

EFFICIENT HEURISTICS GENERATION FOR SOLVING COMBINATORIAL OPTIMIZATION PROBLEMS USING LARGE LANGUAGE MODELS

Anonymous authors

Paper under double-blind review

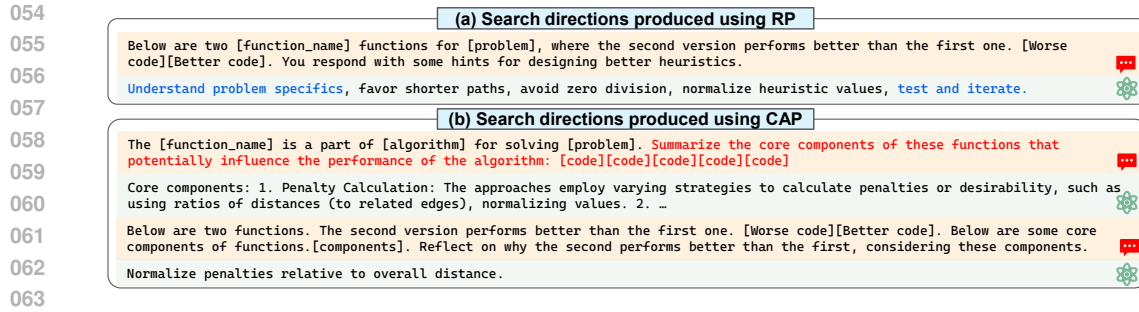
ABSTRACT

Recent studies exploited Large Language Models (LLMs) to autonomously generate heuristics for solving Combinatorial Optimization Problems (COPs), by prompting LLMs to first provide search directions and then derive heuristics accordingly. However, the absence of task-specific knowledge in prompts often leads LLMs to provide unspecific search directions, obstructing the derivation of well-performing heuristics. Moreover, evaluating the derived heuristics remains resource-intensive, especially for those semantically equivalent ones, often requiring unnecessary resource expenditure. To enable LLMs to provide specific search directions, we propose the Hercules algorithm, which leverages our designed Core Abstraction Prompting (CAP) method to abstract the core components from elite heuristics and incorporate them as prior knowledge in prompts. We theoretically prove the effectiveness of CAP in reducing unspecificity and provide empirical results in this work. To reduce the required computing resources for evaluating the derived heuristics, we propose few-shot Performance Prediction Prompting (PPP), a first-of-its-kind method for the Heuristic Generation (HG) task. PPP leverages LLMs to predict the fitness values of newly derived heuristics by analyzing their semantic similarity to previously evaluated ones. We further develop two tailored mechanisms for PPP to enhance predictive accuracy and determine unreliable predictions, respectively. The use of PPP makes Hercules more resource-efficient and we name this variant Hercules-P. Extensive experiments across various HG tasks, COPs, and LLMs demonstrate that Hercules outperforms the state-of-the-art LLM-based HG algorithms, while Hercules-P excels at minimizing computing resources. In addition, we illustrate the effectiveness of CAP, PPP, and the other proposed mechanisms by conducting relevant ablation studies.

1 INTRODUCTION

Heuristic algorithms have long been a preferred approach for solving Combinatorial Optimization Problems (COPs) (Rego et al., 2011). To automate the derivation of heuristics for a given COP, Heuristic Generation (HG) methods have attracted significant attention (Burke et al., 2013). Early HG methods predominantly employ Evolutionary Computation (EC) algorithms to derive heuristics. However, these methods focus on the exploration and exploitation in the micro search space composed of the predefined modules, often resulting in limited performance (Ye et al., 2024a).

Recently, the emergence of Large Language Models (LLMs) has facilitated the autonomous derivation of heuristics, eliminating the need for manually defining the search space (Liu et al., 2023a; 2024a; van Stein & Bäck, 2024). In addition, compared to conventional EC algorithms, LLMs benefit from a broader search space by leveraging their mega-size training corpora, resulting in elevated performance (Yang et al., 2024; Ma et al., 2024; Liu et al., 2024b). Specifically, these LLM-based HG methods exploit LLMs to provide search directions, which are then used to derive (novel) off-spring heuristics (Romera-Paredes et al., 2024). These produced heuristics are subsequently evaluated using COP instances to determine their fitness values, with the better-performing heuristics carried over to the next iteration. For example, Liu et al. (2023a) proposed prompting methods that emulate crossover and mutation operators as search strategies, thereby implicitly providing search directions. To let LLMs offer more explicit search directions, Ye et al. (2024a) proposed Reflection



070

071

072

073

074

075

076

077

078

079

080

081

082

083

084

085

086

087

088

089

090

091

092

093

094

095

096

097

098

099

100

Figure 1: Illustration of the search directions produced using RP and CAP for the task described in Section 4.1. When RP prompts LLMs (GPT-4o-mini used in this example) for search directions directly, the LLMs may respond with unspecific directions (highlighted in blue). Different from RP, our CAP enhances the quality of the produced search directions by first prompting the LLMs to abstract the core components as prior knowledge in a zero-shot manner (highlighted in red).

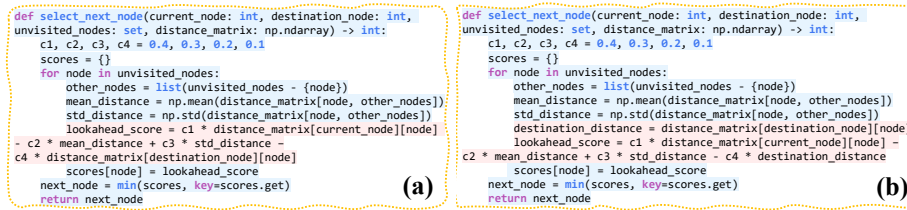


Figure 2: Illustration of two heuristics with identical semantics, produced by LLMs (GPT-3.5-turbo used in this example) for the task described in Section 4.2. Code snippets with literal equivalence are highlighted in blue, while those with semantic equivalence are highlighted in pink.

Prompting (RP), which requires LLMs to reflect on the relative performance of the produced heuristics and provide insights as search directions. These directions are then used to derive heuristics with expected elevated performance in subsequent crossover and mutation promptings.

These existing LLM-based HG methods face two key challenges. Firstly, when prompting LLMs to provide search directions (e.g., reflections on the relative performance of heuristics), the lack of task-specific knowledge in prompts often leads to over-generalized, unspecific directions that hinder the derivation of high-performance heuristics. As illustrated in Figure 1(a), the produced search directions “*Understand problem specifics*” and “*test and iterate*” are vague, over-general, and lack actionable steps required for heuristic generation. Consequently, they contribute little to the derivation of high-performance heuristics. In contrast, other elements of the produced search directions are more specific. For example, “*normalize heuristic values*” provides an actionable step that can be directly applied to derive heuristics. Therefore, it is essential to reduce unspecificity in the produced search directions. Secondly, during the search process, LLM-based HG methods often derive numerous heuristics, some of which may be semantically or even literally identical, as illustrated in Figure 2. Reevaluating these heuristics not only wastes computing resources but also significantly prolongs the search process (Chen et al., 2024). In particular, these heuristics often involve numerous linear operations and conditional branches, which GPUs cannot efficiently accelerate (Wachowiak et al., 2017). In addition, providing LLMs with all historical heuristics to avoid deriving semantically similar ones is impractical. This approach may compel LLMs to derive overly random or unviable heuristics, while significantly increasing the cost of context tokens.

To better address the first challenge, we propose **Heuristic Generation Using Large Language Models (Hercules)**, which exploits our proprietary, straightforward yet effective Core Abstraction Prompting (CAP) method to reduce unspecificity in the produced search directions and thus enable the derivation of high-performance heuristics. Specifically, CAP directs an LLM to abstract the core components from the top- k heuristics in the current population and then provide more specific search directions based on these components (see Section 3.1). Notably, as illustrated in Figure 1(b), CAP operates in a zero-shot manner, abstracting the core components without providing any examples to guide this abstraction process, which leads to significant savings in context token costs. To couple

with CAP, we introduce a rank-based selection mechanism that increases the likelihood of selecting high-performance heuristics as parents, rather than relying on random selection (Ye et al., 2024a). Meanwhile, by incorporating the concept of information gain, we theoretically prove that CAP can reduce unspecificity in the produced search directions in Appendix A.

To better address the second challenge, we propose **Hercules-P**, which integrates CAP with our novel Performance Prediction Prompting (**PPP**) method. PPP operates in a few-shot manner by presenting LLMs with a small set of previously evaluated heuristics as examples and prompting LLMs to predict the fitness values of the newly produced heuristics based on their semantic similarity to the presented examples (see Section 3.2). Generally speaking, to enhance the predictive accuracy of PPP, we can either increase the number of examples or enhance their quality. However, collecting numerous heuristic examples along with their corresponding performance is resource-intensive. This contradicts to the primary purpose of incorporating PPP, which is to reduce resource expenditure during the search process. Moreover, unlike Neural Architecture Search (NAS), which benefits from extensive benchmarks (Ying et al., 2019; Qiu et al., 2023), the HG task lacks benchmarks with pre-evaluated heuristics. Therefore, we opt to provide higher-quality examples through a tailored example selection mechanism, termed EXEMPLAR, which favors distinct parent heuristics with superior performance as examples. Meanwhile, to determine unreliable predictions, we develop the Confidence Stratification (ConS) mechanism that requires the LLM to provide confidence levels for the predicted fitness values, thereby facilitating the identification of heuristics that need reevaluation. In summary, PPP reduces the resource expenditure in heuristic evaluations while maintaining population diversity, making it effective for tasks with a border search space. To the best of our knowledge, **our work proposes the first LLM-based performance predictor for the HG task.**

To assess the performance of the proposed Hercules and Hercules-P algorithms, we conduct extensive experiments on four HG tasks (see Section 4). The experimental results demonstrate that Hercules outperforms the state-of-the-art (SOTA) LLM-based HG algorithms across diverse HG tasks, COPs, and LLMs, without significantly increasing context or generation token costs. By incorporating PPP, Hercules-P significantly reduces the overall search time by 7%~59% when compared to Hercules, while achieving on-par performance on the gain metric. Finally, ablation studies validate the effectiveness of the proposed rank-based selection mechanism, EXEMPLAR, and ConS.

The key contributions of this work are as follows.

- i) We propose the zero-shot CAP method, which reduces unspecificity in the LLM-produced search directions, enabling the derivation of high-performance heuristics. We also provide the theoretical proof of CAP’s effectiveness in reducing unspecificity by utilizing the concept of information gain.
- ii) We propose the few-shot PPP method, a first-of-its-kind LLM-based performance predictor specifically designed for HG tasks. PPP predicts the performance of newly produced heuristics by analyzing their semantic similarity to previously evaluated ones. Moreover, we develop two novel mechanisms: EXEMPLAR and ConS, which significantly enhance the overall performance of PPP.
- iii) The experimental results demonstrate that our proposed Hercules achieves SOTA performance across diverse HG tasks, COPs, and LLMs, while Hercules-P excels at reducing resource expenditure. Finally, ablation study results validate the effectiveness of all proposed methods.

2 RELATED WORK

In this section, we review the relevant literature.

2.1 LLM-BASED HEURISTIC GENERATION ALGORITHMS

Conventional EC-based HG algorithms search for the optimal combination of the predefined heuristic modules (Keller & Poli, 2007), which often limits their performance. In contrast, LLM-based HG algorithms eliminate the need for predefining the search space, liberating researchers from manual customization and enabling the derivation of high-performance heuristics (Zhang et al., 2024; Wu et al., 2024a; Huang et al., 2024). Specifically, these algorithms begin with a seed heuristic to prompt LLMs to derive multiple heuristics as the initial population (Liu et al., 2023a; 2024a; Ye et al., 2024a). Each heuristic is then evaluated using a set of COP instances, with its performance serving as its fitness value. During the iterative process, certain heuristics are selected as parents and presented to LLMs to derive (novel) offspring heuristics. This approach emulates the concepts of

162 crossover and mutation, while implicitly providing search directions for the LLMs to derive heuristics.
163 In addition, certain studies exploit LLMs to provide explicit search directions for deriving
164 well-performing heuristics (Ye et al., 2024a). However, these LLM-based HG algorithms overlook
165 the issue of unspecificity in LLM responses (see Figure 1(a)), which can lead to unspecific search
166 directions that do not contribute to discovering high-performance heuristics.

