All instances are solved in one batch and total time is averaged over the dataset. D2A=1; T=500, 4000, 5000, and 5000 for 4 problem sizes, respectively.
†: The results are drawn from (Sui et al., 2023; Liu et al., 2024) due to closed source code, model, or lack of algorithm details.
‡: They are based on our own GLS implementation.

Cost and Hardware. Heuristics of the same generation are generated, reflected upon, and evaluated in parallel. A single ReEvo run takes about 5 minutes on an average PC, costing about 0.2$. The evaluations of heuristics, when involving runtime comparisons, are conducted using a single core of AMD EPYC 7742 CPU and an NVIDIA GeForce RTX 3090 GPU.

6.2. Evaluation Results

6.2.1. Heuristics for Solution Improvement

We evolve heuristics for solution improvement under a Guided Local Search (GLS) framework (Arnold & Sörensen, 2019). GLS interleaves local search with solution perturbation. The perturbation is guided by heuristics to maximize its utility. ReEvo searches for the heuristic that leads to the best GLS performance.

We implement the best heuristic generated by ReEvo within KGLS (Arnold & Sörensen, 2019) and refer to such coupling as KGLS-ReEvo. In Table 3, we compare KGLS-ReEvo with the original KGLS, other GLS variants (Hudson et al., 2022; Sui et al., 2023; Liu et al., 2024), and SOTA NCO method that learns to improve a solution (Ma et al., 2023a). The results show that ReEvo significantly improves KGLS and outperforms SOTA baselines.

It is worth noting that the creation of KGLS-ReEvo requires only 20 minutes of 3 ReEvo runs on TSP200, while Liu et al. (2024) report that the AEL-GLS takes 2 days. In addition, we use a single heuristic for TSP20 to 200, demonstrating its strong generalization. In contrast, NCO baselines require training models specific to each problem scale, hindered by huge training costs and limited generalizability.

6.2.2. Heuristics for Stochastic Solution Sampling

Solutions to COPs can be stochastically sampled, with heuristics indicating the promise of each solution component and biasing the sampling. Ant Colony Optimization (ACO), which interleaves stochastic solution sampling with pheromone update, builds on this idea. We generate such heuristics for five different COPs: Traveling Salesman Problem (TSP), Capacitated Vehicle Routing Problem (CVRP), Orienteering Problem (OP), Multiple Knapsack Problem (MKP), and Bin Packing Problem (BPP). Each COP is described using white-box and black-box prompts separately. For white-box settings, we include the COP definitions and function descriptions in our prompt. For black-box settings, we do not reveal any information related to the COPs and prompt LHHs in general forms (e.g., "edge_attr" in place of "distance_matrix"). Black-box settings can provide more reliable evaluations to ascertain whether LHHs can design effective heuristics for novel and complex applications, rather than simply retrieving code tailored for prominent COPs from their parameterized knowledge.

Under the ACO framework, we evaluate the best ReEvo-generated heuristics against the expert-designed ones and neural heuristics specifically learned for ACO (Ye et al., 2023). The evolution curves displayed in Figure 2 verify the consistent superiority of ReEvo across COPs and problem sizes. Notably, on 3 out of 5 COPs, ReEvo outperforms DeepACO (Ye et al., 2023) even when the latter overfits the test problem size (TSP50, OP50, and MKP100). The advantages of ReEvo grow as the distributional shift increases for neural heuristics.

On the other hand, Table 4 gathers the evaluation results of the best heuristics generated by ReEvo and AEL. As an LHH, ReEvo demonstrates superior search effectiveness, especially when challenged with black-box prompts. Besides the better neighborhood structure of ReEvo (Section 5), reflections facilitate explicit verbal inference of underlying black-box COP structures. Take black-box TSP as an example. We observe short-term reflections such as: *"Hint: The edge attributes seem to have an inverse relationship with the black-box objective value."* and *"In the worse code, the heuristics prioritize edges with higher attribute values as more promising; in the better code, the heuristics prioritize edges with smaller attribute values as more promising."*. They correctly infer the derivation of TSP objective. Also, long-term reflections like *"Incorporate the correlation co-*

Table 4. ACO evolution results using heuristics designed by human experts (Skinderowicz, 2022; Cai et al., 2022; Sohrabi et al., 2021; Fidanova, 2020; Levine & Ducatelle, 2004), AEL (Liu et al., 2023b), and ReEvo. Each of the 5 COPs is presented to LLMs as a white-box and a black-box problem separately. Baselines for white-box and black-box settings are expert-designed heuristics and trivial uniform heuristics (e.g. an all-ones matrix), respectively. "Degradation" represents the performance drop caused by turning a problem from a white box into a black box. For each problem setting, we perform three independent LHH runs on the smallest problem size among the three, then evaluate the elite heuristic of each run on three problem sizes.

	White-box			Black-box			Degradation ↓		
TSP ↓	TSP20	TSP50	TSP100	TSP20	TSP50	TSP100	TSP20	TSP50	TSP100
Human / Uniform	3.90	6.54	10.11	6.94	20.13	44.39	-	-	-
AEL	3.90 ± 0.05	6.08 ± 0.20	8.97 ± 0.34	5.47 ± 1.10	14.25 ± 4.82	29.66 ± 11.78	1.57	8.17	20.69
ReEvo	**3.87** ± 0.00	**5.84** ± 0.02	**8.40** ± 0.02	**3.88** ± 0.01	**6.07** ± 0.30	**8.96** ± 0.82	**0.01**	**0.23**	**0.56**
CVRP ↓	CVRP20	CVRP50	CVRP100	CVRP20	CVRP50	CVRP100	CVRP20	CVRP50	CVRP100
Human / Uniform	8.98	18.34	30.02	11.06	26.39	51.45	-	-	-
AEL	4.86 ± 0.03	9.62 ± 0.39	16.67 ± 0.90	8.32 ± 1.99	20.42 ± 5.31	40.35 ± 11.21	3.46	10.80	23.68
ReEvo	**4.81** ± 0.05	**9.20** ± 0.18	**15.89** ± 0.27	**5.46** ± 0.84	**11.53** ± 2.92	**21.11** ± 6.83	**0.65**	**2.33**	**5.22**
OP ↑	OP50	OP100	OP200	OP50	OP100	OP200	OP50	OP100	OP200
Human / Uniform	13.52	24.34	36.89	6.17	7.64	8.69	-	-	-
AEL	**15.05** ± 0.10	29.69 ± 0.67	**52.25** ± 2.54	8.45 ± 1.91	11.14 ± 3.12	12.23 ± 3.19	6.60	18.55	40.02
ReEvo	15.03 ± 0.16	**29.71** ± 0.52	52.16 ± 1.72	**10.53** ± 0.34	**15.24** ± 1.34	**16.59** ± 2.81	**4.50**	**14.47**	**35.57**
MKP ↑	MKP100	MKP300	MKP500	MKP100	MKP300	MKP500	MKP100	MKP300	MKP500
Human / Uniform	16.77	51.31	122.35	13.32	32.15	81.59	-	-	-
AEL	**17.00** ± 0.09	**53.71** ± 0.27	129.56 ± 0.88	15.53 ± 0.33	44.07 ± 0.99	110.31 ± 2.34	1.47	9.64	19.25
ReEvo	16.97 ± 0.09	53.61 ± 0.50	**130.10** ± 1.29	**16.92** ± 0.13	**52.55** ± 1.71	**127.29** ± 3.70	**0.06**	**1.07**	**2.81**
BPP ↓	BPP120	BPP500	BPP1000	BPP120	BPP500	BPP1000	BPP120	BPP500	BPP1000
Human / Uniform	51.05	215.73	433.08	51.89	219.64	441.09	-	-	-
AEL	50.58 ± 0.08	211.15 ± 0.47	422.60 ± 1.37	51.21 ± 0.25	214.78 ± 1.56	429.82 ± 3.59	**0.64**	3.63	7.22
ReEvo	**50.01** ± 0.18	**207.31** ± 0.98	**413.74** ± 2.41	**50.66** ± 0.43	**210.83** ± 3.04	**420.81** ± 6.81	0.66	**3.52**	**7.06**

