DIRECTING TO DIVERSIFY: MULTI-DIRECTIONAL SEARCH FOR AUTOMATIC HEURISTIC DESIGN

Anonymous authors
Paper under double-blind review

ABSTRACT

Large language models (LLMs) have demonstrated strong potential to automate heuristic design for NP-hard combinatorial optimization (CO) tasks without human intervention. Recent methods predominantly rely on evolutionary computation frameworks, where heuristic diversification is mainly induced through EC-like operators or downstream selection, while the direction of heuristic transformation remains largely unharnessed. In this paper, we propose PRS, a multi-directional **P**erturbation–**R**ewrite **S**earch (**PRS**) framework for automatic heuristic design that factorizes heuristic transformation through directional search primitives at the generation level. Specifically, PRS operates on heuristic ideas through three complementary directions: expansion toward richer and more expressive designs, compression toward simpler and less specialized designs, and steering for fine-grained refinement, together enabling broader and more sustained exploration of diverse regions in the heuristic design space. PRS further organizes candidate exploration within a breadth-first search-based process with adaptive operator control, ensuring thorough candidate utilization and sustained exploration, thereby maintaining strong search dynamics while preventing premature convergence. Extensive empirical results across diverse CO tasks show that PRS consistently discovers high-quality heuristics, demonstrating the effectiveness and generality of explicitly directional exploration design.

1 INTRODUCTION

Automatic heuristic design (AHD) aims to automatically discover high-performing heuristic algorithms for a wide range of optimization problems (He et al., 2011; Dahmani et al., 2024; Romera-Paredes et al., 2024; Novikov et al., 2025; Rajendran, 1993; Mihoubi et al., 2021). Despite its importance, traditional AHD methods, such as genetic programming, remain fundamentally constrained by their reliance on manually designed, problem-specific operators, which limits scalability and restricts the exploration of the heuristic search space. The recent emergence of large language models (LLMs) has opened a promising new direction for AHD (Liu et al., 2024b), enabling automated heuristic generation through LLM-based code synthesis and reasoning.

Most existing LLM-based AHD approaches build upon evolutionary computation (EC) frameworks, where heuristics are iteratively improved using LLMs (Romera-Paredes et al., 2024; Liu et al., 2024a; Ye et al., 2024; Dat et al., 2025). Maintaining diverse heuristic candidates is important for preventing premature concentration around a narrow family of designs and for preserving access to alternative regions that may become useful at different stages of search. Recent works have attempted to enhance the search process with more structured control, such as diversity-aware population management (Dat et al., 2025) and feature-assisted partitioning of the abstract heuristic search space (Hu & Zhang, 2025). For example, MCTS-AHD (Zheng et al., 2025) replaces population-based evolutionary search with Monte Carlo Tree Search, using progressive widening to regulate exploration and multiple mutation-style actions to promote diversification.

Despite these advances, existing approaches mainly encourage diversification through population management, candidate selection, similarity measures, or search-space partitioning. Such mechanisms regulate which candidates are retained, preferred, or grouped after or around generation, but they do not explicitly structure the generation process itself around different directions of heuristic change. As a result, heuristic diversification is typically an emergent consequence of generic mutation,

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

crossover, reflection, or novelty pressure, rather than a directly controlled search primitive. This leaves an important gap: existing AHD frameworks lack explicit generation primitives for deliberately diversifying heuristic ideas along complementary semantic directions.

In this work, we argue that diversification should be controlled directly at the level of heuristic generation. To this end, we propose **Perturbation–Rewrite Search (PRS)**, a multi-directional search framework that explicitly factorizes heuristic generation into complementary search primitives over the idea space. PRS introduces three perturbation strategies—*expansion*, *compression*, and *steering*—which deliberately transform a parent heuristic toward richer, simpler, and locally refined designs, respectively. Rather than using diversity only as a population property or selection signal, PRS controls *how* new heuristic ideas are produced before they are realized as executable implementations. These abstraction-level perturbations are then instantiated through an LLM-based rewriting process that translates the modified heuristic specification into executable code while preserving its intended decision logic. Together, this primitive-level directional diversification enables broader and more sustained coverage of structural regions in the heuristic space.

To further enhance search effectiveness, PRS organizes candidate exploration within a breadth-first search (BFS) framework. By expanding multiple candidates layer by layer, PRS improves candidate utilization while preventing early domination by a small subset of heuristics. This search strategy is coupled with an adaptive exploration control mechanism, enabling sustained exploration and robustness against stagnation.

Our contributions are summarized as follows:

- We introduce Perturbation–Rewrite Search (PRS) for automatic heuristic design, which makes directional heuristic transformation an explicit generation-level search primitive for structured exploration of the heuristic idea space.
- We develop a multi-directional perturbation mechanism with expansion, compression, and steering, which deliberately transforms heuristic ideas toward richer, simpler, and locally refined designs, respectively. PRS further integrates bounded BFS-based search and adaptive operator control to allocate search effort across these complementary directions.
- We comprehensively evaluate PRS across 11 combinatorial optimization tasks and show that it consistently discovers high-quality heuristics, demonstrating strong effectiveness and generality compared to state-of-the-art LLM-based AHD methods.

2 PRELIMINARY

AHD. Let $\mathcal{P}$ denote a problem class, and let $\mathcal{S}(p)$ be the feasible solution set for an instance $p \in \mathcal{P}$. A heuristic space is defined as $\mathcal{H}$, where each heuristic $h \in \mathcal{H}$ induces a mapping $h : \mathcal{P} \to \bigcup_{p\in\mathcal{P}} \mathcal{S}(p)$, and for any $p \in \mathcal{P}$, $h(p) \in \mathcal{S}(p)$. For a given validation set $\mathcal{D} \subseteq \mathcal{P}$ and an objective function $f : \mathcal{S} \to \mathbb{R}$, the goal of AHD is to find an optimal heuristic $h^\star \in \mathcal{H}$ that maximizes expected performance g:

$$h^\star = \arg\max_{h\in\mathcal{H}} g(h) = \arg\max_{h\in\mathcal{H}} \mathbb{E}_{x\sim\mathcal{D}} f(h(x)). \tag{1}$$

LLM-based AHD. LLM-based AHD leverages LLMs to navigate the typically vast heuristic space $\mathcal{H}$ in CO. In practice, the designed heuristic often constitutes only a key component of a broader optimization algorithm. For example, in Travelling Salesman Problem (TSP), existing LLM-based AHD methods (Liu et al., 2024a) typically focus on designing step-by-step constructive heuristics that select the next node based on the current partial tour. Beyond constructive frameworks, recent studies have also explored learning heuristics that update weighted distance matrices within the broader Guided Local Search (GLS) algorithm, thereby more effectively guiding local search decisions (Ye et al., 2024; Zheng et al., 2025).

A prominent line of LLM-based AHD methods follows an evolutionary paradigm (Liu et al., 2023; Romera-Paredes et al., 2024; Liu et al., 2024a; Ye et al., 2024). Using EC-style operators such as crossover and mutation, these methods progressively explore the heuristic space, with more recent extensions further strengthening this paradigm. For example, HiFo-Prompt (Chen et al., 2026) introduces hierarchical prompt-level guidance through hindsight–foresight control and knowledge distillation for faster and more efficient heuristic search.

