

000 001 002 003 004 005 006 007 008 009 010 011 012 013 014 015 016 017 018 019 020 021 022 023 024 025 026 027 028 029 030 031 032 033 034 035 036 037 038 039 040 041 042 043 044 045 046 047 048 049 050 051 052 053 REDAHD: TOWARD END-TO-END LLM-BASED AUTOMATIC HEURISTIC DESIGN USING REDUCTIONS

Anonymous authors

Paper under double-blind review

ABSTRACT

Solving NP-hard combinatorial optimization problems (COPs) (e.g., traveling salesman problems (TSPs) and capacitated vehicle routing problems (CVRPs)) in practice traditionally involves handcrafting heuristics or specifying a search space for finding effective heuristics. The main challenges from these approaches, however, are the sheer amount of domain knowledge and implementation efforts required from human experts. Recently, significant progress has been made to address these challenges, particularly by using large language models (LLMs) to design heuristics within some predetermined generalized algorithmic framework (GAF, e.g., ant colony optimization and guided local search) for building key functions/components (e.g., *a priori* information on how promising it is to include each edge in a solution for TSP and CVRP). Although existing methods leveraging this idea have shown to yield impressive optimization performance, they are far from being end-to-end and still require considerable manual interventions. In this paper, we propose a novel framework, named RedAHD, that enables these LLM-based heuristic design methods to operate without the need of GAFs. More specifically, RedAHD employs LLMs to automate the process of *reduction*, i.e., transforming the COP at hand into similar COPs that are better-understood, from which LLM-based heuristic design methods can design effective heuristics for directly solving the transformed COPs and, in turn, indirectly solving the original COP. Our experimental results, evaluated on six COPs, show that RedAHD is capable of designing heuristics with competitive or improved results over the state-of-the-art methods with minimal human involvement.

1 INTRODUCTION

Solving NP-hard combinatorial optimization problems (COPs) encountered in real-world applications, such as TSPs (Matai et al., 2010) and CVRPs (Dantzig & Ramser, 1959), traditionally requires extensive domain knowledge and manual efforts from human experts to either design approximation algorithms with provable guarantees or handcraft problem-specific heuristics, with the latter being a more pertinent choice in practice (Desale et al., 2015). In response, automatic heuristic design (AHD), or hyper-heuristics (Burke et al., 2013; Pillay & Qu, 2018), was proposed as a promising alternative, in which the goal is to find the best heuristic among several prespecified options i.e., the heuristic space. Among popular AHD approaches, those employing genetic programming (GP) (Langdon & Poli, 2013), an evolutionary algorithm from machine learning, stands out due to its effectiveness in navigating the heuristic space as well as interpretability (Mei et al., 2022). However, GP-based AHD approaches require a handcrafted set of permissible search operators for generating new heuristics, which can be hard to construct in practice (O’Neill et al., 2010).

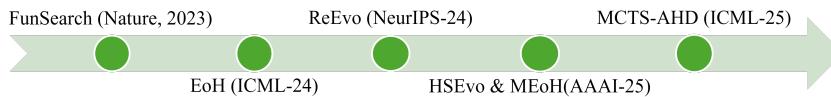


Figure 1: Timeline of LLM-EPS methods developed thus far.

054 **Latest Efforts and Their Limitations.** In recent years, the advent of powerful, readily accessible
 055 large language models (LLMs) such as GPT-3.5 and its successors (Brown et al., 2020) has enabled
 056 new approaches for AHD (Liu et al., 2024b). Among them, integrating LLMs into an evolutionary
 057 computation (EC) procedure for iterative refinement of heuristics, also known as LLM-based evo-
 058 lutionary program search (LLM-EPS) (Liu et al., 2024d; Dat et al., 2025), has attracted increasing
 059 attention. As illustrated in Figure 1, in the past two years, multiple works falling into this cate-
 060 gory have been proposed, each building upon the previous ones to yield incrementally better results.
 061 The common idea from these works is to maintain a set of heuristics with good optimization per-
 062 formance on an evaluation dataset of problem instances and iteratively prompt LLMs to generate
 063 new heuristics using existing ones as references. LLM-EPS methods can not only design novel,
 064 high-quality heuristics but also streamline the implementation process by representing heuristics as
 065 LLM-generated code that can be applied to unseen in-distribution (ID) as well as out-of-distribution
 066 (OOD) problem instances (Liu et al., 2024a; Ye et al., 2024; Yao et al., 2025; Zheng et al., 2025b).
 067 Combined with the current rapid development of LLMs with improved reasoning capabilities (Zheng
 068 et al., 2025a), this approach is expected to revolutionize how heuristics for COPs are developed and
 069 implemented in the near future (Liu et al., 2024b).

070 Table 1: GAFs used for the considered COPs in existing LLM-EPS studies. Legends: IC—iterative construc-
 071 tion; GLS—guided local search; ACO—ant colony optimization; NCO—neural combinatorial optimization (see
 072 Section 4.1 for clarifications on COP acronyms). COPs not considered in the respective studies are shaded.

COP \ Method	FunSearch 2024	EoH 2024a	ReEvo 2024	HSEvo 2025	MEoH 2025	MCTS-AHD 2025b	RedAHD (ours)
TSP		GLS	IC, ACO, GLS, NCO	GLS	GLS	IC, ACO	
OBPP	IC	IC		IC	IC	IC	
BPP			ACO			ACO	
KP						IC	
MKP			ACO			ACO	
CVRP			ACO, NCO			ACO	

073 However, despite their advantages over classical AHD ap-
 074 proaches, existing LLM-EPS methods are far from be-
 075 ing end-to-end (Liu et al., 2024b). That is, they only
 076 design heuristics for building key functions/components
 077 within some predetermined general algorithmic frame-
 078 work (GAF), such as iterative construction (IC) (Asani
 079 et al., 2023), ant colony optimization (ACO) (Dorigo
 080 et al., 2007), and guided local search (GLS) (Voudouris
 081 et al., 2010), as detailed in Table 1, rather than heuristics
 082 for solving COPs directly. When ACO is employed for
 083 TSP, for instance, LLM-EPS methods only aim to design
 084 heuristics that indicate how promising it is to include each
 085 edge in a solution. This heuristic is then used to generate *a priori*
 086 information within the ACO frame-
 087 work to better guide the search/foraging behavior of ants.
 088 Thus, when applying existing LLM-EPS
 089 methods to solve COPs in practice, human users still need to manually specify and design a suitable
 090 GAF for directly solving the problem. Employing complex GAFs such as ACO and GLS may yield
 091 improved performance over handcrafted heuristics, GP-based AHD methods, and even specialized
 092 neural networks (see “NCO” in Appendix A) (Liu et al., 2024a; Ye et al., 2024; Dat et al., 2025; Yao
 093 et al., 2025; Zheng et al., 2025b) but also requires domain knowledge and significant implementation
 094 efforts, whereas resorting to simple GAFs such as IC may result in subpar performances (see Table
 095 2). In either case, a tailored GAF must be implemented for each COP (see Appendix C for compari-
 096 son between ACO code for TSP vs. CVRP). Then, individual components for LLM prompting in
 097 accordance with the built GAF, e.g., the (sub)problem description, the heuristic description, and the
 098 function signature, are carefully designed (see Table S9). Given these limitations, LLM-EPS with
 099 enhanced automation warrants more attention to advance the field of AHD (Liu et al., 2024b).

100 **Our Contributions.** In this paper, we initiate the first attempt toward end-to-end AHD via LLM-
 101 EPS. We summarize our contributions as follows:

102 • We introduce a novel general framework, named **Reduction-based Automatic Heuristic Design**
 103 (RedAHD), that enables existing LLM-EPS methods to function independently without the need

104 Table 2: Performance comparison (lower is
 105 better) between LLM-EPS methods using IC
 106 vs. ACO for TSP (from results in Zheng
 107 et al. (2025b)). n is the number of nodes.
 108 Note: Results came from different test sets
 109 (1,000 and 64 instances for IC and ACO, re-
 110 spectively), hence actual values might vary
 111 slightly.

Method	Setting		TSP w/ IC		TSP w/ ACO	
	$n=50$	$n=100$	$n=50$	$n=100$	$n=50$	$n=100$
EoH 2024a	6.394	8.894	5.828	8.263		
MCTS-AHD 2025b	6.225	8.684	5.801	8.179		

108 of GAFs. RedAHD operates based on the simple-yet-powerful idea of *reduction* in algorithm
 109 design (Crescenzi, 1997) (also formally defined in Section 2), in which a COP of interest is trans-
 110 formed into a similar COP that is better-understood. This process is automated by prompting
 111 the LLM to devise a reduction and implement two corresponding functions (as code) that convert
 112 instances and solutions of one COP to another. By this means, existing LLM-EPS methods can
 113 be utilized to design novel heuristics for directly solving the transformed COP and, in turn, in-
 114 directly solving the original COP. RedAHD not only enhances automation in LLM-based AHD,
 115 substantially reducing the manual efforts involved, but also potentially brings fresh insights to the
 116 COP at hand by uncovering uncharted heuristic space (to be elaborated in Section 3.2) and yields
 117 improved optimization performance over state-of-the-art methods.
 118

- 119 • We incorporate a mechanism within RedAHD that automatically refines reduction functions (for
 120 mapping instances and solutions of one COP to another) whenever the search process stagnates
 121 and seemingly converges to local optima (within the landscape defined by the objective function
 122 of the COP). This extension, in turn, enables RedAHD to yield good performance even when the
 123 initial reductions are not adequately implemented by the LLM.
- 124 • We empirically show in our experiments that when integrating the most representative LLM-EPS
 125 method, EoH (Liu et al., 2024a), into RedAHD to attempt end-to-end AHD for six COPs, the
 126 designed heuristics achieve competitive or better optimization performances compared to existing
 127 LLM-EPS methods even when operated under advanced GAFs such as ACO. Moreover, these
 128 impressive performances are further improved when we employ (i) a more powerful LLM (o3-
 129 mini) or (ii) more sophisticated LLM-EPS methods (ReEvo (Ye et al., 2024) and MEoH (Yao
 130 et al., 2025)).

131 **Outline.** We provide the preliminaries in Section 2. Section 3 describes the proposed RedAHD
 132 framework. We evaluate its efficacy through various experiments in Section 4 and conclude our
 133 work in Section 5. We defer related work to Appendix A and further discussions (e.g., resource
 134 consumption, limitations and future works, and advantage scope of RedAHD) to Appendix D.

135 2 LANGUAGE REDUCTION FOR COMBINATORIAL OPTIMIZATION

136 In this section, we first revisit the LLM-based AHD task as considered in previous LLM-EPS works,
 137 which helps better identify their shared flaw and motivate our approach, then formally define the
 138 concept of *reduction* and *language reduction* upon which our framework is built.

139 Let A be a COP of interest, x be an instance of A , y be a feasible solution of x , and $h(x) = y$
 140 be a heuristic for A . The (supposed) task of LLM-EPS is to search for an optimal heuristic h^* in a
 141 heuristic space H (characterized by prior knowledge from LLMs) such that its expected performance
 142 on solving A is maximized, i.e.,
 143

$$144 \quad h^* \in \arg \max_{h \in H} \mathbb{E}_{x \sim \mathcal{D}} [q(x, h(x))] \quad (1)$$

145 where \mathcal{D} is an arbitrary distribution over problem instances of A and $q(x, y)$ is the objective function
 146 for A (defined in Appendix B for each of our considered COPs). However, existing LLM-EPS
 147 methods (Romera-Paredes et al., 2024; Liu et al., 2024a; Ye et al., 2024; Dat et al., 2025; Yao et al.,
 148 2025; Zheng et al., 2025b) actually design $h'(x') = y'$, which builds a subroutine within some GAF
 149 and hence does not solve the COP on its own. Therefore, in reality, the task of these methods is to
 150 search for
 151

$$152 \quad h^* \in \arg \max_{h' \in H'} \mathbb{E}_{x \sim \mathcal{D}} [q(x, g(h'(f(x))))] \quad (2)$$

153 where $f(x) = x'$ maps an instance of A to an instance of a subproblem B and $g(y') = y$ maps a
 154 solution of B to a solution of A , both of which are given by the manually specified GAF, and H' is
 155 the heuristic space for B .
 156

157 We approach the task by noticing that the tuple (f, g) resembles the following concept of reduction.
 158

159 **Definition 1 (Reduction (Crescenzi, 1997))** *Let A and B be two COPs. A reduction from A to B
 160 is a pair of polynomial-time computable functions (f, g) , such that:*

- 161 • f maps an instance x of A into an instance x' of B , i.e., $f(x) = x'$.

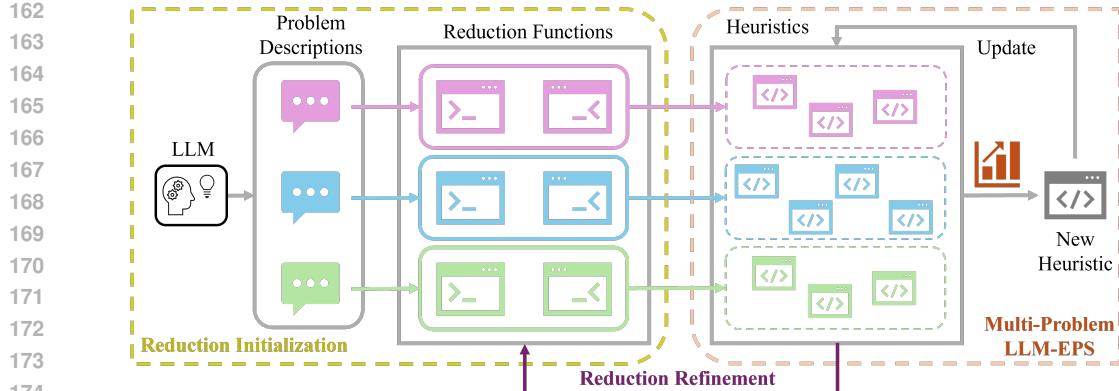


Figure 2: Illustration of RedAHD. First, the designer LLM generates a set of LRs, encoded as two reduction functions (one for mapping instances of A to B and the other for mapping solutions of B to A , see Figure S7-center for an example). The LRs are then used to generate a set of heuristics (exemplified in Figure S8) that are iteratively refined using existing LLM-EPS methods, in which offspring heuristics of an LR may be generated using algorithmic ideas from heuristics of any other LRs. When the overall performance of the heuristics associated with an LR stagnates, the LR is automatically refined by the LLM.

- g maps a solution y' of B to a solution y of A , i.e., $g(y') = y$.

Motivated by this observation, our goal in this paper is thus to automate the design of f and g (Definition 1), which eliminates the need of GAFs and thereby enhance automation in LLM-based AHD. We hereby introduce a novel variant of reduction as follows.

Definition 2 (Language reduction) A language reduction (LR) is an approximate reduction from A to B where f, g are generated by LLMs. The reduction is “approximate” in the sense that g does not necessarily preserve some guarantee of the performance ratio of y with respect to x (Crescenzi, 1997).

3 THE REDAHD FRAMEWORK

In this section, we propose RedAHD, which aims to address the stated flaw of existing LLM-EPS methods via LRs. In essence, RedAHD only takes A ’s specifications as input and outputs h^* defined in Equation 2 with minimal human involvement. It maintains a set P of N LLM-generated heuristics, denoted as $P = \{h'_1, \dots, h'_N\}$ ¹, by adopting some LLM-EPS method to iteratively find heuristics with better objective values subject to a finite set of $D > 0$ problem instances drawn from \mathcal{D} . Each heuristic $h'_i \in P$ is associated with an LR $r_j \in R = \{r_1, \dots, r_M\}$, which transforms A into another COP, B_j . The LRs are automatically refined as needed to avoid premature convergence at locally optimal heuristics. Figure 2 illustrates the schematic of RedAHD, which comprises three steps: (i) reduction initialization, (ii) multi-problem LLM-EPS, and (iii) reduction refinement. The following subsections elaborate each step. Our designed prompts are detailed in Appendix D.1.

