HIFO-PROMPT: PROMPTING WITH HINDSIGHT AND FORESIGHT FOR LLM-BASED AUTOMATIC HEURISTIC DESIGN

Anonymous authors
Paper under double-blind review

ABSTRACT

This paper investigates the application of Large Language Models (LLMs) in Automated Heuristic Design (AHD), where their integration into evolutionary frameworks reveals a significant gap in global control and long-term learning. We propose the Hindsight-Foresight Prompt (HiFo-Prompt), a novel framework for LLM-based AHD designed to overcome these limitations. This is achieved through two synergistic strategies: **Foresight** and **Hindsight**. Foresight acts as a high-level meta-controller, monitoring population dynamics(e.g., stagnation and diversity collapse) to switch the global search strategy between exploration and exploitation explicitly. Hindsight builds a persistent knowledge base by distilling successful design principles from past generations, making this knowledge reusable. This dual mechanism ensures that the LLM is not just a passive operator but an active reasoner, guided by a global plan (Foresight) while continuously improving from its cumulative experience (Hindsight). Empirical results demonstrate that HiFo-Prompt significantly outperforms a comprehensive suite of state-of-the-art AHD methods, discovering higher-quality heuristics with substantially improved convergence speed and query efficiency.

1 INTRODUCTION

Combinatorial Optimization (CO) problems, which involve finding an optimal solution from a discrete set of possibilities, are ubiquitous in science and engineering. Because of their NP-hardness, designing effective heuristics for these problems is a complex work, traditionally based on extensive human experience and intuition (Camacho-Villalón et al., 2023).

The advent of Large Language Models (LLMs) has catalyzed a paradigm shift toward Automated Heuristic Design (AHD) (Wang & Chen, 2023; Liu et al., 2024c). A particularly potent approach marries LLMs with Evolutionary Computation (EC), casting the LLM as a high-level semantic mutation operator. Foundational works such as FunSearch (Romera-Paredes et al., 2024) and EoH (Liu et al., 2024b) established the viability of this LLM+EC paradigm, demonstrating its capacity to discover novel and effective heuristics.

However, as the field progresses, two fundamental challenges have emerged in AHD: *the inability to steer the heuristic generation process based on population dynamics* and *the failure to distill and manage the core design principles of high-performance heuristics to guide the subsequent heuristic generation process*.

First, many approaches lack a mechanism for global adaptive guidance. They often rely on local or reactive signals; for instance, ReEvo (Ye et al., 2024) performs reflection on a single candidate, while methods such as MCTS-AHD (Zheng et al., 2025) passively embed the exploration-exploitation trade-off within their search structure. This localized control does not respond to the macroscopical dynamics of the population and cannot proactively intervene when the search encounters systemic issues such as premature convergence or a decline in diversity. A more aggressive strategy involves in-weight adaptation, where methods such as EvoTune (Šurina et al., 2025) and CALM (Huang et al., 2025) use numerical gradients to fine-tune the LLM's parameters. Although powerful, this approach is computationally intensive and treats the LLM as an opaque policy network, sacrificing the interpretability of high-level symbolic reasoning.

Second, existing frameworks across the board suffer from poor knowledge persistence. Successful heuristic design experiences, the valuable "genes" of the evolutionary process, often remain entangled within their specific code implementations. When parent candidates are discarded, these insights are lost, a phenomenon we term knowledge decay. Other high-level concepts, such as evolving the optimizer itself in MoH (Shi et al., 2025) or reshaping the problem in RedAHD (Thach et al., 2025), are orthogonal and do not address this core issue of knowledge loss. Ultimately, this flaw prevents the system from achieving cumulative learning, forcing it to perpetually rediscover similar concepts rather than building on a foundation of proven wisdom.

To overcome these fundamental limitations, we propose **HiFo-Prompt (Hindsight-Foresight Prompt)**, a framework that establishes a hierarchical control architecture for LLM-based AHD. HiFo-Prompt elevates the LLM from a mere code generator to a symbolic meta-optimizer by endowing it with two synergistic capabilities: First, the **Foresight** module addresses the control problem by serving as a meta-controller(**Evolutionary Navigator**) that observes population dynamics. Upon detecting states like performance plateaus, it explicitly modulates the generative process by switching evolutionary regimes via lightweight, interpretable *verbal gradients* injected at the prompt level—a symbolic alternative to opaque and expensive numerical gradients. Second, the Hindsight module directly tackles knowledge decay by implementing the **Insight Pool**, a persistent and evolving repository of distilled algorithmic knowledge. It distills valuable heuristic generation strategies from their specific code implementations, transforming them into abstract, reusable design principles. This institutional memory allows the system to build upon past successes, effectively seeding subsequent generations with proven wisdom. In summary, the contributions of our approach are as follows:

- We introduce HiFo-Prompt, a novel framework consisting of Hindsight and Foresight modules. It dynamically generates prompts for LLMs by decoupling thoughts from code, thereby enabling independent updates and evaluations. This mechanism leads to a significant reduction in both training time for heuristics and evaluation costs for LLMs.
- To improve the Hindsight and Foresight abilities of our method, we introduce the Insight Pool and Evolutionary Navigator, respectively. The Insight Pool accumulates knowledge from high-performing codes through iterative updates. The Evolutionary Navigator controls population states by monitoring evolution and balancing exploration-exploitation dynamics.
- We evaluated the heuristics designed by HiFo-Prompt on complex optimization tasks, comparing them against advanced handcrafted heuristics and existing AHD approaches. Our results achieve state-of-the-art performance in the AHD domain, with substantial improvements over prior AHD methods, particularly excelling in the Traveling Salesman Problem (TSP) and Flow Shop Scheduling Problem (FSSP).

2 Related Work

LLM-driven Automatic Heuristic Design The integration of Large Language Models (LLMs) into Evolutionary Computation (EC) is a vibrant new direction for Automated Heuristic Design (AHD) (Liu et al., 2024a; Chauhan et al., 2025). Pioneered by works like FunSearch (Romera-Paredes et al., 2024) and EoH (Liu et al., 2024b), this paradigm leverages the LLM as a powerful semantic operator to generate heuristics as code. Recent efforts to advance this paradigm can be categorized along several axes. Some works focus on refining search control through sophisticated prompt engineering and guidance mechanisms (Ye et al., 2024; Dat et al., 2025), or by redesigning the population structure itself (Zheng et al., 2025). A distinct, more model-centric approach directly adapts the LLM's parameters via reinforcement learning-based fine-tuning (Šurina et al., 2025; Huang et al., 2025). At the highest level of abstraction, research has also explored evolving other core components of the optimization process, such as the optimizer (Shi et al., 2025) or the problem representation (Thach et al., 2025).

Knowledge Management in Generative Search Harnessing historical information is a cornerstone of efficient search. In classical EC, methods like Cultural Algorithms (Maheri et al., 2021) formalize this via a structured Belief Space that stores and evolves collective knowledge. Contemporary LLM-based approaches often rely on in-context "reflection" mechanisms (Shinn et al., 2023;

Bo et al., 2024), where the model self-critiques failures to inform its next attempt. However, this knowledge is typically transient, unstructured, and instance-specific. Consequently, these methods lack a mechanism for accumulating and generalizing insights over time, preventing the formation of a persistent, structured knowledge base analogous to those in classical EC.

Adaptive Control in Evolutionary Computation Dynamically adapting search strategies is a long-standing goal in EC. Historically, this has been addressed through low-level, reactive mechanisms like Adaptive Operator Selection (AOS) (Álvaro Fialho, 2010; Tian et al., 2023) and parameter control (Eiben & Smith, 2015; Aleti & Moser, 2016). These methods rely on numerical credit assignment, effectively voting for strategies that recently performed well, but they lack a semantic understanding of the search dynamics (e.g., identifying population stagnation). The reasoning capabilities of LLMs offer a paradigm shift towards higher-level, proactive control (Eiben et al., 1999; Papa, 2021). Instead of merely adjusting parameters, LLMs can interpret population-level statistics to suggest symbolic actions, such as "increase mutation rate to escape a local optimum." This marks a transition from fine-grained numerical tuning to semantic-based strategic adjustment.

3 METHODOLOGY

In this section, we introduce our proposed HiFo-Prompt, a novel framework that equips the evolutionary process driven by LLM with mechanisms for learning and adaptation (shown in Figure 1). HiFo-Prompt integrates two synergistic components: a Foresight module for real-time adaptive control, which monitors evolutionary dynamics to steer the search strategy, and a Hindsight module for long-term knowledge accumulation, which manages a self-evolving repository of successful design principles that we term insights. By uniting foresight-driven strategy with hindsight-informed knowledge, our framework transforms the generative process into a robust, self-regulating system.

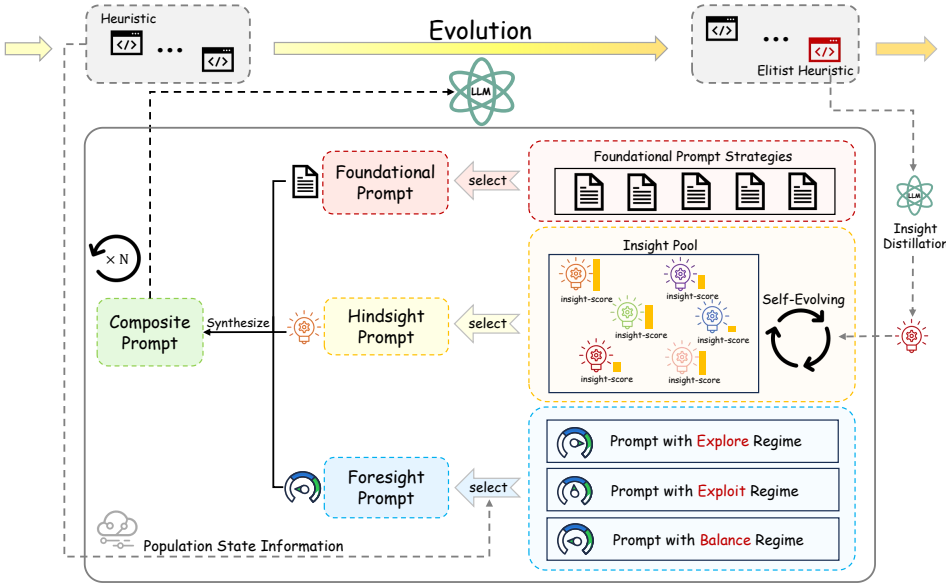

Figure 1: The framework of HiFo-Prompt. **Left**: Prompt Construction builds upon a foundational prompt, augmenting it with strategic Foresight and empirical Hindsight to form the final composite prompt. **Right**: Knowledge Evolution forms a learning loop where elite heuristics are distilled into new insights to continuously enrich the Hindsight knowledge base.

3.1 GUIDED PROMPT SYNTHESIS FOR AHD

Our HiFo-Prompt framework applies a Guided Prompt Synthesis mechanism that constructs each prompt as a context-aware composite instruction. This mechanism integrates three interlocking

modules: Foundational prompt strategies, hint knowledge, and a foresight-driven strategy. The resulting composite prompt provides precise, multifaceted guidance to the LLM. A complete example of prompt generation for TSP with step-by-step construction is provided in Figure 2.

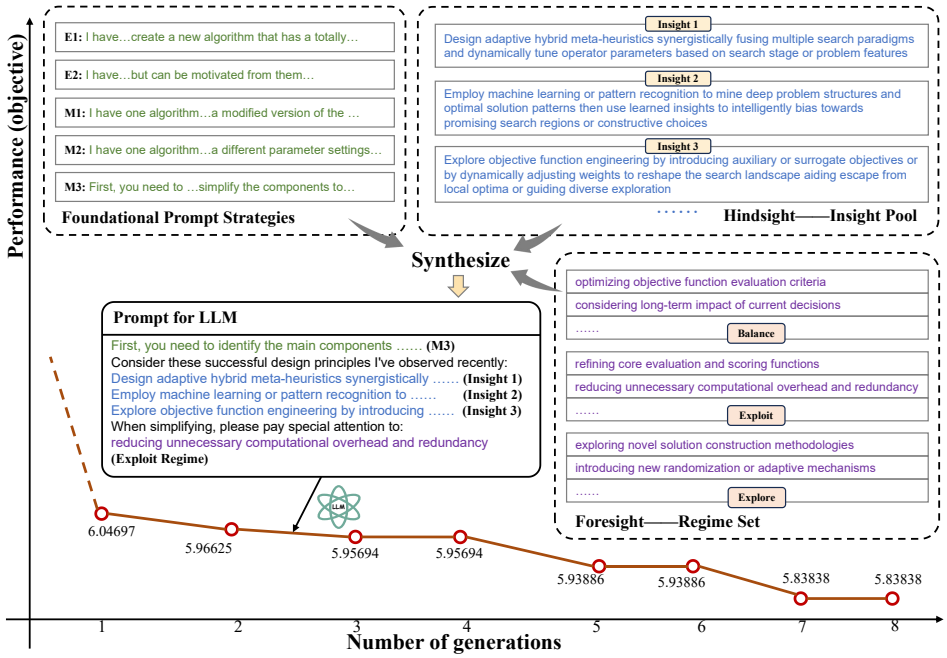

Figure 2: Dynamic prompt generation process of HiFo-Prompt.

Foundational prompt strategies. Our framework's generative foundation is a set of Foundational Prompt Strategies, which function as the LLM-equivalent of genetic operators. We first generate heuristics from scratch using the initial prompt strategy **I1**, then evolve them with five foundational prompts adapted from EoH Liu et al. (2024b). These prompts are organized into two primary strategies: 1) **Reorganization Strategies**, which include **E1**, synthesizing a new algorithm with a novel structure from multiple parents, and **E2**, abstracting shared core ideas to generate conceptually distinct variants; and 2) **Mutation Strategies**, which encompass **M1**, making structural modifications for functionally equivalent variants; **M2**, tuning critical parameters; and **M3**, simplifying components prone to overfitting. While this curated set provides the raw generative capability, its effectiveness depends on contextual guidance. This is the role of the Hindsight and Foresight modules, which inject insights and a design directive into the prompts to align each action with the search's current needs.