Despite differences in evolution strategies, these frameworks remain rooted in EC-like search, where search operators typically recombine multiple heuristics in crossover to inherit parental features, together with mutation-style operators to promote diversification. Consequently, the direction of heuristic change is largely uncontrolled by the search mechanism itself: different transformation patterns may emerge during evolution, but they are not explicitly harnessed as distinct search primitives. In contrast, PRS directly structures heuristic generation through complementary transformation directions, enabling multi-directional exploration without relying primarily on recombination of existing structures.

Breadth First Search. Breadth-first search is a fundamental graph traversal strategy that explores states layer by layer by maintaining a frontier of candidate states at each depth. It has been widely applied in various domains, ranging from social network analysis (Beamer et al., 2013) to image processing (Silvela & Portillo, 2001). Recent work such as Tree-of-Thoughts (Yao et al., 2023) instantiates BFS in the space of intermediate reasoning trajectories, where each layer corresponds to a single reasoning step and the frontier is pruned to a fixed breadth. In a generic BFS procedure, the search starts from an initial state and expands all states in the current frontier using a predefined expansion operator. The resulting states are then scored, and a subset of top-ranked states is retained as the frontier for the next layer. This process repeats until a stopping criterion, such as a maximum search depth or a computational budget, is reached.

3 METHODOLOGY

In this section, we introduce PRS, a multi-directional perturbation–rewrite search framework that makes directional heuristic transformation an explicit search primitive for automatic heuristic design. PRS explores the heuristic space by applying multi-directional perturbations to heuristic ideas and then rewriting the corresponding implementations to realize the modified heuristic as executable candidates. At the search level, PRS maintains a heuristic pool and organizes candidate expansion in a breadth-first manner with selective pruning to enhance candidate utilization. In addition, PRS incorporates an adaptive exploration controller to regulate the exploration–exploitation trade-off and mitigate premature convergence during heuristic search.

PRS Pipeline. The overall pipeline of PRS is illustrated in Fig. 1. Specifically, PRS starts the heuristic search by initializing a heuristic pool, which maintains a set of parent heuristics with a selection mechanism, and terminates after a certain condition. We introduce a novel multi-directional perturbation–rewrite operator that explores the heuristic space through complementary transformation directions. It includes expansion, compression, and steering, which transform heuristic ideas along complementary directions in the heuristic design space. These operators are used to expand parent nodes within a BFS framework, as illustrated in Fig 2. Regulated by an adaptive exploration controller, these operators jointly facilitate an effective search for the optimal heuristic h^* for a given task. After each completed BFS tree, random sampling is applied to the heuristic pool to promote diversity and retain temporarily inferior yet potentially promising candidates, while always preserving the current best heuristic. The complete PRS algorithm and associated prompts are presented in Algorithm 1 and Appendix D.1, respectively. In the following subsections, we describe the perturbation and rewrite process, the BFS-based search structure, and the adaptive exploration controller in detail.

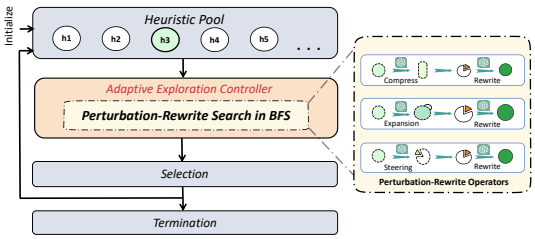


Figure 1: An illustration of PRS pipeline.

3.1 PERTURBATION-REWRITE OPERATOR

PRS introduces a multi-directional perturbation–rewrite process composed of a perturbation step followed by an LLM-based rewrite step. Rather than directly modifying executable heuristics through carefully crafted EC operators, PRS applies directional perturbations to heuristic ideas and subsequently rewrites the corresponding implementations to realize the modified heuristics as executable candidates. This design separates the intended algorithmic modification from its low-level implementation details, enabling controlled changes in heuristic structure while preserving the

executability and semantic coherence required for black-box heuristic evaluation. We denote h_p as the parent heuristic used for perturbation–rewrite search, and elaborate the perturbation and rewriting strategies below.

Multi-directional Perturbation. The purpose of multi-directional perturbation is to explore the heuristic space through complementary transformation directions, thereby overcoming the localized perturbation nature of traditional mutation operators. This design is motivated by the fact that heuristic quality does not vary monotonically with structural complexity: overly simple heuristics may lack expressive power, whereas overly complex ones may become bulky, computationally expensive, or over-fit to in-distributional data (IDD). Accordingly, PRS employs three complementary perturbation actions that address different search needs: expansion increases heuristic expressiveness, compression counteracts excessive specialization, and steering provides refinement around promising heuristic structures.

Expansion (Φ_θ^{expn}). The expansion operator explicitly prompts LLMs to enrich the parent heuristic by introducing additional decision signals, interactions, conditional rules, or adaptive mechanisms, deliberately increasing its structural complexity and expressive capacity. Specifically, expansion instructs LLMs to inject new structural components, such as additional scoring terms or conditional logic. The operator further specifies how these additions affect heuristic decisions rather than simply appending independent components. From an optimization perspective, expansion biases heuristic structures toward more elaborate and expressive programmatic forms, such as deeper control flow or richer operator compositions, thereby increasing its expressive capacity and conceptual richness.

Compression (Φ_θ^{comp}). Complementary to expansion, PRS employs a compression operator that biases LLMs toward reducing the structural complexity of the current parent heuristic by removing redundant mechanisms or over-specialized components that may hinder out-of-distribution (OOD) generalization (e.g., structures tightly coupled to specific problem sizes or data distributions). Intuitively, compression may disrupt the established advantages of the parent heuristic, as simplification can often lead to short-term performance degradation. However, from a broader perspective, it may mitigate the accumulation of dominant structural priors and excessive complexity, acting as a potential refactoring mechanism in the heuristic space that counteracts over-specialization and promotes more robust heuristic representations.

Steering (Φ_θ^{steer}). While expansion and compression enable substantial shifts toward richer or simpler heuristic designs, they are less suited for fine-grained exploitation of an already promising heuristic idea. Steering complements these directional perturbations by performing localized refinements while approximately preserving the overall heuristic structure, allowing PRS to further exploit promising regions without inducing another substantial structural drift.

Together, expansion, compression, and steering constitute a multi-directional perturbation mechanism that enables PRS to explore both richer and more generalized heuristics across complementary regimes. The perturbation process can be formally expressed as

$$\tilde{z} \;=\; \Phi_\theta^{(o)}(z_p \mid \mathcal{C})\,, \tag{2}$$

where z_p denotes the heuristic idea of the parent heuristic, $\tilde{z}$ denotes the perturbed heuristic idea, $\mathcal{C}$ contains the task and function descriptions, and $\Phi_\theta^{(o)} \in \{\Phi_\theta^{\text{steer}}, \Phi_\theta^{\text{expn}}, \Phi_\theta^{\text{comp}}\}$ denotes the selected perturbation operator. At each perturbation step, one of these actions is adaptively selected according to its recent search utility.