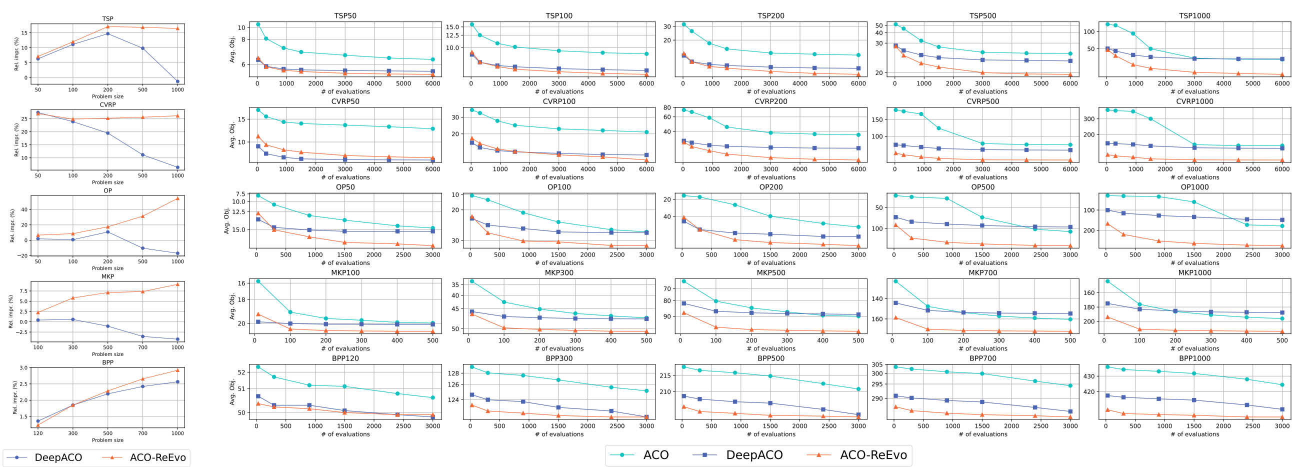

Figure 2. Comparative evaluations of ACO using expert-designed heuristics (Skinderowicz, 2022; Cai et al., 2022; Sohrabi et al., 2021; Fidanova, 2020; Levine & Ducatelle, 2004), SOTA neural heuristics (Ye et al., 2023), and ReEvo heuristics. For each COP, the same neural heuristic or the ReEvo heuristic is applied across all problem sizes; both heuristics are trained exclusively on the smallest problem size among the five. **Left**: Relative performance improvement of DeepACO and ReEvo over human baselines w.r.t. problem sizes. **Right**: ACO evolution curves, plotting the all-time best objective value w.r.t. the number of solution evaluations. The curves are averaged over 3 runs in which only small variances are observed (e.g., ~ 0.01 for TSP50).

efficient between attributes" directly leads to generating the best-performing *TSP_ACO* heuristic. Notably, the best heuristics for TSP and MKP are generated under black-box views, strongly indicating the robustness of ReEvo for novel and complex real-world applications.

6.2.3. Heuristics for Deterministic Solution Construction

Heuristics can be used for deterministic solution construction, via sequentially assigning values to each decision variables. We evaluate the constructive heuristic for TSP generated by ReEvo on both the sythetic benchmarks and the TSPLIB benchmarks, with results presented in Tables 5 and 6, respectively. We show that ReEvo can generate heuristics that are better than those of AEL and GHPP.

Table 5. Comparisons of constructive heuristics on TSP20-1000 datasets, each with 64 uniformly sampled instances (Liu et al., 2023b). We use the best heuristic reported in (Liu et al., 2023b) for AEL and our best found heuristic for ReEvo. For each test instance, a same starting node is randomly selected for all heuristics.

TSP size	Nearest Neighbour	AEL-best (Liu et al., 2023b)	ReEvo-best
20	4.45	**4.08**	4.09
50	6.89	6.23	**6.23**
100	9.65	8.60	**8.57**
200	13.42	12.31	**12.09**
500	20.65	19.24	**18.95**
1000	29.18	27.34	**26.79**

Table 6. Comparisons of constructive heuristics designed by human, genetic programming (Duflo et al., 2019), and ReEvo. We report the average optimality gap of each instance, where the baseline results are drawn from (Duflo et al., 2019) and the results of ReEvo are averaged over 3 runs with different starting nodes.

Instance	Nearest Neighbour	GHPP (Duflo et al., 2019)	ReEvo
ts225	16.8	7.7	**6.6**
rat99	21.8	14.1	**12.4**
rl1889	23.7	21.1	**17.5**
u1817	22.2	21.2	**16.6**
d1655	23.9	18.7	**17.5**
bier127	23.3	15.6	**10.8**
lin318	25.8	**14.3**	16.6
eil51	32.0	10.2	**6.5**
d493	24.0	15.6	**13.4**
kroB100	26.3	14.1	**12.2**
kroC100	25.8	16.2	**15.9**
ch130	25.7	14.8	**9.4**
pr299	31.4	**18.2**	20.6
fl417	32.4	22.7	**19.2**
d657	29.7	16.3	**16.0**
kroA150	26.1	15.6	**11.6**
fl1577	25.0	17.6	**12.1**
u724	28.5	**15.5**	16.9
pr264	17.9	24.0	**16.8**
pr226	24.6	**15.5**	18.0
pr439	27.4	21.4	**19.3**
Avg. opt. gap	25.4	16.7	**14.6**

6.3. Ablation Studies

We conduct ablation studies on *TSP-ACO* settings under both white-box and black-box views. We investigate ReEvo without reflections, long-term reflections, short-term reflections, crossover, and mutation. The results are presented in Table 7. It is shown that all components of ReEvo positively contribute to its performance.

Table 7. Ablation studies of ReEvo on *TSP-ACO*. The results of each setting are obtained over 3 runs.