167 Similar challenges are observed in tasks such as arithmetic and symbolic reasoning, making it crucial
168 to evoke LLM reasoning through a multi-step process and incorporate task-specific knowledge (Yu
169 et al., 2024; Jiang et al., 2024; Lv et al., 2024). For example, Wei et al. (2022) proposed Chain-
170 of-Thought (CoT) prompting, which directs LLMs to emulate the given examples in completing
171 a multi-step solution process, leading to more accurate answers. Subsequently, Zheng et al. (2024)
172 proposed the few-shot Step-back Prompting (SP), which exploits the given examples to enable LLMs
173 to abstract high-level principles and then apply these principles in reasoning. In a similar multi-
174 step fashion, we propose CAP to mitigate unspecificity in the produced search directions for better
175 solving HG tasks. However, unlike CoT and SP, CAP operates in a zero-shot manner, because it
176 abstracts the core components without any examples to guide the abstraction process.

177 2.2 LLM-BASED PERFORMANCE PREDICTION METHODS

178 In the field of NAS, performance predictors, typically Deep Neural Networks, are widely used to
179 reduce search costs by predicting the performance of candidate architectures (Baker et al., 2017;
180 Wu et al., 2021). These predictors model neural architectures as graphs, where nodes represent
181 subnets and edges represent the connections between subnets (Chu et al., 2023; Liu et al., 2022).
182 The graphs are then encoded into vectors, and the mapping between these vectors and the corre-
183 sponding performance metrics is learned. Recently, Jawahar et al. (2024) and Chen et al. (2024)
184 proposed LLM-based predictors for predicting the performance of neural architectures. Specifi-
185 cally, they employed examples of architectures and corresponding performance metrics to prompt
186 LLMs, leveraging semantic similarity to predict the performance of newly searched architectures.

187 In the context of HG, conventional performance predictors may struggle to accurately evaluate
188 heuristics due to the difficulty in modeling these diverse and complex heuristics as graph structures.
189 However, the LLM-based predictor presents a promising alternative by eliminating the need for
190 explicit heuristic modeling. Consequently, this paper leverages LLMs to predict the performance
191 of heuristics for effectively solving HG tasks. However, unlike (Jawahar et al., 2024) and (Chen
192 et al., 2024), which relied on a larger number of examples, our PPP emphasizes the use of only the
193 higher-quality examples to improve predictive performance (see Section 3.2 for more details).
194

195 2.3 NEURAL COMBINATORIAL OPTIMIZATION SOLVERS

196 Neural Combinatorial Optimization (NCO) refers to a class of Neural Network solvers that either
197 independently solve COPs or collaborate with heuristic algorithms (Bengio et al., 2021; Wu et al.,
198 2022; 2024b; Bogrybayeva et al., 2024). To enable the derivation of insights from historical COP
199 instances and efficiently handle batches of instances in parallel, researchers have recently developed
200 numerous NCO solvers (Kwon et al., 2020; Lu et al., 2020; Hudson et al., 2022; Chen et al., 2023;
201 Kim et al., 2024; Dervedde et al., 2024). However, these NCO solvers still face several challenges.
202 Two of the most prominent ones are how to improve their generalization capabilities (Zhou et al.,
203 2023; Xiao et al., 2024; Hottung et al., 2024) and their performance on large-scale COPs (Hou
204 et al., 2023; Sun & Yang, 2023; Min et al., 2023; Ye et al., 2024b). Recently, Wang et al. (2024)
205 proposed a distance-aware heuristic algorithm designed to enhance the generalization ability of NCO
206 solvers trained on small-scale COPs for solving large-scale COPs. To assess the effectiveness of the
207 proposed Hercules and Hercules-P algorithms, we apply them to improve the performance of two
208 classic NCO solvers on both small-scale and large-scale COPs in Section 4.4.
209

210 3 HEURISTIC GENERATION WITH HERCULES AND HERCULES-P

211 The illustrations of Hercules and Hercules-P are schematically presented in Figure 3. In this section,
212 we first introduce CAP, which is designed to provide more specific search directions for deriving
213 heuristics. We then prove that CAP can reduce unspecificity of the produced search directions.
214 Finally, we present the design of PPP, along with tailored EXEMPLAR and ConS mechanisms.
215

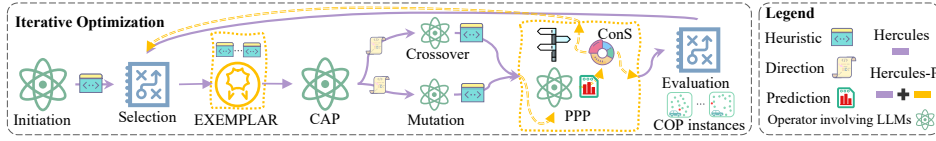


Figure 3: Overview of the proposed Hercules and Hercules-P algorithms. Hercules exploits CAP to provide specific search directions, which are then used to guide LLMs in deriving high-performance heuristics. In Hercules, the performance of all derived heuristics on a set of COP instances determines their respective fitness values. In contrast, Hercules-P evaluates only a subset of the produced heuristics with COP instances, while the rest are assessed using the proposed PPP method.

3.1 CORE ABSTRACTION PROMPTING (CAP)

As aforementioned, when LLMs are tasked with providing search directions, they often generate directions that lack specificity for heuristic derivation. As illustrated in the RP example in Figure 1(a), certain directions, such as “Understand problem specifics” and “test and iterate”, lack relevance to heuristic derivation and fail to derive well-performing heuristics.

In this case and many others, providing prior knowledge in prompts can help LLMs reduce unspecificity in their responses, leading to more focused, specific search directions. To achieve this, we propose the zero-shot CAP method, which can abstract the core components from the top- k heuristics in the current population without additional guidance. Because the core components are essential for heuristic performance (Xue et al., 2016; Liu et al., 2024a), leveraging them enables LLMs to provide more specific search directions. As shown in Figure 1(b), the suggested direction “Normalize penalties relative to overall distance” may lead to more effective heuristic generation (see Appendix B for more comparative examples of search directions produced by RP and CAP). In addition, CAP abstracts the core components once per iteration, instead of abstracting distinct components separately for crossover and elitist mutation operators. Consequently, this approach helps prevent a significant increase in context and generation token costs compared to RP (see Table 2). The details about the adopted crossover and elitist mutation operators are presented in Appendix C.

In the field of information theory, the advantage of CAP can be quantified using the concept of information gain. In the prior study (Hu et al., 2024), information gain was defined as the reduction in entropy between two states. Extending this concept, we use information gain to quantify entropy reduction in scenarios with and without abstraction, facilitating the assessment of CAP in reducing unspecificity. Specifically, the entropy without abstraction (i.e., the core components are not presented to LLMs) in the t th iteration is defined as follows:

$$H(\Omega_t) = - \sum_{i:\omega_i \in \Omega_t} p(\omega_i|\Omega_t) \log p(\omega_i|\Omega_t), \quad (1)$$

where ω_i denotes a direction belonging to the set of all possible directions Ω_t .

When the core components are used as prior knowledge in prompts, an LLM can provide more specific, subdivided search directions either based on one of these core components or disregarding all core components. Consequently, the set of all possible directions, Ω_t , can be partitioned into mutually exclusive subsets, Ω_j , where $\bigcup_{j=0}^k \Omega_j = \Omega_t$. Here, when $j \in \{0, 1, \dots, k-1\}$, Ω_j represents the subset of directions associated with the j th core component, while $j = k$ corresponds to the subset of directions independent of any core component.

Assuming that the produced direction belongs to the j th subset ($j \in \{0, 1, \dots, k\}$) after providing the core components, the remaining entropy is defined as follows:

$$H(\Omega_j) = - \sum_{i:\omega_i \in \Omega_j} p(\omega_i|\Omega_j) \log p(\omega_i|\Omega_j). \quad (2)$$

Then, the entropy with abstraction (i.e., the expected remaining entropy) is defined as $\sum_{j=0}^k p_j H(\Omega_j)$, where p_j denotes the probability that the search direction belongs to the j th subset, i.e., $p_j = p(\Omega_j)/p(\Omega_t)$. Thus, the information gain from abstracting the core components in the t th iteration (the entropy reduction without and with abstraction) is defined as follows:

$$IG(\Omega_t) = H(\Omega_t) - \sum_{j=0}^k p_j H(\Omega_j). \quad (3)$$

270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323

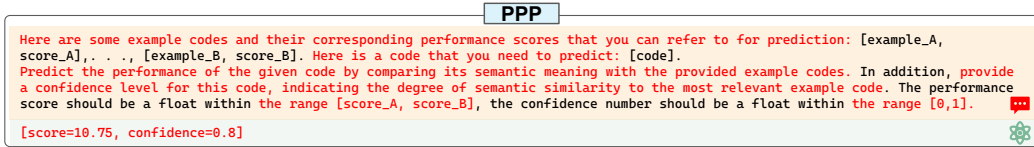


Figure 4: Illustration of the prediction process using the proposed PPP method. By analyzing the semantic similarity between the heuristics to be predicted and the previously evaluated ones, LLMs can respond with a performance score for each heuristic with an associated confidence level.

As proven in Appendix A, (3) simplifies to the following expression, ranging from $(0, \log(k + 1)]$:

$$IG(\Omega_t) = - \sum_{j=0}^k p_j \log p_j. \quad (4)$$

Therefore, in theory, providing the core components reduces unspecificity in LLM responses and yields more specific search directions, subsequently leading to heuristics with higher performance.

To fine-search the space with high-quality heuristics, we adopt a rank-based selection mechanism. Specifically, the probability of selecting the i th heuristic as a parent is computed as follows:

$$p(x_i) = \frac{1}{\text{rank}(x_i) + N} \bigg/ \sum_{j=1}^N \frac{1}{\text{rank}(x_j) + N}, \quad (5)$$

where N denotes the population size, and $\text{rank}(\cdot)$ returns the rank of the associated fitness value in the ascending order. In addition, Hercules adopts the core components of the top- k heuristics as prior knowledge during the first λ percent of iterations ($\lambda \in [0, 1]$). In the later iterations, to better preserve population diversity, Hercules directly applies the core components of the parent heuristics as prior knowledge to provide search directions, bypassing the abstraction process of elite heuristics.

3.2 PERFORMANCE PREDICTION PROMPTING (PPP)

Semantic features have demonstrated significant merits in software engineering tasks, e.g., identifying the defective code regions (Liu et al., 2023b), due to their influence on the overall code performance. Motivated by this concept, we propose the few-shot PPP method, which leverages LLMs to predict the performance of newly produced heuristics by analyzing their semantic similarity to previously evaluated ones, as shown in Figure 4. To achieve higher predictive accuracy with a small number of N_e examples, we propose an example selection mechanism called EXEMPLAR, which operates on a principle similar to providing a more relevant, well-defined knowledge base in retrieval-augmented generation (Gao et al., 2023). Specifically, EXEMPLAR selects the historically best and worst heuristics, i.e., x_{lb} and x_{ub} , respectively, as prediction boundaries (assuming the goal of the HG task is to derive the heuristic with the minimum fitness value), while prioritizing parent heuristics with better performance (i.e., lower fitness value). Parent heuristics with better performance are typically more complex and richer in semantic features than those with inferior performance, highly likely leading to higher prediction accuracy. In addition, any heuristic with the same fitness value as a previously selected example will not be chosen as an example. Because if LLMs encounter multiple examples sharing the same fitness value, their predictions may become biased towards this common fitness value, potentially overlooking semantic features. If each example has a distinct fitness value, LLMs can more effectively leverage semantic features to predict the performance of the new heuristics. The set of examples \mathcal{P}_e is selected as follows:

$$\mathcal{P}_e = \{x_{lb}, x_{ub} \mid x_{lb} = \arg \min_{x \in \mathcal{P}_h} f(x), x_{ub} = \arg \max_{x \in \mathcal{P}_h} f(x)\} \cup \{x \mid \arg \text{top}(N_e-2) f(x)\}, \quad (6)$$

$$\mathcal{P}_t = \{x \in \mathcal{P}_p \setminus \{x_{lb}, x_{ub}\} \mid f(x_i) \neq f(x_j), \forall i \neq j\},$$

where \mathcal{P}_h and \mathcal{P}_p denote the set of all historical heuristics and the set of parent heuristics selected from the current iteration according to (5) to produce offspring, respectively, and $f(\cdot)$ denotes the fitness evaluation function, introduced in the following paragraph. EXEMPLAR selects the set \mathcal{P}_e for each iteration.