Hindsight Module. This module incorporates proven knowledge in the form of insights, which are abstract and generalizable design principles distilled from successful heuristics. These insights are managed in a dynamic Insight Pool, where each is assigned and continuously updated with a credibility score based on its empirical performance. Before generation, high-scoring insights are retrieved and embedded into the prompt. They serve as validated priors to steer the LLM toward promising designs. While effective for historical guidance, this module cannot address real-time evolutionary needs.

Foresight Module. The Foresight module implements real-time strategic control, orchestrated by its core component, the Evolutionary Navigator. The Navigator continuously monitors macroscopic evolutionary indicators, such as performance stagnation and population diversity, to assess the state of the search. Based on this analysis, it selects the governing evolutionary regime for the subsequent generation. This regime dictates the overall strategic focus by choosing one of three explicit modes (Črepinšek et al., 2013): 1) **Explore**, to foster novelty when diversity is low or progress has stalled; 2) **Exploit**, to refine high-performing solutions when progress is consistent; 3) **Balance**, to

maintain a synergistic application of all operators. This high-level directive shapes the final prompt, ensuring that the generative process of the LLM aligns with the immediate strategic needs of the evolution.

3.2 HINDSIGHT: MECHANISMS OF THE SELF-EVOLVING INSIGHT POOL

While foundational prompts provide the generative actions, the Hindsight module steers evolution with empirically validated knowledge managed within the Insight Pool. Analogous to the Belief Space in Cultural Algorithms (Coello & Becerra, 2010), our work introduces a novel instantiation for LLM-based AHD, distinguished by two core features: LLM-driven insight extraction and a utility-based credit assignment tailored for prompt-based generation.

To mitigate the cold-start problem, we initialize the Insight Pool with seed insights from established heuristic design literature. The core objective of the framework then becomes the autonomous discovery and refinement of new, problem-specific insights. This is achieved through a continuous lifecycle that systematically transforms transient evolutionary successes into explicit, reusable knowledge assets, governed by three integrated phases: insight extraction, utility-driven application, and adaptive pruning.

Insight Extraction and Admission. The lifecycle begins by expanding the knowledge base. At the end of each generation, we prompt an LLM to distill generalizable design principles (insights) from the elite individuals of the population (see Appendix G.1). To preserve informational diversity, a candidate insight k_{new} is admitted to the insight pool K_{pool} only if its Jaccard similarity to all existing pool members falls below a novelty threshold θ_{novelty}. For this comparison, each insight's text is preprocessed by converting it to lowercase and tokenizing it based on whitespace. The insights themselves become active candidates for the guidance of future generations.

Insight Retrieval and Credit Assignment. To guide heuristic generation, we employ a utility-based retrieval mechanism. For each new generation attempt, the mechanism selects the top-s insights with the highest adaptive utility score $U(k_i, t)$. This contributing set of insights, denoted as K_c, is then injected into the LLM's prompt. Following the evaluation of the offspring generated, we use credit assignment (Whitacre et al., 2006) to update the utility scores of all the insights in K_c. The utility function is formulated to balance exploitation and exploration:

$$U(k_i, t) = \underbrace{E_i(t)}_{\text{Effectiveness}} \underbrace{- w_u \log(N_i(t) + 1)}_{\text{Usage Penalty}} + \underbrace{B_r(t, t_i^{\text{last}})}_{\text{Recency Bonus}} \tag{1}$$

where $E_i(t)$ is the learned effectiveness of insight i. The penalty term, weighted by w_u, discourages overuse by penalizing an insight based on its total retrieval count $N_i(t)$, thus promoting exploration of less-used ideas. The recency bonus B_r offers a temporary reward for insights used recently; specifically, it grants a fixed bonus if the insight was used within a small generation window τ_r (that is, if $t - t_i^{\text{last}} \leq \tau_r$), promoting strategic coherence.

The effectiveness score $E_i(t)$ is updated via credit assignment. This process first converts the raw fitness $g(h_{\text{new}})$ of an offspring into a normalized, problem-agnostic score $\tilde{\rho}$, scaling its performance relative to the best ($g(h_{\text{best}})$) and worst ($g(h_{\text{worst}})$) solutions of the current population:

$$\tilde{\rho} = \frac{g(h_{\text{worst}}) - g(h_{\text{new}})}{g(h_{\text{worst}}) - g(h_{\text{best}}) + \epsilon} \tag{2}$$

where ϵ is a small constant (e.g., 10^{-6}) to prevent division by zero in the case of a homogeneous population where $g(h_{\text{worst}}) = g(h_{\text{best}})$. We posit that an offspring's evolutionary contribution is non-linear. Therefore, we map $\tilde{\rho}$ to a final credit signal, g_{eff}, using a tiered, piecewise function. This design creates distinct reward regimes for qualitatively different outcomes:

$$g_{\text{eff}}^{\text{raw}} = \begin{cases} 0.8 + 0.2 \cdot \tilde{\rho} & \text{if } g(h_{\text{new}}) \leq g(h_{\text{best}}) \\ 0.2 + 0.6 \cdot \tilde{\rho} & \text{if } g(h_{\text{best}}) < g(h_{\text{new}}) \leq g(h_{\text{avg}}) \\ -0.3 + 0.5 \cdot \tilde{\rho} & \text{if } g(h_{\text{new}}) > g(h_{\text{avg}}) \end{cases} \tag{3}$$

This structure provides strong positive signals for paradigm-changing improvements ($\tilde{\rho} \geq 1$), moderate rewards for incremental progress ($0 \leq \tilde{\rho} < 1$), and penalties for below-average performance. The allowance of negative credit is a deliberate design choice to accelerate the pruning of detrimental ideas. To ensure stable updates, the raw credit is then clipped to a final value

$g_{\text{eff}} = \max(-1.0, \min(1.0, g_{\text{eff}}^{\text{raw}}))$. Finally, for each insight j in the contributing set K_c, its effectiveness score is updated through an Exponential Moving Average (EMA):

$$E_j(t+1) = (1-\alpha) \cdot E_j(t) + \alpha \cdot g_{\text{eff}} \tag{4}$$

where the learning rate $\alpha \in [0, 1]$ smooths the stochastic credit signals from individual evaluations. This allows an insight's robust, long-term utility to emerge statistically, preventing its score from being skewed by outlier performances.

Adaptive Pruning and Pool Maintenance. To maintain quality within its finite capacity C_{pool}, the pool employs an adaptive pruning mechanism triggered when $|K_{\text{pool}}| > C_{\text{pool}}$. This process removes the insight with the minimum eviction score, $\mathcal{S}_{\text{evict}}$, which balances the proven performance of an insight against the risk of its obsolescence:

$$\mathcal{S}_{\text{evict}}(k_i, t) = E_i(t) - R_{\text{decay}} \cdot (t - t_{\text{last\_used}}(k_i)) \tag{5}$$

where $E_i(t)$ is the current effectiveness of the insight, $t_{\text{last\_used}}(k_i)$ is the generation of its last use, and the time decay rate R_{decay} targets not only low-effectiveness insights, but also those that have become inactive. The decay rate is set conservatively to prevent the premature removal of valuable but temporarily dormant knowledge. Furthermore, a grace period grants new insights with usage counts below a threshold T_{usage}, eviction immunity. This ensures novel principles are given a fair opportunity to demonstrate their utility before facing pruning.

3.3 Foresight: The Evolutionary Navigator for State-Aware Guidance

While the Hindsight module grounds the search in historically validated principles, the Foresight module acts as an **Evolutionary Navigator**, providing real-time, state-aware guidance. Its primary role is to dynamically orchestrate the balance between exploration and exploitation by implementing a control policy π that maps the macroscopic evolutionary state S_t to a high-level strategic orientation, the *Evolutionary Regime* θ_t. Crucially, unlike traditional adaptive EAs that solely tune numerical parameters, Foresight establishes a semantic feedback loop that directly steers the conceptual strategy of the generative LLM itself. This policy is a rule-based system that interprets key state indicators to select the most appropriate regime:

$$\theta_t = \begin{cases} \theta_{\text{explore}} & \text{if } C_{\text{stag}}(t) \geq \tau_{\text{stag}} \text{ or } \Delta_p(t) < \delta_p \\ \theta_{\text{exploit}} & \text{if } C_{\text{prog}}(t) \geq \tau_{\text{prog}} \\ \theta_{\text{balance}} & \text{otherwise} \end{cases} \tag{6}$$

To implement this policy, the Navigator continuously monitors two primary aspects of the evolutionary search: its performance trajectory and its population diversity. The performance trajectory is tracked via two mutually exclusive counters, C_{prog} and C_{stag}, based on the improvement in the best raw fitness Δg. If $\Delta g > 10^{-4}$, it is a generation of *progress* (increment C_{prog}, reset C_{stag}); otherwise, it is *stagnation* (increment C_{stag}, reset C_{prog}). Simultaneously, the Navigator assesses population health through a novel, problem-agnostic measure of phenotypic diversity, $\Delta_p(t)$. This metric is crucial because relying on fitness alone can be misleading; a population of high-fitness but structurally similar individuals often indicates entrapment in a local optimum. To counteract this, our metric quantifies the semantic variety within the population by measuring the dissimilarity of the generated algorithms' textual forms, rather than their fitness values. It is computed as the normalized fraction of unique pairs of algorithms in the population P whose textual descriptions are non-identical:

$$\Delta_p(t) = \frac{1}{|P|(|P|-1)/2} \sum_{i=1}^{|P|} \sum_{j=i+1}^{|P|} \mathbb{I}(\text{alg}_i \neq \text{alg}_j) \tag{7}$$

where $\mathbb{I}(\cdot)$ is the indicator function and alg_i denotes the textual description of the i-th algorithm. This hybrid state representation, marrying quantitative performance trends with a qualitative measure of semantic diversity, grants the Navigator a far more holistic understanding of the search landscape than fitness-based metrics alone can provide. These state indicators are evaluated against empirically determined thresholds, with their specific values detailed in our experimental setup. The design principles for these thresholds are key to the Navigator's effectiveness. The design principles for these thresholds are key to the Navigator's effectiveness. The progress and stagnation counters

are designed to track immediate performance trends. This focus on recent history is crucial for capturing the rapid dynamics inherent in LLM-based evolution, enabling swift strategic adjustments in response to even short periods of stagnation or consistent improvement. Similarly, the diversity threshold is calibrated to detect a significant collapse in semantic variety. It acts as an early warning mechanism against premature convergence, triggering stronger exploratory pressure when a substantial portion of the population becomes homogeneous. Once the regime θ_t is determined, it is translated into a natural language **Design Directive** that is injected into the LLM's prompt. For instance, θ_{explore} might yield a directive to *try a significantly different approach from conventional solutions*, whereas θ_{exploit} would instruct the model to *focus on refining and optimizing the most effective patterns*. This mechanism ensures the LLM's generative focus is explicitly aligned with the high-level strategy dictated by the current evolutionary state.

4 EXPERIMENTS

In this section, we present the results of heuristics designed by our proposed HiFo-Prompt on different complex tasks, including Traveling Salesman Problem (TSP)(Matai et al., 2010), Online Bin Packing Problem (Online BPP) (Seiden, 2002), Flow Shop Scheduling Problem (FSSP) (Emmons & Vairaktarakis, 2012), and Bayesian Optimization (BO) (Shahriari et al., 2016). Task definitions and details are given in Appendix B. Results on TSP, online BPP, and FSSP are presented in this section, while the results on BO are provided in Appendix C.4, where HiFo-Prompt demonstrates competitive and reliable performance.

Settings. We use the `qwen2.5-max` model via the `DashScope` API with a temperature of 1.0 and evolve a population of 4 individuals. The Insight Pool in Hindsight module has a capacity of $C_{\text{pool}} = 30$, with a Jaccard similarity threshold of 0.7 for deduplication. Insight retrieval uses a utility function with selection count $s = 3$, usage penalty weight $w_u = 0.1$, and recency bonus $\tau_r = 0.2$. Credit assignment is performed using EMA with a learning rate of $\alpha = 0.3$. For pool maintenance, each insight's eviction score decays at $R_{\text{decay}} = 0.01$ per generation, and new insights receive a grace period of $T_{\text{usage}} = 3$ applications. In the Foresight module, Evolutionary Navigator's thresholds are set to $\tau_{\text{stag}} = 3$ for stagnation, $\tau_{\text{prog}} = 2$ for progress, and $\delta_p = 0.3$ for critical diversity. We run 8 generations for CO tasks and 4 for BO, where rapid convergence demonstrates our method's efficiency. To ensure fairness, all LLM-based AHD baselines use the same LLM endpoint and query budget, with other settings kept as reported in their original papers.

Table 1: Results on TSP with step-by-step construction. Gap(%) denotes the performance gap compared to advanced heuristic algorithms. Time(s) represents the running time of designed heuristics. This result of LLM-based AHD method is the average of three runs. The best-performing LLM-based AHD results are shown in bold.

Method	TSP10		TSP20		TSP50	
	Gap	Time(s)	Gap	Time(s)	Gap	Time(s)
LKH3	0.000%	6.492	0.000%	24.9	0.000%	323.3
POMO	0.246%	-	0.248%	-	0.163%	-
LEHD	0.183%	-	0.010%	-	0.117%	-
EoH	7.148%	0.042	10.064%	0.1	12.820%	1.3
ReEvo	5.227%	0.228	6.811%	1.2	10.239%	21.5
HSEvo	5.461%	0.689	7.950%	3.2	10.467%	89.4
MCTS-AHD	4.829%	0.440	8.045%	4.1	10.642%	91.4
Ours	**1.654%**	0.709	**3.619%**	12.9	**6.625%**	244.7

Baselines. To demonstrate the effectiveness of our proposed method in designing heuristics, we introduce several approaches for solving these complex optimization tasks. (1) handcrafted heuristics, e.g., LKH3(Lin & Kernighan, 1973) for TSP, First Fit and Best Fit (Romera-Paredes et al., 2024) for Online BPP, NEH (Nawaz et al., 1983) and NEHFF (Fernandez-Viagas & Framinan, 2014) for FSSP. (2) Neural Combinatorial Optimization (NCO) methods, e.g., POMO (Kwon et al., 2020) and LEHD (Luo et al., 2023) for TSP, PFSPNet_NEH (Pan et al., 2022) for FSSP. (3) LLM-based

AHD methods, e.g., Funsearch (Romera-Paredes et al., 2024), EoH (Liu et al., 2024b), ReEvo (Ye et al., 2024), HSEvo (Dat et al., 2025) and MCTS-AHD (Zheng et al., 2025). Most of our test datasets follow EoH. Notably, Funsearch, ReEvo, and HSEvo require a seed function to initialize their populations, while EOH, MCTS-AHD, and our method can run without it.