White-box	TSP50	TSP100
w/o reflections	6.11 ± 0.17	9.21 ± 0.41
w/o long-term reflections	5.91 ± 0.05	8.61 ± 0.21
w/o short-term reflections	5.88 ± 0.01	8.46 ± 0.01
w/o crossover	5.86 ± 0.00	8.45 ± 0.02
w/o mutation	5.95 ± 0.03	8.83 ± 0.09
ReEvo	**5.84** ± 0.02	**8.40** ± 0.02
Black-box	**TSP50**	**TSP100**
w/o reflections	6.35 ± 0.29	9.67 ± 0.68
w/o long-term reflections	6.27 ± 0.34	9.32 ± 0.71
w/o short-term reflections	6.08 ± 0.33	9.05 ± 0.83
w/o crossover	6.20 ± 0.50	9.47 ± 1.40
w/o mutation	6.11 ± 0.27	9.34 ± 0.96
ReEvo	**6.07** ± 0.30	**8.96** ± 0.82

7. Conclusion

This paper introduces Language Hyper-Heuristics (LHHs), a rising variant of HHs, alongside Reflective Evolution (ReEvo), an evolutionary framework to elicit the power of LHHs. Across 12 COP settings, ReEvo introduces notable improvements over previous SOTA, showcasing robust LHH effectiveness, stronger heuristic performance, and emergent abilities to tackle black-box COPs.

LHHs feature minimal manual intervention and an open-ended heuristic space compared to traditional HHs. Therefore, we expect ReEvo to lead a comprehensive paradigm shift within HH research. In comparison to NCO solvers, LHHs demonstrate superior computational efficiency, enhanced generalizability of learned heuristics, and greater interpretability by delivering readable code snippets. We anticipate that the heuristics generated by ReEvo will establish strong baselines for future NCO research.

The development of LHHs is still at its emerging stage. We believe that it is promising to explore their broader applications, better dual-level optimization architectures, and solid theoretical foundations. Beyond the scope of LHHs, we hope that ReEvo can enrich the landscape of Evolutionary Computation (EC), by showing that genetic cues can be interpreted and verbalized using LLMs.

References

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

Bengio, Y., Lodi, A., and Prouvost, A. Machine learning for combinatorial optimization: a methodological tour d'horizon. *European Journal of Operational Research*, 290(2):405–421, 2021.

Berto, F., Hua, C., Park, J., Kim, M., Kim, H., Son, J., Kim, H., Kim, J., and Park, J. RL4CO: a unified reinforcement learning for combinatorial optimization library. In *NeurIPS 2023 Workshop: New Frontiers in Graph Learning*, 2023.

Brooks, E., Walls, L. A., Lewis, R., and Singh, S. Large language models can implement policy iteration. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023.

Cai, J., Wang, P., Sun, S., and Dong, H. A dynamic space reduction ant colony optimization for capacitated vehicle routing problem. *Soft Computing*, 26(17):8745–8756, 2022.

Chao, W., Zhao, J., Jiao, L., Li, L., Liu, F., and Yang, S. A match made in consistency heaven: when large language models meet evolutionary algorithms, 2024.

Chen, A., Dohan, D. M., and So, D. R. Evoprompting: Language models for code-level neural architecture search. *arXiv preprint arXiv:2302.14838*, 2023a.

Chen, J., Wang, J., Zhang, Z., Cao, Z., Ye, T., and Siyuan, C. Efficient meta neural heuristic for multi-objective combinatorial optimization. In *Advances in Neural Information Processing Systems*, 2023b.

Chen, X., Lin, M., Schärli, N., and Zhou, D. Teaching large language models to self-debug. *arXiv preprint arXiv:2304.05128*, 2023c.

Drake, J. H., Kheiri, A., Özcan, E., and Burke, E. K. Recent advances in selection hyper-heuristics. *European Journal of Operational Research*, 285(2):405–428, 2020.

Drakulic, D., Michel, S., Mai, F., Sors, A., and Andreoli, J.-M. Bq-nco: Bisimulation quotienting for efficient neural combinatorial optimization. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023.

Duflo, G., Kieffer, E., Brust, M. R., Danoy, G., and Bouvry, P. A gp hyper-heuristic approach for generating tsp heuristics. In *2019 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pp. 521–529. IEEE, 2019.

Fidanova, S. Hybrid ant colony optimization algorithm for multiple knapsack problem. In *2020 5th IEEE International Conference on Recent Advances and Innovations in Engineering (ICRAIE)*, pp. 1–5. IEEE, 2020.

Fu Luo, Xi Lin, F. L. Q. Z. Z. W. Neural combinatorial optimization with heavy decoder: Toward large scale generalization. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023.

Gao, C., Shang, H., Xue, K., Li, D., and Qian, C. Towards generalizable neural solvers for vehicle routing problems via ensemble with transferrable local policy. *arXiv preprint arXiv:2308.14104*, 2023.

Guerriero, F. and Saccomanno, F. P. A hierarchical hyper-heuristic for the bin packing problem. *Soft Computing*, 27(18):12997–13010, 2023.

Guo, Q., Wang, R., Guo, J., Li, B., Song, K., Tan, X., Liu, G., Bian, J., and Yang, Y. Connecting large language models with evolutionary algorithms yields powerful prompt optimizers. *arXiv preprint arXiv:2309.08532*, 2023.

Hemberg, E., Moskal, S., and O'Reilly, U.-M. Evolving code with a large language model, 2024.

Hordijk, W. A measure of landscapes. *Evolutionary computation*, 4(4):335–360, 1996.

Hromkovič, J. *Algorithmics for hard problems: introduction to combinatorial optimization, randomization, approximation, and heuristics*. Springer Science & Business Media, 2013.

Hudson, B., Li, Q., Malencia, M., and Prorok, A. Graph neural network guided local search for the traveling salesperson problem. In *International Conference on Learning Representations*, 2022.

Ji, J., Qiu, T., Chen, B., Zhang, B., Lou, H., Wang, K., Duan, Y., He, Z., Zhou, J., Zhang, Z., et al. Ai alignment: A comprehensive survey. *arXiv preprint arXiv:2310.19852*, 2023.

Jiang, Y., Cao, Z., Wu, Y., Song, W., and Zhang, J. Ensemble-based deep reinforcement learning for vehicle routing problems under distribution shift. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023.

Jin, Y., Ding, Y., Pan, X., He, K., Zhao, L., Qin, T., Song, L., and Bian, J. Pointerformer: Deep reinforced multi-pointer transformer for the traveling salesman problem. *arXiv preprint arXiv:2304.09407*, 2023.

Joshi, C. K., Cappart, Q., Rousseau, L.-M., and Laurent, T. Learning the travelling salesperson problem requires rethinking generalization. *arXiv preprint arXiv:2006.07054*, 2020.

Kim, M., Park, J., and Park, J. Sym-nco: Leveraging symmetricity for neural combinatorial optimization. *Advances in Neural Information Processing Systems*, 35: 1936–1949, 2022.

Kim, M., Yun, T., Bengio, E., Zhang, D., Bengio, Y., Ahn, S., and Park, J. Local search gflownets. *arXiv preprint arXiv:2310.02710*, 2023.

Levine, J. and Ducatelle, F. Ant colony optimization and local search for bin packing and cutting stock problems. *Journal of the Operational Research society*, 55(7):705–716, 2004.

Li, Y., Choi, D., Chung, J., Kushman, N., Schrittwieser, J., Leblond, R., Eccles, T., Keeling, J., Gimeno, F., Dal Lago, A., et al. Competition-level code generation with alphacode. *Science*, 378(6624):1092–1097, 2022.

Li, Y., Guo, J., Wang, R., and Yan, J. T2t: From distribution learning in training to gradient search in testing for combinatorial optimization. In *Advances in Neural Information Processing Systems*, 2023.

Liang, J., Huang, W., Xia, F., Xu, P., Hausman, K., Ichter, B., Florence, P., and Zeng, A. Code as policies: Language model programs for embodied control. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 9493–9500. IEEE, 2023.