Nevertheless, LLMs cannot always accurately predict the performance of each heuristic. To mitigate the potential impact of incorrect predictions, we propose the Confidence Stratification (ConS)

Algorithm 1 Hercules-P for Deriving Heuristics

Input: Maximum iteration number T
Output: Best heuristic x_{best}

```

1 //Omitting Steps 5, 10, and 11 makes Hercules-P fall back to the original Hercules algorithm
2 Initialize and evaluate population  $\mathcal{P}$ ; the number of current iteration  $t = 0$ 
3 while  $t < T$  do
4   Select parent heuristics set  $\mathcal{P}_p$  according to (5) //Rank-based selection
5   Select heuristic examples set  $\mathcal{P}_e$  for PPP according to (6) //EXEMPLAR
6   if  $t \leq \lambda \cdot T$  then Provide search directions using core components of elite heuristics //CAP ;
7   else Provide search directions using core components of parent heuristics;
8   Derive heuristics using crossover based on the produced search directions
9   Derive heuristics using elitist mutation based on the produced search directions
10  Predict the fitness values of newly produced heuristics //PPP
11  Determine fitness values  $f(\cdot)$  according to (7) //ConS
12  Update  $\mathcal{P}$  and  $x_{best}$  with new heuristics

```

mechanism. Other than the LLM-predicted fitness value ξ_i , ConS prompts an LLM to provide a corresponding confidence level $\phi_i \in [0, 1]$ based on the degree of semantic similarity between x_i and the most similar examples in P_e . Subsequently, based on ϕ_i , ConS selectively accepts the predicted fitness values of certain heuristics, while others are reevaluated using COP instances. Intuitively, we implement the following design. For heuristic x_i , if ϕ_i is sufficiently high, ConS deems ξ_i accurate. If ϕ_i is moderately high, only the top-ranked candidates in this category should be trusted to directly adopt ξ_i without reevaluation, reflecting the degraded confidence level. For low ϕ_i values, they can only be directly adopted if ξ_i is greater than a predetermined threshold. Because for these heuristics with an acceptable yet sub-par performance score and a not-too-low confidence level, it is intuitive to deem them having inferior performance, without the need for precise predictions (Xu et al., 2021). Specifically, we heuristically define this threshold gauging the known prediction boundaries, i.e., lb_t and ub_t . When ϕ_i is extremely low, ξ_i is deemed unreliable and the corresponding heuristic must be reevaluated. Such design is implemented as follows to define the fitness function $f(x_i)$:

$$f(x_i) = \begin{cases} \xi_i, & \phi_i \geq 1 - \delta, \\ \xi_i, & 1 - 2\delta \leq \phi_i < 1 - \delta \wedge x_i \in \arg \max_{x \in \mathcal{P}_c} \phi(x), \\ \xi_i, & 1 - 3\delta \leq \phi_i < 1 - 2\delta \wedge \xi_i > lb_t + 3\delta(ub_t - lb_t), \\ \mathcal{F}(x_i), & \text{otherwise,} \end{cases} \quad (7)$$

where $\delta \in [0, 1/3]$ denotes a predefined interval to distinguish the performance range of the produced heuristics (a smaller δ value means ConS only accepts the predicted scores with the highest confidence), \mathcal{P}_c denotes the set of heuristics whose ϕ_i values lie within the $[1 - 2\delta, 1 - \delta]$ interval, and $\mathcal{F}(\cdot)$ denotes the conventional fitness evaluation function, which uses COP instances to evaluate heuristics. Furthermore, we gradually decrease the number of heuristics that do not require reevaluation in \mathcal{P}_c after each iteration. Specifically, we set an acceptance threshold $m_t = \lfloor \alpha \cdot \beta^t \cdot N_o \rfloor$, where $\alpha, \beta \in (0, 1)$, and N_o denotes the number of the produced heuristics in the current iteration.

The pseudocode of Hercules-P is presented in Algorithm 1, and its source code is available online¹.

4 EXPERIMENTAL RESULTS

This section presents extensive experimental results on various HG tasks, COPs, and LLMs to assess the performance of both Hercules and Hercules-P. Please refer to Appendices D, E, F, and G for the experimental setups with predefined hyperparameter values, results on the search time of various LLM-based HG algorithms, prompts used in this paper, and the produced heuristics, respectively.

4.1 DERIVING PENALTY HEURISTICS FOR GLS TO SOLVE TSP

In this subsection, we exploit Hercules and Hercules-P to derive penalty heuristics for Guided Local Search (GLS) to solve the Travelling Salesman Problem (TSP). The seed function is human-designed

¹<https://anonymous.4open.science/r/ICLR-12808>

Table 1: Performance comparison of different GLS algorithms on TSP

Algorithm	Type	Gain (%) ($n = 100$)	Gain (%) ($n = 200$)
KGLS-Random	GLS+Llama3-70b	-137.13	0.47
KGLS-EoH (ICML'24)	GLS+Llama3-70b	-369.10	5.82
KGLS-ReEvo (NeurIPS'24)	GLS+Llama3-70b	-661.69	2.19
KGLS-Hercules-P (ours)	GLS+Llama3-70b	-218.91	4.71
KGLS-Hercules (ours)	GLS+Llama3-70b	-12.48	3.42
KGLS-Random	GLS+GPT-4o-mini	<u>63.64</u>	3.44
KGLS-EoH (ICML'24)	GLS+GPT-4o-mini	25.53	5.62
KGLS-ReEvo (NeurIPS'24)	GLS+GPT-4o-mini	-280.79	2.45
KGLS-Hercules-P (ours)	GLS+GPT-4o-mini	71.05	<u>7.46</u>
KGLS-Hercules (ours)	GLS+GPT-4o-mini	42.98	11.10

Table 2: Search cost comparison of different LLM-based HG algorithms on TSP

Algorithm	Gain (%)	Time (m)	Context Token (k)	Generation Token (k)
KGLS-Random	3.44±1.20	<u>28.5</u> ±2.2	0.2	19.4
KGLS-EoH (ICML'24)	5.62±1.83	37.2±7.2	43.5	26.2
KGLS-ReEvo (NeurIPS'24)	2.45±10.93	37.7±12.2	95.5	42.0
KGLS-Hercules-P (ours)	<u>7.46</u> ±3.36	23.6 ±3.0	143.4	31.2
KGLS-Hercules (ours)	11.10 ±0.69	30.6±1.4	95.8	33.3

GPT-4o-mini

heuristic KGLS (Arnold & Sörensen, 2019). We choose three LLM-based HG algorithms as benchmarking models, namely Random, EoH (Liu et al., 2024a), and ReEvo (Ye et al., 2024a). Random is a straightforward method that derives heuristics directly using LLMs without incorporating search directions and is commonly used as a baseline model in NAS studies (Li & Talwalkar, 2020). In addition, unless specified otherwise, for the performance of LLM-based HG algorithms, namely Random, EoH, ReEvo, Hercules-P, and Hercules, we report the average performance of three independent runs, following the prior study (Ye et al., 2024a). The average gains of the heuristics produced by these algorithms are presented in Table 1, where n denotes the problem scale. The gain measure is calculated as $1 - (\text{the performance of the LLM-produced heuristics}) / (\text{the performance of the original KGLS})$.

As shown in Table 1, for the 200-node TSP, the heuristics produced by Hercules using GPT-4o-mini outperform those produced by the other HG algorithms, yielding the best performance gain of 11.1%. In addition, when GPT-4o-mini is adopted, the average gain of Hercules-P drops by only 3.64% comparing to Hercules, securing the second-best performance. EoH ranks at the third place in the gain metric. The experimental results shown in Table 1 highlight that the choice of LLM significantly impacts the performance of the produced heuristics. Nevertheless, Hercules and Hercules-P consistently outperform ReEvo across all node scales, regardless of the LLM in use.

Table 2 presents the search cost comparison of LLM-based HG algorithms across four metrics, namely gain (identical to the bottom-right cell of Table 1), search time, context token, and generation token. The results show that Hercules yields better gains without substantially increasing the costs of context and generation tokens, compared to ReEvo. Moreover, ReEvo and EoH spend longer search time when compared to the others, likely due to their ineffective search directions, which cause the LLM to derive complex but suboptimal heuristics. The std value of 10.93 for ReEvo further underscores this issue. On the other hand, Hercules-P reduces the overall search time to 77% (23.6/30.6) of that required by Hercules. Although Hercules-P uses approximately 1.5 times more context tokens than Hercules and ReEvo, it does not significantly increase the cost of generation tokens, which are typically more expensive (OpenAI). This makes Hercules-P ideal for environments with limited computing resources. Notably, Random utilizes only 0.2k context tokens, because of its simple prompts used for heuristic generation. However, this simplicity limits its ability to derive well-performing heuristics.

4.2 DERIVING CONSTRUCTIVE HEURISTICS TO SOLVE TSP

To assess the generalization capabilities of Hercules and Hercules-P across different HG tasks, we employ them in this subsection to derive constructive heuristics, which sequentially select unvisited nodes for solving real-world TSPLIB benchmarks (Reinelt, 1991). The seed function is genetic programming hyper-heuristic (Dufflo et al., 2019). As shown in Table 3, Hercules achieves the highest average gain of 4.87% across eighteen TSPLIB instances, followed by EoH with the average

Table 3: Performance comparison of different constructive heuristic algorithms on TSPLIB

instances (total number)	Random	EoH (ICML'24)	ReEvo (NeurIPS'24)	Hercules-P (ours)	Hercules (ours)
$n < 101$ (4)	-3.92	16.68	1.18	14.16	10.52
$101 \leq n \leq 500$ (9)	-3.80	-0.60	-1.17	<u>0.71</u>	2.25
$n > 500$ (5)	-5.73	5.32	0.46	0.95	<u>5.18</u>
Avg. Gain (%) (18)	-4.49	<u>4.80</u>	-0.16	3.42	4.87

GPT-3.5-turbo

Table 4: Performance comparison of different ACO algorithms on BPP and MKP

Algorithm	Type	BPP (Gain (%)), LLM: Llama3.1-405b			MKP (Gain (%)), LLM: Gemma2-27b		
		$n = 120$	$n = 500$	$n = 1,000$	$n = 120$	$n = 500$	$n = 1,000$
ACO+Random	ACO+LLM	0.00	-0.09	0.00	1.24	3.21	4.01
ACO+EoH (ICML'24)	ACO+LLM	0.14	0.16	0.38	<u>1.61</u>	4.42	5.81
ACO+ReEvo (NeurIPS'24)	ACO+LLM	<u>0.66</u>	<u>1.49</u>	2.01	1.59	4.67	<u>6.31</u>
ACO+Hercules-P (ours)	ACO+LLM	0.08	1.47	<u>2.04</u>	1.44	<u>4.73</u>	6.14
ACO+Hercules (ours)	ACO+LLM	0.84	1.64	2.19	1.99	6.40	8.22

Table 5: Performance comparison of different NCO solvers on TSP and CVRP

Algorithm	Type	TSP (Gain (%))			CVRP (Gain (%))		
		$n = 200$	$n = 500$	$n = 1,000$	$n = 200$	$n = 500$	$n = 1,000$
POMO+Random	NCO+GPT-4o-mini	3.05	-18.90	-35.10	3.07	<u>1.14</u>	2.86
POMO+EoH (ICML'24)	NCO+GPT-4o-mini	2.19	<u>1.42</u>	<u>1.47</u>	0.48	-1.83	0.27
POMO+ReEvo (NeurIPS'24)	NCO+GPT-4o-mini	2.38	-5.24	-2.78	0.34	-14.20	-3.01
POMO+Hercules-p (ours)	NCO+GPT-4o-mini	-0.10	-4.81	-3.58	-0.57	-3.29	-0.57
POMO+Hercules (ours)	NCO+GPT-4o-mini	<u>2.49</u>	6.62	16.43	<u>1.53</u>	1.22	<u>1.59</u>
LEHD+Random	NCO+GPT-4o-mini	<u>9.93</u>	8.83	5.44	1.72	2.33	<u>1.68</u>
LEHD+EoH (ICML'24)	NCO+GPT-4o-mini	10.67	<u>7.73</u>	<u>6.09</u>	6.62	3.57	0.47
LEHD+ReEvo (NeurIPS'24)	NCO+GPT-4o-mini	6.94	-1.78	1.56	<u>10.19</u>	<u>4.97</u>	0.70
LEHD+Hercules-p (ours)	NCO+GPT-4o-mini	9.55	7.53	6.89	4.44	2.45	0.75
LEHD+Hercules (ours)	NCO+GPT-4o-mini	7.46	6.64	5.14	14.37	7.90	2.33

gain of 4.8%. In contrast, both Random and ReEvo perform poorly, yielding negative gains on average, i.e., failing to improve the performance of the seed function.