Traveling Salesman Problem. Step-by-Step Construction (Asani et al., 2023) and Guided Local Search (GLS) (Alsheddy et al., 2016) are two different strategies for solving TSP. The detailed procedure of the two strategies can be found in the Appendix B.1. We design key heuristics for these two strategies. Table 1 compares the results of our method with other baselines. We evaluated the performance on 100 instances at each of five sizes. To demonstrate performance on out-of-distribution instances, we also conducted experiments on the TSPLib dataset (Reinelt, 1991), and the results can be found in the Appendix C.1. We further evaluated our method on 100 instances of TSP100, TSP200, and TSP500 in GLS. The results are shown in Table 2. The heuristic designed by our method achieves a significant performance improvement compared to the LLM-based AHD approach.

Table 2: Results on TSP with GLS. Comparison relative to the results of advanced heuristic on TSP100, TSP200 and TSP500. This result of LLM-based AHD method is the average of three runs. The best results are shown in bold.

Method	TSP100		TSP200		TSP500	
	Gap	Time(s)	Gap	Time(s)	Gap	Time(s)
LKH3	0.000%	-	0.000%	-	0.000%	-
EoH	0.026%	210.3	0.453%	368.1	2.037%	1100.7
ReEvo	0.049%	357.1	0.424%	775.8	2.090%	1103.4
HSEvo	0.087%	543.4	0.886%	792.9	2.507%	1105.1
Ours	**0.015%**	217.0	**0.382%**	392.4	**1.520%**	1100.8

Table 3: Results on Online BPP. Gap(%) denotes the ratio of excess bins compared to the lower bound on Weibull instances. Obj. represents the value of the objective function. Results with * are from EoH (Liu et al., 2024b). This result of LLM-based AHD method is the average of three runs. The best results are highlighted in bold.

Method	5k_C100		5k_C300		5k_C500	
	Gap	Obj	Gap	Obj	Gap	Obj
lower bound	0.00%	2006.2	0.00%	1740.2	0.00%	1687.0
First Fit*	4.40%	-	4.18%	-	4.27%	-
Best Fit*	4.08%	-	3.83%	-	3.91%	-
Funsearch*	0.80%	2022.2	1.07%	1758.8	1.47%	1711.8
EOH	1.02%	2026.6	1.00%	1757.6	1.00%	1703.9
ReEvo	0.78%	2021.8	4.47%	1818.0	3.24%	1741.6
HSEvo	1.91%	2044.6	5.47%	1835.4	4.39%	1761.1
MCTS-AHD	0.99%	2026.0	0.95%	1756.8	0.95%	1703.0
Ours	**0.69%**	2020.1	**0.66%**	1751.7	**0.66%**	1698.1

Online Bin Packing Problem. The key function of Online BPP is a scoring function that outputs a score for each bin, based on the current item's size and the remaining capacities of the bins. We evaluate our method on Weibull BPP instances (Romera-Paredes et al., 2024) following the EoH setting. The results with three scales are shown in Table 3. Our method not only significantly outperforms handcrafted heuristics but also surpasses LLM-based AHD approaches. More results are provided in Appendix C.2.

Flow Shop Scheduling Problem. FSSP involves scheduling n jobs on m machines to minimize the makespan, where each job comprises m operations processed in a fixed order. We evaluate

the heuristic designed by our method on Taillard instances (Taillard, 1993). The dataset includes instances with 20 to 100 jobs (n) and 10 to 20 machines (m). Table 4 presents some of the results, with additional results provided in Appendix C.3. Our method performs well on all datasets, consistently delivering strong and reliable results across different scenarios.

Table 4: Results on FSSP. The results show the comparison of makespan relative to the baseline on Taillard instances. n denotes the number of jobs and m denotes the number of machines. Results with * are from EoH (Liu et al., 2024b). This result of LLM-based AHD method is the average of three runs. The best results are highlighted in bold.

Method	n20,m10	n50,m10	n100,m10	n20,m20	n50,m20	n100,m20
NEH*	4.05%	3.47%	2.07%	3.06%	5.48%	3.58%
NEHFF*	4.15%	3.62%	1.88%	2.72%	5.10%	3.73%
PFSPNet_NEH*	4.04%	3.48%	1.72%	2.96%	5.05%	3.56%
EoH	0.31%	0.29%	0.23%	0.20%	0.84%	0.94%
Ours	**0.17%**	**0.17%**	**0.13%**	**0.10%**	**0.58%**	**0.51%**

Ablation Study. To evaluate the contribution of each component in our method, we conducted a series of ablation studies, as shown in Table 5. We separately removed the Hindsight obtained by the insight pool and the Foresight provided by the navigator, and then removed both Hindsight and Foresight entirely. These experiments are conducted on TSP and Online BPP. The design and parameters of the experiments are aligned with those used in the main experiments.

Table 5: Ablations on Insight Pool and Navigator in TSP and Online BPP. The best results are highlighted in bold.

	TSP					
	TSP20		TSP50		TSP100	
	Gap	Obj.	Gap	Obj.	Gap	Obj.
w/o Insight Pool	11.07%	4.29	13.76%	6.50	16.26%	9.06
w/o Navigator	5.82%	4.09	10.31%	6.30	11.48%	8.69
w/o Insight Pool and Navigator	11.49%	4.31	14.36%	6.53	18.80%	9.26
HiFo-Prompt	**2.79%**	**3.97**	**5.81%**	**6.04**	**8.16%**	**8.43**
	Online BPP					
	1k, 100	5k, 100	1k, 300	5k, 300	1k, 500	5k, 500
w/o Insight Pool	4.33%	1.27%	4.07%	1.22%	4.14%	1.29%
w/o Navigator	2.83%	1.26%	2.64%	1.24%	2.60%	1.24%
w/o Insight Pool and Navigator	4.53%	2.19%	4.30%	2.01%	4.26%	2.05%
HiFo-Prompt	**1.99%**	**0.66%**	**2.01%**	**0.63%**	**1.95%**	**0.66%**

5 Conclusion

We introduced HiFo-Prompt, a novel evolutionary framework that advances LLM-driven heuristic design through a hierarchical foresight-hindsight prompting mechanism. By synergizing an Evolutionary Navigator for adaptive control with a self-evolving Insight Pool for knowledge reuse, our framework transforms the search process into a closed-loop, self-regulating system. Across diverse optimization benchmarks, HiFo-Prompt consistently outperforms state-of-the-art methods with remarkable sample efficiency, often finding superior solutions using only 200 LLM requests. This work provides not only a powerful method for heuristic automated algorithm design but also a concrete step towards agents that can learn to invent their problem-solving methodologies.

REFERENCES

Aldeida Aleti and Irene Moser. A systematic literature review of adaptive parameter control methods for evolutionary algorithms. *ACM Computing Surveys*, 49(3):1–35, October 2016. doi: 10.1145/2996355.

Abdullah Alsheddy, Christos Voudouris, Edward P. K. Tsang, and Ahmad Alhindi. Guided local search. In *Handbook of Heuristics*, pp. 1–37. Springer, 2016. doi: 10.1007/978-3-319-07124-4_19-1. Living reference work entry.

Emmanuel O. Asani, Aderemi E. Okeyinka, and Ayodele Ariyo Adebiyi. A computation investigation of the impact of convex hull subtour on the nearest neighbour heuristic. In *2023 International Conference on Science, Engineering and Business for Sustainable Development Goals*, Omu-Aran, Nigeria, April 2023. IEEE. doi: 10.1109/SEB-SDG57117.2023.10124469.

Xiaohe Bo, Zeyu Zhang, Quanyu Dai, Xueyang Feng, Lei Wang, Rui Li, Xu Chen, and Ji-Rong Wen. Reflective multi-agent collaboration based on large language models. In *NeurIPS 2024 Poster*. NeurIPS, 2024. URL https://openreview.net/forum?id=9469. Submitted: 26 Sept 2024, Last Modified: 04 Jan 2025.

Edmund K. Burke, Matthew Hyde, Graham Kendall, Gabriela Ochoa, Ender Ozcan, and Rong Qu. A survey of hyper-heuristics. Technical Report NOTTCS-TR-SUB-0906241418-2747, School of Computer Science and Information Technology, University of Nottingham, Jubilee Campus, Nottingham NG8 1BB, UK, 2009.

Christian L. Camacho-Villalón, Thomas Stützle, and Marco Dorigo. Designing new metaheuristics: Manual versus automatic approaches. *Intelligent Computing*, 2:Article ID 0048, December 2023. doi: 10.34133/icomputing.0048.

Dikshit Chauhan, Bapi Dutta, Indu Bala, Niki van Stein, Thomas Bäck, and Anupam Yadav. Evolutionary computation and large language models: A survey of methods, synergies, and applications, 2025. URL https://arxiv.org/abs/2505.15741.

Carlos A. Coello Coello and Ricardo Landa Becerra. Efficient evolutionary optimization through the use of a cultural algorithm. *Engineering Optimization*, 36(2):219–236, 2010. doi: 10.1080/03052150410001647966.

Pham Vu Tuan Dat, Long Doan, and Huynh Thi Thanh Binh. HSEvo: Elevating automatic heuristic design with diversity-driven harmony search and genetic algorithm using LLMs. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39 of *AAAI-25 Technical Tracks*, pp. 26931–26938, 2025. doi: 10.1609/aaai.v39i25.34898.

A. E. Eiben and J. Smith. From evolutionary computation to the evolution of things. *Nature*, 521 (7553):476–482, 2015.

A. E. Eiben, R. Hinterding, and Z. Michalewicz. Parameter control in evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, 3(2):124–141, July 1999. doi: 10.1109/4235.771166.

Hamilton Emmons and George Vairaktarakis. *Flow Shop Scheduling: Theoretical Results, Algorithms, and Applications*, volume 182 of *International Series in Operations Research & Management Science*. Springer, New York, NY, 2012. ISBN 978-1461451518. doi: 10.1007/978-1-4614-5152-5. URL https://doi.org/10.1007/978-1-4614-5152-5.

Leah Epstein, Lene M. Favrholdt, and Jens S. Kohrt. Comparing online algorithms for bin packing problems. *Journal of Scheduling*, 15(1):13–21, 2012. doi: 10.1007/s10951-009-0129-5. URL https://doi.org/10.1007/s10951-009-0129-5.

Victor Fernandez-Viagas and Jose M. Framinan. On insertion tie-breaking rules in heuristics for the permutation flowshop scheduling problem. *Computers & Operations Research*, 45:60–67, May 2014. doi: 10.1016/j.cor.2013.12.012.

Fred Glover, Gregory Gutin, Anders Yeo, and Alexey Zverovich. Construction heuristics for the asymmetric TSP. *European Journal of Operational Research*, 129(3):555–568, March 2001. doi: 10.1016/S0377-2217(99)00468-3.

Ziyao Huang, Weiwei Wu, Kui Wu, Jianping Wang, and Wei-Bin Lee. Calm: Co-evolution of algorithms and language model for automatic heuristic design. *arXiv preprint arXiv:2505.12285*, 2025. URL https://doi.org/10.48550/arXiv.2505.12285. Neural and Evolutionary Computing (cs.NE).

Angel A. Juan, Helena R. Lourenço, Manuel Mateo, Rachel Luo, and Quim Castella. Using iterated local search for solving the flow-shop problem: parametrization, randomization and parallelization issues. *International Transactions in Operational Research*, 21(1):103–126, 2014. doi: 10.1111/itor.12028. URL https://doi.org/10.1111/itor.12028.

G. M. Komaki, Shaya Sheikh, and Behnam Malakooti. Flow shop scheduling problems with assembly operations: a review and new trends. *International Journal of Production Research*, 57 (10):2926–2955, 2019. doi: 10.1080/00207543.2018.1550269. URL https://doi.org/10.1080/00207543.2018.1550269.

Yeong-Dae Kwon, Jinho Choo, Byoungjip Kim, Iljoo Yoon, Youngjune Gwon, and Seungjai Min. POMO: Policy optimization with multiple optima for reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 33, pp. 21188–21199, 2020. URL https://proceedings.neurips.cc/paper_files/paper/2020/hash/f8b5b1f8f1f3b18a4d7a7a4c0d6a9f3d-Abstract.html.

Remi R. Lam, Karen E. Willcox, and David H. Wolpert. Bayesian optimization with a finite budget: An approximate dynamic programming approach. In *Advances in Neural Information Processing Systems*, volume 29, pp. 883–891, Barcelona, Spain, 2016. Curran Associates, Inc.

Eric Hans Lee, Valerio Perrone, Cédric Archambeau, and Matthias Seeger. Cost-aware bayesian optimization. *arXiv preprint*, 2020. doi: 10.48550/arXiv.2003.10870. URL https://doi.org/10.48550/arXiv.2003.10870.

Shen Lin and B. W. Kernighan. An effective heuristic algorithm for the traveling-salesman problem. *Operations Research*, 21(2):498–516, April 1973. doi: 10.1287/opre.21.2.498.

Fang Liu, Yang Liu, Lin Shi, Houkun Huang, Ruifeng Wang, Zhen Yang, Li Zhang, Zhongqi Li, and Yuchi Ma. Exploring and evaluating hallucinations in llm-powered code generation. arXiv preprint arXiv:2404.00971, 2024a. URL https://arxiv.org/abs/2404.00971. Submitted on 1 Apr 2024, last revised 11 May 2024.