Rewrite. Following multi-directional perturbation, the perturbed offspring idea specifies the intended algorithmic change but does not directly constitute an executable heuristic. The rewrite step therefore prompts the LLM to realize the offspring idea by rewriting the complete parent implementation, preserving existing mechanisms when consistent with the target idea. PRS independently rewrites the same offspring idea into multiple offspring candidates according to the beam size κ, allowing different executable realizations of the same heuristic perturbation. The rewrite process can be formally expressed as:

$$h'_j \;\sim\; \Psi_\theta(h_p,\, \tilde{z},\, o \mid \mathcal{C})\,, \qquad j = 1, \ldots, \kappa, \tag{3}$$

where Ψ_θ denotes the LLM-based rewrite operator, h_p is the parent heuristic implementation, $\tilde{z}$ is the perturbed heuristic idea produced by operator o, and κ denotes the beam size.

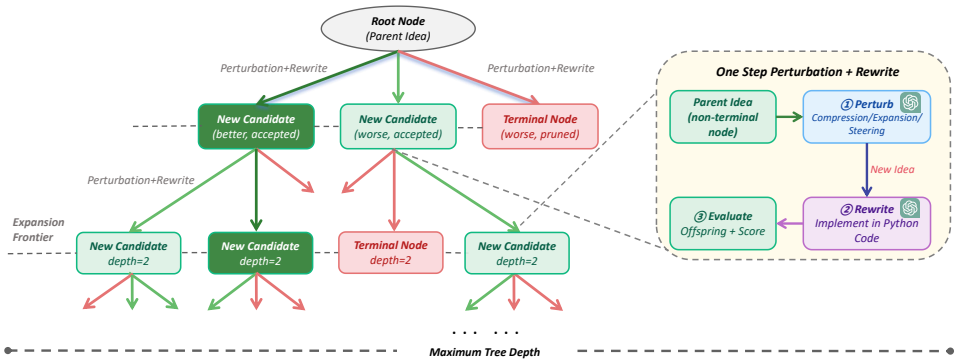


Figure 2: An overview of BFS-based Heuristic Search. Starting from a root parent heuristic with its corresponding idea, PRS performs a BFS-style expansion in which new offspring ideas are generated via multi-directional perturbation and then realized as executable code through rewriting. The offspring codes are then evaluated and selectively retained in a fixed-size frontier at each depth. Terminal nodes correspond to pruned or non-extendable offspring. Dark green boxes denote heuristics that outperform the root parent, while light green (or light red) boxes represent worse-than-root offspring that are accepted (or rejected) for inclusion in the next frontier. Each parent heuristic across BFS paths retains opportunity for further expansion, preventing early domination by a single trajectory. The beam size is set to $\kappa = 3$ for illustration.

3.2 BFS-based Heuristic Search

We employ a bounded BFS-style search mechanism in conjunction with the perturbation–rewrite operator, as BFS naturally promotes candidate utilization through layer-by-layer expansion and prevents early domination by a small subset of heuristics. At the root level, one parent heuristic is selected from the current heuristic pool using an ϵ-greedy strategy based on its performance. Let h^* denote the best heuristic in the current pool $\mathcal{P}$. At the root level of each BFS tree, h^* is selected as the root parent with probability $1 - \epsilon$, while each of the remaining heuristics is selected with probability $\frac{\epsilon}{|\mathcal{P}|-1}$. We set $\epsilon = 0.5$ in all experiments to balance exploitation of the current best heuristic and exploration of alternative parents.

As shown in Fig. 2, for each non-terminal parent at the current BFS frontier, one perturbation operator is selected to generate offspring candidates through the perturbation–rewrite process. After all parents in the current frontier have been expanded, the resulting offspring at the same depth are evaluated based on their average performance on the validation dataset. A subset of candidates is then retained as the next frontier through softmax-based sampling conditioned on their performance scores. For a candidate set $\tilde{\mathcal{F}}$ at the current BFS depth, the probability of retaining a candidate h_i is defined as:

$$p(h_i) = \begin{cases} 1, & \text{if } h_i = \arg\max_{h \in \tilde{\mathcal{F}}} g(h), \\ \text{Softmax}(g(h_i)), & \text{otherwise.} \end{cases} \tag{4}$$

Each retained offspring serves as a parent for the next BFS layer. The BFS expansion terminates when the maximum tree depth is reached or the remaining evaluation budget is exhausted. After each completed BFS tree, the surviving frontier candidates are immediately merged with the current heuristic pool, where the best heuristic is preserved and the remaining pool members are sampled randomly to maintain diversity.

3.3 Adaptive Exploration Controller

The BFS-based heuristic search with the perturbation–rewrite operator constitutes the core of PRS. In this subsection, we introduce an adaptive exploration controller that serves as an auxiliary mechanism to balance exploration and exploitation, and to mitigate search stagnation in local optima.

Adaptive Operator Control. In evolutionary search, the balance between exploration and exploitation critically influences both search breadth and convergence quality. Excessive exploration can lead

to random drift and insufficient refinement of promising elites, whereas overly conservative updates or exploitation may cause the search to collapse into a narrow region of the solution space. Previous studies have shown that static control policies are generally suboptimal in different search phases (Črepinšek et al., 2013).

To address this issue, PRS adaptively selects among expansion, compression, and steering using a UCB-based criterion according to their recent search utility. For an operator $o \in \mathcal{O}$, its selection score at evaluation step t is defined as

$$U_t(o) = \bar{R}_t(o) + c_t \sqrt{\frac{\log N_t}{N_t(o)}}, \tag{5}$$

where $\bar{R}_t(o)$ denotes the mean reward of operator o within a recent sliding window, $N_t(o)$ is the number of times o has been selected within the window, and N_t is the total number of recorded operator selections. For each parent expansion, the operator reward is computed as $R_t(o) = \max\left\{0, \frac{g(h'_*)-g(h_p)}{|g(h_p)|}\right\}$, where h'_* is the best offspring generated from the corresponding parent. Operators that have not yet been observed in the current window are prioritized before applying the UCB criterion. The exploration coefficient c_t controls the trade-off between operator exploration and reward-guided selection, and follows a linear annealing schedule as a function of the evaluation step t. Let T denote the maximum evaluation step. We define:

$$c_t = c_0 + (c_T - c_0)\frac{t}{T}, \tag{6}$$

where c_0 and c_T are the initial and final exploration coefficients, respectively. This schedule encourages broader exploration of different perturbation directions at the beginning and gradually shifts toward exploiting operators with higher observed utility.

Stagnation-Aware Restart. While adaptive operator control improves the overall trade-off between exploration and exploitation, it cannot guarantee escape from stagnation in local optima (Dat et al., 2025). As the search progresses, repeatedly exploiting successful perturbation directions and promising parents may gradually concentrate the search around a narrow set of heuristic structures, making further improvements increasingly difficult. This behavior arises because the best-performing heuristics can repeatedly attract search effort within their local neighborhoods, leading to the proliferation of structurally similar candidates. As a result, the search may enter a *plateau* regime, where heuristic diversity diminishes due to premature convergence or inbreeding (Eshelman, 1991).

To address this issue, PRS triggers a stagnation-aware restart when no improvement in the global best heuristic is observed for a predefined number $\mathcal{C}$ of heuristic evaluations. Once activated, two coordinated actions are performed: a) *Elite Restart*, where the heuristic pool is reset to retain only the globally best heuristic discovered so far; and b) *Operator Reheating*, where the recent operator reward window is reset and the UCB exploration coefficient is temporarily restored to its initial value c_0, shifting operator selection back toward exploration before gradually returning to reward-guided exploitation. Together, these mechanisms enable the heuristic search to escape the current *plateau* by reintroducing diversity into the heuristic pool while ensuring smooth convergence through restarting from the elite heuristic. The complete PRS algorithm and associated prompts are presented in Algorithm 1 and Appendix D.1, respectively.