4.3 DERIVING HEURISTIC MEASURES FOR ACO TO SOLVE BPP AND MKP

In this subsection, we exploit Hercules and Hercules-P to derive heuristic measures for Ant Colony Optimization (ACO) applied to the Bin Packing Problem (BPP) and Multiple Knapsack Problem (MKP). The seed function is a conventional ACO algorithm (Dorigo et al., 2006). We adopt Llama3.1-405b to solve BPP while adopt Gemma2-27b to solve MKP. This is because Llama 3.1-405b fails to improve the seed function of MKP regardless of which LLM-based HG algorithm is executed. As shown in Table 4, Hercules outperforms the other algorithms across all COPs and LLMs, with particularly strong performance observed when solving the 1,000-scale MKP, achieving an 8.22% gain. In addition, when using Llama3.1-405b, Random fails to derive superior heuristics compared to the original ACO, while EoH achieves only a modest improvement, falling short when compared to the more substantial gains obtained by ReEvo, Hercules-P, and Hercules.

4.4 RESHAPING ATTENTION SCORES FOR NCO TO SOLVE TSP AND CVRP

Recently, Wang et al. (2024) demonstrated that reshaping attention scores can enhance the generalization performance of NCO solvers trained on small-scale COPs for solving large-scale COPs. To assess the effectiveness of Hercules and Hercules-P on NCO solvers, following (Ye et al., 2024a), we select DAR (Wang et al., 2024) as the seed function for TSP and the vanilla POMO (Kwon et al., 2020) and LEHD (Luo et al., 2023) as seed functions for CVRP. As shown in Table 5, Random outperforms the other four LLM-based HG algorithms on certain tasks. A plausible reason for this is that the LLM corpora may lack sufficient knowledge of emerging NCO domains, thus limiting the performance of the other four LLM-based HG algorithms. Nevertheless, the heuristics derived by Hercules outperform the corresponding seed functions across a wider range of tasks compared to Random. For example, Hercules performs better than Random on the 500- and 1,000-node scales for the TSP-POMO task. In addition, Appendix E provides a detailed comparison on search time across these five LLM-based HG algorithms. The experimental results show that Hercules-P achieves the shortest search time across all NCO tasks. For example, it solves the 1,000-node CVRP-LEHD task

Table 6: Ablation study results on different design choices

Algorithm	Gain (%)	Algorithm	Gain (%)	Algorithm	Gain (%)	Algorithm	Gain (%)
w/o CAP	3.12	Hercules ($\lambda = 0.5$)	5.96	w/o ConS	-4.06	Hercules-P ($\delta = 0.2$)	7.01
w/o rank-based selection	<u>8.49</u>	Hercules ($\lambda = 0.9$)	<u>8.90</u>	w/o EXEMPLAR	<u>-0.30</u>	Hercules-P ($\delta = 0.3$)	6.21
		Hercules ($\lambda = 1$)	5.60				
Hercules (w/o PPP)	11.10	Hercules ($\lambda = 0.7$)	11.10	Hercules-P	7.46	Hercules-P ($\delta = 0.1$)	7.46

GPT-4o-mini

in roughly five hours, which is approximately 41% of the time needed by Hercules. Across all tasks, Hercules-P effectively reduces the search time by 7%~59% when compared to Hercules.

4.5 ABLATION STUDIES

In this subsection, we conduct ablation studies to investigate the effectiveness of the design choices of Hercules and Hercules-P, and present the results in Table 6. The adopted HG task is deriving penalty heuristics for GLS to solve TSPs (see Section 4.1). Specifically, w/o CAP refers to the setting using RP to provide search directions, w/o rank-based selection refers to the setting that randomly selects parent heuristics, w/o ConS refers to the setting that PPP assumes all predictions are accurate, and w/o EXEMPLAR refers to the setting that heuristic examples are randomly selected from the current population. For all the other experiments presented in this paper, $\lambda = 0.7$ is applied for Hercules, and $\delta = 0.1$ is applied for Hercules-P. As shown in Table 6, when CAP is omitted, the gain decreases by 7.98%, further demonstrating that CAP produces more specific search directions. In addition, the proposed rank-based selection mechanism significantly contributes to the superior performance of Hercules. For Hercules-P, ConS effectively determines unreliable predictions, preventing them from negatively affecting the derivation of high-performance heuristics. Finally, when EXEMPLAR is omitted, the gain decreases by 7.76%, mainly due to the associated degradation in predictive accuracy (elaborated in the following paragraph).

We further present the predictive accuracy of PPP with and without EXEMPLAR, both of which are executed ten times, aiming to perform meaningful statistical tests. In addition, we include w/ EXEMPLAR-U as an additional setting, where EXEMPLAR is able to select heuristics with identical fitness values. To assess whether different versions of EXEMPLAR can accurately predict the fitness values of the produced heuristics, we need to set a quantifying measure. Specifically, we intuitively deem a prediction accurate if the absolute error between the predicted fitness value and the true fitness value is less than $\delta \cdot (ub_t - lb_t)$. As shown in Figure 5, the inclusion of EXEMPLAR improves the median of predictive accuracy by 26% and 37% (both significantly different: $p = 0.048$ and 0.004) when compared to w/ EXEMPLAR-U and w/o EXEMPLAR, respectively. It is imperative to clarify that although the proposed PPP may seem less accurate in predicting heuristic performance, the values shown in Figure 5 are determined by a strict measure of fitness values as afore-defined and they do not exhibit a strong correlation with the overall performance of Hercules-P, because many produced heuristics are reevaluated (see ConS in Section 3.2). As discussed in Sections 4.1 and 4.4, Hercules-P reduces search time by 7%~59% when compared to Hercules, while achieving on-par gain. We strongly believe that PPP is highly beneficial for HG tasks that require rapid solutions, e.g., deriving heuristics for the dynamic, near-real-time allocation of resources in 5G mobile edge cloud networks (Laboni et al., 2024). We plan to extend PPP by integrating it with other methods, such as beam search, to further enhance its predictive accuracy.

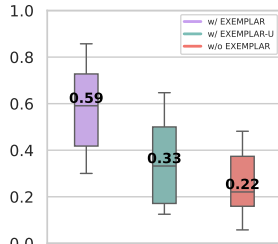


Figure 5: Ablation study on the predictive accuracy of different EXEMPLAR variants.

5 CONCLUSION

To derive well-performing heuristics, we propose Hercules, which exploits our proprietary CAP to abstract the core components from elite heuristics, to produce more specific search directions. In addition, we introduce Hercules-P, a resource-efficient variant that integrates CAP with our novel PPP. PPP exploits previously evaluated heuristics to predict the performance of newly produced ones, thereby reducing the required computing resources for heuristic evaluations. The experimental results demonstrate the effectiveness of Hercules, Hercules-P, and all our designed mechanisms.

REFERENCES

- 540
541
542 Florian Arnold and Kenneth Sörensen. Knowledge-guided local search for the vehicle routing prob-
543 lem. *Computers & Operations Research*, 105:32–46, 2019.
- 544 Bowen Baker, Otkrist Gupta, Nikhil Naik, and Ramesh Raskar. Designing neural network architec-
545 tures using reinforcement learning. In *the International Conference on Learning Representations*,
546 pp. 1–18, 2017.
- 547
548 Yoshua Bengio, Andrea Lodi, and Antoine Prouvost. Machine learning for combinatorial opti-
549 mization: A methodological tour d’horizon. *European Journal of Operational Research*, 290(2):
550 405–421, 2021.
- 551 Aigerim Bogrybayeva, Meraryslan Meraliyev, Taukekhan Mustakhov, and Bissenbay Dauletbayev.
552 Machine learning to solve vehicle routing problems: A survey. *IEEE Transactions on Intelligent*
553 *Transportation Systems*, 25(6):4754–4772, 2024.
- 554
555 Edmund K Burke, Michel Gendreau, Matthew Hyde, Graham Kendall, Gabriela Ochoa, Ender
556 Özcan, and Rong Qu. Hyper-heuristics: A survey of the state of the art. *Journal of the Op-*
557 *erational Research Society*, 64(12):1695–1724, 2013.
- 558 Jinbiao Chen, Zizhen Zhang, Zhiguang Cao, Yaoxin Wu, Yining Ma, Te Ye, and Jiahai Wang.
559 Neural multi-objective combinatorial optimization with diversity enhancement. In *Proceedings*
560 *of the Advances in Neural Information Processing Systems*, pp. 39176–39188, 2023.
- 561
562 Lin Chen, Fengli Xu, Nian Li, Zhenyu Han, Meng Wang, Yong Li, and Pan Hui. Large language
563 model-driven meta-structure discovery in heterogeneous information network. In *Proceedings of*
564 *the ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pp. 307–318, 2024.
- 565
566 Xiangxiang Chu, Shun Lu, Xudong Li, and Bo Zhang. Mixpath: A unified approach for one-
567 shot neural architecture search. In *Proceedings of the IEEE/CVF International Conference on*
568 *Computer Vision*, pp. 5972–5981, 2023.
- 569 Tim Dornedde, Daniela Thyssens, Sören Dittrich, Maximilian Stubbemann, and Lars Schmidt-
570 Thieme. MOCO: A learnable meta optimizer for combinatorial optimization, 2024. arXiv:
571 2402.04915.
- 572
573 Marco Dorigo, Mauro Birattari, and Thomas Stutzle. Ant colony optimization. *IEEE Computational*
574 *Intelligence Magazine*, 1(4):28–39, 2006.
- 575
576 Gabriel Duflou, Emmanuel Kieffer, Matthias R Brust, Grégoire Danoy, and Pascal Bouvry. A GP
577 hyper-heuristic approach for generating TSP heuristics. In *Proceedings of IEEE International*
578 *Parallel and Distributed Processing Symposium Workshops*, pp. 521–529, 2019.
- 579 Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai, Jiawei Sun, and
580 Haofen Wang. Retrieval-augmented generation for large language models: A survey, 2023. arXiv:
581 2312.10997.
- 582
583 André Hottung, Mridul Mahajan, and Kevin Tierney. PolyNet: Learning diverse solution strategies
584 for neural combinatorial optimization, 2024. arXiv: 2402.14048.
- 585
586 Qingchun Hou, Jingwei Yang, Yiqiang Su, Xiaoqing Wang, and Yuming Deng. Generalize learned
587 heuristics to solve large-scale vehicle routing problems in real-time. In *the International Confer-*
ence on Learning Representations, pp. 1–37, 2023.
- 588
589 Zhiyuan Hu, Chumin Liu, Xidong Feng, Yilun Zhao, See-Kiong Ng, Anh Tuan Luu, Junxian He,
590 Pang Wei Koh, and Bryan Hooi. Uncertainty of thoughts: Uncertainty-aware planning enhances
591 information seeking in large language models. In *the International Conference on Learning Rep-*
resentations Workshop on Large Language Model (LLM) Agents, 2024.
- 592
593 Zhehui Huang, Guangyao Shi, and Gaurav S. Sukhatme. Can large language models solve robot
routing?, 2024. arXiv: 2403.10795.