Lim, K. C. W., Wong, L.-P., and Chin, J. F. Simulated-annealing-based hyper-heuristic for flexible job-shop scheduling. *Engineering Optimization*, 55(10):1635–1651, 2023.

Liu, F., Lin, X., Wang, Z., Yao, S., Tong, X., Yuan, M., and Zhang, Q. Large language model for multi-objective evolutionary optimization. *arXiv preprint arXiv:2310.12541*, 2023a.

Liu, F., Tong, X., Yuan, M., and Zhang, Q. Algorithm evolution using large language model. *arXiv preprint arXiv:2311.15249*, 2023b.

Liu, F., Tong, X., Yuan, M., Lin, X., Luo, F., Wang, Z., Lu, Z., and Zhang, Q. An example of evolutionary computation + large language model beating human: Design of efficient guided local search, 2024.

Liu, S., Chen, C., Qu, X., Tang, K., and Ong, Y.-S. Large language models as evolutionary optimizers. *arXiv preprint arXiv:2310.19046*, 2023c.

Liu, S., Zhang, Y., Tang, K., and Yao, X. How good is neural combinatorial optimization? a systematic evaluation on the traveling salesman problem. *IEEE Computational Intelligence Magazine*, 18(3):14–28, 2023d.

Liventsev, V., Grishina, A., Härmä, A., and Moonen, L. Fully autonomous programming with large language models. *arXiv preprint arXiv:2304.10423*, 2023.

Ma, Y., Cao, Z., and Chee, Y. M. Learning to search feasible and infeasible regions of routing problems with flexible neural k-opt. In *Advances in Neural Information Processing Systems*, 2023a.

Ma, Y. J., Liang, W., Wang, G., Huang, D.-A., Bastani, O., Jayaraman, D., Zhu, Y., Fan, L., and Anandkumar, A. Eureka: Human-level reward design via coding large language models. *arXiv preprint arXiv:2310.12931*, 2023b.

Ma, Z., Guo, H., Chen, J., Li, Z., Peng, G., Gong, Y.-J., Ma, Y., and Cao, Z. Metabox: A benchmark platform for meta-black-box optimization with reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 36, 2023c.

Madaan, A., Shypula, A., Alon, U., Hashemi, M., Ranganathan, P., Yang, Y., Neubig, G., and Yazdanbakhsh, A. Learning performance-improving code edits. *arXiv preprint arXiv:2302.07867*, 2023a.

Madaan, A., Tandon, N., Gupta, P., Hallinan, S., Gao, L., Wiegreffe, S., Alon, U., Dziri, N., Prabhumoye, S., Yang, Y., et al. Self-refine: Iterative refinement with self-feedback. *arXiv preprint arXiv:2303.17651*, 2023b.

Mohammad Hasani Zade, B., Mansouri, N., and Javidi, M. M. A new hyper-heuristic based on ant lion optimizer and tabu search algorithm for replica management in cloud environment. *Artificial Intelligence Review*, 56(9): 9837–9947, 2023.

Ochoa, G., Qu, R., and Burke, E. K. Analyzing the landscape of a graph based hyper-heuristic for timetabling problems. In *Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, pp. 341–348, 2009.

Pillay, N. and Qu, R. *Hyper-heuristics: theory and applications*. Springer, 2018.

Pryzant, R., Iter, D., Li, J., Lee, Y. T., Zhu, C., and Zeng, M. Automatic prompt optimization with" gradient descent" and beam search. *arXiv preprint arXiv:2305.03495*, 2023.

Romera-Paredes, B., Barekatain, M., Novikov, A., Balog, M., Kumar, M. P., Dupont, E., Ruiz, F. J., Ellenberg, J. S., Wang, P., Fawzi, O., et al. Mathematical discoveries from program search with large language models. *Nature*, pp. 1–3, 2023.

Shinn, N., Cassano, F., Gopinath, A., Narasimhan, K. R., and Yao, S. Reflexion: Language agents with verbal reinforcement learning. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023.

Skinderowicz, R. Improving ant colony optimization efficiency for solving large tsp instances. *Applied Soft Computing*, 120:108653, 2022.

Sohrabi, S., Ziarati, K., and Keshtkaran, M. Acs-ophs: Ant colony system for the orienteering problem with hotel selection. *EURO Journal on Transportation and Logistics*, 10:100036, 2021.

Son, J., Kim, M., Choi, S., and Park, J. Solving np-hard min-max routing problems as sequential generation with equity context. *arXiv preprint arXiv:2306.02689*, 2023.

Sui, J., Ding, S., Xia, B., Liu, R., and Bu, D. Neuralgls: learning to guide local search with graph convolutional network for the traveling salesman problem. *Neural Computing and Applications*, pp. 1–20, 2023.

Sun, Z. and Yang, Y. Difusco: Graph-based diffusion solvers for combinatorial optimization. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023.

Voudouris, C. and Tsang, E. Guided local search and its application to the traveling salesman problem. *European Journal of Operational Research*, 113(2):469–499, March 1999. ISSN 03772217. doi: 10.1016/S0377-2217(98)00099-X.

Wang, C., Yu, Z., McAleer, S., Yu, T., and Yang, Y. Asp: Learn a universal neural solver! *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2024.

Wang, H., Feng, S., He, T., Tan, Z., Han, X., and Tsvetkov, Y. Can language models solve graph problems in natural language? *arXiv preprint arXiv:2305.10037*, 2023a.

Wang, L., Ling, Y., Yuan, Z., Shridhar, M., Bao, C., Qin, Y., Wang, B., Xu, H., and Wang, X. Gensim: Generating robotic simulation tasks via large language models. *arXiv preprint arXiv:2310.01361*, 2023b.

Wang, L., Ma, C., Feng, X., Zhang, Z., Yang, H., Zhang, J., Chen, Z., Tang, J., Chen, X., Lin, Y., et al. A survey on large language model based autonomous agents. *arXiv preprint arXiv:2308.11432*, 2023c.

Wang, X., Li, C., Wang, Z., Bai, F., Luo, H., Zhang, J., Jojic, N., Xing, E. P., and Hu, Z. Promptagent: Strategic planning with language models enables expert-level prompt optimization. *arXiv preprint arXiv:2310.16427*, 2023d.

Wang, Y., Liu, Z., Zhang, J., Yao, W., Heinecke, S., and Yu, P. S. Drdt: Dynamic reflection with divergent thinking for llm-based sequential recommendation. *arXiv preprint arXiv:2312.11336*, 2023e.

Weinberger, E. Correlated and uncorrelated fitness landscapes and how to tell the difference. *Biological cybernetics*, 63(5):325–336, 1990.

Wu, X., hao Wu, S., Wu, J., Feng, L., and Tan, K. C. Evolutionary computation in the era of large language model: Survey and roadmap, 2024.

Xi, Z., Chen, W., Guo, X., He, W., Ding, Y., Hong, B., Zhang, M., Wang, J., Jin, S., Zhou, E., et al. The rise and potential of large language model based agents: A survey. *arXiv preprint arXiv:2309.07864*, 2023.