4 EXPERIMENTS

This section empirically evaluates PRS on some typical NP-hard COPs. The definitions of these COPs are listed in the Appendix B.1. Importantly, these problems are also applied to a wide range of general frameworks, including step-by-step constructive, guided-local search (GLS) and ant colony optimization (ACO). Details of these general frameworks are included in the Appendix B.2. The metrics used to measure the performance of each method primarily include the average objective and the average gap to the reference value (e.g., best-known solution) of each problem size on the test set. Our evaluated problem setting covers a wide range of problem types and general frameworks: 1) ACO: Capacitated Vehicle Routing Problem (CVRP), Orienteering Problem (OP) (both black-box and white-box), Multiple Knapsack Problem (MKP) (black-box); 2) Step-by-step Construction: Knapsack Problem (KP), Online Bin Packing Problem (OBPP), TSP, Admissible Set Problem (ASP); 3) GLS: TSP, Flow Shop Scheduling Problem (FSSP).

Implementation Setups. To maximize the effectiveness of BFS, we adopt a beam search with beam size $\kappa = 3$ and maximum tree depth $d = 3$, and set the frontier size (i.e., the maximum number of parent nodes for the next layer) to $fs = 3$. For both validation datasets and testing datasets, we follow the procedures and configurations of the MCTS-AHD (Zheng et al., 2025) and use their random seeds to generate all datasets. In addition, to ensure a fair comparison, we adopt the same seed heuristics as MCTS-AHD. For all problems except OBPP, we set the number of evaluated heuristics to $T = 1000$ and constrain all LLM-based AHD methods to a maximum budget of 2,500 LLM queries. The size of the heuristic pool is set to 10 for all problems. Comprehensive hyperparameter settings and experimentation setups can be found in the Appendix E. All experiments were conducted on a machine equipped with a single Intel Core i9-12900KF CPU.

Baselines. We compare PRS with the following baselines. They are mainly composed of four categories: (1) hand-crafted heuristics such as Greedy-Construct (GC) (Rosenkrantz et al., 1977), Best-Fit for OBPP (Kenyon et al., 1996), GPUTA (Gupta, 1971), ACO (Blum, 2005; Dorigo et al., 2006) algorithm for CVRP, OP, MKP, etc. (2) Traditional heuristic solvers, including ILS1 (Stützle, 1998), OR-Tools, etc. (3) Neural Combinatorial Optimization (NCO) methods including POMO (Kwon et al., 2020), DeepACO (Ye et al., 2023), etc. (4) Existing LLM-based AHD methods. They are: FunSearch (Romera-Paredes et al., 2024), EoH (Liu et al., 2024a), ReEvo (Ye et al., 2024), HSEvo (Dat et al., 2025) and MCTS-AHD (Zheng et al., 2025). All methods are run with the same initial seed heuristics and tested on identical validation datasets D. Unless otherwise stated, all LLM-based AHD methods are evaluated using GPT-5-nano and mean performance is based on three independent runs.

4.1 Comparison on ACO Framework

The ACO framework is a classical metaheuristic mechanism for solving combinatorial optimization problems by simulating pheromone-guided stochastic construction (Dorigo et al., 2006; Kim et al., 2024). By updating the pheromone matrix τ to determine the edge priority and crafting a more complicated heuristic matrix η to guide ants' transition probabilities and assess the constructed tours, ACO can iteratively drive the solution towards optimal or near-optimal regions. As shown in Table 1, under GPT-5-nano, PRS consistently outperforms all compared LLM-based AHD baselines on the out-of-distribution settings of both CVRP and OP. On the in-distribution settings, PRS also achieves the best performance on CVRP and remains competitive on OP, outperforming EoH and MCTS-AHD. These results suggest that PRS is particularly effective in discovering heuristics with stronger out-of-distribution generalization under the ACO framework.

Table 1: Performance of ACO-based heuristics on CVRP and OP. All methods are evaluated on three test sets of 64 randomly generated instances each, following the setup in Dat et al. (2025); Zheng et al. (2025). Test sets whose scales match the validation sets are underlined. Optimal solutions are approximated using DeepACO with significantly more ants and iterations than those in the baselines. The best-performing method is shaded, and each test set's overall best result is in bold.

Task	CVRP						OP					
N=	N=50		N=100		N=200		N=50		N=100		N=200	
Method	Obj↓	Gap↓	Obj↓	Gap↓	Obj↓	Gap↓	Obj↑	Gap↓	Obj↑	Gap↓	Obj↑	Gap↓
DeepACO	8.888	–	14.932	–	27.159	–	19.867	–	36.392	–	63.380	–
ACO	18.282	105.69%	30.030	101.11%	48.582	78.88%	13.570	31.70%	24.149	33.64%	37.109	41.45%
LLM-based AHD (GPT-5-nano)												
EOH	9.320	4.85%	16.511	10.57%	29.603	9.00%	15.162	23.68%	29.699	18.39%	53.497	15.59%
ReEvo	9.186	3.35%	16.020	7.29%	28.782	5.98%	**15.290**	**23.04%**	30.820	15.31%	54.459	14.08%
MCTS-AHD	9.025	1.54%	15.974	6.98%	28.572	5.20%	15.192	23.53%	30.752	15.50%	54.718	13.67%
Ours	**8.958**	**0.79%**	**15.576**	**4.31%**	**28.228**	**3.93%**	15.247	23.25%	**30.924**	**15.03%**	**55.821**	**11.93%**

4.2 Comparison on KP and OBPP

The step-by-step constructive framework incrementally builds a feasible solution by repeatedly selecting and adding one component at a time, where each decision is guided by a problem-specific heuristic and must satisfy the corresponding feasibility constraints (Asani et al., 2023). For KP, the heuristic scores the remaining items based on their values, weights, and current residual capacity, and iteratively selects the next item until no feasible additions are possible.

Table 2: **Online bin packing problem results**. The first four problem scales are used during validation, and the remaining are used for held-out test. Notations of problem scales are abbreviated, e.g., 1k_500 indicates 1,000 items and a fixed bin capacity 500.

Test Sets	1k_100	1k_500	5k_100	5k_500	10k_100	10k_500	AVG
LLM-based AHD (GPT-5-nano)							
Best Fit	4.77%	0.25%	4.31%	0.55%	4.05%	0.47%	2.40%
First Fit	5.02%	0.25%	4.65%	0.55%	4.36%	0.50%	2.56%
FunSearch	2.45%	0.66%	1.30%	**0.25%**	1.05%	**0.21%**	0.99%
EoH	2.69%	0.25%	1.63%	0.53%	1.47%	0.45%	1.17%
ReEvo	3.13%	0.25%	1.78%	0.55%	2.63%	0.50%	1.47%
HSEvo	2.24%	0.50%	1.46%	0.38%	1.30%	0.36%	1.04%
MCTS-AHD	1.92%	0.58%	1.08%	0.38%	0.88%	0.37%	0.87%
PRS (ours)	**1.82%**	**0.18%**	**0.86%**	0.35%	**0.63%**	0.45%	**0.72%**

For OBPP, items arrive sequentially and must be assigned immediately; the heuristic maps the incoming item size together with current bin states (e.g., remaining capacities) to priority scores over bins, placing each item into the best feasible bin. Specifically, Online-BPP instances are generated following EoH using the standard Weibull distribution (Castiñeiras et al., 2012). Regarding KP, Table 3 shows that PRS consistently achieves the lowest or near-lowest optimality gaps across all problem sizes and clearly outperforms existing LLM-based AHD baselines under GPT-5-nano. Table 2 further shows that online BPP heuristics designed by PRS have a consistently superior average performance across six test settings.