- 594 Benjamin Hudson, Qingbiao Li, Matthew Malencia, and Amanda Prorok. Graph neural network
595 guided local search for the traveling salesperson problem. In *the International Conference on*
596 *Learning Representations*, pp. 1–20, 2022.
- 597
598 Ganesh Jawahar, Muhammad Abdul-Mageed, Laks V. S. Lakshmanan, and Dujian Ding. LLM per-
599 formance predictors are good initializers for architecture search. In *the Findings of the Association*
600 *for Computational Linguistics*, pp. 10540–10560, 2024.
- 601
602 Zhuoxuan Jiang, Haoyuan Peng, Shanshan Feng, Fan Li, and Dongsheng Li. LLMs can find mathe-
603 matical reasoning mistakes by pedagogical chain-of-thought. In *Proceedings of the International*
604 *Joint Conference on Artificial Intelligence*, pp. 3439–3447, 2024.
- 605
606 R. E. Keller and R. Poli. Linear genetic programming of parsimonious metaheuristics. In *Proceed-*
607 *ings of the IEEE Congress on Evolutionary Computation*, pp. 4508–4515, 2007.
- 608
609 Hyeonah Kim, Minsu Kim, Sungsoo Ahn, and Jinkyoo Park. Symmetric replay training: Enhancing
610 sample efficiency in deep reinforcement learning for combinatorial optimization. In *Proceedings*
611 *of the International Conference on Machine Learning*, pp. 24110–24136, 2024.
- 612
613 Yeong-Dae Kwon, Jinho Choo, Byoungjip Kim, Iljoo Yoon, Youngjune Gwon, and Seungjai Min.
614 POMO: Policy optimization with multiple optima for reinforcement learning. In *Proceedings of*
615 *the Advances in Neural Information Processing Systems*, pp. 21188–21198, 2020.
- 616
617 Nadia Motalib Laboni, Sadia Jahangir Safa, Selina Sharmin, Md. Abdur Razzaque, Md. Mustafizur
618 Rahman, and Mohammad Mehedi Hassan. A hyper heuristic algorithm for efficient resource
619 allocation in 5g mobile edge clouds. *IEEE Transactions on Mobile Computing*, 23(1):29–41,
620 2024.
- 621
622 Liam Li and Ameet Talwalkar. Random search and reproducibility for neural architecture search. In
623 *Proceedings of Uncertainty in Artificial Intelligence*, pp. 367–377, 2020.
- 624
625 Fei Liu, Xialiang Tong, Mingxuan Yuan, and Qingfu Zhang. Algorithm evolution using large lan-
626 guage model, 2023a. arXiv: 2311.15249.
- 627
628 Fei Liu, Xialiang Tong, Mingxuan Yuan, Xi Lin, Fu Luo, Zhenkun Wang, Zhichao Lu, and Qingfu
629 Zhang. Evolution of heuristics: Towards efficient automatic algorithm design using large language
630 mode. In *Proceedings of the International Conference on Machine Learning*, pp. 32201–32223,
631 2024a.
- 632
633 Jingyu Liu, Jun Ai, Minyan Lu, Jie Wang, and Haoxiang Shi. Semantic feature learning for software
634 defect prediction from source code and external knowledge. *Journal of Systems and Software*,
635 204:111753, 2023b.
- 636
637 Tension Liu, Nicolás Astorga, Nabeel Seedat, and Mihaela van der Schaar. Large language models
638 to enhance bayesian optimization. In *the International Conference on Learning Representations*,
639 pp. 1–33, 2024b.
- 640
641 Yuqiao Liu, Yehui Tang, Zeqiong Lv, Yunhe Wang, and Yanan Sun. Bridge the gap between archi-
642 tecture spaces via a cross-domain predictor. In *Proceedings of the Advances in Neural Information*
643 *Processing Systems*, pp. 13355–13366, 2022.
- 644
645 Hao Lu, Xingwen Zhang, and Shuang Yang. A learning-based iterative method for solving vehicle
646 routing problems. In *the International Conference on Learning Representations*, pp. 1–15, 2020.
- 647
648 Fu Luo, Xin Li, Fei Liu, Qingfu Zhang, and Zhenkun Wang. Neural combinatorial optimization
649 with heavy decoder: Toward large scale generalization. In *Proceedings of the Advances in Neural*
650 *Information Processing Systems*, pp. 8845–8864, 2023.
- 651
652 Qitan Lv, Jie Wang, Hanzhu Chen, Bin Li, Yongdong Zhang, and Feng Wu. Coarse-to-fine high-
653 lighting: Reducing knowledge hallucination in large language models. In *Proceedings of the*
654 *International Conference on Machine Learning*, pp. 32612–32642, 2024.

- 648 Zeyuan Ma, Hongshu Guo, Jiacheng Chen, Guojun Peng, Zhiguang Cao, Yining Ma, and Yue-Jiao
649 Gong. LLaMoCo: Instruction tuning of large language models for optimization code generation,
650 2024. arXiv: 2403.01131.
- 651 Yimeng Min, Yiwei Bai, and Carla P. Gomes. Unsupervised learning for solving the travelling
652 salesman problem. In *Proceedings of the Advances in Neural Information Processing Systems*,
653 pp. 47264–47278, 2023.
- 654 OpenAI. <https://openai.com/api/pricing/>.
- 655 Zhengzhong Qiu, Wei Bi, Dong Xu, Hua Guo, Hongwei Ge, Yanchun Liang, Heow Pueh Lee,
656 and Chunguo Wu. Efficient self-learning evolutionary neural architecture search. *Applied Soft
657 Computing*, 146:110671, 2023.
- 660 César Rego, Dorabela Gamboa, Fred Glover, and Colin Osterman. Traveling salesman problem
661 heuristics: Leading methods, implementations and latest advances. *European Journal of Oper-
662 ational Research*, 211(3):427–441, 2011.
- 663 Gerhard Reinelt. TSPLIB—a traveling salesman problem library. *ORSA journal on computing*, 3:
664 376–384, 1991.
- 665 Bernardino Romera-Paredes, Mohammadamin Barekatin, Alexander Novikov, Matej Balog,
666 M. Pawan Kumar, Emilien Dupont, Francisco J. R. Ruiz, Jordan S. Ellenberg, Pengming Wang,
667 Omar Fawzi, Pushmeet Kohli, and Alhussein Fawzi. Mathematical discoveries from program
668 search with large language models. *Nature*, 625:468–475, 2024.
- 669 Zhiqing Sun and Yiming Yang. DIFUSCO: Graph-based Diffusion Solvers for Combinatorial Opti-
670 mization. In *Proceedings of the Advances in Neural Information Processing Systems*, pp. 3706–
671 3731, 2023.
- 672 Niki van Stein and Thomas Bäck. Llamea: A large language model evolutionary algorithm for
673 automatically generating metaheuristics, 2024. arXiv: 2405.20132.
- 674 Mark P. Wachowiak, Mitchell C. Timson, and David J. DuVal. Adaptive particle swarm optimization
675 with heterogeneous multicore parallelism and GPU acceleration. *IEEE Transactions on Parallel
676 and Distributed Systems*, 28(10):2784–2793, 2017.
- 677 Yang Wang, Ya-Hui Jia, Wei-Neng Chen, and Yi Mei. Distance-aware attention reshaping: Enhance
678 generalization of neural solver for large-scale vehicle routing problems, 2024. arXiv: 2401.06979.
- 679 Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, and
680 Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models. In *Pro-
681 ceedings of the Advances in Neural Information Processing Systems*, pp. 24824–24837, 2022.
- 682 Junru Wu, Xiyang Dai, Dongdong Chen, Yinpeng Chen, Mengchen Liu, Ye Yu, Zhangyang Wang,
683 Zicheng Liu, Mei Chen, and Lu Yuan. Stronger NAS with weaker predictors. In *Proceedings of
684 the Advances in Neural Information Processing Systems*, pp. 28904–28918, 2021.
- 685 Xingyu Wu, Sheng-hao Wu, Jibin Wu, Liang Feng, and Kay Chen Tan. Evolutionary computation
686 in the era of large language model: Survey and roadmap, 2024a. arXiv: 2401.10034.
- 687 Xuan Wu, Di Wang, Lijie Wen, Yubin Xiao, Chunguo Wu, Yuesong Wu, Chaoyu Yu, Douglas L.
688 Maskell, and You Zhou. Neural combinatorial optimization algorithms for solving vehicle routing
689 problems: A comprehensive survey with perspectives, 2024b. arXiv: 2406.00415.
- 690 Yaoxin Wu, Wen Song, Zhiguang Cao, Jie Zhang, and Andrew Lim. Learning improvement heuris-
691 tics for solving routing problems. *IEEE Transactions on Neural Networks and Learning Systems*,
692 33(9):5057–5069, 2022.
- 693 Yubin Xiao, Di Wang, Xuan Wu, Yuesong Wu, Boyang Li, Wei Du, Liupu Wang, and You Zhou.
694 Improving generalization of neural vehicle routing problem solvers through the lens of model
695 architecture, 2024. arXiv: 2406.06652.

- 702 Yixing Xu, Yunhe Wang, Kai Han, Yehui Tang, Shangling Jui, Chunjing Xu, and Chang Xu. ReNAS:
703 Relativistic evaluation of neural architecture search. In *Proceedings of the IEEE/CVF Conference*
704 *on Computer Vision and Pattern Recognition*, pp. 4411–4420, 2021.
- 705
706 Bing Xue, Mengjie Zhang, Will N. Browne, and Xin Yao. A survey on evolutionary computation
707 approaches to feature selection. *IEEE Transactions on Evolutionary Computation*, 20(4):606–
708 626, 2016.
- 709 Chengrun Yang, Xuezhi Wang, Yifeng Lu, Hanxiao Liu, Quoc V Le, Denny Zhou, and Xinyun
710 Chen. Large language models as optimizers. In *the International Conference on Learning Repre-*
711 *sentations*, pp. 1–42, 2024.
- 712 Haoran Ye, Jiarui Wang, Zhiguang Cao, Federico Berto, Chuanbo Hua, Haeyeon Kim, Jinkyoo
713 Parkand, and Guojie Song. Large language models as hyper-heuristics for combinatorial opti-
714 mization. In *Proceedings of the Advances in Neural Information Processing Systems*, 2024a.
- 715
716 Haoran Ye, Jiarui Wang, Helan Liang, Zhiguang Cao, Yong Li, and Fanzhang Li. GLOP: Learning
717 global partition and local construction for solving large-scale routing problems in real-time. In
718 *Proceedings of the AAAI Conference on Artificial Intelligence*, pp. 20284–20292, 2024b.
- 719 Chris Ying, Aaron Klein, Eric Christiansen, Esteban Real, Kevin Murphy, and Frank Hutter. NAS-
720 Bench-101: Towards reproducible neural architecture search. In *Proceedings of the International*
721 *Conference on Machine Learning*, pp. 7105–7114, 2019.
- 722
723 Junchi Yu, Ran He, and Zhitao Ying. Thought propagation: An analogical approach to complex rea-
724 soning with large language model. In *the International Conference on Learning Representations*,
725 pp. 1–27, 2024.
- 726 Rui Zhang, Fei Liu, Xi Lin, Zhenkun Wang, Zhichao Lu, and Qingfu Zhang. Understanding the
727 importance of evolutionary search in automated heuristic design with large language models. In
728 *Proceedings of the International Conference on Parallel Problem Solving From Nature*, pp. 185–
729 202, 2024.
- 730 Huaixiu Steven Zheng, Swaroop Mishra, Xinyun Chen, Heng-Tze Cheng, Ed H. Chi, Quoc V Le,
731 and Denny Zhou. Take a step back: Evoking reasoning via abstraction in large language models.
732 In *the International Conference on Learning Representations*, pp. 1–38, 2024.
- 733
734 Jianan Zhou, Yaoxin Wu, Wen Song, Zhiguang Cao, and Jie Zhang. Towards omni-generalizable
735 neural methods for vehicle routing problems. In *Proceedings of the International Conference on*
736 *Machine Learning*, pp. 42769–42789, 2023.
- 737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755

A DERIVATION OF INFORMATION GAIN FORMULA IN CAP

Proposition 1. *The information gain from abstracting core components is equal to:*

$$IG(\Omega_t) = - \sum_{j=0}^k p_j \log p_j \in (0, \log(k+1)]. \quad (8)$$

Proof.

$$\begin{aligned} IG(\Omega_t) &= H(\Omega_t) - p_0 H(\Omega_0) - \dots - p_k H(\Omega_k) \\ &= - \sum_{i:\omega_i \in \Omega_t} p(\omega_i|\Omega_t) \log p(\omega_i|\Omega_t) \\ &\quad + p_0 \sum_{i:\omega_i \in \Omega_0} p(\omega_i|\Omega_0) \log p(\omega_i|\Omega_0) + \dots \\ &\quad + p_k \sum_{i:\omega_i \in \Omega_k} p(\omega_i|\Omega_k) \log p(\omega_i|\Omega_k) \\ &= \sum_{i:\omega_i \in \Omega_0} p(\omega_i|\Omega_0) [\log p(\omega_i|\Omega_0) - \log p(\omega_i|\Omega_t)] + \dots \\ &\quad + \sum_{i:\omega_i \in \Omega_k} p(\omega_i|\Omega_k) [\log p(\omega_i|\Omega_k) - \log p(\omega_i|\Omega_t)] \end{aligned}$$