Xiao, Y., Wang, D., Chen, H., Li, B., Pang, W., Wu, X., Li, H., Xu, D., Liang, Y., and Zhou, Y. Reinforcement learning-based non-autoregressive solver for traveling salesman problems. *arXiv preprint arXiv:2308.00560*, 2023.

Xiao, Y., Wang, D., Li, B., Wang, M., Wu, X., Zhou, C., and Zhou, Y. Distilling autoregressive models to obtain high-performance non-autoregressive solvers for vehicle routing problems with faster inference speed. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2024.

Yang, C., Wang, X., Lu, Y., Liu, H., Le, Q. V., Zhou, D., and Chen, X. Large language models as optimizers. *arXiv preprint arXiv:2309.03409*, 2023a.

Yang, J., Prabhakar, A., Narasimhan, K., and Yao, S. Intercode: Standardizing and benchmarking interactive coding with execution feedback. *arXiv preprint arXiv:2306.14898*, 2023b.

Yang, Y. and Whinston, A. A survey on reinforcement learning for combinatorial optimization. In *2023 IEEE World Conference on Applied Intelligence and Computing (AIC)*, pp. 131–136. IEEE, 2023.

Ye, H., Wang, J., Cao, Z., Liang, H., and Li, Y. Deepaco: Neural-enhanced ant systems for combinatorial optimization. In *Advances in Neural Information Processing Systems*, 2023.

Ye, H., Wang, J., Liang, H., Cao, Z., Li, Y., and Li, F. Glop: Learning global partition and local construction for solving large-scale routing problems in real-time. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2024.

Zambrano-Gutierrez, D., Cruz-Duarte, J., and Castañeda, H. Automatic hyper-heuristic to generate heuristic-based adaptive sliding mode controller tuners for buck-boost converters. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 1482–1489, 2023.

Zelikman, E., Lorch, E., Mackey, L., and Kalai, A. T. Self-taught optimizer (stop): Recursively self-improving code generation. *arXiv preprint arXiv:2310.02304*, 2023.

Zhang, S., Chen, Z., Shen, Y., Ding, M., Tenenbaum, J. B., and Gan, C. Planning with large language models for code generation. In *The Eleventh International Conference on Learning Representations*, 2022.

Zhang, Z.-Q., Wu, F.-C., Qian, B., Hu, R., Wang, L., and Jin, H.-P. A q-learning-based hyper-heuristic evolutionary algorithm for the distributed flexible job-shop scheduling problem with crane transportation. *Expert Systems with Applications*, 234:121050, 2023.

Zhao, W. X., Zhou, K., Li, J., Tang, T., Wang, X., Hou, Y., Min, Y., Zhang, B., Zhang, J., Dong, Z., et al. A survey of large language models. *arXiv preprint arXiv:2303.18223*, 2023.

Zhou, J., Wu, Y., Cao, Z., Song, W., Zhang, J., and Chen, Z. Learning large neighborhood search for vehicle routing in airport ground handling. *IEEE Transactions on Knowledge and Data Engineering*, 2023.

Zhou, Y., Muresanu, A. I., Han, Z., Paster, K., Pitis, S., Chan, H., and Ba, J. Large language models are human-level prompt engineers. *arXiv preprint arXiv:2211.01910*, 2022.

Zhu, L., Zhou, Y., Sun, S., and Su, Q. Surgical cases assignment problem using an efficient genetic programming hyper-heuristic. *Computers & Industrial Engineering*, 178:109102, 2023.

A. Prompts

We gather prompts used for ReEvo in this section. Our prompt structure is flexible and extensible. To adapt ReEvo to a new problem setting, one only needs to define *problem_description*, *function_description* and *function_signature*. Optionally, one can include *seed_function* and *initial_long-term_reflection*.

A.1. Common Prompts

The prompt formats are given below. They are used for all COP settings.

Prompt 1. System prompt for generator LLM.

```
You are an expert in the domain of optimization heuristics. Your task is to design
    heuristics that can effectively solve optimization problems.
Your response outputs Python code and nothing else. Format your code as a Python code
    string: "```python ... ```".
```

Prompt 2. System prompt for reflector LLM.

```
You are an expert in the domain of optimization heuristics. Your task is to give hints to
    design better heuristics.
```

Prompt 3. Task description.

```
Write a {function_name} function for {problem_description}
{function_description}
```

Prompt 4. User prompt for population initialization.

```
{task_description}

{seed_function}

Improve '{function_name}_v1' to give '{function_name}_v2'. Output code only and enclose
    your code with Python code block: ```python ... ```.

{initial_long-term_reflection}
```

Prompt 5. User prompt for short-term reflection.

```
Below are two {function_name} functions for {problem_description}
{function_description}

You are provided with two code versions below, where the second version performs better
    than the first one.

[Worse code]
{worse_code}

[Better code]
{better_code}

You respond with some hints for designing better heuristics, based on the two code
    versions and using less than 20 words.
```

The user prompt used for short-term reflection in black-box COPs is slightly different from the one used for while-box COPs. We explicitly ask the reflector LLM to infer the problem settings and to give hints about how the node and edge attributes correlate with the black-box objective value.

Prompt 6. User prompt for short-term reflection on black-box COPs.

```
Below are two {function_name} functions for {problem_description}
{function_description}
```

```
You are provided with two code versions below, where the second version performs better
    than the first one.

[Worse code]
{worse_code}

[Better code]
{better_code}

Please infer the problem settings by comparing two code versions and give hints for
    designing better heuristics. You may give hints about how edge and node attributes
    correlate with the black-box objective value. Use less than 50 words.
```

Prompt 7. User prompt for crossover.

```
{task_description}

[Worse code]
{function_signature0}
{worse_code}

[Better code]
{function_signature1}
{better_code}

[Reflection]
{short_term_reflection}

[Improved code]
Please write an improved function `{function_name}_v2`, according to the reflection.
    Output code only and enclose your code with Python code block: ```python ... ```.
```

The function signature variables here are used to adjust function names with their versions, which is similar to the design in (Romera-Paredes et al., 2023). For example, when designing "heuristics", the worse code is named "heuristics_v0" while the better code "heuristics_v1". In Prompt 9, the elitist code is named "heuristic_v1".

Prompt 8. User prompt for long-term reflection.

```
Below is your prior long-term reflection on designing heuristics for {problem_description}
{prior_long-term_reflection}

Below are some newly gained insights.
{new_short-term_reflections}

Write constructive hints for designing better heuristics, based on prior reflections and
    new insights and using less than 50 words.
```

Prompt 9. User prompt for elitist mutation.

```
{task_description}

[Prior reflection]
{long-term_reflection}

[Code]
{function_signature1}
{elitist_code}

[Improved code]
Please write a mutated function `{function_name}_v2`, according to the reflection. Output
    code only and enclose your code with Python code block: ```python ... ```.
```

A.2. Problem-Specific Prompt Components

Problem-specific prompt components are given below.

- Problem descriptions of all COP settings are given in Table 8. For more details of COPs involved in ACO evaluations, please refer to (Ye et al., 2023).
- The function descriptions of all COP settings are presented in Table 9. The descriptions crafted for black-box settings avoid disclosing any information that could link to the original COP.
- The function signatures are gathered in Prompt 10.
- The seed functions are shown in Prompt 11. The seed function used for TSP_constructive is drawn from (Liu et al., 2023b). The seed functions used for black-box ACO settings are expert-designed heuristics (Skinderowicz, 2022; Cai et al., 2022; Sohrabi et al., 2021; Fidanova, 2020; Levine & Ducatelle, 2004), while those used for while-box ACO settings are trivial all-ones matrices.
- The initial long-term reflections for some COP settings are presented in Prompt 12, while are left empty for the others.