3.1 REDUCTION INITIALIZATION

LR Representation. We start by describing the components to represent an LR, which include:

1. The natural-language problem description of B in a few sentences.
2. The code snippet for implementing (f, g) in accordance with A and B ’s descriptions. It should follow a predefined format, referred to as “reduction template”, so that it can be seamlessly combined with existing LLM-EPS methods.²

¹For clarity, h' denotes heuristics for an arbitrary B and $h^{(j)}$ denotes heuristics for a specific B_j .

²In the experiments, we choose to implement f and g as two Python functions.

216 3. The code template based on the implemented (f, g) , which is used by the employed LLM-EPS
 217 method to design h' for B . In prior works, this component must be manually designed in accor-
 218 dance with the underlying GAF (see ‘‘Function signature’’ in Table S9).
 219 4. Each LR is assigned a score to quantify its performance on A , which is used for selection and
 220 stagnation tracking (to be elaborated in Section 3.3). We will define this score shortly.
 221

222 We provide illustrative examples of LRs in Appendix D.4.

223 **Candidate LR Generation.** Given A ’s description, RedAHD first prompts the LLM to provide
 224 a list of $M_{init} \geq M$ descriptions for the respective candidate COPs, $\{B_j\}_{j=1}^{M_{init}}$. For each B_j ,
 225 RedAHD generates (f_j, g_j) by prompting the LLM with its description and the reduction template
 226 as input, then uses these functions to prompt the LLM again for the code template associated with
 227 B_j . We do not combine these two sequential calls into one to prevent hallucinations from LLMs
 228 (Huang et al., 2025).
 229

230 **Heuristic Initialization.** We initialize a set of heuristics for each B_j , denoted as P_j , by providing
 231 the LLM with B_j ’s description and its corresponding code template. Once a heuristic $h_i^{(j)1}$ is
 232 generated, its optimization performance, or fitness value, is computed as follows:
 233

$$234 Q(h_i^{(j)}) = \frac{1}{D} \sum_{k=1}^D q(x^{(k)}, y^{(k)})$$

235 where q is the objective function for A (e.g., minus tour length for TSP) and $y^{(k)} =$
 236 $g_j(h_i^{(j)}(f_j(x^{(k)})))$. We repeat this process $\lceil N/M \rceil$ times to obtain $P_j = \{h_1^{(j)}, \dots, h_{\lceil N/M \rceil}^{(j)}\}$.
 237

238 **Selection.** We define the score of an LR, denoted as s_j , as the average fitness values of its top-
 239 l associated heuristics. After evaluating all M_{init} candidate LRs, we select M LRs with highest
 240 scores. Consequently, the initial set of heuristics is $P = \bigcup_{j=1}^M P_j$, for a total of at least N heuristics.
 241

242 Note that for any LR r_j , we do not explicitly check the correctness of (f_j, g_j) . As long as the
 243 solutions $y^{(k)}$ returned from the resulting heuristic $h_i^{(j)}$ are valid (e.g., for TSP, the tour must traverse
 244 all nodes without revisiting non-starting nodes) for *all* respective instances $x^{(k)}$, r_j is deemed valid.
 245 We elaborate on our strategy to consistently obtain valid LRs in Appendix D.1.
 246

247 3.2 MULTI-PROBLEM LLM-EPS

248 Once the set of LRs R and the resulting set of heuristics P are initialized, the evolutionary search
 249 procedure in RedAHD follows existing LLM-EPS methods, which typically consists of: (i) selecting
 250 parent heuristic(s) from P (either randomly or based on Q), (ii) applying variation operators on these
 251 heuristics via LLM prompting to search for new heuristics in H' (as elaborated in Appendix D.1),
 252 and (iii) managing the size of P to be within N by only keeping the fittest heuristics. However, since
 253 there are now multiple options for H' , it is important to apply these works such that the expanded
 254 heuristic space can be efficiently explored without incurring extra costs. Therefore, we extend LLM-
 255 EPS methods to *multi-problem settings* where *any* heuristic from P , regardless of which COP it is
 256 intended to solve, may be indiscriminately selected as parent when designing new heuristics for a
 257 given COP. That is, a heuristic $h_i^{(j)}$ for B_j can be used as algorithmic reference to generate offspring
 258 heuristics for $B_{j'}$, $j' \neq j$. The advantages of this technique over designing heuristics for each
 259 B_j separately are twofold. First, it prevents situations where one LR performs significantly better
 260 than others and hence all heuristics in P are designed for a single COP, making the search for
 261 heuristics for other COPs futile. More importantly, it facilitates the discovery of novel heuristics
 262 from uncharted heuristic space, which may result in improved performance. Figure 3 illustrates a
 263 supporting example for this claim during heuristic design for TSP.
 264

265 In the following, we describe the multi-problem LLM-EPS procedure using EoH (Liu et al., 2024a)
 266 as the reference LLM-EPS method given its close resemblance to traditional evolutionary algorithms
 267 and proven significance to the field of LLM-based AHD, but the same concept can be applied to other
 268 LLM-EPS methods (as detailed in Appendix D.1 for ReEvo (Ye et al., 2024) and MEoH (Yao et al.,
 269 2025)).

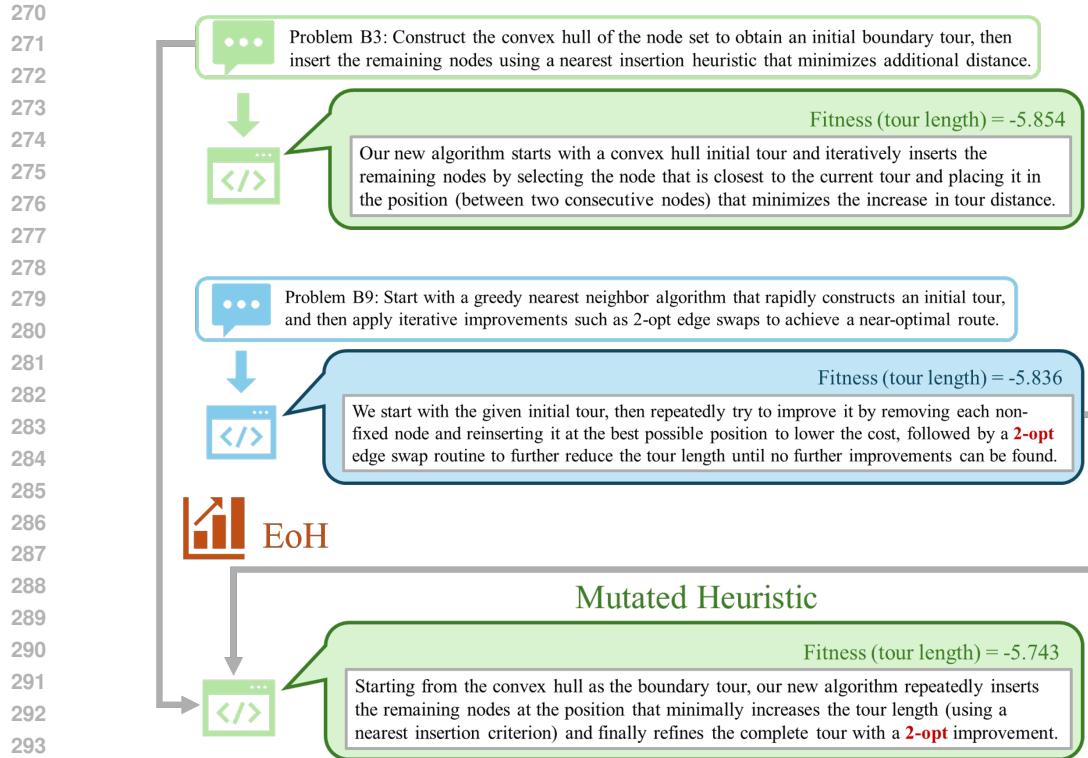


Figure 3: A demonstration of multi-problem LLM-EPS for TSP, in which the parent heuristic (blue) during EoH mutation (Liu et al., 2024a) is not intended to solve the COP at hand (“Problem B3”). As a result, the offspring heuristic for B3 (green) is generated with the novel idea of 2-opt edge swap and hence yields better performance.

LR ration. At each iteration or generation in EoH, each variation operator (e.g., crossover and mutation) is applied N times to generate N new heuristics for B . In multi-problem LLM-EPS, each variation operator now creates heuristics for different COPs in $\{B_j\}_{j=1}^M$. To maintain the number of newly generated heuristics in a generation, each variation operator is applied to generate only $0 < N_j < N$ heuristics for B_j so that $\sum_{j=1}^M N_j = N$. The exact numbers are determined as follows.

LR selection. The number of times B_j is considered for generating new heuristics is determined by sampling N times from R with probability $p_j \propto 1/|s_j|$ if $q(x, y) < 0$ (e.g., TSP) and $p_j \propto s_j$ if $q(x, y) \geq 0$ (e.g., knapsack problems), which resembles the selection method in EoH (Liu et al., 2024a) for selecting parent heuristics. Thus, better-performing reductions are more likely to have larger N_j .

3.3 REDUCTION REFINEMENT

During evolution, one LR may drastically outperform others (e.g., due to inadequate implementations), securing large ration and in turn monopolizing nearly all heuristics in P . Since the search now effectively collapses to typical LLM-EPS, this behavior may lead to premature convergence at local optima (Zheng et al., 2025b). To avoid this, RedAHD automatically refines LRs whenever their score stagnates. In particular, for each r_j when s_j does not improve for T consecutive units of evaluation budget (e.g., number of generations or fitness evaluations), the reduction functions (f_j, g_j) as well as the corresponding code template for B_j are updated by prompting the LLM with both A and B_j ’s descriptions along with their current version. Once updated, the fitness values of the heuristics associated with r_j are recomputed (through the new (f_j, g_j)), which in turn updates s_j . RedAHD keeps the update for r_j only if s_j is improved. The exact prompt used is detailed in Appendix D.1.

324 **4 EXPERIMENTS**

325

326 We start this section by describing the experimental settings and the considered baselines in Sec-
 327 tion 4.1. Section 4.2 presents results on six COPs for evaluating the efficacy of RedAHD. Finally,
 328 we provide several ablation studies in Section 4.3 to grasp its individual components’ impact on
 329 optimization performance. Appendix D includes all implementation details and missing results.

330

331 **4.1 EXPERIMENTAL SETUPS**

332

333 To our best knowledge, the current state-of-the-art LLM-EPS method is MCTS-AHD (Zheng et al.,
 334 2025b). Therefore, we follow their setups whenever possible, including the considered COPs, the
 335 evaluation dataset for each COP, and the respective baselines (from handcrafted heuristics, tradi-
 336 tional AHD methods, NCO methods, and other LLM-EPS methods). The COPs consist of TSPs,
 337 CVRPs, 0/1 knapsack problems (KPs), multiple knapsack problems (MKPs), and online and offline
 338 bin packing problems (OBPPs and BPPs, respectively). For RedAHD, we set $M = 3$, $M_{init} = 10$,
 339 and $l = 3$. We use EoH (Liu et al., 2024a) as the default LLM-EPS method, in which we use only
 340 two variation operators, one for crossover and the other for mutation, instead of five as in the original
 341 work (see prompt specifications and our justifications in Appendix D.1). We set T , the number of
 342 generations in EoH context, to 3. Unless otherwise specified, GPT-4o-mini with temperature fixed
 343 at 1 is employed as the designer LLM for generating both LRs and heuristics, with each run of
 344 RedAHD repeated three times and we report the average performance of h^* .

345

346 **4.2 MAIN RESULTS**

347

348 Recall that existing LLM-EPS methods necessitate some predetermined GAF to operate. Hence, we
 compare RedAHD with LLM-EPS methods when integrated within the IC and ACO frameworks.

349 **Iterative Construction (IC).** This GAF, also known as step-by-step construction, constructs the
 350 solution components of a given COP one by one (Asani et al., 2023). By this means, when dealing
 351 with TSP for example, LLM-EPS methods only need to design h' that takes the distance matrix and
 352 the currently visiting and unvisited nodes as input and returns the next node to visit. It has been
 353 considered in all known LLM-EPS works (see Table 1), particularly for TSP, KP, and OBPP. Table
 354 3 shows the performance of RedAHD on these COPs with respect to the baselines. We see that
 355 for TSP and KP, RedAHD not only outperforms EoH, the underlying LLM-EPS, but also achieves
 356 the best or second best performance on all test sets. For OBPP, despite surpassing the handcrafted
 357 heuristics “Best Fit” and “First Fit” in nearly all settings, RedAHD performs rather unremarkably
 358 compared to LLM-EPS methods. We attribute this decrease in relative performance to the fact that
 359 for OBPP in particular, the additional constraint that each item must be packed sequentially without
 360 knowledge on future items greatly restricts H' and hence exploring novel heuristics via the proposed
 361 multi-problem LLM-EPS is less beneficial. We show in Section 4.3 that RedAHD can still excel with
 362 more capable LLMs.

363 Table 3: Comparative results for (left) TSP & KP and (right) OBPP when LLM-EPS methods (denoted by
 364 an asterisk) employ the IC framework. We use the results reported in Zheng et al. (2025b) for the baselines.
 365 n is the number of nodes to visit for TSP and number of items to consider for KP and OBPP, and W is the
 366 knapsack capacity for KP and bin size for OBPP. ID settings are underlined while OOD settings are not. The
 367 best-performing LLM-based method (with GPT-4o-mini) is shaded, and the overall best method is bolded.

Method	Problem setting	OBPP (% optimality gap ↓)					
		TSP (Obj. ↓)			KP (Obj. ↑)		
		<u>n=50</u>	<u>n=100</u>	<u>n=200</u>	<u>W=12.5</u>	<u>W=25</u>	<u>W=25</u>
Greedy 1977		6.959	9.706	13.461	19.985	40.225	57.395
POMO 2020		5.697	8.001	12.897	19.612	39.676	57.271
Funsearch*		6.357	8.850	12.372	19.988	40.227	57.398
EoH*		6.394	8.894	12.437	19.993	40.231	57.399
MCTS-AHD*		6.225	8.684	12.120	20.015	40.252	57.423
RedAHD		5.767	8.006	11.164	20.006	40.248	57.416

375 Table 4 shows additional results on TSPLib (Reinelt, 1991), a standard real-world TSP benchmark.
 376 Following prior LLM-EPS works (Liu et al., 2024a; Ye et al., 2024; Zheng et al., 2025b), we use
 377 the best-performing heuristic among the three runs of RedAHD to report its performance. Since this
 heuristic (depicted in Appendix D.4 under “TSP”) was found to randomly select a starting node,

378 we run it three times for each TSPLib instance and report the average performance. Clearly, the
 379 heuristic from RedAHD outperforms all baselines on every instance, achieving small optimality gap
 380 even on very large instances with over 1,500 nodes (shaded in green). On the other hand, LLM-
 381 EPS methods often fail to surpass handcrafted heuristics (e.g., Christofides (Christofides, 2022)),
 382 particularly on larger instances with a few hundred nodes or more.

383 Table 4: Results (% optimality gap) on TSPLib when LLM-EPS methods (denoted by an asterisk) employ the
 384 IC framework. The number from each instance’s name corresponds to the number of nodes. We use the results
 385 reported in Duflo et al. (2019); Ye et al. (2024); Zheng et al. (2025b) for the baselines. The best baseline is
 386 shaded in gray, and the overall best is bolded.