According to the conditional probability, $p_j \cdot p(\omega_i|\Omega_j) = p(\omega_i|\Omega_t)$, $\forall j \in \{0, 1, \dots, k\}$. Thus, the j th term simplifies to the following expression:

$$\begin{aligned} &\sum_{i:\omega_i \in \Omega_j} p(\omega_i|\Omega_j) [\log p(\omega_i|\Omega_j) - \log p(\omega_i|\Omega_t)] \\ &= \sum_{i:\omega_i \in \Omega_j} p(\omega_i|\Omega_j) \log \frac{p(\omega_i|\Omega_j)}{p(\omega_i|\Omega_t)} \\ &= - \sum_{i:\omega_i \in \Omega_j} p(\omega_i|\Omega_j) \log p_j \\ &= -p_j \log p_j \end{aligned}$$

Therefore, we conclude that:

$$IG(\Omega_t) = - \sum_{j=0}^k p_j \log p_j. \quad (9)$$

When $\forall j \in \{0, 1, \dots, k\}$, $p_j = \frac{1}{k+1}$, $IG(\Omega_t)$ reaches its maximum value of $\log(k+1)$. When $\exists j \in \{0, 1, \dots, k\}$ s.t. $p_j = 1$, $IG(\Omega_t)$ reaches its minimum value of 0. However, due to the diverse nature of LLM training corpora, the LLM will not consistently provide the same direction. Therefore, by abstracting core components, the unspecificity (entropy) can decrease within the $(0, \log(k+1)]$ interval. \square

B THE SEARCH DIRECTIONS PRODUCED BY RP AND CAP

In this section, we present additional search directions produced by RP (Ye et al., 2024a) and CAP (our method) across various HG tasks, COPs and LLMs. Additionally, all produced unspecific search directions are highlighted in blue. For example, GPT-4o-mini frequently suggests the term “edge clustering”, when performing RP. This direction “edge clustering” is frequently applied in tasks like recommendation systems, where it helps identify patterns in user interactions and preferences. However, it is not commonly used in heuristic algorithms for solving COPs and is, therefore, considered unspecific.

810
811
812 **Direction 1: The produced search directions for deriving penalty heuristics to solve TSP**

813 # The LLM used to provide search directions is GPT-4o-mini.
814 **RP:**
815 Consider **edge_clustering**, incorporate **historical_edge_frequencies**, and adapt penalties dynamically based on
816 current path exploration.
817 **CAP:**
818 Focus on relative edge scoring, incorporate multiple factors like connectivity and distance, and enhance
819 normalization techniques.
820 # The LLM used to provide search directions is Llama-3-70b.
821 **RP:**
822 Normalize and **symmetrize** heuristics; consider **the_opposite** (not including an edge) for more effective
823 penalties.
824 **CAP:**
825 Focus on relative edge costs (e.g., proximity concept) rather than absolute deviations from average distance.

826
827
828 **Direction 2: The produced search directions for deriving constructive heuristics to solve TSP**

830 # The LLM used to provide search directions is GPT-3.5-turbo.
831 **RP:**
832 Use sorted future nodes by distance, limit look-ahead steps, and minimize total future score.
833 **CAP:**
834 Simplify heuristic by focusing on coefficients for better node selection efficiency.

836
837 **Direction 3: The produced search directions for deriving ACO heuristic measures to solve BPP**

839 # The LLM used to provide search directions is Llama3.1-405b.
840 **RP:**
841 Consider non-linear relationships between demand ratios and heuristics, and **experiment** with different
842 sparsification thresholds for better performance.
843 **CAP:**
844 Simplification and normalization of demand values can lead to more effective heuristics, reducing
845 computational complexity.

846
847 **Direction 4: The produced search directions for deriving ACO heuristic measures to solve MKP**

849 # The LLM used to provide search directions is Gemma2-27b.
850 **RP:**
851 Focus on item value density, consider weight imbalance across dimensions, and refine sparsity thresholds.
852 **CAP:**
853 Consider the interaction between dimensionality constraints and item value across dimensions.

854
855 **Direction 5: The produced search directions for reshaping attention scores of POMO to solve TSP**

857 # The LLM used to provide search directions is GPT-4o-mini.
858 **RP:**
859 Incorporate dynamic adjustment of K based on node density. Use heuristics from **successful_TSP_solutions** as
860 weight modifiers. Explore **edge_clustering** to reduce focus on distant nodes.
861 **CAP:**
862 Prioritize distance quantiles, and apply exponential decay for promising edges while suppressing undesirable
863 ones more effectively.

Direction 6: The produced search directions for reshaping attention scores of POMO to solve CVRP

```
# The LLM used to provide search directions is GPT-4o-mini.
RP:
Incorporate route_clustering, demand distribution analysis, and consider multi-vehicle interactions for
enhanced heuristics.
CAP:
Emphasize vectorization over loops for performance. Enhance demand penalties to better reflect capacity
constraints. Normalize distances effectively to balance contributions.
```

Direction 7: The produced search directions for reshaping attention scores of LEHD to solve TSP

```
# The LLM used to provide search directions is GPT-4o-mini.
RP:
Incorporate edge_connectivity to prioritize clusters. Consider spatial locality using coordinates for
refinement. Adaptively adjust weights based on current_solution_state.
CAP:
Use logarithmic scaling for distances, increase top-K selection, and implement normalization for better
convergence and stability.
```

Direction 8: The produced search directions for reshaping attention scores of LEHD to solve CVRP

```
# The LLM used to provide search directions is GPT-4o-mini.
# N=200
RP:
Utilize matrix operations for demand calculations to enhance efficiency. Introduce adaptive penalties based
on demand-to-capacity ratios. Explore additional features, like clustering of nodes, to improve routing
logic.
CAP:
Focus on vectorized operations, minimize nested loops, penalize exceeding capacity more effectively, and
integrate distance-demand balancing.
# N=500
RP:
Consider integrating real-time clustering and demand_forecasting for optimized routing. Explore adaptive
penalties and multi-objective criteria.
CAP:
Prioritize vectorized operations, minimize nested loops, reward feasible short connections, and enhance
penalties for exceeding capacities.
# N=1,000
RP:
Incorporate vehicle_utilization_metrics. Explore clustering techniques. Include demand prioritization based
on proximity. Optimize candidate edge selection dynamically. Use adaptive penalties for infeasible
edges. Consider adding multiple objectives in assessment.
CAP:
Incorporate vectorized calculations, normalize scores, and prioritize low-distance/high-demand paths for
improved efficiency and effectiveness.
```

C THE ADOPTED CROSSOVER AND ELITIST MUTATION OPERATORS

For consistency, we adopt the crossover and mutation operators from the prior study (Ye et al., 2024a) in all the experiments presented in this paper. Specifically, for the adopted crossover operator, two distinct parent heuristics are selected according to (5). Subsequently, the relative fitness values of these two heuristics determine which one serves as the primary learning exemplar for deriving an offspring heuristic. The employed mutation operator is elitist mutation, which derives multiple heuristics based on the historically best heuristic, aiming to produce high-performance ones. The prompting formats for both the crossover and elitist mutation operators, as well as the other promptings (e.g., CAP and PPP) used in this paper are shown in Appendix F.

D DETAILED HYPER-PARAMETERS AND EXPERIMENTAL SETUPS

Hyper-parameters In Table 7, we present the hyper-parameters of Hercules and Hercules-P. In addition, following the prior study (Ye et al., 2024a), the temperature of the LLM is added by 0.3 to enhance the diversity of the initial population.

Hardware We comprehensively evaluate the performance of all algorithms, using a computer equipped with an Intel(R) Xeon(R) W-2235 CPU.

Table 7: Parameters of Hercules and Hercules-P

Parameter	Value
LLM temperature	1
Population size N	15
CAP coefficients k, λ	5, 0.7
Maximum number of evaluations	100
Crossover rate	1
Mutation rate	0.5
ConS coefficients δ, α, β	0.1, 0.5, 0.8

To ensure a fair comparison, we adopt the parameter configurations of all seed functions (e.g., KGLS parameters) as specified in the prior study (Ye et al., 2024a), which also documented the definitions of all HG tasks used in this paper. In addition, for all experiments in this paper, we exploit the training and test datasets to derive well-performing heuristics and assess the final derived heuristics, respectively. Specifically, during the search process, the performance of heuristics on the training dataset determines their fitness values. The heuristic with the best performance on the training dataset is selected as the final derived heuristic. We then further assess the performance of all final derived heuristics on test datasets and report the experimental results in Section 4. In the following part of this section, we present the details of training datasets and test datasets of all HG tasks.

Generating Penalty Heuristics for Guided Local Search During the search process, the performance of newly produced heuristics is evaluated using a training dataset comprising the number of 20 TSP instances, each with 200 nodes. Subsequently, we assess the performance of the final derived heuristics on two test datasets and report the results. Both test datasets contain 64 TSP instances, but differ in node scale, with one consisting of 100-node instances and the other of 200-node instances. All instances in both training and test datasets are uniformly distributed.

Generating Constructive Heuristics During the search process, the performance of newly produced heuristics is evaluated on a training dataset comprising the number of 64 TSP instances, each with 50 nodes, following a uniform distribution. Subsequently, the performance of the final derived heuristics on TSPLIB instances is reported in Table 3.

Generating Heuristic Measures for Ant Colony Optimization For BPP, during the search process, the performance of heuristics is evaluated on the training dataset consisting of 30 instances with 500 items each. The three test datasets each consist of 1,000 instances, with 120, 500, and 1,000 items, respectively. The bin capacity across all instances is fixed at 150, and item sizes are uniformly sampled from the range $[20, 100]$.

For MKP, the training dataset includes 30 instances, each with 120 items. The three test datasets each consist of 1,000 instances, with 120, 500, and 1,000 items, respectively. Both item values and weights are uniformly sampled from the range $[0, 1]$.

Reshaping Attention Scores for Neural Combinatorial Optimization For TSP-POMO and CVRP-POMO tasks, during the search process, the performance of newly produced heuristics is evaluated on a training dataset comprising 64 instances, each with 200 nodes. Subsequently, we report the performance of the final derived heuristics on three test datasets of different scales, namely 200-node, 500-node, and 1,000-node scales. Each test dataset contains 64 instances. All instances are uniformly distributed. In addition, for CVRP-POMO, customer locations are uniformly sampled

within the unit square, and customer demands are drawn from the discrete set $\{1, 2, \dots, 9\}$, each vehicle’s capacity is set to 50, and the depot is centrally located in the unit square.

For the TSP-LEHD task, during the search process, the performance of newly produced heuristics is evaluated on a training dataset consisting of 64 instances, each with 200 nodes. Subsequently, we report the performance of the final derived heuristics on three test datasets, namely 200-node, 500-node, and 1,000-node datasets, each containing 64 instances. Both the training and test datasets are sourced from (Luo et al., 2023). For the CVRP-LEHD task, following the prior study (Ye et al., 2024a), we apply LLM-based HG algorithms to derive heuristics for three training datasets, corresponding to problem sizes of $n = 200, 500,$ and $1,000,$ respectively. Subsequently, we assess these final derived heuristics on the corresponding scale test datasets and report the experimental results. The training dataset for $n = 200$ consists of 64 instances, while those for $n = 500$ and $n = 1,000$ contain 32 instances each. All test datasets consist of 64 instances. In addition, all the training and test datasets are sourced from (Luo et al., 2023).

E SEARCH TIME COMPARISON OF DIVERSE LLM-BASED HG ALGORITHMS

In Table 8, we present the search time of different LLM-based HG algorithms across diverse NCO tasks. As shown in Table 8, Hercules-P outperforms the other LLM-based HG algorithms in terms of search time, while Random ranks at the second place. On these NCO tasks, Hercules-P reduce the search time by 48%, 7%, 31%, 27%, 38%, and 59%, respectively, when compared to Hercules. This reduction in search time is especially significant for large-scale COPs, where search can extend to several hours. In these cases, incorporating PPP demonstrates highly effective in reducing the resource expenditure.