Fei Liu, Xialiang Tong, Mingxuan Yuan, and Qingfu Zhang. Algorithm evolution using large language model, 2023. URL https://arxiv.org/abs/2311.15249. arXiv preprint arXiv:2311.15249.

Fei Liu, Tong Xialiang, Mingxuan Yuan, Xi Lin, Fu Luo, Zhenkun Wang, Zhichao Lu, and Qingfu Zhang. Evolution of heuristics: Towards efficient automatic algorithm design using large language model. In *Proceedings of the Forty-first International Conference on Machine Learning*, 2024b. URL https://icml.cc/Conferences/2024/AuthorGuide.

Fei Liu, Yiming Yao, Ping Guo, Zhiyuan Yang, Zhe Zhao, Xi Lin, Xialiang Tong, Mingxuan Yuan, Zhichao Lu, Zhenkun Wang, and Qingfu Zhang. A systematic survey on large language models for algorithm design. *arXiv preprint arXiv:2410.14716*, November 2024c. URL https://doi.org/10.48550/arXiv.2410.14716. v3.

Fu Luo, Xi Lin, Fei Liu, Qingfu Zhang, and Zhenkun Wang. Neural combinatorial optimization with heavy decoder: Toward large scale generalization. In *Advances in Neural Information Processing Systems 36*, 2023. URL https://papers.nips.cc/paper_files/paper/2023/hash/f0e9cba66c49e382e4f3e7351d47e2ed-Abstract-Conference.html.

Alireza Maheri, Shahin Jalili, Yousef Hosseinzadeh, Reza Khani, and Mirreza Miryahyavi. A comprehensive survey on cultural algorithms. *Swarm and Evolutionary Computation*, 62:100846, 2021. doi: 10.1016/j.swevo.2021.100846.

Yannis Marinakis. Heuristic and metaheuristic algorithms for the traveling salesman problem. In *Encyclopedia of Optimization*, pp. 1–12. Springer International Publishing, living reference work entry edition, 2024. doi: 10.1007/978-3-030-54621-2_262-1. URL https://doi.org/10.1007/978-3-030-54621-2_262-1.

Rafael Martí and Gerhard Reinelt. Heuristic methods. In *The Linear Ordering Problem*, volume 175 of *Applied Mathematical Sciences*, pp. 17–40. Springer, Berlin, Heidelberg, 2011. ISBN 978-3-642-16729-4. doi: 10.1007/978-3-642-16729-4_2. URL https://doi.org/10.1007/978-3-642-16729-4_2.

Rajesh Matai, Surya Prakash Singh, and Murari Lal Mittal. Traveling salesman problem: An overview of applications, formulations, and solution approaches. In Donald Davendra (ed.), *Traveling Salesman Problem, Theory and Applications*, pp. 1–25. InTech, Rijeka, Croatia, 2010. ISBN 978-953-307-426-9. doi: 10.5772/12909.

Muhammad Nawaz, E. Emory Enscore Jr, and Inyong Ham. A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem. *Omega*, 11(1):91–95, 1983. doi: 10.1016/0305-0483(83)90088-9.

Zixiao Pan, Ling Wang, Jingjing Wang, and Jiawen Lu. Deep reinforcement learning based optimization algorithm for permutation flow-shop scheduling. *IEEE Transactions on Emerging Topics in Computational Intelligence*, 7(4):900–911, 2022. doi: 10.1109/TETCI.2022.3154977.

Gregor Papa. Applications of dynamic parameter control in evolutionary computation. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pp. 1064–1088. ACM, July 2021. doi: 10.1145/3449726.3461435. URL https://doi.org/10.1145/3449726.3461435.

Carl Edward Rasmussen and Christopher K. I. Williams. *Gaussian Processes for Machine Learning*. Adaptive Computation and Machine Learning. The MIT Press, Cambridge, MA, 2006. ISBN 978-0-262-22567-4. doi: 10.7551/mitpress/3206.001.0001. URL https://doi.org/10.7551/mitpress/3206.001.0001. Open Access Edition.

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

Imma Ribas, Rainer Leisten, and Jose M. Framiñan. Review and classification of hybrid flow shop scheduling problems from a production system and a solutions procedure perspective. *Computers & Operations Research*, 37(8):1439–1454, 2010. doi: 10.1016/j.cor.2009.11.001. URL https://doi.org/10.1016/j.cor.2009.11.001.

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

Steven S. Seiden. On the online bin packing problem. *Journal of the ACM*, 49(5):640–671, September 2002. doi: 10.1145/585265.585269. URL https://doi.org/10.1145/585265.585269.

Lahari Sengupta, Radu Mariescu-Istodor, and Pasi Fränti. Which local search operator works best for the open-loop TSP? *Applied Sciences*, 9(19):3985, September 2019. doi: 10.3390/app9193985. URL https://doi.org/10.3390/app9193985.

Jiří Sgall. Online bin packing: Old algorithms and new results. In Arnold Beckmann, Erzèbet Csuhaj-Varjú, and Klaus Meer (eds.), *Language, Life, Limits: 10th Conference on Computability in Europe, CiE 2014, Budapest, Hungary, June 23–27, 2014, Proceedings*, volume 8493 of *Lecture Notes in Computer Science*, pp. 362–372. Springer, 2014. doi: 10.1007/978-3-319-08019-2_38. URL https://doi.org/10.1007/978-3-319-08019-2_38.

Bobak Shahriari, Kevin Swersky, Ziyu Wang, Ryan P. Adams, and Nando de Freitas. Taking the human out of the loop: A review of bayesian optimization. *Proceedings of the IEEE*, 104(1):148–175, January 2016. doi: 10.1109/JPROC.2015.2494218. URL https://doi.org/10.1109/JPROC.2015.2494218.

Yiding Shi, Jianan Zhou, Wen Song, Jieyi Bi, Yaoxin Wu, and Jie Zhang. Generalizable heuristic generation through large language models with meta-optimization. *arXiv preprint arXiv:2505.20881*, 2025. URL https://doi.org/10.48550/arXiv.2505.20881. Machine Learning (cs.LG); Artificial Intelligence (cs.AI).

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

Jasper Snoek, Hugo Larochelle, and Ryan P. Adams. Practical bayesian optimization of machine learning algorithms. In *Advances in Neural Information Processing Systems*, volume 25, pp. 2951–2959, Red Hook, NY, 2012. Curran Associates, Inc.

E. Taillard. Benchmarks for basic scheduling problems. *European Journal of Operational Research*, 64(2):278–285, 1993. doi: 10.1016/0377-2217(93)90182-M.

Nasser Tairan and Qingfu Zhang. Population-based guided local search: Some preliminary experimental results. In *Proceedings of the IEEE Congress on Evolutionary Computation*. IEEE, 2010. doi: 10.1109/CEC.2010.5586062. URL https://doi.org/10.1109/CEC.2010.5586062.

Nguyen Thach, Aida Riahifar, Nathan Huynh, and Hau Chan. Redahd: Reduction-based end-to-end automatic heuristic design with large language models. *arXiv preprint arXiv:2505.20242*, 2025. URL https://doi.org/10.48550/arXiv.2505.20242. Machine Learning (cs.LG).

Ye Tian, Xiaopeng Li, Haiping Ma, Xingyi Zhang, Kay Chen Tan, and Yaochu Jin. Deep reinforcement learning based adaptive operator selection for evolutionary multi-objective optimization. *IEEE Transactions on Emerging Topics in Computational Intelligence*, 7(4):1051–1064, August 2023. doi: 10.1109/TETCI.2022.3146882.

Jimi P. Tuononen. Analysis of rebuild local search operators for TSP. Master's thesis, University of Eastern Finland, School of Computing, Faculty of Forestry and Natural Sciences, Joensuu, Finland, April 2022. URL https://cs.uef.fi/sipu/pub/MSc_TuononenJimi.pdf.

R. J. M. Vaessens, E. H. L. Aarts, and J. K. Lenstra. Job shop scheduling by local search. *INFORMS Journal on Computing*, 8(3):302–317, 1996. doi: 10.1287/ijoc.8.3.302. URL https://doi.org/10.1287/ijoc.8.3.302.

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

Christos Voudouris, Abdullah Alsheddy, Edward P. K. Tsang, and Ahmad Alhindi. *Handbook of Heuristics*. Springer International Publishing AG, 2016. ISBN 978-3-319-07153-4. doi: 10.1007/978-3-319-07153-4.

Jianxun Wang and Yixiang Chen. A review on code generation with llms: Application and evaluation. In *Proceedings of the 2023 IEEE International Conference on Medical Artificial Intelligence*, Beijing, China, November 2023. doi: 10.1109/MedAI59581.2023.00044.

James M. Whitacre, Tuan Q. Pham, and Ruhul A. Sarker. Credit assignment in adaptive evolutionary algorithms. In *Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation*, pp. 1353–1360, New York, NY, USA, July 2006. ACM. doi: 10.1145/1143997.1144206. URL https://doi.org/10.1145/1143997.1144206.

Yuezhong Wu, Thomas Weise, and Raymond Chiong. Local search for the traveling salesman problem: A comparative study. In *Proceedings of the 2015 IEEE 14th International Conference on Cognitive Informatics & Cognitive Computing (ICCI*CC)*, pp. 213–220. IEEE, 2015. doi: 10.1109/ICCI-CC.2015.7259388. URL https://doi.org/10.1109/ICCI-CC.2015.7259388.

Xuesong Yan, Tao Song, and Qinghua Wu. An improved cultural algorithm and its application in image matching. *Multimedia Tools and Applications*, 76:14951–14968, 2017. doi: 10.1007/s11042-016-4041-x.

Shunyu Yao, Fei Liu, Xi Lin, Zhichao Lu, Zhenkun Wang, and Qingfu Zhang. Multi-objective evolution of heuristic using large language model. In *Proceedings of the Thirty-Ninth AAAI Conference on Artificial Intelligence (AAAI-25)*, Hong Kong, China, 2025. AAAI Press.

Yiming Yao, Fei Liu, Ji Cheng, and Qingfu Zhang. Evolve cost-aware acquisition functions using large language models. In Michael Affenzeller, Stephan M. Winkler, Anna V. Kononova, Heike Trautmann, Tea Tušar, Penousal Machado, and Thomas Bäck (eds.), *Parallel Problem Solving from Nature – PPSN XVIII: 18th International Conference, PPSN 2024, Hagenberg, Austria, September 14–18, 2024, Proceedings, Part II*, volume 15149 of *Lecture Notes in Computer Science*, pp. 374–390. Springer Cham, 2024. doi: 10.1007/978-3-031-70068-2. URL https://doi.org/10.1007/978-3-031-70068-2.

Ahmet Yarimcam, Shahriar Asta, Ender Özcan, and Andrew J. Parkes. Heuristic generation via parameter tuning for online bin packing. In *2014 IEEE Symposium on Evolving and Autonomous Learning Systems (EALS)*, pp. 102–108. IEEE, 2014. doi: 10.1109/EALS.2014.7009510. URL https://doi.org/10.1109/EALS.2014.7009510.

Haoran Ye, Jiarui Wang, Zhiguang Cao, Federico Berto, Chuanbo Hua, Haeyeon Kim, Jinkyoo Park, and Guojie Song. Reevo: Large language models as hyper-heuristics with reflective evolution. In *Proceedings of the 38th International Conference on Neural Information Processing Systems*, pp. 43571–43608, June 2024. URL https://nips.cc/Conferences/2024/AuthorGuide.

Zhi Zheng, Changliang Zhou, Tong Xialiang, Mingxuan Yuan, and Zhenkun Wang. Udc: A unified neural divide-and-conquer framework for large-scale combinatorial optimization problems. In *Advances in Neural Information Processing Systems 37*, 2024. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/0b8e4c8468273ee3bafb288229c0acbc-Paper-Conference.pdf.

Zhi Zheng, Zhuoliang Xie, Zhenkun Wang, and Bryan Hooi. Monte carlo tree search for comprehensive exploration in llm-based automatic heuristic design. In *Proceedings of the 2025 International Conference on Machine Learning (ICML 2025)*, 2025. URL https://icml.cc/Conferences/2025/AuthorGuide.

Álvaro Fialho. *Adaptive Operator Selection for Optimization*. PhD thesis, Université Paris Sud- Paris XI, Paris, France, 2010. URL https://theses.hal.science/tel-00578431v1. Submitted on 20 Mar 2011, HAL Id: tel-00578431.

Matej Črepinšek, Shih-Hsi Liu, and Marjan Mernik. Exploration and exploitation in evolutionary algorithms: A survey. *ACM Computing Surveys*, 45(3):1–33, July 2013. doi: 10.1145/2480741.2480752. URL https://doi.org/10.1145/2480741.2480752.

Anja Šurina, Amin Mansouri, Lars C.P.M. Quaedvlieg, Amal Seddas, Maryna Viazovska, Emmanuel Abbe, and Caglar Gulcehre. Algorithm discovery with llms: Evolutionary search meets reinforcement learning. *COLM 2025*, 2025. URL https://doi.org/10.48550/arXiv.2505.12285.

A Preliminary

A.1 Problem Formulation of Automatic Heuristic Design

Automatic Heuristic Design (AHD) (Burke et al., 2009; Voudouris et al., 2016) aims to identify an optimal heuristic h^* from a vast search space $\mathcal{H}$ for a given computational task $\mathcal{P}$. This process can be formally expressed as the following optimization problem (Zheng et al., 2025):

$$h^* = \arg\max_{h \in \mathcal{H}} g(h), \tag{8}$$

where $g(h)$ is a fitness function that maps a heuristic to a real number, estimating its quality based on its expected performance on a representative set of problem instances D. For a task with a minimization objective f (e.g., minimizing cost or error), this performance metric is defined as:

$$g(h) = \mathbb{E}_{\text{ins} \in D} \left[-f(h(\text{ins})) \right]. \tag{9}$$

This formulation reframes the original task as a maximization problem over the heuristic space $\mathcal{H}$, thereby enabling the search for robust heuristics that yield high-quality solutions across diverse instances.