TSPLib instance	Christofides 2022	Greedy 2015	Nearest insertion	Nearest neighbor 1977	GPHH-best 2019	EoH*	ReEvo*	MCTS-AHD*	RedAHD
ts225	5.67	5.38	19.93	16.82	7.71	5.57	6.56	10.84	2.29 ± 0.21
rat99	9.43	22.30	21.05	21.79	14.09	18.78	12.41	10.46	3.47 ± 0.08
rl1889	7.60	19.44	24.34	23.74	21.09	-	17.5	-	6.87 ± 0.61
u1817	14.15	19.78	24.07	22.20	21.21	-	16.6	-	6.42 ± 0.16
d1655	12.65	16.31	21.35	23.86	18.69	-	17.5	-	7.10 ± 0.34
bier127	13.03	19.50	23.05	23.25	15.64	14.05	10.79	7.56	2.32 ± 0.38
lin318	13.80	18.75	24.44	25.78	14.30	14.03	16.63	14.07	5.39 ± 0.17
eil51	15.18	13.03	16.14	31.96	10.20	8.37	6.47	15.98	2.29 ± 0.48
d493	9.52	16.68	20.39	24.00	15.58	12.41	13.43	11.73	3.83 ± 0.28
kroB100	9.82	16.59	21.53	26.26	14.06	13.46	12.20	11.43	2.12 ± 0.84
kroC100	9.08	12.94	24.25	25.76	16.22	16.85	15.88	8.27	3.64 ± 0.24
ch130	10.09	28.40	19.21	25.66	14.77	12.26	9.40	10.18	4.51 ± 0.69
pr299	11.23	31.42	25.05	31.42	18.24	23.58	20.63	11.23	5.45 ± 0.33
f417	15.57	12.64	25.52	32.42	22.72	20.47	19.15	10.20	3.43 ± 0.52
d657	10.41	15.76	22.84	29.74	16.30	-	16.0	-	5.34 ± 0.61
kroA150	13.44	20.24	19.09	26.08	15.59	18.36	11.62	10.08	3.62 ± 0.31
f1577	8.84	15.60	24.17	25.01	17.60	-	12.1	-	3.17 ± 0.38
u724	12.04	17.20	25.58	28.45	15.54	-	16.9	-	5.08 ± 0.38
pr264	11.28	11.89	34.28	17.87	23.96	18.03	16.78	12.27	4.97 ± 1.05
pr226	14.17	21.44	28.02	24.65	15.51	19.90	18.02	7.15	1.97 ± 1.13
pr439	11.16	20.08	24.67	27.36	21.36	21.96	19.25	15.12	5.65 ± 0.50

403
 404 **Ant Colony Optimization (ACO).** ACO (Dorigo et al., 2007) is an advanced and well-known
 405 GAF that had been applied to more complex COPs such as CVRP and MKP (which are respectively
 406 more general COPs than TSP and KP). Under this framework, LLM-EPS methods only need to
 407 design heuristics for estimating the potential of each solution component, which is then used as prior
 408 information to bias the stochastic sampling of solutions (Ye et al., 2024; Zheng et al., 2025b). Our
 409 results for RedAHD on TSP, CVRP, MKP, and BPP with respect to baselines employing ACO are
 410 shown in Table 5. Being self-contained, RedAHD still outperforms LLM-EPS methods in nearly all
 411 OOD settings and yields competitive performance against them in ID settings. RedAHD also stays
 412 competitive against DeepACO (Ye et al., 2023), a representative NCO method based on ACO, in all
 413 COPs except CVRP. We show in Section 4.3 that the lackluster performance of RedAHD on CVRP,
 414 which we believe to be due to the lack of domain knowledge from GPT-4o-mini, can be significantly
 415 improved and even tops DeepACO with more capable LLMs.

416 Table 5: Comparative results for TSP, CVRP, MKP, and BPP when LLM-EPS methods (denoted by an asterisk)
 417 employ the ACO framework. We use the results reported in Zheng et al. (2025b) for the baselines. n : number
 418 of nodes to visit for TSP and CVRP and number of items to consider for MKP and BPP; C : vehicle capacity
 419 for CVRP; m : number of knapsacks for MKP; W : bin size for BPP. ID settings are underlined while OOD
 420 settings are not. The best-performing LLM-based method (with GPT-4o-mini) is shaded, and the overall best
 421 method is bolded.

Method	Problem setting	TSP (Obj. ↓)		CVRP (Obj. ↓)		MKP (Obj. ↑)		BPP (Obj. ↓)	
		<u>$n=50$</u>	<u>$n=100$</u>	<u>$n=50$</u>	<u>$n=100$</u>	<u>$n=100$</u>	<u>$n=200$</u>	<u>$n=500$</u>	<u>$n=1,000$</u>
ACO 2007		5.992	8.948	11.355	18.778	22.738	40.672	208.828	417.938
DeepACO 2023		5.842	8.282	8.888	14.932	23.093	41.988	203.125	405.172
EoH*		5.828	8.263	9.359	15.681	23.139	41.994	204.646	408.599
ReEvo*		5.856	8.340	9.327	16.092	23.245	42.416	206.693	413.510
MCTS-AHD*		5.801	8.179	9.286	15.782	23.269	42.498	204.094	407.323
RedAHD		5.819	8.039	9.826	15.726	23.164	42.682	203.344	405.359

4.3 ABLATION STUDIES

428 **Reduction Refinement.** In our experiments, for $T = 3$, the reduction refinement step in RedAHD
 429 was called at least once up to three times. We validate the necessity of this step by rerunning the
 430 experiments in Table 5 without it. As shown in Table 6, RedAHD exhibits a decrease in perfor-

mance across all COPs and barely surpasses EoH. This performance drop is likely due to premature convergence at local optima during search as discussed in Section 3.3.

Table 6: Ablation of the reduction refinement step. Results from EoH in Table 5 are used as references.

Method	Problem setting	TSP (Obj. \downarrow)		CVRP (Obj. \downarrow)		MKP (Obj. \uparrow)		BPP (Obj. \downarrow)	
		$n=50$	$n=100$	$n=50$	$n=100$	$n=100$	$n=200$	$n=500$	$n=1,000$
		$C=50$	$C=50$	$C=50$	$C=50$	$m=5$	$m=5$	$W=150$	$W=150$
EoH*		5.828	8.263	9.359	15.681	23.139	41.994	204.646	408.599
RedAHD (w/o reduction refinement)		5.847	8.322	10.218	16.175	23.126	41.978	204.561	407.639
RedAHD		5.819	8.039	9.826	15.726	23.164	42.682	203.344	405.359

The Designer LLM. The impressive performance from RedAHD across multiple COPs up to this point was achieved using GPT-4o-mini, a lightweight general-purpose LLM that had been shown to be poor at algorithmic reasoning (Yang et al., 2025). Therefore, we should expect RedAHD to improve when more capable LLMs, particularly reasoning models such as o3-mini, are employed. Table 7 verifies our claim, where the originally unremarkable performance from RedAHD on OBPP and CVRP is significantly improved and even surpasses the best baseline on multiple settings. Notably, for the OOD setting of CVRP ($N = 100$ and $C = 50$), RedAHD yields objective values even better than those returned from OR-Tools, an optimization library dedicated for vehicle routing problems (Furnon & Perron).

Table 7: Ablation of the designer LLM. Truncated results from Tables 3 and 5 are used as references.

OBPP (% optimality gap \downarrow)						
n (number of items)	1k	1k	5k	5k	10k	10k
W (bin capacity)	100	500	100	500	100	500
Best baseline	2.45	0.25	1.06	0.25	0.74	0.21
EoH*	2.69	0.25	1.63	0.53	1.47	0.45
RedAHD (GPT-4o-mini)	3.78	0.99	2.82	0.55	2.61	0.40
RedAHD (o3-mini)	3.13	0.00	2.33	0.30	2.02	0.20

CVRP (Obj. \downarrow)		
n (number of nodes)	50	100
C (vehicle capacity)	50	50
OR-Tools (Furnon & Perron)	8.314	13.948
Best baseline (DeepACO)	8.888	14.932
EoH*	9.359	15.681
RedAHD (GPT-4o-mini)	9.826	15.726
RedAHD (o3-mini)	8.348	13.516

The LLM-EPS Method. We demonstrate that RedAHD can work with LLM-EPS methods other than EoH, namely ReEvo (Ye et al., 2024) and MEoH (Yao et al., 2025). As shown in Table 8, RedAHD improves the performance of the corresponding LLM-EPS methods even without the need of GAFs. In particular, RedAHD[EoH] and RedAHD[ReEvo] respectively outperform EoH and ReEvo, where the latter two operate under the ACO framework. Moreover, as LLM-EPS methods improve, exemplified here by MEoH (which extends EoH to multi-objective heuristic search with runtime as the additional fitness criterion), RedAHD may yield further improvement, now outperforming the best baseline in the ID setting of TSP ($N = 50$). This result verifies the applicability of our proposed framework in the emerging field of LLM-based AHD.

Table 8: Ablation of the underlying LLM-EPS method. Truncated results from Table 5 are used as references. RedAHD[EoH] is RedAHD reported in earlier results. For RedAHD[MEoH], which also optimizes runtime, we report the average performance from heuristics that yield the best objective values.

TSP (Obj. \downarrow)		
n (number of nodes)	50	100
Best baseline	5.801	8.179
EoH*	5.828	8.263
ReEvo*	5.856	8.340
RedAHD[EoH]	5.819	8.039
RedAHD[ReEvo]	5.835	8.251
RedAHD[MEoH]	5.730	7.883

5 CONCLUSION

In this paper, we propose RedAHD, the first framework toward end-to-end automatic design of heuristics with LLMs. RedAHD leverages the concept of reduction for enabling contemporary LLM-EPS methods to operate without the need of GAFs, which significantly reduces manual efforts from human designers. Furthermore, RedAHD facilitates the discovery of novel heuristics from uncharted heuristic space, resulting in improved optimization performance over state-of-the-art methods. As the capabilities of LLMs and LLM-EPS methods continue to grow, we envision the efficacy of RedAHD in solving COPs would be more evident.

6 REPRODUCIBILITY STATEMENT

We refer readers to Section 4.1 as well as Appendix D.1 for complete details on reproducing our results. We also include the code for our work in the supplemental material.

486 REFERENCES
487

488 Emmanuel O Asani, Aderemi E Okeyinka, and Ayodele Ariyo Adebiyi. A computation investi-
489 gation of the impact of convex hull subtour on the nearest neighbour heuristic. In *2023 Inter-
490 national Conference on Science, Engineering and Business for Sustainable Development Goals
491 (SEB-SDG)*, volume 1, pp. 1–7. IEEE, 2023.

492 Thomas Bäck, David B Fogel, and Zbigniew Michalewicz. Handbook of evolutionary computation.
493 *Release*, 97(1):B1, 1997.

494 Yoshua Bengio, Andrea Lodi, and Antoine Prouvost. Machine learning for combinatorial opti-
495 mization: a methodological tour d’horizon. *European Journal of Operational Research*, 290(2):
496 405–421, 2021.

497 Judith Brecklinghaus and Stefan Hougardy. The approximation ratio of the greedy algorithm for the
498 metric traveling salesman problem. *Operations Research Letters*, 43(3):259–261, 2015.

499 500 Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal,
501 Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are
502 few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

503 504 Edmund K Burke, Michel Gendreau, Matthew Hyde, Graham Kendall, Gabriela Ochoa, Ender
505 Özcan, and Rong Qu. Hyper-heuristics: A survey of the state of the art. *Journal of the Op-
506 erational Research Society*, 64(12):1695–1724, 2013.

507 508 Herbert G Campbell, Richard A Dudek, and Milton L Smith. A heuristic algorithm for the n job, m
509 machine sequencing problem. *Management science*, 16(10):B–630, 1970.

510 511 Jinbiao Chen, Jiahai Wang, Zizhen Zhang, Zhiguang Cao, Te Ye, and Siyuan Chen. Efficient meta
512 neural heuristic for multi-objective combinatorial optimization. *Advances in Neural Information
Processing Systems*, 36:56825–56837, 2023.

513 514 Nicos Christofides. Worst-case analysis of a new heuristic for the travelling salesman problem. In
515 *Operations Research Forum*, volume 3, pp. 20. Springer, 2022.

516 517 Pierluigi Crescenzi. A short guide to approximation preserving reductions. In *Proceedings of Com-
518 putational Complexity. Twelfth Annual IEEE Conference*, pp. 262–273. IEEE, 1997.

519 520 George B Dantzig and John H Ramser. The truck dispatching problem. *Management science*, 6(1):
521 80–91, 1959.

522 523 Pham Vu Tuan Dat, Long Doan, and Huynh Thi Thanh Binh. Hsevo: Elevating automatic heuristic
524 design with diversity-driven harmony search and genetic algorithm using llms. In *Proceedings of
525 the AAAI Conference on Artificial Intelligence*, volume 39, pp. 26931–26938, 2025.

526 527 Sachin Desale, Akhtar Rasool, Sushil Andhale, and Priti Rane. Heuristic and meta-heuristic algo-
528 rithms and their relevance to the real world: a survey. *Int. J. Comput. Eng. Res. Trends*, 351(5):
529 2349–7084, 2015.

530 531 Marco Dorigo, Mauro Birattari, and Thomas Stutzle. Ant colony optimization. *IEEE computational
532 intelligence magazine*, 1(4):28–39, 2007.

533 534 John H Drake, Ahmed Kheiri, Ender Özcan, and Edmund K Burke. Recent advances in selection
535 hyper-heuristics. *European Journal of Operational Research*, 285(2):405–428, 2020.

536 537 Gabriel Duflo, Emmanuel Kieffer, Matthias R Brust, Grégoire Danoy, and Pascal Bouvry. A gp
538 hyper-heuristic approach for generating tsp heuristics. In *2019 IEEE International Parallel and
539 Distributed Processing Symposium Workshops (IPDPSW)*, pp. 521–529. IEEE, 2019.

540 541 Agoston E Eiben and Jim Smith. From evolutionary computation to the evolution of things. *Nature*,
542 521(7553):476–482, 2015.

543 544 Hamilton Emmons and George Vairaktarakis. *Flow shop scheduling: theoretical results, algorithms,
545 and applications*, volume 182. Springer Science & Business Media, 2012.

540 Victor Fernandez-Viagas and Jose M Framinan. On insertion tie-breaking rules in heuristics for the
 541 permutation flowshop scheduling problem. *Computers & Operations Research*, 45:60–67, 2014.

542

543 Vincent Furnon and Laurent Perron. Or-tools routing library. URL <https://developers.google.com/optimization/routing/>.

544

545 Qingyan Guo, Rui Wang, Junliang Guo, Bei Li, Kaitao Song, Xu Tan, Guoqing Liu, Jiang Bian,
 546 and Yujiu Yang. Connecting large language models with evolutionary algorithms yields powerful
 547 prompt optimizers. In *The Twelfth International Conference on Learning Representations*, 2024.
 548 URL <https://openreview.net/forum?id=ZG3RaNIso8>.

549

550 Jatinder ND Gupta. A functional heuristic algorithm for the flowshop scheduling problem. *Journal
 551 of the Operational Research Society*, 22(1):39–47, 1971.

552

553 Erik Hemberg, Stephen Moskal, and Una-May O'Reilly. Evolving code with a large language model.
 554 *Genetic Programming and Evolvable Machines*, 25(2):21, 2024.

555

556 Lei Huang, Weijiang Yu, Weitao Ma, Weihong Zhong, Zhangyin Feng, Haotian Wang, Qianglong
 557 Chen, Weihua Peng, Xiaocheng Feng, Bing Qin, et al. A survey on hallucination in large language
 558 models: Principles, taxonomy, challenges, and open questions. *ACM Transactions on Information
 559 Systems*, 43(2):1–55, 2025.

560

561 Brian Kallehauge, Jesper Larsen, Oli BG Madsen, and Marius M Solomon. Vehicle routing problem
 562 with time windows. In *Column generation*, pp. 67–98. Springer, 2005.

563

564 Subbarao Kambhampati, Karthik Valmeeckam, Lin Guan, Mudit Verma, Kaya Stechly, Siddhant
 565 Bhambri, Lucas Paul Saldyt, and Anil B Murthy. Position: Llms can't plan, but can help planning
 566 in llm-modulo frameworks. In *Forty-first International Conference on Machine Learning*, 2024.