Table 8: Search time comparison of different LLM-based HG algorithms on diverse HG tasks

	Algorithm	Random	EOH (ICML’24)	ReEvo (NeurIPS’24)	Hercules-P (ours)	Hercules (ours)
Time (m)	TSP-POMO	<u>15.95</u>	18.17	17.89	11.50	22.12
	CVRP-POMO	<u>16.86</u>	30.54	29.57	9.51	10.28
	TSP-LEHD	<u>30.58</u>	39.55	37.25	28.72	41.43
	CVRP-LEHD ($n = 200$)	<u>45.73</u>	67.27	61.58	31.20	42.80
	CVRP-LEHD ($n = 500$)	<u>149.31</u>	224.01	215.61	110.28	178.01
	CVRP-LEHD ($n = 1,000$)	<u>639.83</u>	854.25	854.71	310.98	757.67

F PROMPTS USED IN HERCULES AND HERCULES-P

Prompts used for Hercules or Hercules-P can be categorized as problem-specific prompts and general prompts. This section provides a detailed overview of the used general prompts, while problem-specific prompts (including the heuristic description, COP description, seed function, and function signature) are documented in the prior study (Ye et al., 2024a).

Prompt 9: System prompt for elitist mutation and crossover operators.

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

#### Prompt 10: System prompt for abstracting core components.

```
You are an expert in the domain of automatic heuristics algorithm design. Your task is to give some hints for Large Language Model evolutionary framework to evolve better heuristic methods.
```

1026  
1027  
1028  
1029  
1030  
1031  
1032  
1033  
1034  
1035  
1036  
1037  
1038  
1039  
1040  
1041  
1042  
1043  
1044  
1045  
1046  
1047  
1048  
1049  
1050  
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

**Prompt 11: System prompt for providing search directions.**

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

**Prompt 12: System prompt for predicting heuristic performance.**

You are an expert in the domain of heuristics evaluation. Your task is to predict the performance of heuristics.

**Prompt 13: User prompt for population initialization.**

```
{task_description}
{seed_function}

Refer to the format of a trivial design above. Be very creative and give `{func_name}_v2`. Output code only and
enclose your code with Python code block: ``python ... ``.
```

**Prompt 14: User prompt for abstracting core components.**

```
The {func_name} function is a part of {alg} for solving {pro}.
{func_desc}

Below are five {func_name} functions:
{code_0}
{code_0}

{code_1}
{code_1}

{code_2}
{code_2}

{code_3}
{code_3}

{code_4}
{code_4}

Summarize the key code components of these functions that potentially influence the effectiveness and
performance of the algorithm, using less than 200 words.
```

**Prompt 15: User prompt for providing short-term search directions.**

```
Below are two {func_name} functions for {problem_desc}
{func_desc}

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

[Worse code]
{worse_code}

[Better code]
{better_code}

Below are some core components of the previous {func_name} functions.

{component}
{component}

Reflect about why the second code performs better than the first, considering the core components.
Only output some hints on designing better {func_name} functions base your reflections, using less than
20 words.
```

1080

1081

1082

**Prompt 16: User prompt for providing long-term search directions.**

1083

1084

```
Below is your prior long-term search directions on designing heuristics for {problem_desc}
{prior_direction}
```

1085

1086

```
Below are some newly gained insights.
{new_direction}
```

1087

1088

1089

```
Below are some core components of the previous {func_name} functions.
```

```
{component}
{component}
```

1090

1091

```
Write constructive hints for designing better heuristics, based on prior search directions, new insights, and
the core components, using less than 50 words.
```

1092

1093

1094

1095

**Prompt 17: User prompt for crossover.**

1096

1097

1098

1099

1100

1101

1102

1103

1104

1105

```
{task_description}

[Worse code]
{function_signature0}
{worse_code}

[Better code]
{function_signature1}
{better_code}

[direction]
{short_term_direction}

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

1106

1107

1108

1109

**Prompt 18: User prompt for elitist mutation.**

1110

1111

1112

1113

1114

1115

1116

```
{task_description}

[Prior direction]
{long-term_direction}

[Code]
{function_signature1}
{elitist_code}

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

1117

1118

1119

**Prompt 19: User prompt for predicting heuristic performance.**

1120

1121

1122

1123

1124

1125

1126

1127

1128

1129

1130

1131

1132

1133

```
The {func_name} function is a part of {alg}, which is used to solve {pro}.
{func_desc}
Here are some example codes and their corresponding performance scores that you can refer to for
predicting heuristic functions:
[example code 0]
{code_0}
[performance score of example code 0]
{score_0}

[example code 1]
{code_1}
[performance score of example code 1]
{score_1}

[example code 2]
{code_2}
[performance score of example code 2]
{score_2}

[example code 3]
{code_3}
[performance score of example code 3]
{score_3}

```

```

1134 [example code 4]
1135 {code_4}
1136 [performance score of example code 4]
1137 {score_4}
1138 ---
1139 [example code 5]
1140 {code_5}
1141 [performance score of example code 5]
1142 {score_5}
1143 ---
1144 [example code 6]
1145 {code_6}
1146 [performance score of example code 6]
1147 {score_6}
1148 ---
1149 [example code 7]
1150 {code_7}
1151 [performance score of example code 7]
1152 {score_7}
1153 ---
1154 [example code 8]
1155 {code_8}
1156 [performance score of example code 8]
1157 {score_8}
1158 ---
1159 [example code 9]
1160 {code_9}
1161 [performance score of example code 9]
1162 {score_9}
1163 ---
1164 Here are some codes that you need to predict:
1165 {code_10}
1166 {code_10}
1167 ---
1168 {code_11}
1169 {code_11}
1170 ---
1171 {code_12}
1172 {code_12}
1173 ---
1174 {code_13}
1175 {code_13}
1176 ---
1177 {code_14}
1178 {code_14}
1179 ---
1180 {code_15}
1181 {code_15}
1182 ---
1183 {code_16}
1184 {code_16}
1185 ---
1186 {code_17}
1187 {code_17}
1188 ---
1189 {code_18}
1190 {code_18}
1191 ---
1192 {code_19}
1193 {code_19}
1194 Predict the performance of the above codes by comparing their semantic meanings with the produced example
1195 codes. Provide a performance score and a confidence number based on your evaluation for each code. The
1196 performance score should be a float within the range [{score_0}, {score_1}], where a lower score indicates a
1197 better-performing heuristic. The confidence number should be a float within the range [0,1], indicating how
1198 similar the semantics of the code is to the most similar example code. Note that you can only give a confidence
1199 level = 1 if the code is semantically identical to the produced example code. Output only the performance score
1200 and confidence number of these codes that need to be predicted, strictly adhering to the following format. No
1201 other words and punctuation should be included in the output.
1202 '''code_10: score, confidence,
1203 code_11: score, confidence,
1204 code_12: score, confidence,
1205 code_13: score, confidence,
1206 code_14: score, confidence,
1207 code_15: score, confidence,
1208 code_16: score, confidence,
1209 code_17: score, confidence,
1210 code_18: score, confidence,
1211 code_19: score, confidence'''
1212
1213
1214
1215
1216
1217
1218
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1279
1280
1281
1282
1283
1284
1285
1286
1287
1288
1289
1290
1291
1292
1293
1294
1295
1296
1297
1298
1299
1300

```

## G HIGH-PERFORMANCE HEURISTICS PRODUCED BY HERCULES

In this section, we present the best heuristics produced by Hercules for all tasks.

**Heuristic 1:** The high-performance GLS heuristic produced by Hercules using GPT-4o-mini for solving TSP.

```

1194 def heuristic(distance_matrix: np.ndarray) -> np.ndarray:
1195 n = distance_matrix.shape[0]
1196 heuristics_scores = np.zeros_like(distance_matrix)
1197
1198 # Penalty function for edges based on both distance and connectivity
1199 for i in range(n):
1200 for j in range(n):
1201 if i != j:
1202 # Increase penalty for longer distances
1203 heuristics_scores[i, j] = distance_matrix[i, j]
1204
1205 # Reduce score if this edge is part of a densely connected horizon
1206 connections = np.sum(distance_matrix[i] < distance_matrix[i, j]) - 1 # excluding itself
1207 heuristics_scores[i, j] *= (1 + connections * 0.1) # penalizing connected edges more
1208
1209 return heuristics_scores

```

**Heuristic 2:** The high-performance constructive heuristic produced by Hercules using GPT-3.5-turbo for solving TSP.

```

1207 def heuristic(current_node: int, destination_node: int, unvisited_nodes: set, distance_matrix: np.ndarray) ->
1208 int:
1209 """Select the next node to visit from the unvisited nodes with look-ahead mechanism."""
1210
1211 def calculate_look_ahead_score(node, next_node):
1212 if len(unvisited_nodes) == 1:
1213 return 0
1214 lookahead_nodes = unvisited_nodes - {next_node}
1215 min_lookahead_distance = min([distance_matrix[node][i] for i in lookahead_nodes if i != node])
1216 return -0.1 * min_lookahead_distance
1217
1218 c1, c2, c3, c4 = 0.4, 0.3, 0.2, 0.1
1219 scores = {}
1220 for node in unvisited_nodes:
1221 all_distances = [distance_matrix[node][i] for i in unvisited_nodes if i != node]
1222 average_distance_to_unvisited = np.mean(all_distances)
1223 std_dev_distance_to_unvisited = np.std(all_distances)
1224 lookahead_score = calculate_look_ahead_score(current_node, node)
1225 score = c1 * distance_matrix[current_node][node] - c2 * average_distance_to_unvisited + c3 *
1226 std_dev_distance_to_unvisited - c4 * distance_matrix[destination_node][node] + lookahead_score
1227 scores[node] = score
1228 next_node = min(scores, key=scores.get)
1229 return next_node

```

**Heuristic 3:** The high-performance ACO heuristic measure produced by Hercules using Llama3.1-405b for solving BPP.

```

1224 def heuristic(demand: np.ndarray, capacity: int) -> np.ndarray:
1225 """
1226 This function calculates the heuristics for the Bin Packing Problem (BPP).
1227 Parameters:
1228 demand (np.ndarray): A 1D array representing the sizes of the items.
1229 capacity (int): The capacity of each bin.
1230 Returns:
1231 np.ndarray: A 2D array where heuristics[i][j] represents how promising it is to put item i and item j in
1232 the same bin.
1233 """
1234 # Calculate the complementarity of each pair of items
1235 # The complementarity is the difference between the capacity and the sum of the demands of the two items
1236 complementarity = capacity - np.add.outer(demand, demand)
1237
1238 # Apply exponential decay to the complementarity values
1239 # This reduces the dominance of large values and emphasizes the importance of small values
1240 decayed_complementarity = np.exp(-complementarity / capacity)
1241
1242 # Normalize the demand values to be between 0 and 1
1243 normalized_demand = demand / demand.max()
1244
1245 # Calculate the heuristic value for each pair of items
1246 # The heuristic value is the product of the normalized demands and the decayed complementarity
1247 heuristics = np.outer(normalized_demand, normalized_demand) * decayed_complementarity
1248
1249 # Sparsify the matrix by setting unpromising elements to zero
1250 # Here, we consider elements with a value less than 0.5 as unpromising
1251 heuristics[heuristics < 0.5] = 0
1252
1253 return heuristics

```

1242 Heuristic 4: The high-performance ACO heuristic measure produced by Hercules using Gemma2-  
1243 27b for solving MKP.

```
1244 def heuristic(prize: np.ndarray, weight: np.ndarray) -> np.ndarray:
1245 prize_per_unit_weight = prize / np.sum(weight, axis=1)
1246 max_weight_ratios = np.max(weight / np.expand_dims(np.sum(weight, axis=1), axis=1), axis=1)
1247 density_score = prize_per_unit_weight * (1 - max_weight_ratios)
1248
1249 # Weight Magnitude Awareness
1250 weight_magnitude = np.sum(weight, axis=1)
1251 magnitude_bonus = np.exp(-weight_magnitude / np.max(weight_magnitude))
1252
1253 # Distribution Awareness with Adaptive IQR
1254 density_percentile_75 = np.percentile(density_score, 75)
1255 density_percentile_25 = np.percentile(density_score, 25)
1256 iqr = density_percentile_75 - density_percentile_25
1257 adaptive_iqr_window = 0.3 * iqr
1258 distribution_factor = np.where(density_score > density_percentile_75, 1.2,
1259 np.where(density_score > density_percentile_75 - adaptive_iqr_window, 1, 0.5))
1260
1261 # Dimensionality-Weighted Density Scores (Tighter Coupling and Exponent Tuning)
1262 dimensionality_weights = np.sum(weight > 0, axis=1) / weight.shape[1]
1263 dimensionality_bonus = density_score ** (1 + dimensionality_weights * 2)
1264
1265 # Sparsity Penalty
1266 sparsity_penalty = np.where(np.sum(weight > 0, axis=1) < weight.shape[1], 1.2, 1)
1267
1268 heuristics = density_score * magnitude_bonus * distribution_factor * dimensionality_bonus *
1269 sparsity_penalty
1270 heuristics[heuristics < np.percentile(heuristics, 5)] = 0
1271
1272 return heuristics
```