A.2 LLM-DRIVEN EVOLUTIONARY COMPUTATION

The LLM-driven Evolutionary Computation (LLM+EC) framewor(Liu et al., 2024b; Romera-Paredes et al., 2024; Yao et al., 2025; Chauhan et al., 2025) casts the AHD problem as an iterative, population-based search process. Let $P^{(t)} = \{h_1^{(t)}, \ldots, h_M^{(t)}\}$ be the population of M heuristics at generation t, where each heuristic $h_i^{(t)} \in \mathcal{H}$ is represented by its thought and code. The transition from $P^{(t)}$ to $P^{(t+1)}$ is governed by a stochastic evolutionary kernel $\mathcal{F}$, which is parameterized by a control vector $\theta^{(t)} \in \Theta$:

$$P^{(t+1)} \sim \mathcal{F}(P^{(t)} \mid \theta^{(t)}). \tag{10}$$

Here, the control vector $\theta^{(t)}$ encapsulates contextual information that guides heuristic generation, such as prompting strategies or performance feedback. In the absence of adaptive control, $\theta^{(t)}$ can be considered constant or null, i.e., $\theta^{(t)} = \theta_{\text{const}}$ or $\theta^{(t)} = \varnothing$. The kernel $\mathcal{F}$ comprises two phases:

1. **Generation Phase**: An LLM, acting as a conditional generator $\mathcal{L}$, creates new heuristics through prompted crossover and mutation. A set of parents $h_p \subseteq P^{(t)}$ is selected, and their symbolic representations $\rho(h_p)$, which include the `thought-and-code`, are used. A prompt function Π constructs a conditional prompt from $\rho(h_p)$ and $\theta^{(t)}$ (e.g., exploration or modification strategies). The LLM then generates offspring:
$$h_c = \mathcal{L}(\Pi(\rho(h_p), \theta^{(t)})), \quad O^{(t)} = \{h_{c,1}, h_{c,2}, \ldots, h_{c,N}\}, \tag{11}$$
where N is the number of offspring (e.g., $N = \lambda M$, where λ is the reproduction rate).
2. **Selection Phase**: The parent and offspring populations are merged into a candidate pool, $U^{(t)} = P^{(t)} \cup O^{(t)}$. A selection operator $\mathcal{S}$ then chooses the top M heuristics from this pool based on the fitness metric g to form the next generation:
$$P^{(t+1)} = \mathcal{S}(U^{(t)}; g). \tag{12}$$

This framework, introduced within the "heuristic evolution" paradigm, leverages LLM-driven generation to explore the heuristic space and a selection mechanism to exploit high-performing solutions.

A.3 KNOWLEDGE-AUGMENTED EVOLUTIONARY COMPUTATION

To extend the capabilities of evolutionary algorithms beyond simple adaptive control, a prominent research direction involves incorporating an explicit knowledge component. Cultural Algorithms (**?**Coello & Becerra, 2010; Yan et al., 2017) provide a classic framework for this idea, decoupling the evolutionary system into two interacting spaces: a population space containing candidate solutions $P^{(t)}$ and a belief space K_t serving as a repository of experiential knowledge (Maheri et al., 2021). These two spaces co-evolve through a dual-inheritance communication protocol.

Crucially, knowledge k_s extracted from the belief space K_t becomes part of the high-level control vector $\theta^{(t)}$. This knowledge then guides the generation of offspring via an influence mechanism:

$$h_c \sim \mathcal{L}\big(\Pi(\rho(h_p), \theta^{(t)})\big), \quad \text{where } k_s \subseteq \theta^{(t)}. \tag{13}$$

Currently, the successes of the population are fed back into the belief space. An acceptance function $\mathcal{A}$ identifies and extracts potentially valuable experiences, $\mathcal{A}(P^{(t)})$, from the current population. These are then used by a knowledge update function $\mathcal{U}_K$ to update the belief space:

$$K_{t+1} = \mathcal{U}_K(K_t, \mathcal{A}(P^{(t)})). \tag{14}$$

Knowledge-augmented evolution is thus a coupled dynamical system where the population and knowledge base co-evolve interdependently. This explicit mechanism for knowledge management allows the framework to achieve a more sophisticated form of learning based on abstracted experience.

B OPTIMIZATION PROBLEM DETAILS

B.1 TRAVELING SALESMAN PROBLEM

The Traveling Salesman Problem (TSP) (Matai et al., 2010; Voudouris & Tsang, 1999) is a canonical NP-hard combinatorial optimization problem. Given a set of N cities and the distances between

each pair, the objective is to find the shortest possible tour that visits each city exactly once before returning to the starting city (**?**).

Formally, let $V = \{v_1, v_2, \ldots, v_N\}$ be the set of cities. We model the problem on a complete, undirected graph $G = (V, E)$, where a non-negative distance d_{ij} is associated with each edge $(v_i, v_j) \in E$. A tour is a permutation π of the indices $\{1, 2, \ldots, N\}$. The goal is to find a permutation π^* that minimizes the total tour length (Zheng et al., 2024):

$$\min_{\pi} \left(\sum_{i=1}^{N-1} d_{\pi_i, \pi_{i+1}} + d_{\pi_N, \pi_1} \right)$$

where v_{π_i} denotes the i-th city in the tour.

Step-by-step Construction. Construction heuristics build a feasible solution from an empty set by making a sequence of decisions (Glover et al., 2001). At each step, a component is added to the partial solution based on a specific greedy criterion until a complete tour is formed (Martí & Reinelt, 2011). A fundamental example is the Nearest Neighbor (NN) heuristic. Starting from a node v_{π_1}, it constructs a tour by iteratively selecting the closest unvisited node. At step t, with a partial tour $S_t = (v_{\pi_1}, \ldots, v_{\pi_t})$ and the set of unvisited nodes $U_t = V \setminus \{v_{\pi_1}, \ldots, v_{\pi_t}\}$, the next node $v_{\pi_{t+1}}$ is chosen according to the rule (Marinakis, 2024):

$$v_{\pi_{t+1}} = \arg \min_{v_j \in U_t} d_{\pi_t, j}$$

While fast, the myopic nature of such simple rules often leads to suboptimal solutions.

Our approach, HIFO-Prompt, introduces a new paradigm for solving combinatorial optimization problems by shifting from traditional, fixed construction heuristics to dynamically generated, instance-specific policies. We retain the foundational step-by-step framework of construction methods in which a solution is built incrementally. However, we fundamentally revolutionize the core decision-making logic. Instead of relying on a universal, handcrafted rule (e.g., Nearest Neighbor), which is often myopic and susceptible to poor initial choices leading to local optima, our method leverages a LLM to synthesize a sophisticated decision-making function on-the-fly for each problem instance. At every construction step t, the LLM acts as a reasoning engine. It is provided with the complete state of the problem, including the current partial tour S_t, the set of unvisited candidate nodes U_t, the global distance matrix C, and the tour's origin node to facilitate reasoning about the final closing cost (Liu et al., 2024b; 2023).

The advantage of HIFO-Prompt lies in its ability to offload the complex, knowledge-intensive task of heuristic design to the LLM. Through carefully engineered prompts, we guide the model not merely to select a node but to generate a computational policy that embodies advanced strategic principles. The synthesized policy can perform non-trivial reasoning, such as conducting evaluations of multi-step consequences to look beyond immediate greedy gains. It can be guided to implement mechanisms analogous to maintaining a belief over a set of promising candidates, exploring their short-term implications before committing to a single path. Furthermore, the generated function can incorporate stochastic simulations or rollouts to approximate the long-term value of different choices, mimicking the exploration capabilities of advanced search algorithms.

Guided Local Search Improvement heuristics, such as local search, start with a complete tour and iteratively refine it. Guided Local Search (GLS) is an advanced metaheuristic that enhances local search by introducing a guidance mechanism to escape local optima (**?**Voudouris & Tsang, 1999; Voudouris et al., 2016). GLS achieves this by modifying the objective function with penalties on certain solution features that appear in locally optimal solutions (Wu et al., 2015). For the Traveling Salesperson Problem (TSP), the most natural features to penalize are the edges of the tour (Tairan & Zhang, 2010).

The standard GLS cost function is augmented with a penalty term:

$$L_{\text{aug}}(s) = L(s) + \lambda \sum_{(u,v) \in s} p_{uv} \tag{15}$$

where $L(s)$ is the original tour length, the sum is over all edges (u, v) in tour s, p_{uv} is a penalty counter for using the edge between cities u and v, and λ is a regularization parameter. An efficient

implementation involves creating a penalized distance matrix D' for the local search:

$$D'_{uv} = d_{uv} + \lambda \cdot p_{uv} \quad . \tag{16}$$

Minimizing the tour length using D' is equivalent to minimizing $L_{\text{aug}}(s)$. The critical challenge lies in designing the rule for updating the penalty matrix $P = \{p_{uv}\}$. When the local search becomes trapped in a local optimum s^*, a state-dependent update heuristic, H_{update}, is invoked to determine which edges in s^* should be penalized:

$$P_{\text{new}} = H_{\text{update}}(P_{\text{old}}, s^*, D, N) \quad , \tag{17}$$

where N is a matrix of edge usage frequencies. Typically, this heuristic is a static, handcrafted rule that increments the penalties for a subset of edges in s^*. This update reshapes the search landscape to guide the search away from the current basin of attraction.

HiFo-Prompt automates the discovery of the update heuristic H_{update}. Instead of relying on a static, human-designed rule, we task a LLM with synthesizing a complete, executable Python function to serve as H_{update}.

At each GLS iteration, after converging to a local optimum s^*, this LLM-generated function is invoked. It receives the full state necessary for intelligent penalization—the current penalty matrix (P_{old}), the local optimum tour (s^*), the original distance matrix (D), and the edge frequency matrix (N)—and outputs a new penalty matrix, P_{new}. This updated matrix then modifies the search costs via Eq. 16 for the next major iteration. The local search itself is driven by fundamental prompt strategies like **Relocate** (Tuononen, 2022) and **2-opt** (Sengupta et al., 2019). Our contribution lies not in these prompt strategies but in the automated design of the sophisticated H_{update} function through HiFo-Prompt, which provides the intelligence to guide these prompt strategies effectively.

We further elevate this concept by embedding heuristic generation within an evolutionary framework. We treat the distinct H_{update} functions as a population of individuals. The LLM itself serves as the primary genetic operator. Through structured prompts for crossover and mutation, the LLM intelligently combines or modifies existing high-performing strategies to produce novel offspring (Liu et al., 2024b).

B.2 Online Bin Packing Problem

The Online Bin Packing Problem (OBP) (Seiden, 2002) is a classic sequential decision-making problem. We are presented with a sequence of items, $A = (a_1, a_2, \ldots, a_T)$, arriving one at a time. Each item a_t has a size $s_t \in (0, 1]$. We have an unlimited supply of bins, each with a unit capacity of 1. The core constraints of the OBP are:

- **Online Constraint:** When item a_t arrives, a decision must be made to place it into a bin without any knowledge of future items $(a_{t+1}, \ldots, a_T)$.
- **Irrevocable Placement:** Once an item is placed in a bin, it cannot be moved.
- **Capacity Constraint:** For any bin B_j, the sum of the sizes of all items placed within it must not exceed 1.

The objective is to minimize the total number of bins used after placing all T items (Epstein et al., 2012; Yarimcam et al., 2014). If we let $y_j = 1$ if bin j is used and $y_j = 0$ otherwise, the goal is to minimize $\sum_j y_j$ (Sgall, 2014).

Given the online nature of the problem, optimal solutions are generally not achievable. Instead, high-performance algorithms rely on sophisticated greedy placement policies or heuristics. Following the approach of (Romera-Paredes et al., 2024), we frame the task as learning a superior placement heuristic. Specifically, we use the HiFo-Prompt framework to design a scoring function, H_{score}, that determines the most suitable bin for an incoming item.

When an item a_t with size s_t arrives, the HiFo-Prompt-generated heuristic is invoked. It takes as input the item's size and the state of all currently open bins. The state of a bin B_j is captured by its residual capacity, $c_j = 1 - \sum_{a_k \in B_j} s_k$. The scoring function produces a scalar value for each candidate bin:

$$\sigma_j = H_{\text{score}}(s_t, c_j) \tag{18}$$

where σ_j represents the "desirability" of placing item a_t into bin B_j. The placement policy is then to assign the item to the valid bin with the highest score:

$$j^* = \underset{j \mid c_j \geq s_t}{\arg\max}\, \sigma_j \tag{19}$$

If no existing bin can accommodate the item (i.e., the set of valid bins is empty), a new bin is opened. The intelligence of our method lies in HiFo-Prompt's ability to discover a non-trivial scoring function H_{score} that implicitly balances competing objectives, such as leaving space for potentially larger future items versus consolidating small items efficiently. This contrasts with classic heuristics like Best Fit (which is equivalent to $H_{\text{score}}(s_t, c_j) = -c_j$) or First Fit, by allowing for a much richer and more adaptive decision-making process.

B.3 Flow Shop Scheduling Problem

The Flow Shop Scheduling Problem (FSSP) (Emmons & Vairaktarakis, 2012; Komaki et al., 2019) is a canonical NP-hard scheduling challenge. The task is to schedule a set of n jobs, $J = \{1, \ldots, n\}$, on a series of m machines, $M = \{1, \ldots, m\}$. Each job $i \in J$ requires m operations, with the j-th operation occurring on machine j. The processing time for job i on machine j is given by T_{ij} from a processing time matrix T (Vaessens et al., 1996; Ribas et al., 2010). A solution (Juan et al., 2014) is a permutation of jobs, $\pi = (\pi_1, \ldots, \pi_n)$, which dictates the processing order on all machines. Key constraints include that no machine can process multiple jobs at once, and no job can be on multiple machines simultaneously.

The objective is to find a permutation π^* that minimizes the makespan, $C_{\max}$. This is the total time elapsed until the last job completes its final operation. The completion time $C(\pi_i, j)$ for job π_i on machine j is calculated recursively:

$$C(\pi_i, j) = \max\big(C(\pi_{i-1}, j), C(\pi_i, j-1)\big) + T_{\pi_i, j} \tag{20}$$

with base cases $C(\pi_0, j) = 0$ and $C(\pi_i, 0) = 0$. The makespan for a sequence π is therefore $C_{\max}(\pi) = C(\pi_n, m)$.