567

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

571

572 William B Langdon and Riccardo Poli. *Foundations of genetic programming*. Springer Science &
 573 Business Media, 2013.

574

575 Robert Lange, Yingtao Tian, and Yujin Tang. Large language models as evolution strategies. In
 576 *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pp. 579–582,
 577 2024.

578

579 Joel Lehman, Jonathan Gordon, Shawn Jain, Kamal Ndousse, Cathy Yeh, and Kenneth O Stanley.
 580 Evolution through large models. In *Handbook of evolutionary machine learning*, pp. 331–366.
 581 Springer, 2023.

582

583 Fei Liu, Tong Xialiang, Mingxuan Yuan, Xi Lin, Fu Luo, Zhenkun Wang, Zhichao Lu, and Qingfu
 584 Zhang. Evolution of heuristics: Towards efficient automatic algorithm design using large language
 585 model. In *Forty-first International Conference on Machine Learning*, 2024a.

586

587 Fei Liu, Yiming Yao, Ping Guo, Zhiyuan Yang, Xi Lin, Xialiang Tong, Mingxuan Yuan, Zhichao Lu,
 588 Zhenkun Wang, and Qingfu Zhang. A systematic survey on large language models for algorithm
 589 design. *arXiv preprint arXiv:2410.14716*, 2024b.

590

591 Fei Liu, Rui Zhang, Zhuoliang Xie, Rui Sun, Kai Li, Xi Lin, Zhenkun Wang, Zhichao Lu, and
 592 Qingfu Zhang. Llm4ad: A platform for algorithm design with large language model. 2024c.
 593 URL <https://arxiv.org/abs/2412.17287>.

594

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

598

599 Zeyuan Ma, Hongshu Guo, Jiacheng Chen, Zhenrui Li, Guojun Peng, Yue-Jiao Gong, Yining Ma,
 600 and Zhiguang Cao. Metabox: A benchmark platform for meta-black-box optimization with re-
 601inforcement learning. *Advances in Neural Information Processing Systems*, 36:10775–10795,
 602 2023.

594 Kyle Mahowald, Anna A Ivanova, Idan A Blank, Nancy Kanwisher, Joshua B Tenenbaum, and
 595 Evelina Fedorenko. Dissociating language and thought in large language models. *Trends in*
 596 *cognitive sciences*, 2024.

597 Rajesh Matai, Surya Prakash Singh, and Murari Lal Mittal. Traveling salesman problem: an
 598 overview of applications, formulations, and solution approaches. *Traveling salesman problem,*
 599 *theory and applications*, 1(1):1–25, 2010.

600 Yi Mei, Qi Chen, Andrew Lensen, Bing Xue, and Mengjie Zhang. Explainable artificial intelligence
 601 by genetic programming: A survey. *IEEE Transactions on Evolutionary Computation*, 27(3):
 602 621–641, 2022.

603 Elliot Meyerson, Mark J Nelson, Herbie Bradley, Adam Gaier, Arash Moradi, Amy K Hoover,
 604 and Joel Lehman. Language model crossover: Variation through few-shot prompting. *ACM*
 605 *Transactions on Evolutionary Learning*, 4(4):1–40, 2024.

606 Muhammad Nawaz, E Emory Enscore Jr, and Inyong Ham. A heuristic algorithm for the m-machine,
 607 n-job flow-shop sequencing problem. *Omega*, 11(1):91–95, 1983.

608 Michael O’Neill, Leonardo Vanneschi, Steven Gustafson, and Wolfgang Banzhaf. Open issues in
 609 genetic programming. *Genetic Programming and Evolvable Machines*, 11:339–363, 2010.

610 Zixiao Pan, Ling Wang, Jingjing Wang, and Jiawen Lu. Deep reinforcement learning based opti-
 611 mization algorithm for permutation flow-shop scheduling. *IEEE Transactions on Emerging Topics*
 612 *in Computational Intelligence*, 7(4):983–994, 2021.

613 Nelishia Pillay and Rong Qu. *Hyper-heuristics: theory and applications*. Springer, 2018.

614 Gerhard Reinelt. Tsplib—a traveling salesman problem library. *ORSA journal on computing*, 3(4):
 615 376–384, 1991.

616 Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Matej Balog,
 617 M Pawan Kumar, Emilien Dupont, Francisco JR Ruiz, Jordan S Ellenberg, Pengming Wang,
 618 Omar Fawzi, et al. Mathematical discoveries from program search with large language models.
 619 *Nature*, 625(7995):468–475, 2024.

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

622 Marcus Tantakoun, Christian Muise, and Xiaodan Zhu. Llms as planning formalizers: A survey
 623 for leveraging large language models to construct automated planning models. In *Findings of the*
 624 *Association for Computational Linguistics: ACL 2025*, pp. 25167–25188, 2025.

625 Christos Voudouris, Edward PK Tsang, and Abdullah Alsheddy. Guided local search. In *Handbook*
 626 *of metaheuristics*, pp. 321–361. Springer, 2010.

627 Hui Wei, Zihao Zhang, Shenghua He, Tian Xia, Shijia Pan, and Fei Liu. PlanGenLLMs: A modern
 628 survey of LLM planning capabilities. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and
 629 Mohammad Taher Pilehvar (eds.), *Proceedings of the 63rd Annual Meeting of the Association*
 630 *for Computational Linguistics (Volume 1: Long Papers)*, pp. 19497–19521, Vienna, Austria, July
 631 2025. Association for Computational Linguistics. ISBN 979-8-89176-251-0. doi: 10.18653/v1/
 632 2025.acl-long.958. URL <https://aclanthology.org/2025.acl-long.958/>.

633 Xingyu Wu, Sheng-hao Wu, Jibin Wu, Liang Feng, and Kay Chen Tan. Evolutionary computation
 634 in the era of large language model: Survey and roadmap. *IEEE Transactions on Evolutionary*
 635 *Computation*, 2024.

636 Chang Yang, Ruiyu Wang, Junzhe Jiang, Qi Jiang, Qinggang Zhang, Yanchen Deng, Shuxin Li,
 637 Shuyue Hu, Bo Li, Florian T. Pokorny, Xiao Huang, and Xinrun Wang. Nondeterministic
 638 polynomial-time problem challenge: An ever-scaling reasoning benchmark for llms, 2025. URL
 639 <https://arxiv.org/abs/2504.11239>.

648 Yunhao Yang and Andrew Whinston. A survey on reinforcement learning for combinatorial opti-
 649 mization. In *2023 IEEE World Conference on Applied Intelligence and Computing (AIC)*, pp.
 650 131–136. IEEE, 2023.

651

652 Shunyu Yao, Fei Liu, Xi Lin, Zhichao Lu, Zhenkun Wang, and Qingfu Zhang. Multi-objective
 653 evolution of heuristic using large language model. In *Proceedings of the AAAI Conference on*
 654 *Artificial Intelligence*, volume 39, pp. 27144–27152, 2025.

655

656 Haoran Ye, Jiarui Wang, Zhiguang Cao, Helan Liang, and Yong Li. Deepaco: Neural-enhanced ant
 657 systems for combinatorial optimization. *Advances in neural information processing systems*, 36:
 658 43706–43728, 2023.

659

660 Haoran Ye, Jiarui Wang, Zhiguang Cao, Federico Berto, Chuanbo Hua, Haeyeon Kim, Jinkyoo Park,
 661 and Guojie Song. Reevo: Large language models as hyper-heuristics with reflective evolution. In
 662 *Advances in Neural Information Processing Systems*, 2024. [https://github.com/ai4co/](https://github.com/ai4co/reevo)
 663 reevo.

664

665 Qi Zhao, Qiqi Duan, Bai Yan, Shi Cheng, and Yuhui Shi. Automated design of metaheuristic
 666 algorithms: A survey. *arXiv preprint arXiv:2303.06532*, 2023.

667

668 Yue Zheng, Yuhao Chen, Bin Qian, Xiufang Shi, Yuanchao Shu, and Jiming Chen. A review on
 669 edge large language models: Design, execution, and applications. *ACM Computing Surveys*, 57
 670 (8):1–35, 2025a.

671

672 Zhi Zheng, Changliang Zhou, Tong Xialiang, Mingxuan Yuan, and Zhenkun Wang. Udc: A unified
 673 neural divide-and-conquer framework for large-scale combinatorial optimization problems. *arXiv*
 674 *preprint arXiv:2407.00312*, 2024.

675

676

677 Zhi Zheng, Zhuoliang Xie, Zhenkun Wang, and Bryan Hooi. Monte carlo tree search for com-
 678 prehensive exploration in llm-based automatic heuristic design. In *Forty-Second International*
 679 *Conference on Machine Learning*, 2025b.

680

A RELATED WORK

681 **Automatic Heuristic Design (AHD).** The field of AHD, or hyper-heuristics (Pillay & Qu, 2018),
 682 aims to provide more generalized approaches for solving COPs via selecting the best-performing
 683 heuristic from a predefined set (Drake et al., 2020) or generating new heuristics through the com-
 684 bination of simpler heuristic components (Duflo et al., 2019; Zhao et al., 2023). By this means,
 685 human experts are only required to specify the heuristic space rather than handcrafting heuristics
 686 from scratch. However, traditional AHD approaches such as those employing GP (Langdon & Poli,
 687 2013) necessitate substantial domain knowledge and implementation efforts (Pillay & Qu, 2018;
 688 O’Neill et al., 2010).

689 **LLMs for AHD.** Recent advances in LLMs have enabled new approaches for AHD. (Please refer
 690 to the latest survey by Liu et al. (2024b)) for a comprehensive review.) Since standalone LLMs
 691 with prompt engineering are arguably incapable of producing novel algorithmic ideas beyond their
 692 encoded knowledge (Mahowald et al., 2024), most active research in this area focuses on integrating
 693 LLMs into an evolutionary computation (EC) procedure to iteratively refine a set of heuristics. EC
 694 is a generic optimization principle inspired by natural evolution (Bäck et al., 1997; Eiben & Smith,
 695 2015). Its idea involves iteratively improving a set of candidate solutions through score-based selec-
 696 tion (i.e., identifying the “fittest” candidate solutions subject to a so-called fitness function such as
 697 the optimality gap) and stochastic variation operators (e.g., crossover and mutation among the fittest
 698 candidate solutions as inspired by biological evolution). In recent years, LLMs have been employed
 699 via prompt engineering to emulate these variation operators (Lehman et al., 2023; Meyerson et al.,
 700 2024; Lange et al., 2024), with already widespread applications in code generation (Hemberg et al.,
 701 2024), text generation (Guo et al., 2024), planning (Kambhampati et al., 2024), as well as AHD,
 702 known in the literature as LLM-based evolutionary program search (LLM-EPS) (Liu et al., 2024d;
 703 Dat et al., 2025). Representative LLM-EPS methods include FunSearch (Romera-Paredes et al.,
 704 2024), EoH (Liu et al., 2024a), ReEvo (Ye et al., 2024), HSEvo (Dat et al., 2025), MeoH (Yao et al.,

2025), and most recently MCTS-AHD (Zheng et al., 2025b) (Figure 1). Despite generally outperforming handcrafted heuristics and GP-based AHD methods while reducing manual interventions, as mentioned in Section 1, they rely on some predetermined GAF such as IC and ACO to operate, which still involves domain knowledge and implementation efforts from human users, and hence are far from being end-to-end. In response, our work enables existing LLM-EPS methods to circumvent this limitation and potentially improves their performance in the process.

Neural Combinatorial Optimization (NCO). NCO is an end-to-end AHD approach that employs neural networks to search for the optimal parameter settings within a parameterized heuristic space (Bengio et al., 2021; Yang & Whinston, 2023). Despite not requiring domain knowledge and being applicable to multiple COPs (Chen et al., 2023; Ma et al., 2023), compared to LLM-EPS methods, they are resource-intensive (Kwon et al., 2020), hard to implement (Zheng et al., 2024), and may yield subpar results in various experimental settings (Liu et al., 2024a; Ye et al., 2024; Zheng et al., 2025b), being outperformed by the state-of-the-art LLM-EPS method, MCTS-AHD, even under the simple IC framework when solving TSP and the 0/1 knapsack problem (Zheng et al., 2025b) for instance. (Please refer to existing LLM-EPS works for a more comprehensive comparison with NCO methods.)

B CONSIDERED COPs

In this appendix, we introduce the considered COPs and define the objective function for each (q in Equations 1 and 2). We follow the problem definitions and setups from Zheng et al. (2025b) (which followed Ye et al. (2024)). TSP, CVRP, BPP, and OBPP are minimization problems while KP and MKP are maximization problems.

Traveling Salesman Problem (TSP). TSP aims to find the shortest path to visit each of the n nodes once and return to the starting node. Each TSP instance contains the Euclidean distance matrix \mathbf{D} where d_{ij} denotes the cost between node i and j . The solution of TSP is a permutation of all node indices $\mathbf{s} = (s_1, s_2, \dots, s_n)$. Thus, the (negated) objective function is

$$-\left(\sum_{t=1}^{n-1} d_{s_t, s_{t+1}} + d_{s_n, s_1}\right).$$

Capacitated Vehicle Routing Problem (CVRP). CVRP aims to plan several capacity-constrained vehicles starting at and returning to a depot, meeting the demands of multiple customers, and minimizing the total travel distance. Each CVRP instance contains a depot (the 0-th node) and n customers. Let \mathbf{D} be the Euclidean distance matrix. The (negated) objective function is

$$\begin{aligned} & -\sum_{j=1}^q C(\rho^j), \\ & C(\rho^j) = \sum_{t=0}^{|\rho^j|-1} d_{\rho_t^j, \rho_{t+1}^j}^j + d_{\rho_{n_j}^j, \rho_0^j}, \\ & \text{s.t. } 0 \leq \delta_i \leq C, \quad \sum_{i \in \rho^j} \delta_i \leq C, \quad i \in \{1, \dots, n\}, j \in \{1, \dots, q\}, \end{aligned}$$

where \mathbf{s} is a solution representing the complete route of vehicles and consists of q sub-routes $\mathbf{s} = \{\rho^1, \rho^2, \dots, \rho^q\}$. Each sub-route $\rho^j = (\rho_1^j, \dots, \rho_{n_j}^j)$, $j \in \{1, \dots, q\}$ starts from the depot s_0 and goes back to s_0 ; n_j represents the number of customer nodes in it such that $n = \sum_{j=1}^q n_j$. δ_i denotes the demand of node i , and C denotes the capacity of the vehicles.

(Offline) Bin Packing Problem (BPP). BPP aims to place a set of n items with different sizes into as few bins as possible, each of which has capacity of W . The solution of BPP is $\mathbf{s} = \{s^1, s^2, \dots, s^K\}$ where s^i is the set of item indices for the i -th bin and K is the number of bins used. The (negated) objective function is

$$\begin{aligned} & -K, \\ & \text{s.t. } \sum_{j \in s^i} w_j \leq W, \quad i \in \{1, \dots, K\}. \end{aligned}$$

Online Bin Packing Problem (OBPP). OBPP additionally requires making an immediate decision on which bin to place once a new item arrives, without any information on future items. The objective function is similar to BPP.

756 **0/1 Knapsack Problem (KP).** KP aims to pack items of maximum total value to a knapsack with
 757 capacity W . Each of the n available items can only be picked once. The solution of KP is the set
 758 of indices of the selected items $s \subseteq \{1, 2, \dots, n\}$. Let w_j and v_j be the weight and value of item j ,
 759 respectively. The objective function is

760
$$\text{s.t.} \quad \sum_{j \in s} v_j,$$

 761
$$\sum_{j \in s} w_j \leq W.$$

763 **Multiple Knapsack Problem (MKP).** MKP extends KP to $m > 1$ knapsacks. The solution of
 764 MKP is now $s = \{s^1, s^2, \dots, s^m\}$ where s^i is the set of indices of the selected items for the i -th
 765 knapsack. The objective function is

766
$$\text{s.t.} \quad \sum_{i=1}^m \sum_{j \in s^i} v_j,$$

 767
$$\sum_{j \in s^i} w_j \leq W_i, \quad i \in \{1, \dots, m\}.$$

770
 771
 772
 773
 774
 775
 776
 777
 778
 779
 780
 781
 782
 783
 784
 785
 786
 787
 788
 789
 790
 791
 792
 793
 794
 795
 796
 797
 798
 799
 800
 801
 802
 803
 804
 805
 806
 807
 808
 809

810 C HOW ACO IS EMPLOYED IN PRIOR LLM-EPS WORKS.

812 As described in Zheng et al. (2025b), ACO is an evolutionary algorithm inspired by the behavior of
 813 ants to find the shortest route between their colony and food sources (Dorigo et al., 2007).