Prompt 10. Function signatures used in ReEvo.

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

# TSP_ACO
def heuristics(distance_matrix: np.ndarray) -> np.ndarray:

# CVRP_ACO
def heuristics(distance_matrix: np.ndarray, coordinates: np.ndarray, demands: np.ndarray, capacity: int) -> np.ndarray:

# OP_ACO
def heuristics(prize: np.ndarray, distance: np.ndarray, maxlen: float) -> np.ndarray:

# MKP_ACO
def heuristics(prize: np.ndarray, weight: np.ndarray) -> np.ndarray:

# BPP_ACO
def heuristics(demand: np.ndarray, capacity: int) -> np.ndarray:

# TSP_ACO (black-box)
def heuristics(edge_attr: np.ndarray) -> np.ndarray:

# CVRP_ACO (black-box)
def heuristics(edge_attr: np.ndarray, node_attr: np.ndarray) -> np.ndarray: # For simplicity, we omit 'coordinates' and 'capacity'
     after using capacity to normalize demands, i.e. node_attr

# OP_ACO (black-box)
def heuristics(node_attr: np.ndarray, edge_attr: np.ndarray, node_constraint: float) -> np.ndarray:

# MKP_ACO (black-box)
def heuristics(item_attr1: np.ndarray, item_attr2: np.ndarray) -> np.ndarray:

# BPP_ACO (black-box)
def heuristics(node_attr: np.ndarray, node_constraint: int) -> np.ndarray:

# TSP_constructive
def select_next_node(current_node: int, destination_node: int, unvisited_nodes: set, distance_matrix: np.ndarray) -> int:
```

Prompt 11. Seed heuristics used for ReEvo.

```python
# TSP_GLS
def heuristics(distance_matrix: np.ndarray) -> np.ndarray:
   return distance_matrix

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

# CVRP_ACO
def heuristics(distance_matrix: np.ndarray, coordinates: np.ndarray, demands: np.ndarray, capacity: int) -> np.ndarray:
   return 1 / distance_matrix

# OP_ACO
def heuristics(prize: np.ndarray, distance: np.ndarray, maxlen: float) -> np.ndarray:
   return prize[np.newaxis, :] / distance

# MKP_ACO
def heuristics(prize: np.ndarray, weight: np.ndarray) -> np.ndarray:
   return prize / np.sum(weight, axis=1)
```

```python
# BPP_ACO
def heuristics(demand: np.ndarray, capacity: int) -> np.ndarray:
   return np.tile(demand/demand.max(), (demand.shape[0], 1))

# TSP_ACO (black-box)
def heuristics(edge_attr: np.ndarray) -> np.ndarray:
   return np.ones_like(edge_attr)

# CVRP_ACO (black-box)
def heuristics(edge_attr: np.ndarray, node_attr: np.ndarray) -> np.ndarray:
   return np.ones_like(edge_attr)

# OP_ACO (black-box)
def heuristics(node_attr: np.ndarray, edge_attr: np.ndarray, edge_constraint: float) -> np.ndarray:
   return np.ones_like(edge_attr)

# MKP_ACO (black-box)
def heuristics(item_attr1: np.ndarray, item_attr2: np.ndarray) -> np.ndarray:
   n, m = item_attr2.shape
   return np.ones(n,)

# BPP_ACO (black-box)
def heuristics(node_attr: np.ndarray, node_constraint: int) -> np.ndarray:
   n = node_attr.shape[0]
   return np.ones((n, n))

# TSP_constructive
def select_next_node(current_node: int, destination_node: int, unvisited_nodes: set, distance_matrix: np.ndarray) -> int:
   threshold = 0.7
   c1, c2, c3, c4 = 0.4, 0.3, 0.2, 0.1
   scores = {}
   for node in unvisited_nodes:
      all_distances = [distance_matrix[node][i] for i in unvisited_nodes if i != node]
      average_distance_to_unvisited = np.mean(all_distances)
      std_dev_distance_to_unvisited = np.std(all_distances)
      score = c1 * distance_matrix[current_node][node] - c2 * average_distance_to_unvisited + c3 * std_dev_distance_to_unvisited - c4 *
            distance_matrix[destination_node][node]
      scores[node] = score
   next_node = min(scores, key=scores.get)
   return next_node
```

Prompt 12. Initial long-term reflections

```
# White-box COP_ACO
- Try combining various factors to determine how promising it is to select a solution
  component.
- Try sparsifying the matrix by setting unpromising elements to zero.