Due to the problem's complexity, we employ a local search metaheuristic (**?**Liu et al., 2024b). To prevent the search from being trapped in local optima, we use the HiFo-Prompt framework to design a sophisticated guidance strategy automatically. When the search converges to a locally optimal sequence π^*, HiFo-Prompt-generated heuristic, H_{guide}, is invoked. It takes the current sequence, the original processing time matrix, and problem dimensions to produce both a new time matrix T' and a designated list of jobs to perturb, J_{perturb}:

$$(T', J_{\text{perturb}}) = H_{\text{guide}}(\pi^*, T, n, m) \tag{21}$$

This dual output provides a powerful guidance mechanism. The new matrix T' reshapes the search landscape by penalizing attributes of the local optimum, effectively steering the search toward unexplored regions. Concurrently, the list J_{perturb} directs subsequent local search operators, such as insertions or swaps, to focus their computational effort on a specific subset of critical jobs. This combined strategy of altering the problem's cost structure while focusing on the search operators constitutes a complete and intelligent guidance component, designed automatically by our framework.

B.4 Bayesian Optimization

Bayesian Optimization (BO) (Shahriari et al., 2016) and its ongoing development are of paramount importance, as it provides the leading framework for sample-efficiently navigating the complex, high-cost search spaces prevalent in modern science and engineering. Bayesian Optimization (BO) has emerged as a principal framework for this task, excelling in applications like hyperparameter tuning and automated scientific discovery. The power of BO lies in its sample efficiency. It builds a probabilistic surrogate model of the objective function. It then uses an acquisition function (**?**Lam et al., 2016) to intelligently decide where to sample next, thereby minimizing the number of costly evaluations.

Our work addresses a particularly demanding variant: cost-aware BO (Snoek et al., 2012; Yao et al., 2024; Zheng et al., 2025). In this setting, each function evaluation $f(\mathbf{x})$ has a heterogeneous and

unknown cost, denoted by $c(\mathbf{x})$. The goal is to find the global maximum of $f(\mathbf{x})$ within a fixed total budget B_{total}. This requires sequentially choosing evaluation points $\{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$ to maximize the final outcome, subject to the constraint that $\sum_{i=1}^{N} c(\mathbf{x}_i) \leq B_{\text{total}}$ (Lee et al., 2020). To manage this, the BO agent maintains two surrogate models, typically Gaussian Processes (GPs) (Rasmussen & Williams, 2006). One GP models the objective function, predicting its posterior mean $\mu_f(\mathbf{x})$ and standard deviation $\sigma_f(\mathbf{x})$. A second GP models the evaluation cost, providing a cost prediction $\mu_c(\mathbf{x})$.

The core intelligence of the agent is encoded in its acquisition function, $\alpha(\mathbf{x})$. This function must navigate a complex trade-off between seeking high rewards (exploitation), reducing model uncertainty (exploration), and managing evaluation costs. At each iteration t, the next evaluation point $\mathbf{x}_{t+1}$ is selected by maximizing this utility:

$$\mathbf{x}_{t+1} = \arg\max_{\mathbf{x} \in \mathcal{X}} \alpha(\mathbf{x} | \mathcal{D}_t, B_{\text{rem}}) \tag{22}$$

where $\mathcal{D}_t = \{(\mathbf{x}_i, y_i, c_i)\}_{i=1}^{t}$ is the set of previously evaluated points and B_{rem} is the remaining budget. The design of $\alpha(\mathbf{x})$ is the single most critical factor for achieving high performance.

We propose to automate the discovery of superior acquisition functions using the HiFo-Prompt framework. Rather than relying on static, human-designed heuristics, HiFo-Prompt generates a novel utility function, H_{utility}, from scratch. This generated function is highly context-aware, synthesizing all critical information available at each decision step. It explicitly considers the surrogate models' predictions, the best-found solution so far ($y_t^* = \max_i y_i$), and the dynamic state of the budget:

$$\alpha(\mathbf{x}) = H_{\text{utility}}\big(\mu_f(\mathbf{x}), \sigma_f(\mathbf{x}), \mu_c(\mathbf{x}), y_t^*, B_{\text{used}}, B_{\text{total}}\big) \tag{23}$$

By generating a holistic function that reasons about the interplay between potential gain, uncertainty, cost, and remaining resources, HiFo-Prompt creates powerful and adaptive sampling strategies. This approach moves beyond hand-crafted designs, enabling superior performance in complex, budget-constrained optimization scenarios.

C More Results

C.1 Traveling Salesman Problem

To demonstrate the generality and robustness of the heuristic designed by our proposed method, we conduct a comprehensive evaluation using the real-world benchmark dataset TSPLib (Reinelt, 1991). TSPLib is a well-established collection of the TSP instances, widely used in the research community for benchmarking optimization algorithms. For our experiments, we select a diverse subset of TSP instances from TSPLib, specifically focusing on those containing no more than 500 nodes. This selection criterion ensures a manageable problem scale while still retaining the complexity necessary to assess algorithmic performance effectively.

In our evaluation framework, we adopt a step-by-step construction approach to solve the TSP, which incrementally builds a tour by selecting the next node based on a learned heuristic. This paradigm allows us to evaluate the quality of decisions made at each step and better observe the contribution of the designed heuristic to the final solution quality. To rigorously assess the effectiveness and competitiveness of our proposed heuristic, we compare its performance against state-of-the-art LLM-based AHD baselines. These methods represent recent advances in leveraging large language models for combinatorial optimization tasks and serve as strong comparative baselines in our study. The experimental results, which include performance metrics, are summarized in Table **??**. These results provide empirical evidence that our method not only performs competitively but also generalizes well across a variety of TSP instances in real-world scenarios.

C.2 Online Bin Packing Problem

To provide a more comprehensive empirical validation and to assess the robustness of our framework, we conducted further evaluations of the online Bin Packing Problem (BPP) on a broader and more challenging set of Weibull-distributed instances. These instances, which more closely mimic real-world scenarios than uniform distributions, were generated following the protocol established

in prior work on LLM-based heuristic discovery (Romera-Paredes et al., 2024). Table 6 presents a detailed comparative analysis of our method against a wide spectrum of competitors, including both classical, widely-adopted handcrafted heuristics (First Fit, Best Fit) and a suite of contemporary approaches based on Large Language Models.

For each combination of bin capacity and problem size, the dataset includes five unique instances. Performance is quantified by the average percentage gap to the known theoretical lower bound across these instances, where a smaller value signifies a more efficient and effective packing solution. The empirical results unequivocally demonstrate the superior performance of the heuristic we have discovered. As shown in the table, our method consistently outperforms all baseline methods in nearly every setting, securing the smallest average gap in 8 out of the 9 distinct configurations tested. This highlights a remarkable level of consistency and dominance. The only exception is the (Capacity=100, Size=10k) case, where Funsearch achieves a marginally lower gap. However, our method's strong performance across the entire parameter space underscores its greater reliability and generalizability compared to methods that may excel only in specific, narrow scenarios. Notably, our approach scales gracefully, achieving extremely low gap values (e.g., 0.41%, 0.42%) on the largest and most complex problems, significantly surpassing both traditional algorithms and other state-of-the-art LLM-driven frameworks. This comprehensive evaluation on challenging instances further validates the efficacy of our evolutionary framework, showcasing its capacity to discover sophisticated and high-performance heuristics that are robust across varied problem characteristics.

Table 6: Results on Online BPP in Weibull instances with varied capacities and problem sizes. Results marked with * denote values taken from (Liu et al., 2024b).

Capacity	Size	First Fit	Best Fit	EoH	ReEvo	HSEvo	MCTS-AHD	Ours
100	1k	5.32%	4.87%	3.10%	3.63%	3.51%	3.38%	**2.19%**
	5k	4.40%	4.08%	1.02%	0.78%	1.91%	0.99%	**0.69%**
	10k	4.44%	4.09%	0.80%	**0.35%**	1.68%	0.84%	0.42%
300	1k	4.93%	4.48%	3.04%	7.34%	6.88%	3.21%	**2.08%**
	5k	4.18%	3.83%	1.00%	4.47%	5.47%	0.95%	**0.66%**
	10k	4.20%	3.87%	0.78%	4.05%	5.26%	0.85%	**0.39%**
500	1k	4.97%	4.50%	3.04%	5.92%	5.80%	3.20%	**2.07%**
	5k	4.27%	3.91%	1.00%	3.24%	4.39%	0.95%	**0.66%**
	10k	4.28%	3.95%	0.78%	2.79%	4.14%	0.85%	**0.40%**

C.3 Flow Shop Scheduling Problem

Table 7 presents a comparative analysis of the performance of EoH and our proposed method on FSSP of varying scales, defined by the number of jobs n and machines m. Performance is evaluated using two key metrics: the objective function value and the relative gap, where the gap is measured with respect to the solution quality of an advanced handcrafted heuristic. A smaller gap reflects a solution that is closer to the heuristic baseline and, therefore, indicates higher solution quality. The results demonstrate that our method consistently achieves smaller gaps across all tested problem configurations, regardless of scale. Moreover, in many cases, it outperforms the advanced heuristic itself in terms of the raw objective value. This consistent superiority suggests that our approach not only generalizes well across diverse FSSP instances (Liu et al., 2024b) but also offers a competitive alternative to domain-specific heuristics, exhibiting both strong effectiveness and robustness.

C.4 Bayesian Optimization

Table 8 presents the experimental results on the design of cost-aware acquisition functions (CAFs) (Yao et al., 2024) in Bayesian Optimization. We compare our method against both manually crafted CAFs and those generated by LLM-based AHD methods. To ensure a fair comparison, all LLM-based AHD approaches utilize the same underlying language model. Our method achieves superior performance on the majority of benchmark functions, outperforming traditional CAFs (e.g., EI, EIpu, EI-cool) (Yao et al., 2024) as well as recent LLM-based AHD baselines (e.g., EoH (Liu et al., 2024b), MCTS (Zheng et al., 2025)). The performance advantage is particularly pronounced

Table 7: Results on FSSP. The best results are highlighted in bold.

n	m	EoH		Ours	
		Gap	Time(s)	Gap	Time(s)
20	5	0.25%	7.5	**-0.01%**	5.5
	10	0.31%	13.0	**0.17%**	8.8
	20	0.20%	23.8	**0.10%**	18.5
50	5	0.01%	45.1	**0.00%**	29.7
	10	0.29%	83.9	**0.17%**	50.6
	20	0.84%	168.4	**0.58%**	101.1
100	5	-0.02%	230.2	**-0.04%**	146.7
	10	0.23%	299.6	**0.13%**	243.5
	20	0.94%	305.2	**0.51%**	303.0
200	10	0.37%	337.1	**0.12%**	317.2
	20	1.23%	334.6	**0.71%**	321.4

Table 8: Results on BO. The results denote the absolute error relative to the optimal solution. Results marked with * are from (Yao et al., 2024). The best results are highlighted in bold.

	Ackley	Rastrigin	Griewank	Rosenbrock	Levy	ThreeHumpCamel
EI*	2.66	4.74	0.49	**1.26**	0.01	0.05
EIpu*	2.33	5.62	**0.34**	2.36	0.01	0.12
EI-cool*	2.74	5.78	**0.34**	2.29	0.01	0.07
EOH	3.11	3.48	0.72	2.57	0.04	0.18
MCTS	3.23	0.87	0.43	1.30	0.01	0.05
Ours	**1.78**	**0.45**	0.41	1.50	**0.00**	**0.01**
	StyblinskiTang	Hartmann	Powell	Shekel	Hartmann	Cosine8
EI*	0.03	**0.00**	18.89	7.91	**0.03**	0.47
EIpu*	**0.02**	**0.00**	19.83	7.92	**0.03**	0.47
EI-cool*	0.03	**0.00**	14.95	8.21	**0.03**	0.54
EOH	2.89	0.01	13.71	8.71	0.47	1.04
MCTS	**0.02**	**0.00**	**1.91**	5.14	0.08	0.29
Ours	**0.02**	0.01	2.65	**4.08**	0.57	**0.28**

on challenging functions such as Rastrigin, ThreeHumpCamel, and Shekel. These results underscore the robustness and strong generalization ability of our approach across a wide range of optimization landscapes.

C.5 Comparative Results

We compared the progression of objective function values during the evolutionary process of our method and EoH on both the TSP and Online BPP. Figure 3 presents the convergence analysis curves. The figure demonstrates that our method converges more rapidly while requiring fewer individuals in the population, indicating higher efficiency. Additionally, we have evaluated our method on a variety of large language models, with the results presented in Table 9.

C.6 Parameter Sensitivity Analysis

Based on the parameter sensitivity analysis results (see in Figure 4), the default parameter settings of the genetic algorithm (population size 8, maximum generations 12, insight pool size 30, stagnation threshold 3, diversity threshold 0.3) demonstrate robust overall performance. The analysis reveals that population size significantly influences convergence behavior: excessively small values (e.g., 4) tend to cause premature convergence, while overly large values (e.g., 12) reduce convergence

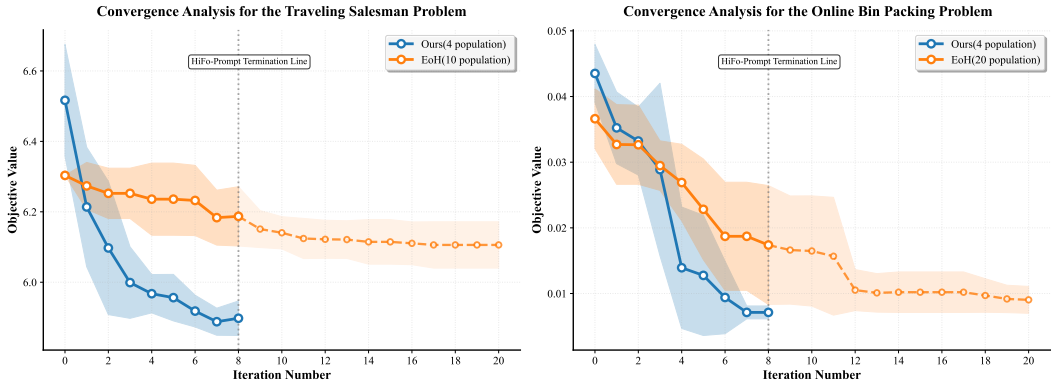

Figure 3: Convergence Analysis for TSP and Online BPP.