814 ACO records a pheromone matrix τ and a heuristic matrix η . Each item τ_{ij} in τ indicates the
 815 priority of including an edge (i, j) in a solution. The pheromone trails are iteratively updated based
 816 on the quality of the solutions found, encouraging future ants to follow better paths. The heuristic
 817 information on each edge, i.e., η_{ij} , is a problem-specific measure that indicates the immediate benefit
 818 of choosing a particular path. For solving TSP with ACO, for example, η_{ij} is often set to be the
 819 inverse of the distance between cities i and j , i.e., $\eta_{ij} = 1/d_{ij}$. In response, LLM-EPS methods aim
 820 to design a more effective heuristic matrix η based on the problem-specific inputs.

821 Given η , the virtual ants then construct solutions by moving from node to node, probabilistically
 822 choosing the next node based on a combination of pheromone and heuristic information. After all
 823 the ants have constructed their solutions, the pheromone levels update. An ACO iteration typically
 824 involves solution construction, optional local search, and pheromone update. By iteratively applying
 825 these steps, ACO algorithms can effectively explore the solution space and converge toward optimal
 826 or near-optimal solutions for COPs.

827 **Implementation.** The following listings respectively show the Python implementation of ACO
 828 for TSP and CVRP in both ReEvo (Ye et al., 2024) and MCTS-AHD (Zheng et al., 2025b). Albeit
 829 using the same GAF, there are substantial differences between the two pieces of code, which means
 830 significant manual efforts are necessary when adopting ACO (and other GAFs in general) for a
 831 particular COP.

```

833
834 import torch
835 from torch.distributions import Categorical
836
837 class ACO():
838     def __init__(self,
839                  distances,
840                  heuristic,
841                  n_ants=30,
842                  decay=0.9,
843                  alpha=1,
844                  beta=1,
845                  device='cpu'):
846
847         self.problem_size = len(distances)
848         self.distances = torch.tensor(distances, device=device) if not isinstance(distances, torch.Tensor) else
849             distances
850         self.n_ants = n_ants
851         self.decay = decay
852         self.alpha = alpha
853         self.beta = beta
854
855         self.pheromone = torch.ones_like(self.distances)
856         self.heuristic = torch.tensor(heuristic, device=device) if not isinstance(heuristic, torch.Tensor) else
857             heuristic
858
859         self.shortest_path = None
860         self.lowest_cost = float('inf')
861
862         self.device = device
863
864     @torch.no_grad()
865     def run(self, n_iterations):
866         for _ in range(n_iterations):
867             paths = self.gen_path(require_prob=False)
868             costs = self.gen_path_costs(paths)
869
870             best_cost, best_idx = costs.min(dim=0)
871             if best_cost < self.lowest_cost:
872                 self.shortest_path = paths[:, best_idx]
873                 self.lowest_cost = best_cost
874
875             self.update_pheronome(paths, costs)
876
877     @torch.no_grad()
878     def update_pheronome(self, paths, costs):
879         """
880             Args:
881                 paths: torch tensor with shape (problem_size, n_ants)
882                 costs: torch tensor with shape (n_ants,
```

```

864
865    """
866        self.pheromone = self.pheromone * self.decay
867        for i in range(self.n_ants):
868            path = paths[:, i]
869            cost = costs[i]
870            self.pheromone[path, torch.roll(path, shifts=1)] += 1.0/cost
871            self.pheromone[torch.roll(path, shifts=1), path] += 1.0/cost
872
873    @torch.no_grad()
874    def gen_path_costs(self, paths):
875        """
876            Args:
877                paths: torch tensor with shape (problem_size, n_ants)
878            Returns:
879                Lengths of paths: torch tensor with shape (n_ants,)
880
881            assert paths.shape == (self.problem_size, self.n_ants)
882            u = paths.T # shape: (n_ants, problem_size)
883            v = torch.roll(u, shifts=1, dims=1) # shape: (n_ants, problem_size)
884            assert (self.distances[u, v] > 0).all()
885            return torch.sum(self.distances[u, v], dim=1)
886
887    def gen_path(self, require_prob=False):
888        """
889            Tour construction for all ants
890            Returns:
891                paths: torch tensor with shape (problem_size, n_ants), paths[:, i] is the constructed tour of the
892                    ith ant
893                log_probs: torch tensor with shape (problem_size, n_ants), log_probs[i, j] is the log_prob of the
894                    ith action of the jth ant
895
896            start = torch.randint(low=0, high=self.problem_size, size=(self.n_ants,), device=self.device)
897            mask = torch.ones(size=(self.n_ants, self.problem_size), device=self.device)
898            mask[torch.arange(self.n_ants, device=self.device), start] = 0
899
900            paths_list = [] # paths_list[i] is the ith move (tensor) for all ants
901            paths_list.append(start)
902
903            log_probs_list = [] # log_probs_list[i] is the ith log_prob (tensor) for all ants' actions
904
905            prev = start
906            for _ in range(self.problem_size-1):
907                actions, log_probs = self.pick_move(prev, mask, require_prob)
908                paths_list.append(actions)
909                if require_prob:
910                    log_probs_list.append(log_probs)
911                    mask = mask.clone()
912                prev = actions
913                mask[torch.arange(self.n_ants, device=self.device), actions] = 0
914
915            if require_prob:
916                return torch.stack(paths_list), torch.stack(log_probs_list)
917            else:
918                return torch.stack(paths_list)
919
920    def pick_move(self, prev, mask, require_prob):
921        """
922            Args:
923                prev: tensor with shape (n_ants,), previous nodes for all ants
924                mask: bool tensor with shape (n_ants, p_size), masks (0) for the visited cities
925
926            pheromone = self.pheromone[prev] # shape: (n_ants, p_size)
927            heuristic = self.heuristic[prev] # shape: (n_ants, p_size)
928            dist = ((pheromone ** self.alpha) * (heuristic ** self.beta) * mask) # shape: (n_ants, p_size)
929            dist = Categorical(dist)
930            actions = dist.sample() # shape: (n_ants,)
931            log_probs = dist.log_prob(actions) if require_prob else None # shape: (n_ants,)
932            return actions, log_probs

```

Listing 1: Implementation of the ACO framework for TSP in ReEvo (Ye et al., 2024) and MCTS-AHD (Zheng et al., 2025b).

```

907
908
909    import torch
910    from torch.distributions import Categorical
911    import random
912    import itertools
913    import numpy as np
914
915    class ACO():
916        def __init__(self, # 0: depot
917                     distances, # (n, n)
918                     demand, # (n, )
919                     heuristic, # (n, n)
920                     capacity,
921                     n_ants=30,
922                     decay=0.9,
923                     alpha=1,
924                     beta=1,
925                     device='cpu',

```

```

918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971

```

```

        ):

        self.problem_size = len(distances)
        self.distances = torch.tensor(distances, device=device) if not isinstance(distances, torch.Tensor) else
            distances
        self.demand = torch.tensor(demand, device=device) if not isinstance(demand, torch.Tensor) else demand
        self.capacity = capacity

        self.n_ants = n_ants
        self.decay = decay
        self.alpha = alpha
        self.beta = beta

        self.pheromone = torch.ones_like(self.distances)
        self.heuristic = torch.tensor(heuristic, device=device) if not isinstance(heuristic, torch.Tensor) else
            heuristic

        self.shortest_path = None
        self.lowest_cost = float('inf')

        self.device = device

    @torch.no_grad()
    def run(self, n_iterations):
        for _ in range(n_iterations):
            paths = self.gen_path()
            costs = self.gen_path_costs(paths)

            best_cost, best_idx = costs.min(dim=0)
            if best_cost < self.lowest_cost:
                self.shortest_path = paths[:, best_idx]
                self.lowest_cost = best_cost

            self.update_pheromone(paths, costs)

        return self.lowest_cost

    @torch.no_grad()
    def update_pheromone(self, paths, costs):
        """
        Args:
            paths: torch tensor with shape (problem_size, n_ants)
            costs: torch tensor with shape (n_ants,)
        """
        self.pheromone = self.pheromone * self.decay
        for i in range(self.n_ants):
            path = paths[:, i]
            cost = costs[i]
            self.pheromone[path[:-1], torch.roll(path, shifts=-1)[:-1]] += 1.0/cost
            self.pheromone[self.pheromone < 1e-10] = 1e-10

    @torch.no_grad()
    def gen_path_costs(self, paths):
        u = paths.permute(1, 0) # shape: (n_ants, max_seq_len)
        v = torch.roll(u, shifts=-1, dims=1)
        return torch.sum(self.distances[u[:, :-1], v[:, :-1]], dim=1)

    def gen_path(self):
        actions = torch.zeros((self.n_ants,), dtype=torch.long, device=self.device)
        visit_mask = torch.ones(size=(self.n_ants, self.problem_size), device=self.device)
        visit_mask = self.update_visit_mask(visit_mask, actions)
        used_capacity = torch.zeros(size=(self.n_ants,), device=self.device)

        used_capacity, capacity_mask = self.update_capacity_mask(actions, used_capacity)

        paths_list = [actions] # paths_list[i] is the ith move (tensor) for all ants

        done = self.check_done(visit_mask, actions)
        while not done:
            actions = self.pick_move(actions, visit_mask, capacity_mask)
            paths_list.append(actions)
            visit_mask = self.update_visit_mask(visit_mask, actions)
            used_capacity, capacity_mask = self.update_capacity_mask(actions, used_capacity)
            done = self.check_done(visit_mask, actions)

        return torch.stack(paths_list)

    def pick_move(self, prev, visit_mask, capacity_mask):
        pheromone = self.pheromone[prev] # shape: (n_ants, p_size)
        heuristic = self.heuristic[prev] # shape: (n_ants, p_size)
        dist = ((pheromone ** self.alpha) * (heuristic ** self.beta) * visit_mask * capacity_mask) # shape:
            # (n_ants, p_size)
        dist = Categorical(dist)
        actions = dist.sample() # shape: (n_ants,)

        return actions

    def update_visit_mask(self, visit_mask, actions):
        visit_mask[torch.arange(self.n_ants, device=self.device), actions] = 0
        visit_mask[:, 0] = 1 # depot can be revisited with one exception
        visit_mask[(actions==0) * (visit_mask[:, 1:]!=0).any(dim=1), 0] = 0 # one exception is here
        return visit_mask

```

```

972
973     def update_capacity_mask(self, cur_nodes, used_capacity):
974         """
975             Args:
976                 cur_nodes: shape (n_ants, )
977                 used_capacity: shape (n_ants, )
978                 capacity_mask: shape (n_ants, p_size)
979             Returns:
980                 ant_capacity: updated capacity
981                 capacity_mask: updated mask
982         """
983         capacity_mask = torch.ones(size=(self.n_ants, self.problem_size), device=self.device)
984         # update capacity
985         used_capacity[cur_nodes==0] = 0
986         used_capacity = used_capacity + self.demand[cur_nodes]
987         # update capacity_mask
988         remaining_capacity = self.capacity - used_capacity # (n_ants, )
989         remaining_capacity_repeat = remaining_capacity.unsqueeze(-1).repeat(1, self.problem_size) # (n_ants,
990                                         p_size)
991         demand_repeat = self.demand.unsqueeze(0).repeat(self.n_ants, 1) # (n_ants, p_size)
992         capacity_mask[demand_repeat > remaining_capacity_repeat] = 0
993
994         return used_capacity, capacity_mask
995
996     def check_done(self, visit_mask, actions):
997         return (visit_mask[:, 1:] == 0).all() and (actions == 0).all()

```

Listing 2: Implementation of the ACO framework for CVRP in ReEvo (Ye et al., 2024) and MCTS-AHD (Zheng et al., 2025b).

990

991

Manually Designed Prompts for TSP and CVRP in Existing Works. In prior LLM-EPS works, prompt components for calling LLMs must be designed in accordance with the employed GAF, rather than the COP at hand. In Table S9, we compare these components when ACO is employed for TSP vs. CVRP.

Table S9: Prompt components used in ReEvo (Ye et al., 2024) and MCTS-AHD (Zheng et al., 2025b) under the ACO framework.

992

993

TSP

994

995

Prompt component	Specification
Problem description	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.
Heuristic description	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.
Function signature	<code>def heuristics(distance_matrix: np.ndarray) -> np.ndarray:</code>

996

CVRP

997

998

Prompt component	Specification
Problem description	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.
Heuristic description	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.
Function signature	<code>def heuristics(distance_matrix: np.ndarray, coordinates: np.ndarray, demands: np.ndarray, capacity: int) -> np.ndarray:</code>

999

1000

1001

1002

1003

1004

1005

1006

1007

1008

1009

1010

1011

1012

1013

1014

1015

1016

1017

1018

1019

1020

1021

1022

1023

1024

1025

1026
1027

D ADDITIONAL EXPERIMENTS AND DISCUSSIONS

1028
1029

D.1 COMPLETE IMPLEMENTATION DETAILS

1030
1031
1032

All experiments were conducted under Ubuntu 20.04 on a Linux virtual machine equipped with NVIDIA GeForce RTX 3050 Ti GPU and 12th Gen Intel(R) Core(TM) i7-12700H CPU @2.3GHz. The code for our implementation in Python 3.10 is uploaded as supplementary material.

1033
1034
1035
1036

We adopt the experimental setups from MCTS-AHD (Zheng et al., 2025b), the state-of-the-art LLM-EPS method, to better gauge the efficacy of RedAHD in solving COPs. For the evaluation datasets, we use their publicly available data³ during both training and testing for all considered COPs.

1037
1038
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1049
1050

RedAHD Settings. We set $M = 3$, $M_{init} = 10$, and $l = 3$. Prompts for LR generation and refinement are specified in Figures S5 and S4, respectively. The running time of each heuristic on the evaluation dataset for any COP is limited to 60 seconds. We use EoH (Liu et al., 2024a) as the default LLM-EPS method, in which we use only two variation operators, one for crossover and the other for mutation, instead of five as in the original work (see prompt specifications and our justifications in EoH Settings). We set T , the number of generations in EoH context, to 3. Additionally, during population management at early stages of evolution, we do not discard heuristics with identical objective values if they are from different LRs. This ensures every LR has sufficient heuristics (at least l) for obtaining a valid score. Unless otherwise specified, GPT-4o-mini with temperature fixed at 1 is employed as the designer LLM for generating both LRs and heuristics, with each run of RedAHD repeated three times and we report the average performance of h^* .

Because RedAHD is self-contained, solution checks are necessary to ensure the validity of the generated heuristics and LRs. That is, during fitness evaluation, we check the solution of each instance as follows:

1051
1052
1053
1054
1055
1056
1057
1058
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079

- **TSP.** All nodes must be visited exactly once.
- **CVRP.** (i) Each customer from a sub-route must be visited exactly once; (ii) sum of demands from customers served by a sub-route must not exceed the vehicle capacity; (iii) all customers must be visited exactly once.
- **BPP.** All items must be packed in one of the bins without exceeding the capacity of any bin.
- **OBPP.** The selected bin must have sufficient capacity for packing the current item.
- **KP.** All selected items must be unique and their total weight must not exceed the knapsack capacity.
- **MKP.** All selected items across all knapsacks must be unique and the total weight of the items in any knapsack must not exceed its capacity.