# TSP_constructive
- Try look-ahead mechanisms.
```

B. Reproduction of AEL

We adhere to the original paper (Liu et al., 2023b) to reproduce AEL, except for two modifications: (1) We follow the later version of AEL (Liu et al., 2024) to use rank-based selection instead of random selection. We empirically find that rank-based selection leads to better performance of AEL. (2) To implement AEL for black-box COPs, we add a seed function, the same as we used for ReEvo, to initialize the population. This leads to more valid heuristics in the initial population and thus to better performance of AEL.

C. Problem Settings

This section details the GLS and ACO settings in our experiments.

C.1. Guided Local Search

Guided Local Search (GLS) explores solution space through local search operations under the guidance of heuristics. We aim to use ReEvo to find the most effective heuristics to enhance GLS. In our experimental setup, we employed a variation of the classical GLS algorithm (Voudouris & Tsang, 1999) that incorporated perturbation phases (Arnold & Sörensen, 2019), wherein edges with higher heuristic values will be prioritized for penalty. In the training phase, we evaluate each heuristic with TSP200 using 1200 GLS iterations. For generating results in Table 3, we use the parameters in Table 10. The iterations stop when reaching the predefined threshold or when the optimality gap is reduced to zero.

Table 8. Problem descriptions used in prompts.

Problem	Problem description
TSP_GLS	Solving Traveling Salesman Problem (TSP) via guided local search. TSP requires finding the shortest path that visits all given nodes and returns to the starting node.
TSP_ACO	Solving Traveling Salesman Problem (TSP) via stochastic solution sampling following ”heuristics”. TSP requires finding the shortest path that visits all given nodes and returns to the starting node.
TSP_ACO_black-box	Solving a black-box graph combinatorial optimization problem via stochastic solution sampling following ”heuristics”.
CVRP_ACO	Solving Capacitated Vehicle Routing Problem (CVRP) via stochastic solution sampling. CVRP requires finding the shortest path that visits all given nodes and returns to the starting node. Each node has a demand and each vehicle has a capacity. The total demand of the nodes visited by a vehicle cannot exceed the vehicle capacity. When the total demand exceeds the vehicle capacity, the vehicle must return to the starting node.
CVRP_ACO_black-box	Solving a black-box graph combinatorial optimization problem via stochastic solution sampling following ”heuristics”.
OP_ACO	Solving Orienteering Problem (OP) via stochastic solution sampling following ”heuristics”. OP is an optimization problem where the goal is to find the most rewarding route, starting from a depot, visiting a subset of nodes with associated prizes, and returning to the depot within a specified travel distance.
OP_ACO_black-box	Solving a black-box graph combinatorial optimization problem via stochastic solution sampling following ”heuristics”.
MKP_ACO	Solving Multiple Knapsack Problems (MKP) through stochastic solution sampling based on ”heuristics”. MKP involves selecting a subset of items to maximize the total prize collected, subject to multi-dimensional maximum weight constraints.
MKP_ACO_black-box	Solving a black-box combinatorial optimization problem via stochastic solution sampling following ”heuristics”.
BPP_ACO	Solving Bin Packing Problem (BPP). BPP requires packing a set of items of various sizes into the smallest number of fixed-sized bins.
BPP_ACO_black-box	Solving a black-box combinatorial optimization problem via stochastic solution sampling following ”heuristics”.
TSP_constructive	Solving Traveling Salesman Problem (TSP) with constructive heuristics. TSP requires finding the shortest path that visits all given nodes and returns to the starting node.

Table 9. Function descriptions used in prompts.

Problem	Function description
TSP_GLS	The ‘heuristics‘ function takes as input a distance matrix, and returns prior indicators of how bad it is to include each edge in a solution. The return is of the same shape as the input.
TSP_ACO	The ‘heuristics‘ function takes as input a distance matrix, and returns prior indicators of how promising it is to include each edge in a solution. The return is of the same shape as the input.
TSP_ACO_black-box	The ‘heuristics‘ function takes as input a matrix of edge attributes and returns prior indicators of how promising it is to include each edge in a solution. The return is of the same shape as the input matrix.
CVRP_ACO	The ‘heuristics‘ function takes as input a distance matrix (shape: n by n), Euclidean coordinates of nodes (shape: n by 2), a vector of customer demands (shape: n), and the integer capacity of vehicle capacity. It returns prior indicators of how promising it is to include each edge in a solution. The return is of the same shape as the distance matrix. The depot node is indexed by 0.
CVRP_ACO_black-box	The ‘heuristics‘ function takes as input a matrix of edge attributes (shape: n by n) and a vector of node attributes (shape: n). A special node is indexed by 0. ‘heuristics‘ returns prior indicators of how promising it is to include each edge in a solution. The return is of the same shape as the input matrix of edge attributes.
OP_ACO	Suppose ‘n‘ represents the number of nodes in the problem, with the depot being the first node. The ‘heuristics‘ function takes as input a ‘prize‘ array of shape (n,), a ‘distance‘ matrix of shape (n,n), and a ‘max_len‘ float which is the constraint to total travel distance, and it returns ‘heuristics‘ of shape (n, n), where ‘heuristics[i][j]‘ indicates the promise of including the edge from node #i to node #j in the solution.
OP_ACO_black-box	The ‘heuristics‘ function takes as input a vector of node attributes (shape: n), a matrix of edge attributes (shape: n by n), and a constraint imposed on the sum of edge attributes. A special node is indexed by 0. ‘heuristics‘ returns prior indicators of how promising it is to include each edge in a solution. The return is of the same shape as the input matrix of edge attributes.
MKP_ACO	Suppose ‘n‘ indicates the scale of the problem, and ‘m‘ is the dimension of weights each item has. The constraint of each dimension is fixed to 1. The ‘heuristics‘ function takes as input a ‘prize‘ of shape (n,), a ‘weight‘ of shape (n, m), and returns ‘heuristics‘ of shape (n,). ‘heuristics[i]‘ indicates how promising it is to include item i in the solution.
MKP_ACO_black-box	Suppose ‘n‘ indicates the scale of the problem, and ‘m‘ is the dimension of some attributes each involved item has. The ‘heuristics‘ function takes as input an ‘item_attr1‘ of shape (n,), an ‘item_attr2‘ of shape (n, m), and returns ‘heuristics‘ of shape (n,). ‘heuristics[i]‘ indicates how promising it is to include item i in the solution.
BPP_ACO	Suppose ‘n‘ represents the number of items in the problem. The heuristics function takes as input a ‘demand‘ array of shape (n,) and an integer as the capacity of every bin, and it returns a ‘heuristics‘ array of shape (n,n). ‘heuristics[i][j]‘ indicates how promising it is to put item i and item j in the same bin.
BPP_ACO_black-box	Suppose ‘n‘ represents the scale of the problem. The heuristics function takes as input an ‘item_attr‘ array of shape (n,) and an integer as a certain constraint imposed on the item attributes. The heuristics function returns a ‘heuristics‘ array of shape (n, n). ‘heuristics[i][j]‘ indicates how promising it is to group item i and item j.
TSP_constructive	The select_next_node function takes as input the current node, the destination node, a set of unvisited nodes, and a distance matrix, and returns the next node to visit.

Table 10. GLS parameters used for the evaluations in Table 3.

Problem	Perturbation moves	Number of iterations	Scale parameter λ
TSP20	5	73	0.1
TSP50	30	175	
TSP100	40	1800	
TSP200	40	800	

Table 11. ACO parameters used for heuristic evaluations during training.

Problem	Population size	Number of iterations
TSP	30	100
CVRP	30	100
OP	20	50
MKP	10	50
BPP	20	15

C.2. Ant Colony Optimization

Ant Colony Optimization is an evolutionary algorithm that interleaves solution samplings with the update of pheromone trails. Stochastic solution samplings are biased toward more promising solution space by heuristics, and ReEvo searches for the best of such heuristics. For more details, please refer to (Ye et al., 2023).

Table 11 presents the ACO parameters used for heuristic evaluations during LHH evolution. They are adjusted to maximize ACO performance while ensuring efficient evaluations. Instance generations and ACO implementations follow Ye et al. (2023). To conduct tests in Figure 2, we increase the number of iterations to ensure full convergence.

D. Generated Heuristics

This section presents the best heuristics generated by ReEvo for all problem settings.

Heuristic 1. The best ReEvo-generated heuristic for TSP_GLS.

```python
def heuristics(distance_matrix: np.ndarray) -> np.ndarray:
   # Calculate the average distance for each node
   average_distance = np.mean(distance_matrix, axis=1)

   # Calculate the distance ranking for each node
   distance_ranking = np.argsort(distance_matrix, axis=1)

   # Calculate the mean of the closest distances for each node
   closest_mean_distance = np.mean(distance_matrix[np.arange(distance_matrix.shape[0])[:, None], distance_ranking[:, 1:5]], axis=1)

   # Initialize the indicator matrix and calculate ratio of distance to average distance
   indicators = distance_matrix / average_distance[:, np.newaxis]

   # Set diagonal elements to np.inf
   np.fill_diagonal(indicators, np.inf)

   # Adjust the indicator matrix using the statistical measure
   indicators += closest_mean_distance[:, np.newaxis] / np.sum(distance_matrix, axis=1)[:, np.newaxis]

   return indicators
```

Heuristic 2 presents the best heuristic found for TSP_ACO, which is generated when viewing TSP as a black-box COP. 'edge_attr' represents the distance matrix.

Heuristic 2. The best ReEvo-generated heuristic for TSP_ACO.