Table 9: Results with different LLMs.

Method	TSP-construction	Online BPP
qwen2.5-max	0.274%	0.584%
GPT-4o-mini	3.835%	0.634%
DeepSeek-v3	4.306%	0.694%
Deepseek-r1	5.343%	1.248%
claude3.5-sonnet	5.036%	0.734%

efficiency. The maximum generations parameter achieves a near-optimal solution at 12 generations, with diminishing returns observed beyond this point. An insight pool size of 30 effectively balances diversity maintenance and noise avoidance. The dual-threshold parameters at their default values optimally balance exploration and exploitation. Overall, the algorithm essentially converges within 6–8 generations, confirming that the current default configuration represents the optimal choice, ensuring solution quality while maintaining computational efficiency.

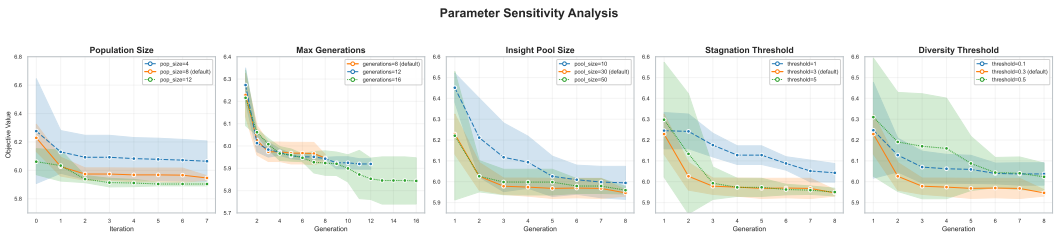

Figure 4: Parameter Sensitivity Analysis.

C.7 Consumption of Time and Token

Compared to other LLM-based AHD methods, our approach demonstrates advantages in both time and token consumption, with particularly significant reductions in computational time. Based on `qwen2.5-max` model, we calculated the runtime and token consumption of various methods across three problem domains: TSP-construction, Online BPP, and BO-CAF.

Table 10: Time and token consumption with different methods.

Methods	Consumption	TSP-construction	Online BPP	BO-CAF
EoH	Time	1.2h	1h	2h
	Input Token	0.8M	0.5M	1.2M
	Output Token	0.2M	0.2M	0.5M
ReEvo	Time	2h	-	-
	Input Token	1.1M	-	-
	Output Token	0.4M	-	-
MCTS-AHD	Time	4h	3h	14h
	Input Token	1M	1M	1.3M
	Output Token	0.3M	0.2M	0.6M
Ours	Time	40 min	36min	1h
	Input Token	0.5M	0.28M	0.8M
	Output Token	0.2M	0.12M	0.2M

D Algorithm Details

This section provides detailed pseudocode for the core components of the HiFo-Prompt framework. We first present the main algorithm, followed by the specific implementations for the survival selection and parent selection mechanisms.

D.1 Main Framework Algorithm

Algorithm 3 outlines the complete, end-to-end procedure for the HiFo-Prompt framework. It details the main evolutionary loop, starting from population initialization and proceeding through iterative generations. Each generation consists of four primary phases: (1) Foresight and Hindsight, where the Evolutionary Navigator and Insight Pool generate the control vector; (2) Construction, where the LLM generates new heuristics under the guidance of the control vector; (3) Evaluation & Feedback, where offspring are evaluated and the Insight Pool learns from the results; and (4) Selection, where the next generation's population is formed.

Algorithm 1 Greedy Population Management

Input: A population of individuals P, Target population size N_{target}
Output: A new population P_{new} with size at most N_{target}

$P_{valid} \leftarrow$ Filter out individuals from P with null objective values
Initialize an empty list P_{unique}
Initialize an empty set of seen objectives O_{seen}
for each individual $i \in P_{valid}$ **do**
  if $i.objective \notin O_{seen}$ **then**
    Add i to P_{unique}
    Add $i.objective$ to O_{seen}
  end if
end for
$N_{actual} \leftarrow \min(N_{target}, |P_{unique}|)$
Sort P_{unique} in ascending order based on objective values.
$P_{new} \leftarrow$ the first N_{actual} individuals from sorted P_{unique}
return P_{new}

D.2 Survival Selection Mechanism

To maintain a constant population size across generations, we employ a survival selection mechanism after new offspring have been generated and evaluated. Algorithm 1 details this procedure. It implements a greedy, elitist strategy that first filters out any invalid or unsuccessfully evaluated

Algorithm 2 Rank-Based Parent Selection

Input: Sorted population P (best to worst), Number of parents to select m
Output: A list of selected parents $P_{parents}$

Let $N \leftarrow$ size of P
Initialize an empty list for probabilities $W \leftarrow \{\}$
for $k = 0$ to $N - 1$ **do**
    Calculate probability $w_k \propto 1/(k + 1 + N)$ {Rank-based weighting}
    Append w_k to W
end for
Select m individuals from P with replacement, using probabilities W, to form $P_{parents}$.
return $P_{parents}$

individuals. It then ensures uniqueness based on objective values to maintain diversity in the performance space. Finally, it sorts the unique, valid individuals by their performance and truncates the population to the target size N_{target}, ensuring that only the highest-performing heuristics are retained for the next generation.

D.3 Parent Selection Mechanism

For the generative operators that require parent heuristics (e.g., crossover and mutation), a parent selection mechanism is used to choose individuals from the current population. Algorithm 2 describes the rank-based selection method used in our framework. This approach assigns a selection probability to each individual that is inversely proportional to its performance rank. This ensures that higher-performing individuals are more likely to be selected as parents, creating a strong selection pressure that guides the search towards promising regions of the heuristic space, while still allowing lower-ranked individuals a chance to contribute.

E Limitation and Future Work

E.1 Limitation

While HiFo-Prompt demonstrates significant advancements in automated heuristic design, its current architecture possesses inherent limitations that define the boundaries of its present capabilities.

First, the core decision-making mechanism of the Evolutionary Navigator is predicated on a static, handcrafted control logic. This rule-based system, while interpretable, lacks the capacity for self-adaptation. Its fixed thresholds for stagnation, progress, and diversity are calibrated for the evaluated problems but may not be universally optimal, potentially constraining its performance on novel problem landscapes or over different evolutionary timescales. The Navigator can react to predefined states but cannot learn or refine its control strategy from experience.

Second, the knowledge evolution within the Insight Pool is fundamentally intra-task. The framework excels at capturing, refining, and reusing design principles within the context of a single optimization problem. However, the true generalizability of these learned insights across different problem domains remains an unevaluated and open question. We have not yet systematically validated whether insights distilled from solving one problem class can effectively bootstrap the learning process on a structurally different, unseen one.

E.2 Future Work

The limitations mentioned above naturally chart a course for several high-impact avenues for future research, aimed at enhancing the framework's autonomy, generality, and strategic depth.

A primary research thrust will be to transcend the Evolutionary Navigator's current heuristic-driven design by developing it into a learned metacontroller. We propose parameterizing its control function and leveraging techniques from the domain of Meta-Reinforcement Learning (Meta-RL). In this paradigm, the Navigator would operate as a high-level agent, learning a control policy that dynamically maps observational data from the evolutionary process—such as population diversity

metrics, fitness landscape topology, and rates of convergence—to a nuanced control vector. This vector would modulate critical evolutionary parameters in real-time, including operator probabilities, selection pressure, and even strategic alterations to the LLM prompts themselves. The reward signal for this meta-agent would be carefully engineered to reflect overarching goals of evolutionary efficiency, such as maximizing the rate of fitness improvement or minimizing the computational cost to reach a performance threshold. Successfully implementing this would elevate the framework from a system guided by static rules to one capable of learning and deploying its own adaptive control strategies online, thereby achieving a superior order of autonomy and problem-specificity.

Another critical, complementary direction is the systematic investigation of inter-task knowledge transfer, intending to evolve the framework from a single-task solver into a more general algorithmic discovery platform. This research will proceed along two parallel vectors. First, we will conduct a rigorous empirical evaluation of the framework's zero-shot and few-shot transfer capabilities. By applying a mature Insight Pool—developed for a source task—to novel target domains, we can precisely quantify the generality of the learned principles and measure the extent to which discovery costs can be amortized across problems. Second, we will pioneer the exploration of more abstract and powerful knowledge representations that transcend the inherent ambiguities of natural language strings. This includes investigating structured canonical forms, such as predicate logic or probabilistic program sketches, and rich semantic representations like knowledge graphs. The central hypothesis is that such formalisms can more effectively decouple a core algorithmic invariant from its domain-specific instantiation, thereby creating a more robust foundation for seamless and compositional cross-domain knowledge transfer.

F PROMPTS WITH FORESIGHT AND HINDSIGHT

This section provides the specific, concrete templates of the prompts used to guide the LLM, offering a transparent and reproducible view of the core interaction engine at the heart of our HiFo-Prompt framework. The 'HiFo' (Hindsight-Foresight) name is not arbitrary; it directly reflects our core methodology: the strategic deployment of distinct, context-aware prompts at different stages of the evolutionary process. 'Foresight' prompts are strategically employed during the initial generation and creative mutation stages, encouraging the LLM to explore a diverse space of novel heuristic possibilities. In stark contrast, 'Hindsight' prompts play a critical role in reflection and data-driven refinement, leveraging empirical performance feedback from past evaluations to methodically prune the search space and systematically improve promising solutions.

To ground these abstract concepts and make our prompts tangible, we anchor our examples in the canonical Online Bin Packing (OBP) problem. The objective in OBP is to assign an incoming sequence of items of varying sizes into a minimum number of fixed-capacity bins, with the crucial constraint that each item must be placed without knowledge of future items. Within this problem context, our framework's specific task is to evolve a high-performance Python scoring function, score(item, bin), which acts as the core decision-making logic. This function evaluates the suitability of a specific candidate bin for an incoming item. A higher returned score signifies a more desirable placement, and the overarching control logic of the framework places the item in the bin that yields the maximum score. The following subsections provide verbatim templates for each distinct phase of the evolutionary process, clearly demonstrating how the system's guidance dynamically shifts its focus—from broad exploration to focused exploitation—as the search progresses.

F.1 INITIAL PROMPT STRATEGY I1

The i1 operator uses the following prompt template to generate the initial population of heuristics. The guidance provided to the LLM at this stage comes from the initial state of the framework's components, including a set of high-quality seed insights.

Prompt for Operator i1

Given a sequence of items and a set of identical bins with a fixed capacity, you need to assign each item to a bin to minimize the total number of bins used. The task can be solved step-by-step by taking the next item and deciding which bin to place it in based on a score.
First, describe your new algorithm and main steps in one sentence. The description must be inside a brace. Next, implement it in Python as a function named **score**.
This function should accept 2 input(s): `'item'`, `'bins'`.
The function should return 1 output(s): `'scores'`.
The **score** function is designed to evaluate the placement options for a given item. It takes the item to be placed and the current list of bins as input. It returns a list of numerical scores, with each score corresponding to a bin in the input list. This list of scores guides the heuristic in selecting the most suitable bin for the item according to the generated logic.
Consider these successful design principles I've observed recently:

- *<A successful design principle from Insights pool>*
- *<Another successful design principle...>*

For the evolutionary regime, please pay special attention to: *<A specific instruction from Design Directive>*
Depending on the regime, try significantly different parameter values (focus_exploration), or fine-tune existing ones (focus_exploitation), or combine both strategies (balanced_search).
Do not give additional explanations.

F.2 Recombination Prompt Strategy E1

The following template for the e1 operator is a representative example of a prompt used during the main evolutionary loop. It demonstrates how parent heuristics and the full, dynamic guidance from the meta-cognitive components are integrated.

Prompt for Operator e1

Given a sequence of items and a set of identical bins with a fixed capacity, you need to assign each item to a bin to minimize the total number of bins used. The task can be solved step-by-step by taking the next item and deciding which bin to place it in based on a score.
I have k existing algorithms with their codes as follows:
No.1 algorithm and the corresponding code are:
<Description of the first algorithm>
<The Python code implementation of the first algorithm>
...
No.k algorithm and the corresponding code are:
<Description of the last algorithm>
<The Python code implementation of the last algorithm>
Please help me create a new algorithm that has a totally different form from the given ones.
First, describe your new algorithm and main steps in one sentence, enclosed in braces {}. Next, implement it in Python as a function named **score**. This function should accept 2 input(s): <'item', 'bins'>. The function should return 1 output(s): <'scores'>. *<Additional info on inputs & outputs> <Other constraints or requirements>*
Consider these successful design principles I've observed recently:

- *<A successful design principle from Insights pool>*
- *<Another successful design principle...>*

For the evolutionary regime, please pay special attention to: *<A specific Design Directive for promoting structural novelty, preserving diversity, and avoiding premature convergence>*
Depending on the regime, try significantly different parameter values (focus_exploration), or fine-tune existing ones (focus_exploitation), or combine both strategies (balanced_search).
Do not give additional explanations.

Prompt for Operator m2

Given a sequence of items and a set of identical bins with a fixed capacity, you need to assign each item to a bin to minimize the total number of bins used. The task can be solved step-by-step by taking the next item and deciding which bin to place it in based on a score.
I have one algorithm with its code as follows:
Algorithm description: *<Description of the parent algorithm>*
Code:
`<The Python code implementation of the parent algorithm>`
Please identify the main algorithm parameters and assist me in creating a new algorithm that has different parameter settings of the score function provided.
First, describe your new algorithm and main steps in one sentence, enclosed in braces {}. Next, implement it in Python as a function named **score**. This function should accept 2 input(s): `<'item', 'bins'>`. The function should return 1 output(s): `<'scores'>`. *<Additional info on inputs & outputs> <Other constraints or requirements>*
Consider these successful design principles I've observed recently:

- *<A successful design principle from Insights pool>*
- *<Another successful design principle>*

When adjusting parameters, please pay special attention to: *<A specific Design Directive for parameter tuning>*
Depending on the regime, try significantly different parameter values (focus_exploration), or fine-tune existing ones (focus_exploitation), or combine both strategies (balanced_search).
Do not give additional explanations.