Prompt for Reduction Refinement

Problem A: [Problem A Description]

I want to transform Problem A into another problem, Problem B, that can be solved efficiently while still providing near-optimal solutions to Problem A. I have one option for Problem B as follows:

Problem description: [Problem B Description]

Please help me modify the following code for transforming Problem A to Problem B and vice versa while remaining as efficient as possible.

Code:

[Reduction Functions]

Do not give additional explanations.

Figure S4: Prompts used for reduction refinement in RedAHD as described in Section 3.3.

³<https://github.com/zz1358m/MCTS-AHD-master/tree/main>

1080

1081

1082

1083

1084

1085

1086

1087

1088

1089

1090

1091

1092

1093

1094

1095

1096

1097

1098

1099

1100

1101

1102

1103

1104

1105

1106

1107

1108

1109

1110

1111

1112

1113

1114

1115

1116

1117

1118

1119

1120

1121

1122

1123

1124

1125

1126

1127

1128

1129

1130

1131

1132

1133

Prompt for Candidate LR Initialization

Problem A: [Problem Description]

I want to transform Problem A into another problem, Problem B, that can be solved efficiently while still providing near-optimal solutions to Problem A. Please help me devise M_{init} different Problem B's. Describe each Problem B in a sentence or two (without mentioning Problem A) and enclose it inside a double brace as follows:

```
{ {Problem B1 involves ...} }
{ {Problem B2 involves ...} }
...
```

Do not give additional explanations.

Prompt for Generating Reduction Functions

Problem A: [Problem A Description]

Problem B: [Problem B Description]

Implement 2 Python functions for transforming Problem A into Problem B using the following templates:

[Reduction Template]

Only provide me the code without any further explanations.

Prompt for Code Template Generation

I have the following code for transforming a Problem A into a simplified Problem B and vice versa.

Code:
[Reduction Functions]

Using this information, fill in the blanks of the following Python function template.

Code template:
[Heuristic Template]

First, determine $<INPUT_B>$ from output of ‘convert_input_A_to_B()’. Then, determine $<SOLUTION_B>$ from ‘solution_B’ variable in ‘convert_solution_B_to_A()’. Finally, complete the docstring at $<ARGS>$ and $<RETURNS>$ with as detailed type hints as possible. Do not attempt to solve the problem directly and do not give additional explanations.

```
import numpy as np
from typing import Tuple

def convert_input_A_to_B(coord_matrix, distance_matrix):
    """ Convert input of Problem A into input of Problem B
    Args:
        [ARGS]
    Returns:
        input_B: A tuple storing the corresponding input of
        Problem B.
    """
    # Placeholder (replace with your actual implementation)
    input_B = ...

    return input_B

def convert_solution_B_to_A(solution_B):
    """ Convert solution of Problem B into solution of
    Problem A
    Args:
        solution_B: The output of Problem B.
    Returns:
        [RETURN]
    """
    # Placeholder (replace with your actual implementation)
    [PLACEHOLDER]
```

```
from typing import Tuple

def solve_B(<INPUT_B>):
    """
    Args:
        <ARGS>
    Returns:
        <RETURNS>
    """
    return <SOLUTION_B>
```

Figure S5: Prompts used for candidate LR generation in RedAHD as described in Section 3.1. The chronological order for LLM prompting is (top) ▶ (center left) ▶ (center right). The (bottom left) code snippet is the “Reduction Template”, where [ARGS], [RETURN], [PLACEHOLDER] are COP-specific and detailed in Table S11. The (bottom right) code snippet is the “Heuristic Template”.

1134 **EoH Settings.** Following Zheng et al. (2025b), the number of generations in EoH is set to 20 and
 1135 the population size N is set to 20 for CVRP, BPP, OBPP, MKP and 10 for TSP and KP. EoH utilizes
 1136 five variation operators in total, two for crossover (E1 and E2) and three for mutation (M1, M2,
 1137 M3). RedAHD only uses E2 and M1 from EoH (see Figure S6, bottom) since we actually observed
 1138 reduced optimization performance when either E1, M2, or M3 is included (and significant increase
 1139 in runtime and API cost). In particular, we notice the heuristics generated by E1 are often erroneous
 1140 (due to e.g., code errors or returning invalid solutions). We attribute this behavior to the fact that E1
 1141 prompts the designer LLM to generate a completely new heuristic from the provided ones, which
 1142 might not be well-suited for multi-problem LLM-EPS within RedAHD that already enables ample
 1143 exploration of novel heuristics.



1170 Figure S6: Prompts used for initialization, exploration, and modification in EoH. “Problem Description” and
 1171 “Code Template” are with respect to B from the LLM-generated LR (see Figure S7 for an example).

1172 **MEoH Settings.** MEoH (Yao et al., 2025) extends EoH to additionally consider runtime during
 1173 fitness evaluation via the proposed dominance-dissimilarity mechanism for multi-objective parent
 1174 selection and population management. We similarly use two variation operators as detailed in EoH
 1175 Settings. Importantly, each LR now records two scores, one with respect to the objective value and
 1176 the other with respect to runtime. The latter is defined as the average runtime of its top- l associated
 1177 heuristics with best objective values. For stagnation tracking, if neither score improves after T , then
 1178 the reduction refinement step is invoked for the LR.

1180 **ReEvo Settings.** ReEvo (Ye et al., 2024) incorporates reflections into the evolutionary search by
 1181 prompting the designer LLM to analyze and revise previously generated heuristics. We make the
 1182 following changes to ReEvo. During parent selection, LR ration is similarly applied to maintain the
 1183 number of generated offspring heuristics from the two crossover and mutation operators. Short- and
 1184 long-term reflections are performed for each LR. For short-term reflection, the problem description
 1185 is with respect to B_j . Importantly, in accordance with our proposed multi-problem LLM-EPS in
 1186 RedAHD, the two provided heuristics can be from $B_{j'}, j' \neq j$.

1188 **COP-Specific Prompts.** Tables S10 and S11 respectively list the problem descriptions and reduction
 1189 templates used in prompts. To facilitate the generation of valid LRs that can generalize to OOD
 1190 instances (i.e., instances with smaller or larger sizes than what originally encountered during training),
 1191 when specifying problem descriptions and reduction templates, we ensure all COP parameters
 1192 are abstracted, such as ‘N’ instead of the actual number of nodes in training instances.
 1193

Table S10: Problem descriptions used in prompts.

COP	Problem description
TSP	Given a set of N nodes with their 2D coordinates, the problem involves finding the shortest route that visits each node exactly once and returns to the starting node.
CVRP	Given a set of N customers and a fleet of vehicles with limited capacity, the problem involves finding a corresponding set of optimal routes to deliver goods to all customers.
BPP	Given a set of N items with different sizes and some bins each with fixed capacity, the problem involves placing each item inside one of the bins in a way that minimizes the number of bins used without exceeding the bin capacity.
OBPP	Given an item with certain size and a set of M bins each with finite capacity, the problem involves finding a priority score for each bin. The bin with the highest priority score will be selected for inserting the item.
KP	Given a set of N items with weights and values, the problem involves selecting a subset of items that maximizes the total value without exceeding the knapsack’s weight capacity.
MKP	Given a set of N items with values and M-dimensional weights, the problem involves selecting a subset of items to maximize the total value without exceeding the multi-dimensional maximum weight constraints.

Table S11: COP-specific components for reduction templates.

Component	Specification
TSP	
ARGS	<pre>... coord_matrix (np.ndarray): A Nx2 matrix storing the 2D coordinates of the nodes. distance_matrix (np.ndarray): A NxN matrix where the entry at i-th row and j-th column (or vice versa) stores the Euclidean distance between nodes i and j. ...</pre>
RETURN	<pre>... route: A Numpy 1D array of length N storing the unique node IDs to visit in order. ...</pre>
PLACEHOLDER	<pre>route = ... return route</pre>
CVRP	
ARGS	<pre>... coord_matrix (np.ndarray): A (N+1)-by-2 matrix storing the Euclidean coordinates of the depot (first row) and the customers. distance_matrix (np.ndarray): A (N+1)-by-(N+1) distance matrix. demands (np.ndarray): An array of length N+1 storing the customer demands, where the first entry is 0 (placeholder for the depot). capacity (int): The capacity of each vehicle for satisfying the customer demands. ...</pre>
RETURN	<pre>... routes (List[List[int]]): A list of routes; each route is represented as a list of unique customer indices (1 to N) to visit in order, subject to the capacity constraint. ...</pre>
PLACEHOLDER	<pre>routes = [] ... return routes</pre>
BPP	
ARGS	<pre>... items (np.ndarray): Array of length N storing the item sizes to be considered in exact order. bins (np.ndarray): Array of capacities for each bin. ...</pre>
RETURN	<pre>... packed_bins (np.ndarray): Array of remaining capacities for each bin after packing all items. ...</pre>
PLACEHOLDER	<pre>packed_bins = return packed_bins</pre>
OBPP	
ARGS	<pre>... item_size (float): Size of the item to be added to one of the bins. bin_caps (np.ndarray): Array of length M storing capacities of each bin. ...</pre>
RETURN	<pre>... scores (np.ndarray): Array of priority scores for the bins. ...</pre>

1242		
1243	PLACEHOLDER	scores = return scores
1244		KP
1245		
1246	ARGS	... weights (np.ndarray): A 1D float array of length {problem_size} storing the item weights. values (np.ndarray): A 1D float array of length {problem_size} storing the associated item values. capacity (float): The weight capacity of the knapsack. ...
1247		
1248	RETURN	... items: A list storing the indices of selected items subject to the capacity constraint. ...
1249		
1250	PLACEHOLDER	items = [] ... return items
1251		MKP
1252		
1253	ARGS	... values (np.ndarray): A 1D float array of length N storing the item values. weights (np.ndarray): A (M x N) float matrix storing the multi-dimensional weights, where each row is associated with a constraint. constraints (np.ndarray): A 1D float array of length M storing weight constraints. ...
1254		
1255	RETURN	... items: A list storing the indices of selected items subject to the weight constraints. ...
1256		
1257	PLACEHOLDER	items = [] ... return items
1258		
1259		
1260		
1261		
1262		
1263		
1264		
1265		

D.2 ADDITIONAL RESULTS

1269 **Ablation of Multi-Problem LLM-EPS.** We validate the necessity of multi-problem LLM-EPS by
 1270 limiting M to 1, which means the search now becomes typical LLM-EPS. As shown in Table S12,
 1271 compared to $M = 3$ as we did throughout our previous experiments, there is a significant decrease
 1272 in optimization performance across all COPs. This result supports our claims of multi-problem
 1273 LLM-EPS advantages as discussed in Section 3.2.

Table S12: Ablation of the proposed multi-problem LLM-EPS. “RedAHD ($M = 3$)” is RedAHD reported in earlier results. Results from EoH in Table 5 are used as references.

Problem setting	TSP (Obj. ↓)		CVRP (Obj. ↓)		MKP (Obj. ↑)		BPP (Obj. ↓)	
	n=50 n=100		n=50 n=100		n=100 n=200		n=500 n=1,000	
	$C=50$	$C=50$	$C=50$	$C=50$	$m=5$	$m=5$	$W=150$	$W=150$
EoH*	5.828	8.263	9.359	15.681	23.139	41.994	204.646	408.599
RedAHD ($M = 1$)	5.931	8.479	10.327	16.252	22.925	41.569	205.983	411.428
RedAHD ($M = 3$, without multi-problem LLM-EPS)	5.943	8.602	10.537	16.985	22.916	41.497	206.013	412.220
RedAHD ($M = 3$)	5.819	8.039	9.826	15.726	23.164	42.682	203.344	405.359

Sensitivity to Initial LRs. We investigate the sensitivity of RedAHD performance to the quality of the initial pool of LRs. Table S13 compares the average test performance of all heuristics in the initial generation (second column) to the test performance of the best heuristic in the final generation (last column). Even though the quality of the initial LRs differs across runs, the final performance of RedAHD remains consistent.

Table S13: RedAHD performance on TSP ($n = 50$) across five independent runs (lower values are better).

Run	Quality of initial LRs	Final performance
1	6.831	5.784
2	6.378	5.761
3	6.494	5.775
4	6.796	5.770
5	6.592	5.766

RedAHD for Solving Flow Shop Scheduling Problems (FSSPs). FSSP (Emmons & Vairaktaris, 2012) is a complex COP considered in EoH (Liu et al., 2024a) that concerns scheduling n jobs on m machines, where each job involves m operations that must be performed in a predetermined order on the respective machine. The objective is to minimize the total schedule length.

known as the makespan. We apply RedAHD to FSSP by adopting the same experimental setups from Liu et al. (2024a) (with consistent evaluation datasets and EoH settings) while keeping the same RedAHD settings detailed in Appendix D.1. Table S14 shows that RedAHD attains second-best optimization performance in nearly all settings, surpassing classical FSSP heuristics and even dedicated deep learning solvers. Note that in the original paper, Liu et al. (2024a) employed GLS (Voudouris et al., 2010) as the GAF for EoH, which yields the overall best performance in exchange for additional manual efforts. For completeness, we also run EoH under the IC framework, which seeks to design a heuristic for selecting the next operation given the current status of each machine and job and the set of feasible operations. When employing this simple GAF, we see a substantial drop in EoH performance. Our results thus demonstrate that even without relying on GAFs, RedAHD can effectively handle complex COPs beyond vehicle routing (TSP, CVRP) and packing problems (OBPP, BPP, KP, MKP).

Table S14: Comparative results for FSSP captured by the average (%) gap with respect to the best known makespan (lower is better). We use the results reported in Liu et al. (2024a) for the baselines other than EoH-IC. The best and second-best methods are respectively bolded and shaded.

		n20m10	n20m20	n50m10	n50m20	n100m10	n100m20
Handcrafted	GUPTA 1971	23.42	21.79	20.11	22.78	15.03	21.00
	CDS 1970	12.87	10.35	12.72	15.03	9.36	13.55
	NEH 1983	4.05	3.06	3.47	5.48	2.07	3.58
	NEHFF 2014	4.15	2.72	3.62	5.10	1.88	3.73
Deep learning	PFSPNet 2021	14.78	14.69	11.95	16.95	8.21	16.47
	PFSPNet_NEH 2021	4.04	2.96	3.48	5.05	1.72	3.56
LLM-EPS	EoH-GLS	0.30	0.10	0.19	0.60	0.14	0.41
	EoH-IC	3.76	50.6	14.2	10.4	12.5	21.2
	RedAHD	3.27	2.40	3.32	4.10	1.78	3.00

D.3 BLACK-BOX SETTINGS

Black-box settings were considered in ReEvo (Ye et al., 2024) and MCTS-AHD (Zheng et al., 2025b), in which all information regarding the COP (e.g., the problem description, the heuristic description, and the function signature in accordance with the designed GAF as shown in Table S9) is not provided. The goal is to fairly evaluate the efficacy of LLM-EPS methods in designing effective heuristics for a wide range of COPs, rather than merely retrieving code tailored to prominent COPs from LLMs’ parameterized knowledge. Since RedAHD solves the COP at hand directly without the need of GAFs, the proposed black-box settings in these works are not applicable to RedAHD.

To address the stated concerns regarding mere code retrieval by LLMs, in every considered COP, we do not mention its commonly known name in the problem description (see Table S10. That is, we do not refer to the COPs plainly as e.g., “the traveling salesman problem”, but rather vaguely “the problem”. Moreover, when prompting the designer LLM for generating heuristics for B , we do not leak any information on A (which also helps mitigate hallucinations (Huang et al., 2025)). By this means, **RedAHD already operates under black-box settings by default**.

D.4 EXAMPLES OF DESIGNED LRS AND HEURISTICS FROM REDAHD

MKP. Figures S7 and S8 respectively show an example of the designed LR and the corresponding heuristic for MKP.