Table 3: Performance of step-by-step constructive heuristics on KP. Near-optimal objectives are obtained by OR-Tools. The best-performing method is shaded, and each test set's overall best result is in bold.

Knapsack Problem (KP)						
Test Sets	N=50		N=100		N=200	
Method	Obj.↑	Gap↓	Obj.↑	Gap↓	Obj.↑	Gap↓
Optimal	20.037	–	40.271	–	57.448	–
GC	19.830	1.04%	40.030	0.60%	57.115	0.58%
POMO	19.612	2.12%	39.676	1.48%	57.271	0.31%
LLM-based AHD (GPT-5-nano)						
FunSearch	19.988	0.24%	40.227	0.11%	57.398	0.09%
EoH	19.994	0.22%	40.233	0.10%	57.398	0.09%
MCTS-AHD	20.010	0.13%	40.250	0.05%	57.422	0.04%
Ours	**20.023**	**0.07%**	**40.262**	**0.02%**	**57.436**	**0.02%**

4.3 Comparison on FSSP

GLS acts as a perturbation-and-improvement framework, which iteratively perturbs a current schedule using neighborhood operators (e.g., swap and relocate) and employs a heuristic matrix to guide which parts to penalize or prioritize for better exploration (Arnold & Sörensen, 2019). Table 11 reports relative improvements in the makespan in Taillard instances. PRS exhibits a consistent advantage over other methods in nearly all test sets and on average.

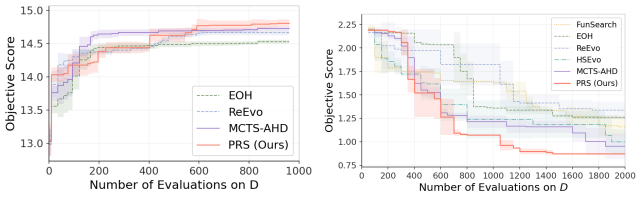
5 Discussions

Ablations on Mechanisms. We analyze the necessity of each component in PRS. As shown in Table 4, all three perturbation operators contribute to search effectiveness: expansion and compression support complementary transformation directions by encouraging richer and simpler heuristic designs, respectively, while steering provides fine-grained exploitation around promising heuristic structures. Disabling UCB-based operator selection and replacing it with random selection consistently degrades performance, indicating that adaptively allocating search effort among the three operators is beneficial. Disabling the stagnation-aware restart mechanism ($C = 10^5$) also weakens performance, further showing the importance of adaptive search control for sustained exploration. Beyond component removal, replacing expansion and compression with EC-style operators (M1&M2) degrades performance within PRS, suggesting that explicitly targeting complementary structural regimes is more effective than the default EC-style transformations in this setting.

Evolution Curves of Different AHD methods. In Fig. 3, we compare the search dynamics of PRS with representative AHD methods on two representative tasks: OP with ACO and OBPP with the constructive framework. Most baselines exhibit rapid early improvements followed by early convergence, indicating a limited ability to sustain effective exploration as the search progresses. In contrast, PRS shows smoother and more continuous performance gains throughout the search. This behavior can be attributed to its multi-directional perturbation mechanism, which actively diversifies heuristic generation and mitigates early homogenization, together with the stagnation-aware controller that restores exploration when progress slows. Empirically, the evolution curves suggest that sustained search effectiveness can be achieved through explicitly controlling diversification through directional search primitives without relying on elaborate EC-based operator orchestration.

Table 4: Ablation study of PRS on key mechanisms and operators. Results are reported as mean ± standard deviation over five independent runs.

Methods	CVRP-50↓	KP-100↑
PRS (default setting)	**8.9583**±0.0726	**40.2603**±0.0165
w/o Expansion	9.7492±0.2840	40.2307±0.0001
w/o Compression	9.3737±0.0707	40.2321±0.0015
w/o Steering	9.0728±0.1207	40.2581±0.0754
w/o UCB Selection	9.3781±0.1142	40.2483±0.0336
$C = 10^5$	9.1399±0.1106	40.2432±0.0137
EC Operators (M1&M2)	9.2277±0.0570	40.2516±0.0071



(a) OP. (b) OBPP.

Figure 3: Evolution curves of the best heuristic in training on two scenarios. Each curve is averaged over five independent runs and the standard deviation is shaded.

Additional Experimental Results. We provide additional experiments in Appendix F–H. In particular, we first provide diversity analyses at both the idea and implementation levels to examine whether the proposed directional search primitives indeed broaden heuristic-space exploration. We then include statistical significance tests (p-values), LLM cost analysis, and sensitivity analysis on the key hyperparameters. Importantly, we further examine how different perturbation directions relate to ID and OOD performance, and an analysis on the contributions of each directional search operator on different problem settings. Moreover, we provide further experimental results on the TSP and TSPLib (Reinelt, 1991) setting, black-box settings, and benchmarking across different LLMs.

6 Conclusion

In this paper, we propose PRS, a multi-directional search framework for automatic heuristic design that makes directional heuristic transformation an explicit generation-level search primitive. By organizing heuristic generation through complementary transformation directions and combining them with breadth-first search and adaptive operator control, PRS actively directs diversification, sustains exploration, and mitigates premature convergence. Extensive experiments across diverse combinatorial optimization tasks demonstrate that PRS consistently discovers high-quality heuristics and outperforms existing LLM-based AHD methods in various settings, emphasizing the importance of exploring directionally diverse regions of the heuristic search space.

AI use statement

Generative AI tools were used solely for language polishing and improving the clarity of the manuscript. They were not used for research idea development, methodology design, code implementation, experiment execution, or initial manuscript drafting. All scientific content and results were produced and verified by the authors, who take full responsibility for the final work.

Ethics statement

This work complies with the ICLR Code of Ethics. The research does not involve human participants, animal experimentation, or personally identifiable information. All datasets and benchmark instances

used in this work are obtained from publicly available sources or generated following established procedures in prior work. We do not identify any specific concerns regarding privacy, security, fairness, or other harmful societal impacts beyond those generally associated with the use of large language models. We have taken care to ensure transparency and integrity in the design, evaluation, and reporting of the proposed method.

Reproducibility statement

We provide the information necessary to reproduce the main experimental results of this work. The complete PRS algorithm is provided in Appendix C, while the prompts and detailed methodology are described in Appendix D. Experimental configurations, datasets, evaluation protocols, and hyperparameter settings are documented in Appendix E and the corresponding experimental sections. We will release the full source code upon acceptance of the paper.