Prompt for Operator m1

Given a sequence of items and a set of identical bins with a fixed capacity, you need to assign each item to a bin to minimize the total number of bins used. The task can be solved step-by-step by taking the next item and deciding which bin to place it in based on a score.
I have one algorithm with its code as follows:
Algorithm description: *<Description of the parent algorithm>*
Code:
`<The Python code implementation of the parent algorithm>`
Please assist me in creating a new algorithm that has a different form but can be a modified version of the algorithm provided.
First, describe your new algorithm and main steps in one sentence, enclosed in braces {}. Next, implement it in Python as a function named **score**. This function should accept 2 input(s): `<'item', 'bins'>`. The function should return 1 output(s): `<'scores'>`. *<Additional info on inputs & outputs> <Other constraints or requirements>*
Consider these successful design principles I've observed recently:

- *<A successful design principle from Insights pool>*
- *<Another successful design principle>*

For this mutation, please pay special attention to: *<A specific Design Directive for mutation>*
Depending on the regime, try significantly different parameter values (focus_exploration), or fine-tune existing ones (focus_exploitation), or combine both strategies (balanced_search).
Do not give additional explanations.

F.3 Recombination Prompt Strategy E2

The e2 operator focuses on "motivated recombination." It prompts the LLM to first identify a common "backbone" or core principle shared by the parent heuristics and then to create a new, improved algorithm based on that shared foundation.

Prompt for Operator e2

Given a sequence of items and a set of identical bins with a fixed capacity, you need to assign each item to a bin to minimize the total number of bins used. The task can be solved step-by-step by taking the next item and deciding which bin to place it in based on a score.
I have k existing algorithms with their codes as follows:
No.1 algorithm and the corresponding code are:
<Description of the first algorithm>
<The Python code implementation of the first algorithm>
...
No.k algorithm and the corresponding code are:
<Description of the last algorithm>
<The Python code implementation of the last algorithm>
Please help me create a new algorithm that has a totally different form from the given ones but can be motivated from them.
Firstly, identify the common backbone idea in the provided algorithms. Secondly, based on the backbone idea, describe your new algorithm and main steps in one sentence, enclosed in braces {}. Thirdly, implement it in Python as a function named **score**. This function should accept 2 input(s): <'item', 'bins'>. The function should return 1 output(s): <'scores'>. *<Additional info on inputs & outputs>* *<Other constraints or requirements>*
Consider these successful design principles I've observed recently:

- *<A successful design principle from Insights pool>*
- *<Another successful design principle>*

For this recombination, please pay special attention to: *<A specific Design Directive for simplification prune low-impact features>*
Depending on the regime, try significantly different parameter values (focus_exploration), or fine-tune existing ones (focus_exploitation), or combine both strategies (balanced_search).
Do not give additional explanations.

F.4 Mutation Prompt Strategy M1

The M1 operator implements a form of targeted mutation, generating a single-parent descendant by refining its most critical components. It performs surgical modifications to elements like scoring rules or parameter weights, while meticulously safeguarding the parent's established, high-performing logic. This process is not random; it is guided by a synthesis of recent, high-utility design "insights" and a high-level strategic directive. By injecting controlled, purposeful diversity, M1 enables the search to escape local optima without dismantling the parent's effective architecture. This deliberate balance between exploitation (refining what works) and exploration (seeking novelty) is crucial for accelerating convergence towards superior heuristics.

F.5 Mutation Prompt Strategy M2

The M2 operator implements *parameter mutation*, a targeted process to modify a heuristic's numerical behavior. It instructs the LLM to first deconstruct the parent's score function to identify its key hyperparameters. Following this analysis, the model is prompted to generate a new set of parameter values. This generation can either explore novel configurations through significant changes or fine-tune existing ones via subtle adjustments. This surgical approach methodically injects parametric diversity into the population while preserving the integrity of the core algorithmic logic.

F.6 Mutation Prompt Strategy M3

The M3 operator is designed for *structural simplification* to enhance heuristic robustness and combat overfitting. It instructs the LLM to perform a critical analysis of the parent score function, specifically targeting components suspected of being over-specialized to in-distribution data. These potentially brittle or overly complex segments are then strategically pruned or streamlined. The outcome is a more parsimonious and computationally lean implementation that is theorized to exhibit superior generalization to out-of-distribution scenarios, all while preserving the original function signature to ensure architectural compatibility.

Prompt for Operator m2

Given a sequence of items and a set of identical bins with a fixed capacity, you need to assign each item to a bin to minimize the total number of bins used. The task can be solved step-by-step by taking the next item and deciding which bin to place it in based on a score.
I have one algorithm with its code as follows:
Algorithm description: *<Description of the parent algorithm>*
Code:
`<The Python code implementation of the parent algorithm>`
Please identify the main algorithm parameters and assist me in creating a new algorithm that has different parameter settings of the score function provided.
First, describe your new algorithm and main steps in one sentence, enclosed in braces {}. Next, implement it in Python as a function named **score**. This function should accept 2 input(s): `<'item', 'bins'>`. The function should return 1 output(s): `<'scores'>`. *<Additional info on inputs & outputs> <Other constraints or requirements>*
Consider these successful design principles I've observed recently:

- *<A successful design principle from Insights pool>*
- *<Another successful design principle>*

When adjusting parameters, please pay special attention to: *<A specific Design Directive for parameter tuning>*
Depending on the regime, try significantly different parameter values (focus_exploration), or fine-tune existing ones (focus_exploitation), or combine both strategies (balanced_search).
Do not give additional explanations.

Prompt Template for Insight Extraction

The following are core descriptions and/or code of high-performance optimization algorithms evolved recently:

Algorithm 1: *<Natural language description and/or code of elite individual 1>*
Algorithm 2: *<Natural language description and/or code of elite individual 2>*
Algorithm n: ... *(and so on for the top 30% of the population)*

Please extract 1-2 concise, generic, and performance-positive **[design principles]** or **[effective patterns]** from the above algorithms. These principles should be applicable to various combinatorial optimization problems, not just the specific problem domain. When formulating these principles, it is essential to draw insights from *both* the conceptual natural language descriptions *and* their corresponding code implementations. Focus on identifying the underlying strategic design choices and algorithmic methodologies rather than superficial characteristics or specific implementation minutiae.

Each principle/pattern must be expressed as an independent sentence in the following format:

- *Balance local optimization with global solution structure when making decisions.*
- *Prioritize choices that maintain flexibility for future decision-making steps.*
- *Implement adaptive mechanisms that respond to problem instance characteristics.*

Provide only the list of principles, without any preamble or other explanatory text.

G Managing Foresight and Hindsight Knowledge

G.1 Hindsight Evolution via Insight Distillation

To continually enrich our system's understanding of effective optimization strategies, we employ a Large Language Model (LLM) to distill high-level design principles from high-performing algorithms. This process is guided by a structured prompt, shown below, which provides the LLM

with the descriptions and/or code of elite solutions discovered during an evolutionary run. The core instruction tasks the model with synthesizing concise, generalizable, and performance-positive patterns, drawing insights from both the conceptual descriptions and the code implementations. This automated extraction mechanism allows our system to learn from its own successes and progressively build a more sophisticated knowledge base.

Seed Insights

- *Design adaptive hybrid meta-heuristics synergistically fusing multiple search paradigms and dynamically tune operator parameters based on search stage or problem features.*
- *Employ machine learning to mine problem structures and use learned insights to intelligently bias towards promising search regions.*
- *Explore objective function engineering by introducing auxiliary objectives or dynamically adjusting weights to reshape the search landscape.*
- *Construct problem-specialized solution representations and co-design dedicated operators to fully leverage the representation's structure.*
- *Implement intelligent diversification based on solution feature space analysis to systematically target uncovered regions and escape local optima.*

G.2 Insight Seed Pool

At the inception of the framework's execution, the LLM bootstraps the heuristic generation process by drawing from a curated repository of Seed Insights, a mechanism designed to mitigate the classic "cold start" problem and channel the model's creativity. These insights, which encapsulate established principles and canonical rules-of-thumb distilled from decades of human expertise in heuristic design—such as the "Shortest Processing Time" principle in scheduling or the "Nearest Neighbor" concept in routing—are not rigid constraints but rather high-level conceptual guidelines. They are strategically injected into the foundational prompts, often framed within a dedicated "human knowledge" block, to act as an intellectual scaffold. This initial infusion of well-vetted knowledge serves to ground the search, preventing the generation of naive or logically flawed heuristics and providing a potent directional bias that immediately steers the early stages of algorithmic evolution away from vast, unproductive regions of the design space. By ensuring the process begins not from a tabula rasa but from a high-quality, well-founded starting point, these seed insights exert a persistent influence that accelerates convergence and significantly enhances the quality and novelty of all subsequent automated discovery.

G.3 The Directive Pool for Foresight

To operationalize the Evolutionary Navigator's high-level strategy, we map the chosen regime—Exploration, Exploitation, or Balance—to a specific Design Directive. Each directive is a fine-grained textual instruction, uniformly sampled from a predefined pool corresponding to the active regime. This sampled directive is then integrated into the generation prompt. This two-tiered mechanism facilitates fine-grained control over the generation process, ensuring model outputs are precisely aligned with the overarching evolutionary objective.

Design Directive

- **Balance:**
  - *Optimizing objective function evaluation criteria.*
  - *Considering the long-term impact of current decisions.*
  - *Balancing local optimality with global search strategies.*
  - *Improving algorithm robustness across different problem instances.*
  - *Managing computational complexity and time efficiency.*
- **Exploitation:**
  - *Refining core evaluation and scoring functions.*
  - *Fine-tuning critical algorithm parameters and thresholds.*
  - *Improving the precision of existing heuristics and rules.*
  - *Reducing unnecessary computational overhead.*
- **Exploration:**
  - *Exploring novel solution construction methodologies.*
  - *Investigating alternative problem decomposition approaches.*
  - *Introducing new randomization or adaptive mechanisms.*
  - *Experimenting with hybrid strategy combinations.*

H THE USE OF LARGE LANGUAGE MODELS (LLMS)

Throughout the drafting and revision process of this manuscript, we employed Google's Gemini large language model as an advanced writing and editing tool. Its use was strictly limited to the refinement of the language and presentation of our pre-existing ideas and research. The model's contributions include: (1) correcting grammatical, spelling, and punctuation errors; (2) rephrasing complex sentences to improve readability and precision; and (3) suggesting adjustments to tone and style to ensure consistency with academic standards.

Critically, the LLM was not used for any substantive intellectual contribution. All aspects of the research, including the formulation of research questions, literature review, methodology, data analysis, and the drawing of conclusions, were conducted exclusively by the human authors. Each suggestion provided by the LLM was critically evaluated by the authors for accuracy and appropriateness, and we retain full responsibility for all claims, arguments, and the final articulation of the work.

Algorithm 3 HiFo-Prompt

Input: Problem definition $\mathcal{F}$, optional seed algorithms $\mathcal{P}_{\text{seed}}$
Parameter: Population size N_{pop}, number of generations $G_{\max}$, set of evolutionary foundation prompt strategies $\mathcal{O}$ with weights $\mathcal{W}$, number of parents m
Output: The final population of evolved algorithms $\mathcal{P}$

Initialize Population $\mathcal{P}$:
if $\mathcal{P}_{\text{seed}}$ is provided **then**
  $\mathcal{P} \leftarrow$ Evaluate and populate from $\mathcal{P}_{\text{seed}}$
else
  $\mathcal{P} \leftarrow \emptyset$
  while $|\mathcal{P}| < N_{\text{pop}}$ **do**
    $\text{Ind}_{\text{new}} \leftarrow$ Generate an initial algorithm via LLM
    Evaluate fitness of Ind_{new} using $\mathcal{F}$
    Add valid Ind_{new} to $\mathcal{P}$
  end while
end if
Initialize Insight Pool $\mathcal{M}_{\text{insight}}$ with a set of default design principles
Initialize Evolutionary Navigator $\mathcal{C}_{\text{meta}}$

for $g = 1$ **to** $G_{\max}$ **do**
  Update $\mathcal{C}_{\text{meta}}$ with current population state
  if a condition for wisdom extraction is met **then**
    $\mathcal{P}_{\text{top}} \leftarrow$ Select top-performing individuals from $\mathcal{P}$
    new_tips $\leftarrow$ Prompt LLM to extract generic design principles from $\mathcal{P}_{\text{top}}$
    Add new_tips to $\mathcal{M}_{\text{Insight}}$
  end if

  for each foundation prompt strategy $o \in \mathcal{O}$ **do**
    if random() $< \mathcal{W}[o]$ **then**
      (evolutionary regime, design directive) $\leftarrow$ $\mathcal{C}_{\text{meta}}$.get_guidance() {Get strategic guidance}
      insights $\leftarrow$ $\mathcal{M}_{\text{Insight}}$.get_insights() {Get inspiring principles}
      $\mathcal{P}_{\text{parents}} \leftarrow$ Select m parents from $\mathcal{P}$
      $\text{Ind}_{\text{new}} \leftarrow$ Generate offspring via LLM using o, $\mathcal{P}_{\text{parents}}$, guidance, and insights
      Evaluate fitness of Ind_{new} using $\mathcal{F}$
      if Ind_{new} is valid and not duplicated **then**
        effectiveness $\leftarrow$ Calculate effectiveness of tips based on Ind_{new}'s performance
        $\mathcal{M}_{\text{Insight}}$.update_insight_stats(insights, effectiveness) {Feedback loop}
        Add Ind_{new} to $\mathcal{P}$
      end if
    end if
  end for
  $\mathcal{P} \leftarrow$ SurvivalSelection($\mathcal{P}$, N_{pop}) {Manage population size}
end for
return Best individuals from $\mathcal{P}$