```python
import numpy as np
from sklearn.preprocessing import StandardScaler

def heuristics(edge_attr: np.ndarray) -> np.ndarray:
   num_edges = edge_attr.shape[0]
```

```
    num_attributes = edge_attr.shape[1]

    heuristic_values = np.zeros_like(edge_attr)

    # Apply feature engineering on edge attributes
    transformed_attr = np.log1p(np.abs(edge_attr)) # Taking logarithm of absolute value of attributes

    # Normalize edge attributes
    scaler = StandardScaler()
    edge_attr_norm = scaler.fit_transform(transformed_attr)

    # Calculate correlation coefficients
    correlation_matrix = np.corrcoef(edge_attr_norm.T)

    # Calculate heuristic value for each edge attribute
    for i in range(num_edges):
        for j in range(num_attributes):
            if edge_attr_norm[i][j] != 0:
                heuristic_values[i][j] = np.exp(-8 * edge_attr_norm[i][j] * correlation_matrix[j][j])

    return heuristic_values
```

Heuristic 3. The best ReEvo-generated heuristic for CVRP_ACO.

```
def heuristics(distance_matrix: np.ndarray, coordinates: np.ndarray, demands: np.ndarray, capacity: int) -> np.ndarray:
    num_nodes = distance_matrix.shape[0]

    # Calculate the inverse of the distance matrix
    inverse_distance_matrix = np.divide(1, distance_matrix, where=(distance_matrix != 0))

    # Calculate total demand and average demand
    total_demand = np.sum(demands)
    average_demand = total_demand / num_nodes

    # Calculate the distance from each node to the starting depot
    depot_distances = distance_matrix[:, 0]

    # Calculate the remaining capacity of the vehicle for each node
    remaining_capacity = capacity - demands

    # Initialize the heuristic matrix
    heuristic_matrix = np.zeros_like(distance_matrix)

    # Calculate the demand factor and distance factor
    demand_factor = demands / total_demand
    normalized_distance = distance_matrix / np.max(distance_matrix)
    distance_factor = depot_distances / (normalized_distance + np.finfo(float).eps)

    # Iterate over each node
    for i in range(num_nodes):

        # Calculate the heuristic value based on distance and capacity constraints
        heuristic_values = inverse_distance_matrix[i] * (1 / (normalized_distance[i] ** 2))

        # Adjust the heuristic values based on the remaining capacity
        heuristic_values = np.where(remaining_capacity >= demands[i], heuristic_values, 0)

        # Adjust the heuristic values based on the demand factor
        heuristic_values *= demand_factor[i] / average_demand

        # Adjust the heuristic values based on the distance factor
        heuristic_values *= distance_factor[i]
        heuristic_values[0] = 0 # Exclude the depot node

        # Adjust the heuristic values based on the capacity utilization
        utilization_factor = np.where(remaining_capacity >= demands[i], capacity - demands[i], 0)
        heuristic_values *= utilization_factor

        # Set the heuristic values for the current node in the heuristic matrix
        heuristic_matrix[i] = heuristic_values

    return heuristic_matrix
```

Heuristic 4. The best ReEvo-generated heuristic for OP_ACO.

```
def heuristics(prize: np.ndarray, distance: np.ndarray, maxlen: float) -> np.ndarray:
    n = prize.shape[0]
    heuristics = np.zeros((n, n))

    # Calculate the prize-to-distance ratio with a power transformation
    prize_distance_ratio = np.power(prize / distance, 3)

    # Find the indices of valid edges based on the distance constraint
    valid_edges = np.where(distance <= maxlen)

    # Assign the prize-to-distance ratio to the valid edges
    heuristics[valid_edges] = prize_distance_ratio[valid_edges]

    return heuristics
```

Heuristic 5 presents the best heuristic found for MKP_ACO, which is generated when viewing MKP as a black-box COP. 'item_attr1' and 'item_attr2' represent the prizes and multi-dimensional weights of items, respectively.

Heuristic 5. The best ReEvo-generated heuristic for MKP_ACO.

```
def heuristics(item_attr1: np.ndarray, item_attr2: np.ndarray) -> np.ndarray:
    n, m = item_attr2.shape

    # Normalize item_attr1 and item_attr2
    item_attr1_norm = (item_attr1 - np.min(item_attr1)) / (np.max(item_attr1) - np.min(item_attr1))
    item_attr2_norm = (item_attr2 - np.min(item_attr2)) / (np.max(item_attr2) - np.min(item_attr2))

    # Calculate the average value of normalized attribute 1
    avg_attr1 = np.mean(item_attr1_norm)

    # Calculate the maximum value of normalized attribute 2 for each item
    max_attr2 = np.max(item_attr2_norm, axis=1)

    # Calculate the sum of normalized attribute 2 for each item
    sum_attr2 = np.sum(item_attr2_norm, axis=1)

    # Calculate the standard deviation of normalized attribute 2 for each item
    std_attr2 = np.std(item_attr2_norm, axis=1)

    # Calculate the heuristics based on a combination of normalized attributes 1 and 2,
    # while considering the average, sum, and standard deviation of normalized attribute 2
    heuristics = (item_attr1_norm / max_attr2) * (item_attr1_norm / avg_attr1) * (item_attr1_norm / sum_attr2) * (1 / std_attr2)

    # Normalize the heuristics to a range of [0, 1]
    heuristics = (heuristics - np.min(heuristics)) / (np.max(heuristics) - np.min(heuristics))

    return heuristics
```

Heuristic 6. The best ReEvo-generated heuristic for BPP_ACO.

```
def heuristics(demand: np.ndarray, capacity: int) -> np.ndarray:
    n = demand.shape[0]
    demand_normalized = demand / demand.max()

    same_bin_penalty = np.abs((capacity - demand[:, None] - demand) / capacity)
    overlap_penalty = (demand[:, None] + demand) / capacity

    heuristics = demand_normalized[:, None] + demand_normalized - same_bin_penalty - overlap_penalty

    threshold = np.percentile(heuristics, 90)
    heuristics[heuristics < threshold] = 0

    return heuristics
```

Heuristic 7 gives the the best-ReEvo generated constructive heuristic for TSP. We used the best heuristic found in AEL (Liu et al., 2023b) as the seed for ReEvo. As a result, our heuristic closely mirrors the one in AEL, scoring each node mostly using a weighted combination of the four factors.

Heuristic 7. The best ReEvo-generated heuristic for TSP_constructive.

```
def select_next_node(current_node: int, destination_node: int, unvisited_nodes: set, distance_matrix: np.ndarray) -> int:
    weights = {'distance_to_current': 0.4,
               'average_distance_to_unvisited': 0.25,
               'std_dev_distance_to_unvisited': 0.25,
               'distance_to_destination': 0.1}
    scores = {}
    for node in unvisited_nodes:
        future_distances = [distance_matrix[node, i] for i in unvisited_nodes if i != node]
        if future_distances:
            average_distance_to_unvisited = sum(future_distances) / len(future_distances)
            std_dev_distance_to_unvisited = (sum((x - average_distance_to_unvisited) ** 2 for x in future_distances) / len(
                future_distances)) ** 0.5
        else:
            average_distance_to_unvisited = std_dev_distance_to_unvisited = 0
        score = (weights['distance_to_current'] * distance_matrix[current_node, node] -
                 weights['average_distance_to_unvisited'] * average_distance_to_unvisited +
                 weights['std_dev_distance_to_unvisited'] * std_dev_distance_to_unvisited -
                 weights['distance_to_destination'] * distance_matrix[destination_node, node])
        scores[node] = score
    next_node = min(scores, key=scores.get)
    return next_node
```