References

H. Abgaryan, A. Harutyunyan, and T. Cazenave. LLMs can schedule, 2024. *arXiv preprint arXiv:2408.06993*.

A. Aleti and I. Moser. A systematic literature review of adaptive parameter control methods for evolutionary algorithms. *ACM Computing Surveys*, 49(3):1–35, 2016.

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

E.O. Asani, A.E. Okeyinka, and A.A. 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 (SEB-SDG)*, volume 1, pp. 1–7, 2023.

T. Bäck, D.B. Fogel, and Z. Michalewicz. Handbook of evolutionary computation. *Release*, 97(1): B1, 1997.

S. Beamer, K. Asanović, and D. Patterson. Direction-optimizing breadth-first search. *Scientific Programming*, 21(3-4):137–148, 2013.

L.C.T. Bezerra, M. López-Ibáñez, and T. Stützle. Automatic component-wise design of multiobjective evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, 20(3):403–417, 2015.

C. Blum. Ant colony optimization: Introduction and recent trends. *Physics of Life Reviews*, 2(4): 353–373, 2005.

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

E.K. Burke, M. Gendreau, M. Hyde, G. Kendall, G. Ochoa, E. Özcan, and R. Qu. Hyper-heuristics: A survey of the state of the art. *Journal of the Operational Research Society*, 64(12):1695–1724, 2013.

E.K. Burke, M.R. Hyde, G. Kendall, G. Ochoa, E. Özcan, and J.R. Woodward. A classification of hyper-heuristic approaches: Revisited. In *Handbook of Metaheuristics*, pp. 453–477. Springer, 2018.

H.G. Campbell, R.A. Dudek, and M.L. Smith. A heuristic algorithm for the n job, m machine sequencing problem. *Management Science*, 16(10):B–630, 1970.

I. Castiñeiras, M. De Cauwer, and B. O'Sullivan. Weibull-based benchmarks for bin packing. In *International Conference on Principles and Practice of Constraint Programming*, pp. 207–222, 2012.

C. Chen, M. Zhong, Y. Fan, J. Shi, and J. Sun. HiFo-Prompt: prompting with hindsight and foresight for LLM-based automatic heuristic design. In *International Conference on Learning Representations*, 2026.

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

M. Črepinšek, S.-H. Liu, and M. Mernik. Exploration and exploitation in evolutionary algorithms: A survey. *ACM Computing Surveys*, 45(3):1–33, 2013.

N. Dahmani, I. Sbai, T. Tlili, and S. Krichen. On solving the 2L-CVRP using an adaptive chemical reaction algorithm: Postal transportation real-case. *International Journal of System Assurance Engineering and Management*, pp. 1–25, 2024.

P.V.T. Dat, L. Doan, and H.T.T. 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, pp. 26931–26938, 2025.

M.G.P. de Lacerda, L.F. de Araujo Pessoa, F.B. de Lima Neto, T.B. Ludermir, and H. Kuchen. A systematic literature review on general parameter control for evolutionary and swarm-based algorithms. *Swarm and Evolutionary Computation*, 60:100777, 2021.

M. Dorigo. *Optimization, learning and natural algorithms*. PhD thesis, 1992. PhD thesis, Politecnico di Milano.

M. Dorigo, M. Birattari, and T. Stutzle. Ant colony optimization. *IEEE Computational Intelligence Magazine*, 1(4):28–39, 2006.

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

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

M. Elhenawy, A. Abutahoun, T.I. Alhadidi, A. Jaber, H.I. Ashqar, S. Jaradat, A. Abdelhay, S. Glaser, and A. Rakotonirainy. Visual reasoning and multi-agent approach in multimodal large language models (MLLMs): Solving TSP and MTSP combinatorial challenges, 2024. *arXiv preprint arXiv:2407.00092*.

H. Emmons and G. Vairaktarakis. *Flow shop scheduling: Theoretical results, algorithms, and applications*. Springer Science & Business Media, 2012.

L.J. Eshelman. Preventing premature convergence in genetic algorithms by preventing incest. In *Proceedings of the Fourth International Conference on Genetic Algorithms*, 1991.

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

Á. Fialho. *Adaptive operator selection for optimization*. PhD thesis, 2010. PhD thesis, Université Paris Sud-Paris XI.

C. Gao, H. Shang, K. Xue, D. Li, and C. Qian. Towards generalizable neural solvers for vehicle routing problems via ensemble with transferrable local policy. In *Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence*, 2024. Article 764.

P.-F. Guo, Y.-H. Chen, Y.-D. Tsai, and S.-D. Lin. Towards optimizing with large language models, 2023. *arXiv preprint arXiv:2310.05204*.

J.N.D. Gupta. A functional heuristic algorithm for the flowshop scheduling problem. *Journal of the Operational Research Society*, 22(1):39–47, 1971.

Q. He, K.L. Head, and J. Ding. Heuristic algorithm for priority traffic signal control. *Transportation Research Record*, 2259(1):1–7, 2011.

A. Hottung, F. Berto, C. Hua, N.G. Zepeda, D. Wetzel, M. Römer, H. Ye, D. Zago, M. Poli, S. Massaroli, et al. VRPAgent: LLM-driven discovery of heuristic operators for vehicle routing problems, 2025. *arXiv preprint arXiv:2510.07073*.

Q. Hu and Q. Zhang. Partition to Evolve: Niching-Enhanced Evolution with LLMs for Automated Algorithm Discovery. In *The Thirty-Ninth Annual Conference on Neural Information Processing Systems*, 2025.

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

X. Jiang, Y. Wu, Y. Wang, and Y. Zhang. UNCO: Towards unifying neural combinatorial optimization through large language model, 2024.

X. Jiang, Y. Wu, M. Li, Z. Cao, and Y. Zhang. Large language models as end-to-end combinatorial optimization solvers, 2025. *arXiv preprint arXiv:2509.16865*.

M. E. Johnson, L. M. Moore, and D. Ylvisaker. Minimax and maximin distance designs. *Journal of Statistical Planning and Inference*, 26(2):131–148, 1990. doi: 10.1016/0378-3758(90)90122-B.

L. Jost. Entropy and diversity. *Oikos*, 113(2):363–375, 2006.

G. Karafotias, M. Hoogendoorn, and Á.E. Eiben. Parameter control in evolutionary algorithms: Trends and challenges. *IEEE Transactions on Evolutionary Computation*, 19(2):167–187, 2014.

C. Kenyon et al. Best-Fit bin-packing with random order. In *Proceedings of SODA*, volume 96, pp. 359–364, 1996.

M. Kim, S. Choi, H. Kim, J. Son, J. Park, and Y. Bengio. Ant colony sampling with GFlowNets for combinatorial optimization, 2024. *arXiv preprint arXiv:2403.07041*.

W. Kool, H. van Hoof, and M. Welling. Attention, learn to solve routing problems! In *International Conference on Learning Representations*, 2019.

Y.-D. Kwon, J. Choo, B. Kim, I. Yoon, Y. Gwon, and S. Min. POMO: Policy optimization with multiple optima for reinforcement learning. *Advances in Neural Information Processing Systems*, 33:21188–21198, 2020.

W.B. Langdon and R. Poli. *Foundations of genetic programming*. Springer Science & Business Media, 2013.

Robert Tjarko Lange, Yuki Imajuku, and Edoardo Cetin. Shinkaevolve: Towards open-ended and sample-efficient program evolution. *arXiv preprint arXiv:2509.19349*, 2025.