1350

1351

1352

1353

1354

1355

1356

1357

1358

1359

1360

Problem Description

1361

1362

1363

1364

1365

1366

1367

1368

1369

1370

1371

1372

1373

1374

1375

1376

1377

1378

1379

1380

1381

1382

1383

1384

1385

1386

1387

1388

1389

1390

1391

1392

1393

1394

1395

1396

1397

1398

1399

1400

1401

1402

1403

Problem B1 involves selecting a subset of N items such that the total value is maximized while ensuring that the total weights in each of the M dimensions do not exceed specific limits, using a greedy heuristic approach based on value-to-weight ratios.

```

import numpy as np
from typing import Tuple, List

def convert_input_A_to_B(values: np.ndarray, weights: np.ndarray, constraints: np.ndarray) ->
    Tuple[np.ndarray, np.ndarray, np.ndarray]:
    """ Convert input of Problem A into input of Problem B
    Args:
        values (np.ndarray): A 1D float array of length N storing the item values.
        weights (np.ndarray): A (M x N) float matrix storing the multi-dimensional weights, where each row is
            associated with a constraint.
        constraints (np.ndarray): A 1D float array of length M storing weight constraints.

    Returns:
        input_B: A tuple storing the corresponding input of Problem B.
    """
    # Calculate value-to-weight ratios for each item
    ratios = values / np.sqrt(np.sum(np.square(weights), axis=0)) # Changed to root of sum of squares for
        better ratio
    input_B = (values, weights, constraints, ratios)

    return input_B

def convert_solution_B_to_A(solution_B: List[int]) -> List[int]:
    """ Convert solution of Problem B into solution of Problem A
    Args:
        solution_B: The output of Problem B, which contains indices of selected items.

    Returns:
        items: A list storing the indices of selected items subject to the weight constraints.
    """
    items = list(solution_B)
    return items

def convert_input_A_to_B(values, weights, constraints):
    """ Convert input of Problem A into input of Problem B
    Args:
        values (np.ndarray): A 1D float array of length N storing the item values.
        weights (np.ndarray): A (M x N) float matrix storing the multi-dimensional weights, where each row is
            associated with a constraint.
        constraints (np.ndarray): A 1D float array of length M storing weight constraints.

    Returns:
        input_B: A tuple storing the corresponding input of Problem B.
    """
    # Calculate value-to-weight ratios for each item
    ratios = values / np.sum(weights, axis=0)
    input_B = (values, weights, constraints, ratios)

    return input_B

def convert_solution_B_to_A(solution_B):
    """ Convert solution of Problem B into solution of Problem A
    Args:
        solution_B: The output of Problem B.

    Returns:
        items: A list storing the indices of selected items subject to the weight constraints.
    """
    selected_items = solution_B # Assuming solution_B contains the indices of selected items
    items = list(selected_items)

    return items

```

Figure S7: Designed LR for MKP using RedAHD. (Top) Problem description of B , (center) refined and (bottom) original implementation of (f, g) for transforming MKP to B .

```

1404
1405
1406     from typing import Tuple, List
1407
1408     def solve_B(input_B: Tuple[np.ndarray, np.ndarray, np.ndarray, np.ndarray]) -> List[int]:
1409         """
1410             Args:
1411                 input_B (Tuple[np.ndarray, np.ndarray, np.ndarray, np.ndarray]): A tuple containing:
1412                     - values (np.ndarray): A 1D float array of length N storing the item values.
1413                     - weights (np.ndarray): A (M x N) float matrix storing the multi-dimensional weights, where each row is
1414                         associated with a constraint.
1415                     - constraints (np.ndarray): A 1D float array of length M storing weight constraints.
1416                     - ratios (np.ndarray): A 1D float array of length N storing the value-to-weight ratios for each item.
1417
1418             Returns:
1419                 List[int]: A list storing the indices of selected items subject to the weight constraints.
1420             """
1421
1422     return solution_B

```

Figure S7 (cont.): Code template for solving B .

Problem Description

Problem B1 involves selecting a subset of N items such that the total value is maximized while ensuring that the total weights in each of the M dimensions do not exceed specific limits, using a greedy heuristic approach based on value-to-weight ratios.

Heuristic Description

A new algorithm that selects items iteratively, calculating the best score considering both value and the remaining capacity left in multi-dimensional space, while simultaneously updating the constraints as items are selected.

```

1433
1434     from typing import Tuple, List
1435     import numpy as np
1436
1437     def solve_B(input_B: Tuple[np.ndarray, np.ndarray, np.ndarray, np.ndarray]) -> List[int]:
1438         values, weights, constraints, ratios = input_B
1439         M, N = weights.shape
1440         selected_items = []
1441         total_weights = np.zeros(M)
1442
1443         # Calculate the remaining capacity to define the score more effectively
1444         remaining_capacity = constraints.copy()
1445
1446         while True:
1447             best_score = -np.inf
1448             best_item = -1
1449             for idx in range(N):
1450                 if idx in selected_items:
1451                     continue
1452                 item_weight = weights[:, idx]
1453
1454                 if all(total_weights + item_weight <= constraints):
1455                     # Calculate new score based on value and remaining capacity
1456                     score = values[idx] / (np.sum(item_weight / remaining_capacity) + 1e-9) # Avoid division by zero
1457                     if score > best_score:
1458                         best_score = score
1459                         best_item = idx
1460
1461             if best_item == -1: # No feasible item can be added
1462                 break
1463
1464             selected_items.append(best_item)
1465             total_weights += weights[:, best_item]
1466             remaining_capacity -= weights[:, best_item]
1467
1468     return selected_items

```

Figure S8: Designed heuristic using the LR for MKP in Figure S7.

1458 **TSP.** Figure S9 shows the learned heuristic whose performance was reported in Table 4.
 1459

1460 **Problem Description**

1461 Problem B7 involves applying a nearest neighbor heuristic starting from each node and selecting the
 1462 route with the least total distance as a near-optimal solution.
 1463

1464 **Heuristic Description**

1465 Repeatedly select a random starting point and build a route while ensuring that each node is added
 1466 without forming a local loop, then refine the tour using a hill-climbing strategy to find better routes.
 1467

```

 1470 from typing import Tuple
 1471 import numpy as np
 1472
 1473 def solve_B(input_B: Tuple[np.ndarray]) -> np.ndarray:
 1474     """
 1475     Args:
 1476         input_B (Tuple[np.ndarray]): A tuple containing a single np.ndarray, which is a NxN matrix where the entry
 1477             at i-th row and j-th column (or vice versa) stores the Euclidean distance between
 1478             nodes i and j.
 1479
 1480     Returns:
 1481         np.ndarray: A 1D array of length N containing the unique node IDs to visit in order, representing the
 1482             solution
 1483             of Problem B converted to the corresponding solution of Problem A.
 1484     """
 1485     distances = input_B[0]
 1486     num_nodes = distances.shape[0]
 1487     best_route = []
 1488     best_length = float('inf')
 1489
 1490     # (Repeatedly select a random starting point and build a route while ensuring that each node is added
 1491     # without forming a local loop,
 1492     # then refine the tour using a hill-climbing strategy to find better routes.)
 1493     for _ in range(10): # Number of iterations to random start points
 1494         start_node = np.random.randint(num_nodes)
 1495         visited = [start_node]
 1496         unvisited = set(range(num_nodes))
 1497         unvisited.remove(start_node)
 1498
 1499         while unvisited:
 1500             current_node = visited[-1]
 1501             next_node = min(unvisited, key=lambda x: distances[current_node][x])
 1502             visited.append(next_node)
 1503             unvisited.remove(next_node)
 1504
 1505             # Hill-climbing optimization
 1506             improvement = True
 1507             while improvement:
 1508                 improvement = False
 1509                 for i in range(len(visited)):
 1510                     for j in range(i + 2, len(visited)):
 1511                         if j == len(visited) - 1 and i == 0: # Skip the final edge to start
 1512                             continue
 1513                         current_cost = (distances[visited[i]][visited[(i + 1) % len(visited)]] +
 1514                             distances[visited[j]][visited[(j + 1) % len(visited)]]))
 1515                         new_cost = (distances[visited[i]][visited[j]] +
 1516                             distances[visited[(i + 1) % len(visited)]][visited[(j + 1) % len(visited)]]))
 1517
 1518                         if new_cost < current_cost:
 1519                             visited[i + 1:j + 1] = reversed(visited[i + 1:j + 1])
 1520                             improvement = True
 1521                             break
 1522                         if improvement:
 1523                             break
 1524
 1525             current_length = sum(distances[visited[k]][visited[(k + 1) % len(visited)]] for k in
 1526             range(len(visited)))
 1527
 1528             if current_length < best_length:
 1529                 best_length = current_length
 1530                 best_route = visited
 1531
 1532     return np.array(best_route)
 1533
 1534
 1535
 1536
 1537
 1538
 1539
 1540
 1541
 1542
 1543
 1544
 1545
 1546
 1547
 1548
 1549
 1550
 1551
 1552
 1553
 1554
 1555
 1556
 1557
 1558
 1559
 1560
 1561
 1562
 1563
 1564
 1565
 1566
 1567
 1568
 1569
 1570
 1571
 1572
 1573
 1574
 1575
 1576
 1577
 1578
 1579
 1580
 1581
 1582
 1583
 1584
 1585
 1586
 1587
 1588
 1589
 1590
 1591
 1592
 1593
 1594
 1595
 1596
 1597
 1598
 1599
 1600
 1601
 1602
 1603
 1604
 1605
 1606
 1607
 1608
 1609
 1610
 1611
 1612
 1613
 1614
 1615
 1616
 1617
 1618
 1619
 1620
 1621
 1622
 1623
 1624
 1625
 1626
 1627
 1628
 1629
 1630
 1631
 1632
 1633
 1634
 1635
 1636
 1637
 1638
 1639
 1640
 1641
 1642
 1643
 1644
 1645
 1646
 1647
 1648
 1649
 1650
 1651
 1652
 1653
 1654
 1655
 1656
 1657
 1658
 1659
 1660
 1661
 1662
 1663
 1664
 1665
 1666
 1667
 1668
 1669
 1670
 1671
 1672
 1673
 1674
 1675
 1676
 1677
 1678
 1679
 1680
 1681
 1682
 1683
 1684
 1685
 1686
 1687
 1688
 1689
 1690
 1691
 1692
 1693
 1694
 1695
 1696
 1697
 1698
 1699
 1700
 1701
 1702
 1703
 1704
 1705
 1706
 1707
 1708
 1709
 1710
 1711
 1712
 1713
 1714
 1715
 1716
 1717
 1718
 1719
 1720
 1721
 1722
 1723
 1724
 1725
 1726
 1727
 1728
 1729
 1730
 1731
 1732
 1733
 1734
 1735
 1736
 1737
 1738
 1739
 1740
 1741
 1742
 1743
 1744
 1745
 1746
 1747
 1748
 1749
 1750
 1751
 1752
 1753
 1754
 1755
 1756
 1757
 1758
 1759
 1760
 1761
 1762
 1763
 1764
 1765
 1766
 1767
 1768
 1769
 1770
 1771
 1772
 1773
 1774
 1775
 1776
 1777
 1778
 1779
 1780
 1781
 1782
 1783
 1784
 1785
 1786
 1787
 1788
 1789
 1790
 1791
 1792
 1793
 1794
 1795
 1796
 1797
 1798
 1799
 1800
 1801
 1802
 1803
 1804
 1805
 1806
 1807
 1808
 1809
 1810
 1811
 1812
 1813
 1814
 1815
 1816
 1817
 1818
 1819
 1820
 1821
 1822
 1823
 1824
 1825
 1826
 1827
 1828
 1829
 1830
 1831
 1832
 1833
 1834
 1835
 1836
 1837
 1838
 1839
 1840
 1841
 1842
 1843
 1844
 1845
 1846
 1847
 1848
 1849
 1850
 1851
 1852
 1853
 1854
 1855
 1856
 1857
 1858
 1859
 1860
 1861
 1862
 1863
 1864
 1865
 1866
 1867
 1868
 1869
 1870
 1871
 1872
 1873
 1874
 1875
 1876
 1877
 1878
 1879
 1880
 1881
 1882
 1883
 1884
 1885
 1886
 1887
 1888
 1889
 1890
 1891
 1892
 1893
 1894
 1895
 1896
 1897
 1898
 1899
 1900
 1901
 1902
 1903
 1904
 1905
 1906
 1907
 1908
 1909
 1910
 1911
 1912
 1913
 1914
 1915
 1916
 1917
 1918
 1919
 1920
 1921
 1922
 1923
 1924
 1925
 1926
 1927
 1928
 1929
 1930
 1931
 1932
 1933
 1934
 1935
 1936
 1937
 1938
 1939
 1940
 1941
 1942
 1943
 1944
 1945
 1946
 1947
 1948
 1949
 1950
 1951
 1952
 1953
 1954
 1955
 1956
 1957
 1958
 1959
 1960
 1961
 1962
 1963
 1964
 1965
 1966
 1967
 1968
 1969
 1970
 1971
 1972
 1973
 1974
 1975
 1976
 1977
 1978
 1979
 1980
 1981
 1982
 1983
 1984
 1985
 1986
 1987
 1988
 1989
 1990
 1991
 1992
 1993
 1994
 1995
 1996
 1997
 1998
 1999
 2000
 2001
 2002
 2003
 2004
 2005
 2006
 2007
 2008
 2009
 2010
 2011
 2012
 2013
 2014
 2015
 2016
 2017
 2018
 2019
 2020
 2021
 2022
 2023
 2024
 2025
 2026
 2027
 2028
 2029
 2030
 2031
 2032
 2033
 2034
 2035
 2036
 2037
 2038
 2039
 2040
 2041
 2042
 2043
 2044
 2045
 2046
 2047
 2048
 2049
 2050
 2051
 2052
 2053
 2054
 2055
 2056
 2057
 2058
 2059
 2060
 2061
 2062
 2063
 2064
 2065
 2066
 2067
 2068
 2069
 2070
 2071
 2072
 2073
 2074
 2075
 2076
 2077
 2078
 2079
 2080
 2081
 2082
 2083
 2084
 2085
 2086
 2087
 2088
 2089
 2090
 2091
 2092
 2093
 2094
 2095
 2096
 2097
 2098
 2099
 2100
 2101
 2102
 2103
 2104
 2105
 2106
 2107
 2108
 2109
 2110
 2111
 2112
 2113
 2114
 2115
 2116
 2117
 2118
 2119
 2120
 2121
 2122
 2123
 2124
 2125
 2126
 2127
 2128
 2129
 2130
 2131
 2132
 2133
 2134
 2135
 2136
 2137
 2138
 2139
 2140
 2141
 2142
 2143
 2144
 2145
 2146
 2147
 2148
 2149
 2150
 2151
 2152
 2153
 2154
 2155
 2156
 2157
 2158
 2159
 2160
 2161
 2162
 2163
 2164
 2165
 2166
 2167
 2168
 2169
 2170
 2171
 2172
 2173
 2174
 2175
 2176
 2177
 2178
 2179
 2180
 2181
 2182
 2183
 2184
 2185
 2186
 2187
 2188
 2189
 2190
 2191
 2192
 2193
 2194
 2195
 2196
 2197
 2198
 2199
 2200
 2201
 2202
 2203
 2204
 2205
 2206
 2207
 2208
 2209
 2210
 2211
 2212
 2213
 2214
 2215
 2216
 2217
 2218
 2219
 2220
 2221
 2222
 2223
 2224
 2225
 2226
 2227
 2228
 2229
 2230
 2231
 2232
 2233
 2234
 2235
 2236
 2237
 2238
 2239
 2240
 2241
 2242
 2243
 2244
 2245
 2246
 2247
 2248
 2249
 2250
 2251
 2252
 2253
 2254
 2255
 2256
 2257
 2258
 2259
 2260
 2261
 2262
 2263
 2264
 2265
 2266
 2267
 2268
 2269
 2270
 2271
 2272
 2273
 2274
 2275
 2276
 2277
 2278
 2279
 2280
 2281
 2282
 2283
 2284
 2285
 2286
 2287
 2288
 2289
 2290
 2291
 2292
 2293
 2294
 2295
 2296
 2297
 2298
 2299
 2300
 2301
 2302
 2303
 2304
 2305
 2306
 2307
 2308
 2309
 2310
 2311
 2312
 2313
 2314
 2315
 2316
 2317
 2318
 2319
 2320
 2321
 2322
 2323
 2324
 2325
 2326
 2327
 2328
 2329
 2330
 2331
 2332
 2333
 2334
 2335
 2336
 2337
 2338
 2339
 2340
 2341
 2342
 2343
 2344
 2345
 2346
 2347
 2348
 2349
 2350
 2351
 2352
 2353
 2354
 2355
 2356
 2357
 2358
 2359
 2360
 2361
 2362
 2363
 2364
 2365
 2366
 2367
 2368
 2369
 2370
 2371
 2372
 2373
 2374
 2375
 2376
 2377
 2378
 2379
 2380
 2381
 2382
 2383
 2384
 2385
 2386
 2387
 2388
 2389
 2390
 2391
 2392
 2393
 2394
 2395
 2396
 2397
 2398
 2399
 2400
 2401
 2402
 2403
 2404
 2405
 2406
 2407
 2408
 2409
 2410
 2411
 2412
 2413
 2414
 2415
 2416
 2417
 2418
 2419
 2420
 2421
 2422
 2423
 2424
 2425
 2426
 2427
 2428
 2429
 2430
 2431
 2432
 2433
 2434
 2435
 2436
 2437
 2438
 2439
 2440
 2441
 2442
 2443
 2444
 2445
 2446
 2447
 2448
 2449
 2450
 2451
 2452
 2453
 2454
 2455
 2456
 2457
 2458
 2459
 2460
 2461
 2462
 2463
 2464
 2465
 2466
 2467
 2468
 2469
 2470
 2471
 2472
 2473
 2474
 2475
 2476
 2477
 2478
 2479
 2480
 2481
 2482
 2483
 2484
 2485
 2486
 2487
 2488
 2489
 2490
 2491
 2492
 2493
 2494
 2495
 2496
 2497
 2498
 2499
 2500
 2501
 2502
 2503
 2504
 2505
 2506
 2507
 2508
 2509
 2510
 2511
 2512
 2513
 2514
 2515
 2516
 2517
 2518
 2519
 2520
 2521
 2522
 2523
 2524
 2525
 2526
 2527
 2528
 2529
 2530
 2531
 2532
 2533
 2534
 2535
 2536
 2537
 2538
 2539
 2540
 2541
 2542
 2543
 2544
 2545
 2546
 2547
 2548
 2549
 2550
 2551
 2552
 2553
 2554
 2555
 2556
 2557
 2558
 2559
 2560
 2561
 2562
 2563
 2564
 2565
 2566
 2567
 2568
 2569
 2570
 2571
 2572
 2573
 2574
 2575
 2576
 2577
 2578
 2579
 2580
 2581
 2582
 2583
 2584
 2585
 2586
 2587
 2588
 2589
 2590
 2591
 2592
 2593
 2594
 2595
 2596
 2597
 2598
 2599
 2600
 2601
 2602
 2603
 2604
 2605
 2606
 2607
 2608
 2609
 2610
 2611
 2612
 2613
 2614
 2615
 2616
 2617
 2618
 2619
 2620
 2621
 2622
 2623
 2624
 2625
 2626
 2627
 2628
 2629
 2630
 2631
 2632
 2633
 2634
 2635
 2636
 2637
 2638
 2639
 2640
 2641
 2642
 2643
 2644
 2645
 2646
 2647
 2648
 2649
 2650
 2651
 2652
 2653
 2654
 2655
 2656
 2657
 2658
 2659
 2660
 2661
 2662
 2663
 2664
 2665
 2666
 2667
 2668
 2669
 2670
 2671
 2672
 2673
 2674
 2675
 2676
 2677
 2678
 2679
 2680
 2681
 2682
 2683
 2684
 2685
 2686
 2687
 2688
 2689
 2690
 2691
 2692
 2693
 2694
 2695
 2696
 2697
 2698
 2699
 2700
 2701
 2702
 2703
 2704
 2705
 2706
 2707
 2708
 2709
 2710
 2711
 2712
 2713
 2714
 2715
 2716
 2717
 2718
 2719
 2720
 2721
 2722
 2723
 2724
 2725
 2726
 2727
 2728
 2729
 2730
 2731
 2732
 2733
 2734
 2735
 2736
 2737
 2738
 2739
 2740
 2741
 2742
 2743
 2744
 2745
 2746
 2747
 2748
 2749
 2750
 2751
 2752
 2753
 2754
 2755
 2756
 2757
 2758
 2759
 2760
 2761
 2762
 2763
 2764
 2765
 2766
 2767
 2768
 2769
 2770
 2771
 2772
 2773
 2774
 2775
 2776
 2777
 2778
 2779
 2780
 2781
 2782
 2783
 2784
 2785
 2786
 2787
 2788
 2789
 2790
 2791
 2792
 2793
 2794
 2795
 2796
 2797
 2798
 2799
 2800
 2801
 2802
 2803
 2804
 2805
 2806
 2807
 2808
 2809
 2810
 2811
 2812
 2813
 2814
 2815
 2816
 2817
 2818
 2819
 2820
 2821
 2822
 2823
 2824
 2825
 2826
 2827
 2828
 2829
 2830
 2831
 2832
 2833
 2834
 2835
 2836
 2837
 2838
 2839
 2840
 2841
 2842
 2843
 2844
 2845
 2846
 2847
 2848
 2849
 2850
 2851
 2852
 2853
 2854
 2855
 2856
 2857
 2858
 2859
 2860
 2861
 2862
 2863
 2864
 2865
 2866
 2867
 2868
 2869
 2870
 2871
 2872
 2873
 2874
 2875
 2876
 2877
 2878
 2879
 2880
 2881
 2882
 2883
 2884
 2885
 2886
 2887
 2888
 2889
 2890
 2891
 2892
 2893
 2894
 2895
 2896
 2897
 2898
 2899
 2900
 2901
 2902
 2903
 2904
 2905
 2906
 2907
 2908
 2909
 2910
 2911
 2912
 2913
 2914
 2915
 2916
 2917
 2918
 2919
 2920
 2921
 2922
 2923
 2924
 2925
 2926
 2927
 2928
 2929
 2930
 2931
 2932
 2933
 2934
 2935
 2936
 2937
 2938
 2939
 2940
 2941
 2942
 2943
 2944
 2945
 2946
 2947
 2948
 2949
 2950
 2951
 2952
 2953
 2954
 2955
 2956
 2957
 2958
 2959
 2960
 2961
 2962
 2963
 2964
 2965
 2966
 2967
 2968
 2969
 2970
 2971
 2972
 2973
 2974
 2975
 2976
 2977
 2978
 2979
 2980
 2981
 2982
 2983
 2984
 2985
 2986
 2987
 2988
 2989
 2990
 2991
 2992
 2993
 2994
 2995
 2996
 2997
 2998
 2999
 3000
 3001
 3002
 3003
 3004
 3005
 3006
 3007
 3008
 3009
 3010
 3011
 3012
 3013
 3014
 3015
 3016
 3017
 3018
 3019
 3020
 3021
 3022
 3023
 3024
 3025
 3026
 3027
 3028
 3029
 3030
 3031
 3032
 3033
 3034
 3035
 3036
 3037
 3038
 3039
 3040
 3041
 3042
 3043
 3044
 3045
 3046
 3047
 3048
 3049
 3050
 3051
 3052
 3053
 3054
 3055
 3056
 3057
 3058
 3059
 3060
 3061
 3062
 3063
 3064
 3065
 3066
 3067
 3068
 3069
 3070
 3071
 3072
 3073
 3074
 3075
 3076
 3077
 3078
 3079
 3080
 3081
 3082
 3083
 3084
 3085
 3086
 3087
 3088
 3089
 3090
 3091
 3092
 3093
 3094
 3095
 3096
 3097
 3098
 3099
 3100
 3101
 3102
 3103
 3104
 3105
 3106
 3107
 3108
 3109
 3110
 3111
 3112
 3113
 3114
 3115
 3116
 3117
 3118
 3119
 3120
 3121
 3122
 3123
 3124
 3125
 3126
 3127
 3128
 3129
 3130
 3131
 3132
 3133
 3134
 3135
 3136
 3137
 3138
 3139
 3140
 3141
 3142
 3143
 3144
 3145
 3146
 3147
 3148
 3149
 3150
 3151
 3152
 3153
 3154
 3155
 3156
 3157
 3158
 3159
 3160
 3161
 3162
 3163
 3164
 3165
 3166
 3167
 3168
 3169
 3170
 3171
 3172
 3173
 3174
 3175
 3176
 3177
 3178
 3179
 3180
 3181
 3182
 3183
 3184
 3185
 3186
 3187
 3188
 3189
 3190
 3191
 3192
 3193
 3194
 3195
 3196
 3197
 3198
 3199
 3200
 3201
 3202
 3203
 3204
 3205
 3206
 3207
 3208
 3209
 3210
 3211
 