1262 Heuristic 5: The high-performance POMO heuristic produced by Hercules using GPT-4o-mini for  
1263 solving TSP.

```
1264 def heuristic(distance_matrix: torch.Tensor) -> torch.Tensor:
1265 """
1266 heuristics computes a refined heuristic for TSP based on the distance matrix by evaluating edges
1267 and applying adaptive, non-linear transformations for better edge prioritization.
1268 The heuristic incorporates clustering dynamics and balances exploration-exploitation strategies.
1269 """
1270 distance_matrix[distance_matrix == 0] = 1e5
1271 K = 5 # Top-K nearest neighbors for refined edge selection
1272 alpha = 0.9 # Increased weight for promoting close edges
1273 beta = 0.1 # Reduced weighting factor for penalizing distant edges
1274 epsilon = 1e-5 # Small constant to prevent division by zero
1275
1276 # Start with heuristic values based on a transformation of the distance matrix
1277 heu = -distance_matrix.clone()
1278
1279 # Find the top-K nearest neighbors
1280 _, indices = torch.topk(distance_matrix, k=K, largest=False, dim=1)
1281
1282 # Create masks for top-K edges
1283 topk_mask = torch.zeros_like(distance_matrix, dtype=torch.bool)
1284 topk_mask.scatter_(1, indices, True)
1285
1286 # Adaptive transformations on selected edges with logarithmic weighting
1287 transformation_term = -alpha * torch.log(1 + distance_matrix[topk_mask])
1288 penalty_term = beta * (1 / (distance_matrix[topk_mask] + epsilon))
1289
1290 # Combine results for top-K and retain default penalties elsewhere
1291 heu[topk_mask] = transformation_term + penalty_term
1292
1293 # Employ edge clustering insights by grouping nearly equal distances
1294 distance_mean = distance_matrix.mean(dim=1, keepdim=True)
1295 distance_std = distance_matrix.std(dim=1, keepdim=True)
1296 cluster_mask = torch.abs(distance_matrix - distance_mean) < distance_std
1297
1298 # Apply a refinement for edges within the same cluster with increased adjustment
1299 heu[cluster_mask] += 0.3 # Increased favor for edges within the same cluster
1300
1301 # Additional adjustment for edges based on their proximity to the mean distance
1302 solution_proximity = distance_matrix.mean() # Example proximity metric
1303 adjustment_term = heu - (distance_matrix - solution_proximity)
1304 heu += adjustment_term * 0.15 # Slightly refine penalties based on distance to the mean solution proximity
1305
1306 return heu
```

1291 Heuristic 6: The high-performance POMO heuristic produced by Hercules using GPT-4o-mini for  
1292 solving CVRP.

```
1293 def heuristic(distance_matrix: torch.Tensor, demands: torch.Tensor) -> torch.Tensor:
1294 """Enhanced adaptive heuristic function for CVRP with refined scoring aggregation and weight parameters."""
1295
1296 # Total vehicle capacity, normalized to the highest demand
1297 vehicle_capacity = demands.max()
```



```

1296
1297 # Initialize distance scores (negative for minimization)
1298 distance_scores = -distance_matrix.clone()
1299
1300 # Compute combined demand interactions with broadcasting
1301 demand_matrix = demands.unsqueeze(1) + demands.unsqueeze(0) # Shape (n, n)
1302
1303 # Identify edges exceeding vehicle capacity
1304 exceeding_capacity_mask = demand_matrix > vehicle_capacity
1305
1306 # Calculate demand scores with adaptive penalties and strong incentives for valid demands
1307 demand_scores = torch.where(
1308 exceeding_capacity_mask,
1309 -5 * (demand_matrix - vehicle_capacity) ** 2, # Higher penalty for exceeding capacity
1310 3 * (vehicle_capacity - demand_matrix) # Incentive for satisfying demands
1311)
1312
1313 # Combine distance and demand scores with an aggregation weight
1314 alpha = 0.7 # Weight for distance scoring
1315 beta = 0.3 # Weight for demand scoring
1316 combined_scores = alpha * distance_scores + beta * demand_scores
1317
1318 # Normalize combined scores for consistent indicator range
1319 combined_scores_normalized = (combined_scores - combined_scores.min()) / (combined_scores.max() -
1320 combined_scores.min() + 1e-10)
1321
1322 return combined_scores_normalized

```

1313 **Heuristic 7: The high-performance LEHD heuristic produced by Hercules using GPT-4o-mini for**  
1314 **solving TSP.**

```

1315 def heuristic(distance_matrix: torch.Tensor) -> torch.Tensor:
1316 """
1317 Improved heuristics for the TSP utilizing adaptive thresholds, robust statistical measures,
1318 and dynamic edge scoring systems to enhance edge desirability evaluation.
1319 """
1320 distance_matrix[distance_matrix == 0] = 1e5
1321 N = distance_matrix.size(0)
1322
1323 # Calculate mean and robust median as a central tendency measure
1324 mean_distances = distance_matrix.mean(dim=1, keepdim=True)
1325 median_distances = distance_matrix.median(dim=1, keepdim=True).values
1326
1327 # Calculate edge scores based on how far they deviate from both mean and median
1328 deviations_from_mean = -(distance_matrix - mean_distances) / (mean_distances + 1e-5)
1329 deviations_from_median = -(distance_matrix - median_distances) / (median_distances + 1e-5)
1330
1331 # Initialize heuristic scores with a combination of deviations
1332 heuristics_scores = (deviations_from_mean + deviations_from_median) / 2
1333
1334 # Apply dynamic proximity boosts for edges that are closer than a weighted threshold
1335 dynamic_threshold = 0.5 * (mean_distances + median_distances)
1336 proximity_boosts = torch.where(distance_matrix <= dynamic_threshold,
1337 (1 / N * dynamic_threshold - distance_matrix).clamp(min=0),
1338 torch.tensor(0.0, device=distance_matrix.device))
1339
1340 # Update heuristic scores with proximity boosts
1341 heuristics_scores += proximity_boosts
1342
1343 return heuristics_scores

```

1333 **Heuristic 8: The high-performance LEHD heuristic produced by Hercules using GPT-4o-mini for**  
1334 **solving CVRP.**

```

1335 #N=200
1336 def heuristic(distance_matrix: torch.Tensor, demands: torch.Tensor) -> torch.Tensor:
1337 """An improved heuristic implementation for the Capacitated Vehicle Routing Problem (CVRP) with refined
1338 dynamic penalties and transformations."""
1339
1340 vehicle_capacity = 1.0 # Normalize demands with respect to maximum capacity
1341 num_customers = demands.shape[0]
1342
1343 # Create a matrix for combined demand
1344 demand_matrix = demands.unsqueeze(1) + demands.unsqueeze(0) # Shape: [n, n]
1345
1346 # Create a mask for viable connections based on vehicle capacity
1347 is_viable = (demand_matrix <= vehicle_capacity).float()
1348
1349 # Compute distance scores, avoiding self-distances by adding a large penalty
1350 distance_scores = 1 / (distance_matrix + torch.eye(num_customers) * 1e6)
1351
1352 # Calculate promising indicators
1353 promising_indicators = is_viable * distance_scores
1354
1355 # Dynamic penalties based on excess demand
1356 excess_demand_penalty = (demand_matrix - vehicle_capacity).clamp(min=0)
1357 penalty_factor = excess_demand_penalty ** 2 / (vehicle_capacity ** 2 + 1e-6)
1358 promising_indicators -= penalty_factor * (distance_scores * 2 - 1)

```

```

1350 # Clustering for improved route planning with a more responsive threshold
1351 cluster_threshold = 0.3 # Adaptive threshold for clustering based on distance
1352 clusters = (distance_matrix < cluster_threshold).float()
1353 promising_indicators *= clusters
1354
1355 # Normalize scores to range between -1 and 1
1356 min_value = promising_indicators.min()
1357 max_value = promising_indicators.max()
1358
1359 if max_value != min_value:
1360 promising_indicators = (promising_indicators - min_value) / (max_value - min_value) * 2 - 1
1361
1362 # Enhance promising connections via a non-linear transformation
1363 promising_indicators = promising_indicators ** 3 * torch.sign(promising_indicators + 1e-6) # Added epsilon
1364 # for stability
1365
1366 return promising_indicators
1367
1368 #N=500
1369 def heuristic(distance_matrix: torch.Tensor, demands: torch.Tensor) -> torch.Tensor:
1370 """Enhanced heuristic implementation for Capacitated Vehicle Routing Problem that evaluates edge
1371 desirability."""
1372 num_customers = demands.shape[0]
1373 vehicle_capacity = 1.0 # Normalized capacity
1374
1375 # Initialize cost matrix
1376 cost_matrix = distance_matrix.clone()
1377
1378 # Calculate total demand and initialize demand density
1379 demand_density = demands / demands.sum()
1380 total_demand_matrix = demands.unsqueeze(1) + demands.unsqueeze(0)
1381
1382 # Calibrated penalties for demand violation
1383 penalties = (total_demand_matrix > vehicle_capacity).float() * 3.0 # Increased penalties for more emphasis
1384
1385 # Evaluate edge desirability based on demand compatibility and distance
1386 mask_compatible = total_demand_matrix <= vehicle_capacity
1387 mask_incompatible = total_demand_matrix > vehicle_capacity
1388
1389 # Adjust cost matrix based on compatibility and added penalties
1390 cost_matrix[1:, 1:] = torch.where(mask_compatible[1:, 1:], -distance_matrix[1:, 1:], distance_matrix[1:,
1391 1:] * penalties[1:, 1:])
1392
1393 # For depot connections, favorably adjust edges
1394 cost_matrix[0, 1:] = -distance_matrix[0, 1:] * 0.5 # Strongly favor depot-to-customer
1395 cost_matrix[1:, 0] = -distance_matrix[1:, 0] * 0.5 # Strongly favor customer-to-depot
1396
1397 # Return normalized desirability
1398 return cost_matrix
1399
1400 #N=1,000
1401 def heuristic(distance_matrix: torch.Tensor, demands: torch.Tensor) -> torch.Tensor:
1402 n = distance_matrix.shape[0]
1403 vehicle_capacity = 1.0 # normalized vehicle capacity
1404 heuristic_scores = torch.zeros_like(distance_matrix)
1405
1406 # Create a mask for valid edges based on capacity constraints (non-self-loops)
1407 demand_within_capacity = (demands.unsqueeze(1) + demands.unsqueeze(0) <= vehicle_capacity) & (
1408 distance_matrix != 0)
1409
1410 # Calculate effective distance score
1411 effective_distances = torch.where(distance_matrix > 0, 1.0 / (distance_matrix + 1e-6), torch.zeros_like(
1412 distance_matrix))
1413
1414 # Initialize promising edges
1415 heuristic_scores[demand_within_capacity] = effective_distances[demand_within_capacity]
1416
1417 # Assign stronger penalties for infeasible edges
1418 heuristic_scores[~demand_within_capacity] = -200.0 # Strong penalty for infeasible edges
1419
1420 # Scale scores for promising paths using min-max normalization
1421 positive_scores = heuristic_scores[heuristic_scores > 0]
1422
1423 if positive_scores.numel() > 0:
1424 min_positive = positive_scores.min()
1425 max_positive = positive_scores.max()
1426
1427 # Normalize to [0, 1]
1428 heuristic_scores[heuristic_scores > 0] = (heuristic_scores[heuristic_scores > 0] - min_positive) / (
1429 max_positive - min_positive)
1430
1431 # Apply additional penalties based on demand
1432 demand_excess = demands.unsqueeze(1) - vehicle_capacity
1433 demand_excess[demand_excess < 0] = 0 # No penalty for nodes within capacity
1434 heuristic_scores -= demand_excess * 15.0 # Apply strong penalty for edges leading to high demand
1435
1436 return heuristic_scores

```