Craig A. Layman, D. Albrey Arrington, Carmen G. Montaña, and David M. Post. Can stable isotope ratios provide for community-wide measures of trophic structure? *Ecology*, 88(1):42–48, 2007. doi: 10.1890/0012-9658(2007)88[42:CSIRPF]2.0.CO;2.

S. Lin and B.W. Kernighan. An effective heuristic algorithm for the traveling-salesman problem. *Operations Research*, 21(2):498–516, 1973.

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

F. Liu, X. Tong, M. Yuan, X. Lin, F. Luo, Z. Wang, Z. Lu, and Q. Zhang. Evolution of heuristics: towards efficient automatic algorithm design using large language model. In *International Conference on Machine Learning*, 2024a.

F. Liu, Y. Yao, P. Guo, Z. Yang, X. Lin, Z. Zhao, X. Tong, K. Mao, Z. Lu, Z. Wang, et al. A systematic survey on large language models for algorithm design. *ACM Computing Surveys*, 2024b.

F. Liu, R. Zhang, Z. Xie, R. Sun, K. Li, X. Lin, Z. Wang, Z. Lu, and Q. Zhang. LLM4AD: A platform for algorithm design with large language model, 2024c. *arXiv preprint arXiv:2412.17287*.

F. Liu, Y. Liu, Q. Zhang, X. Tong, and M. Yuan. EoH-S: Evolution of heuristic set using LLMs for automated heuristic design, 2025. *arXiv preprint arXiv:2508.03082*.

S. Liu, C. Chen, X. Qu, K. Tang, and Y.-S. Ong. Large language models as evolutionary optimizers. In *2024 IEEE Congress on Evolutionary Computation (CEC)*, pp. 1–8, 2024d.

Y. Ma, J. Li, Z. Cao, W. Song, L. Zhang, Z. Chen, and J. Tang. Learning to iteratively solve routing problems with dual-aspect collaborative transformer. *Advances in Neural Information Processing Systems*, 34:11096–11107, 2021.

Y. Mei, Q. Chen, A. Lensen, B. Xue, and M. Zhang. Explainable artificial intelligence by genetic programming: A survey. *IEEE Transactions on Evolutionary Computation*, 27(3):621–641, 2022.

B. Mihoubi, B. Bouzouia, and M. Gaham. Reactive scheduling approach for solving a realistic flexible job shop scheduling problem. *International Journal of Production Research*, 59(19): 5790–5808, 2021.

Max D. Morris and Toby J. Mitchell. Exploratory designs for computational experiments. *Journal of Statistical Planning and Inference*, 43(3):381–402, 1995. doi: 10.1016/0378-3758(94)00035-T.

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

A. Novikov, N. Vũ, M. Eisenberger, E. Dupont, P.-S. Huang, A.Z. Wagner, S. Shirobokov, B. Kozlovskii, F.J.R. Ruiz, A. Mehrabian, et al. AlphaEvolve: A coding agent for scientific and algorithmic discovery, 2025. *arXiv preprint arXiv:2506.13131*.

M. O'Neill, C. Ryan, M. Keijzer, and M. Cattolico. Crossover in grammatical evolution: The search continues. In *European Conference on Genetic Programming*, pp. 337–347, 2001.

Z. Pan, L. Wang, J. Wang, and J. Lu. Deep reinforcement learning based optimization algorithm for permutation flow-shop scheduling. *IEEE Transactions on Emerging Topics in Computational Intelligence*, 7(4):983–994, 2021.

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

E.J.S. Pires, J.A.T. Machado, and P.B. de Moura Oliveira. Diversity study of multi-objective genetic algorithm based on Shannon entropy. In *2014 Sixth World Congress on Nature and Biologically Inspired Computing (NaBIC 2014)*, pp. 17–22, 2014.

E.J.S. Pires, J.A.T. Machado, and P.B. de Moura Oliveira. Dynamic Shannon performance in a multiobjective particle swarm optimization. *Entropy*, 21(9):827, 2019.

R. Qu, G. Kendall, and N. Pillay. The general combinatorial optimization problem: Towards automated algorithm design. *IEEE Computational Intelligence Magazine*, 15(2):14–23, 2020.

C. Rajendran. Heuristic algorithm for scheduling in a flowshop to minimize total flowtime. *International Journal of Production Economics*, 29(1):65–73, 1993.

G. Reinelt. TSPLIB—A traveling salesman problem library. *ORSA Journal on Computing*, 3(4): 376–384, 1991.

B. Romera-Paredes, M. Barekatain, A. Novikov, M. Balog, M.P. Kumar, E. Dupont, F.J.R. Ruiz, J.S. Ellenberg, P. Wang, O. Fawzi, et al. Mathematical discoveries from program search with large language models. *Nature*, 625(7995):468–475, 2024.

D.J. Rosenkrantz, R.E. Stearns, and P.M. Lewis. An analysis of several heuristics for the traveling salesman problem. *SIAM Journal on Computing*, 6(3):563–581, 1977.

X. Sánchez-Díaz, J.C. Ortiz-Bayliss, I. Amaya, J.M. Cruz-Duarte, S.E. Conant-Pablos, and H. Terashima-Marín. A feature-independent hyper-heuristic approach for solving the knapsack problem. *Applied Sciences*, 11(21):10209, 2021.

L. Sengupta, R. Mariescu-Istodor, and P. Fränti. Which local search operator works best for the open-loop TSP? *Applied Sciences*, 9(19):3985, 2019.

J. Silvela and J. Portillo. Breadth-first search and its application to image processing problems. *IEEE Transactions on Image Processing*, 10(8):1194–1199, 2001.

T. Stützle. Applying iterated local search to the permutation flow shop problem. Technical report, 1998. Technical Report AIDA-98-04, FG Intellektik, TU Darmstadt.

T. Stützle and M. López-Ibáñez. Automated design of metaheuristic algorithms. In *Handbook of Metaheuristics*, pp. 541–579. Springer, 2018.

E. Taillard. Benchmarks for basic scheduling problems. *European Journal of Operational Research*, 64(2):278–285, 1993.

Y. Tian, X. Li, H. Ma, X. Zhang, K.C. Tan, and Y. 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, 2022.

J. Tuononen. Analysis of rebuild local search operator for TSP, 2022.

N. van Stein and T. Bäck. LLaMEA: A large language model evolutionary algorithm for automatically generating metaheuristics. *IEEE Transactions on Evolutionary Computation*, 2024.

N. Veerapen, J. Maturana, and F. Saubion. An exploration-exploitation compromise-based adaptive operator selection for local search. In *Proceedings of the 14th Annual Conference on Genetic and Evolutionary Computation*, pp. 1277–1284, 2012.

L. Wang and Y. Chen. Diversity based on entropy: A novel evaluation criterion in multi-objective optimization algorithm. *International Journal of Intelligent Systems and Applications*, 4(10): 113–124, 2012.

X. Wang, J. Wei, D. Schuurmans, Q. Le, E. Chi, S. Narang, A. Chowdhery, and D. Zhou. Self-consistency improves chain of thought reasoning in language models, 2022. *arXiv preprint arXiv:2203.11171*.

Z. Wu, Q. Qi, Z. Zhuang, H. Sun, and J. Wang. Pre-tokenization of numbers for large language models. In *The Second Tiny Papers Track at ICLR 2024*, 2024.