```

1512 D.5 RESOURCE CONSUMPTION
1513

1514 Using our employed settings (detailed in Appendix D.1), RedAHD costs at most \$0.3 (GPT-4o-mini)
 1515 or \$2 (o3-mini) and 1.5 hour to complete training. The authors of ReEvo argued that the efficiency
 1516 benchmarking for LLM-EPS methods should prioritize the number of fitness evaluations over the
 1517 number of LLM calls (Section 7 in Ye et al. (2024)). Additionally, the work of MCTS-AHD, which
 1518 is the latest LLM-EPS method at the time of submission, also adopted this benchmarking scheme
 1519 (Appendix D in Zheng et al. (2025b)). Therefore, we estimate the number of fitness evaluations
 1520 as follows. Since we mainly consider EoH in this work (with two variation operators), RedAHD
 1521 requires at least $(M_{init} \times \lceil N/M \rceil) + (T_{gen} \times N \times 2) = (10 \times \lceil 20/3 \rceil) + (20 \times 20 \times 2) = 870$
 1522 fitness evaluations, where T_{gen} is the number of generations. Each LR refinement additionally
 1523 requires $N_j < N$ evaluations. Overall, RedAHD needs no more than 1,000 evaluations, which is
 1524 similar to or lower than the budget used in prior LLM-EPS works (Liu et al., 2024a; Zheng et al.,
 1525 2025b). In general, the actual costs from running RedAHD naturally follow the costs associated
 1526 with existing LLM-EPS methods. There are no extra incurred costs during the evolutionary search
 1527 given our proposed LR ration technique (Section 3.2). The additional number of LLM queries is
 1528 negligible: $1 + 2 \times M_{init}$ for reduction initialization and 1 for each refinement of an LR.
 1529

1530 D.6 LIMITATIONS AND FUTURE WORKS
1531

1532 First, while RedAHD significantly reduces human involvement in LLM-based AHD for solving
 1533 COPs, it is yet to be fully end-to-end. That is, RedAHD minimally requires the manual design
 1534 of (i) prompts for candidate LR generation (Figure S5), which include COP-specific components,
 1535 and (ii) solution checks during fitness evaluation (bullet points in RedAHD Settings). We believe
 1536 works from the burgeoning field of LLM planning (Tantakoun et al., 2025; Wei et al., 2025) could
 1537 be employed to achieve full automation.

1538 Second, effective reductions from RedAHD rely on the encoded knowledge of the designer LLM. In
 1539 the absence of relevant domain knowledge, it is possible that the designed LRs are trivial. That is, f
 1540 would simply return the input for A and g would return the raw output of the designed heuristics. In
 1541 other words, the heuristics would be designed for solving A directly without any reduction involved,
 1542 which likely results in subpar optimization performance. Thus, when encountering such behavior
 1543 in practice, we recommend using more capable LLM models during the reduction initialization step
 1544 (which should be inexpensive as stated in Appendix D.5), before switching back to more budget-
 friendly models during the remaining steps of RedAHD.

1545 Lastly, as observed in our results with OBPP in Table 3, RedAHD might not perform as well
 1546 on COPs with restricted heuristic space. To investigate this observation, we further experiment
 1547 RedAHD on the vehicle routing problem with time windows (VRPTW) (Kallehauge et al., 2005),
 1548 which is a more restrictive variant of CVRP where each customer i is only available during a specific
 1549 time window $[t_i^{start}, t_i^{end}]$. VRPTW is a challenging COP with no feasibility guarantees even with
 1550 the IC framework, and hence employing LLM-EPS methods requires even more manual efforts.⁴
 1551 Using the library developed by Liu et al. (2024c) to generate 64 50-node training instances and 64
 1552 50-node test instances, we run RedAHD (following the aforementioned setups with GPT-4o-mini)
 1553 and notice the solutions returned from the designed heuristics are not consistently valid. As shown
 1554 in Figure S10, even after 1000 fitness evaluations, we observe violations of time window constraints
 1555 in more than 40% of the test instances. When we relax the constraints of VRPTW by lifting t_i^{start} ,
 1556 which allows vehicles to fulfill customers' demands early, there is a significant decrease in the per-
 1557 centage of violations, down to approximately 25%. To ensure validity of the generated heuristics
 1558 across all instances in this challenging setting, given RedAHD's flexibility, a potential workaround
 1559 could be adopting the structure of existing GAFs from LLM-EPS methods (which guarantee fea-
 1560 sible solutions⁵) during fitness evaluation, at the cost of reduced automation. Future studies may
 1561 consult the EC literature (Wu et al., 2024) for devising better ways of navigating the search within
 1562 the confined heuristic space H' .
 1563

⁴All prior LLM-EPS methods did not consider this COP, though we are aware of a recent Python library for LLM-based AHD (Liu et al., 2024c) that implements a variant of the IC framework specifically for VRPTW.

⁵As an example, please refer to the tailored IC framework for VRPTW in the library from Liu et al. (2024c).

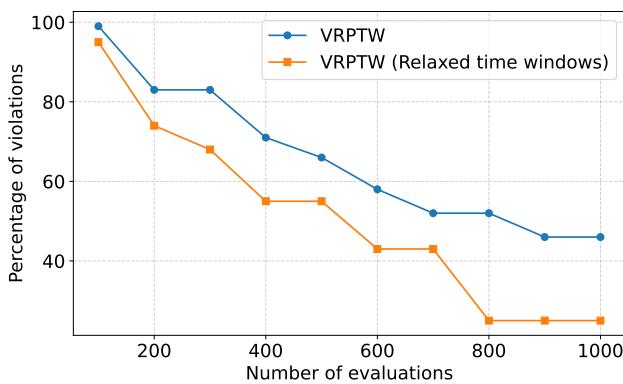


Figure S10: Percentage of test instances for VRPTW in which the heuristics designed by RedAHD violate time window constraints. The orange line considers a relaxed version of VRPTW where vehicles can arrive and serve customers early (i.e., the time windows are $[0, t_i^{end}]$ for all customers i).

D.7 THE ADVANTAGE SCOPE OF REDAHD

Knowing the strengths and current limitations of RedAHD, we summarize application scenarios where our framework would excel.

- *COPs with large heuristic space.* Using reductions and multi-problem LLM-EPS, RedAHD benefits from the many alternatives for H' , which indicates RedAHD is more suitable for COPs with less restrictive heuristic space H (e.g., BPP). In practice, the application scenarios could involve designing effective heuristics for a newly formulated COP with moderate constraints.
- *Well-studied COPs in need of performance enhancement.* Since the quality of the designed LRs relies on the domain knowledge of LLMs, we believe RedAHD would perform particularly well on application scenarios where the problem of interest can be formulated as classical COPs (e.g., TSP and MKP) and available off-the-shelf methods (e.g., approximation algorithms and handcrafted heuristics) yield unsatisfactory optimization performance.