X. Xu, T. Xiao, Z. Chao, Z. Huang, C. Yang, and Y. Wang. Can LLMs solve longer math word problems better?, 2024. *arXiv preprint arXiv:2405.14804*.

C. Yang, X. Wang, Y. Lu, H. Liu, Q.V. Le, D. Zhou, and X. Chen. Large language models as optimizers. In *The Twelfth International Conference on Learning Representations*, 2023.

S. Yao, D. Yu, J. Zhao, I. Shafran, T. Griffiths, Y. Cao, and K. Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. *Advances in Neural Information Processing Systems*, 36:11809–11822, 2023.

H. Ye, J. Wang, Z. Cao, H. Liang, and Y. Li. DeepACO: Neural-enhanced ant systems for combinatorial optimization. *Advances in Neural Information Processing Systems*, 36:43706–43728, 2023.

H. Ye, J. Wang, Z. Cao, F. Berto, C. Hua, H. Kim, J. Park, and G. Song. Reevo: Large language models as hyper-heuristics with reflective evolution. *Advances in Neural Information Processing Systems*, 37:43571–43608, 2024.

R. Zhang and Z. Lu. Rethinking Code Similarity for Automated Algorithm Design with LLMs. In *The Fourteenth International Conference on Learning Representations*, 2026. URL `https://openreview.net/pdf?id=HIUqeO9OOr`.

Z. Zheng, Z. Xie, Z. Wang, and B. Hooi. Monte Carlo tree search for comprehensive exploration in LLM-based automatic heuristic design. In *International Conference on Machine Learning*, 2025.

APPENDIX CONTENTS

A Related Work **16**

A.1 Automated Heuristic Design 16

A.2 LLM-based AHD 16

A.3 LLM for CO 16

A.4 Operators in AHD and LLM-based AHD. 17

A.5 Adaptive Control in EC and LLM-based AHD. 18

B Evaluated Problem Settings **18**

B.1 Definition of Tasks 18

B.2 Evaluated Framework for Tasks 20

C Complete Algorithm **22**

D Detailed Methodology **23**

D.1 PRS Prompt 23

D.2 Generated Heuristics 29

E Details of Evaluation Configurations **31**

F More Experimental Results **33**

F.1 Search Dynamic Analysis 33

F.2 Comparison on Black-box Settings 37

F.3 TSP Step-by-step Constructive Framework 37

F.4 TSP Heuristic under Guided Local Search Framework 38

F.5 Flow Shop Scheduling Problem under Guided Local Search Framework 40

F.6 Results on TSPLib instances 40

F.7 Statistical Significance 41

F.8 Search Strategy Variants 42

F.9 Experiments with different LLMs 43

G Cost Analysis **44**

H Sensitivity Analysis **45**

I Example Response for Multi-directional Perturbations **46**

J Limitations **48**

K Baselines & Licenses **48**

A Related Work

A.1 Automated Heuristic Design

Automated Heuristic Design (AHD), often referred to as hyper-heuristics (Burke et al., 2018), aims to automatically discover high-performing heuristics from a large space of valid heuristic candidates rather than directly solving problem instances. Most AHD methods adopt evolutionary algorithms as the underlying search strategy to iteratively refine heuristics automatically through selection and generation, as well as a series of defined evolutionary operators for combination and adaptation (Stützle & López-Ibáñez, 2018; Pillay & Qu, 2018; Bezerra et al., 2015; Qu et al., 2020). Among them, genetic programming (GP) (Langdon & Poli, 2013; Mei et al., 2022) has emerged as a prevailing approach for heuristic generation due to its flexibility and interpretability, enabling the synthesis of reusable or instance-specific heuristics via tree-based representations. However, GP-based AHD typically relies on carefully hand-crafted mutation and crossover operators, whose effectiveness is highly sensitive to representation choices and domain knowledge (Duflo et al., 2019; Sánchez-Díaz et al., 2021). Beyond GP, recent AHD frameworks further categorize heuristic generation into constructive and perturbative paradigms, or integrate multiple operators and stages through component-based unified solvers, with the goal of enhancing generality across diverse optimization problems (Bezerra et al., 2015; Qu et al., 2020). Despite these advances, existing AHD approaches largely depend on manually designed components and expert knowledge (Liu et al., 2025), which remain a major bottleneck for scalability and portability.

A.2 LLM-based AHD

Recent work has introduced large language models into automated heuristic design, leading to a class of population-based LLM-driven AHD methods that largely follow an evolutionary paradigm (Liu et al., 2023; Romera-Paredes et al., 2024; Liu et al., 2024a; Ye et al., 2024; Liu et al., 2024c). In general, these methods iterate over three core steps after initially sampling a set of heuristic functions: selecting mutation or crossover operators together with parent heuristics from the population, prompting LLMs to generate new heuristic implementations—typically in executable code form—and updating the population (or the tree) based on empirical performance evaluated on a validation dataset until a fixed maximum evaluation step is reached. Representative frameworks within this paradigm include FunSearch (Romera-Paredes et al., 2024), which adopts a multi-island evolutionary structure with a fixed prompt strategy for algorithm refinement, EoH (Liu et al., 2024a), which expands the search space through multiple prompt strategies that evolve both intermediate reasoning and code, and ReEvo (Ye et al., 2024), which incorporates short- and long-term reflection mechanisms to guide heuristic evolution. More recently, diversity has received increasing attention as an important factor for sustaining exploration in LLM-based AHD. For example, HSEvo (Dat et al., 2025) explicitly promotes population diversity through diversity-aware search and population management, showing the value of preserving heterogeneous heuristic candidates during evolution.

Building on this direction, recent studies have introduced more explicit mechanisms for measuring diversity and structuring the search space. BehaveSim (Zhang & Lu, 2026) measures algorithmic similarity from execution trajectories and incorporates behavioral diversity into candidate selection. PartEvo (Hu & Zhang, 2025) projects candidate algorithms into feature spaces and constructs semantic niches to allocate search effort across different regions of the abstract language space. ShinkaEvolve (Lange et al., 2025) further promotes exploration through archive-based parent sampling and code-novelty rejection based on embedding similarity. These approaches demonstrate the importance of maintaining diverse candidates during LLM-based algorithm search, but primarily regulate diversity through similarity assessment, population organization, novelty filtering, or search-space partitioning. In contrast, PRS controls diversification directly at the generation level by explicitly selecting how a parent heuristic is transformed before realization, using expansion, compression, and steering as complementary directional perturbation operators.

A.3 LLM for CO

Beyond LLM-based AHD, the use of LLMs for tackling combinatorial optimization problems has also emerged as a promising direction, and one line of research treats LLMs as optimizers. Starting with OPRO (Yang et al., 2023) that iteratively refines candidate solutions for TSP by conditioning on

previously generated tours and their objective values, the existing LLM-as-optimizer methods (Guo et al., 2023; Liu et al., 2024d; Elhenawy et al., 2024) improve solution quality through repeated in-context prompting, evaluation, and reinjection, forming a closed-loop optimization process. Another closely related direction employs LLMs as end-to-end solvers (LLM-as-E2E-solver) that directly map problem descriptions to complete solutions, either via prompting or with additional fine-tuning within task-specific environments (Abgaryan et al., 2024; Jiang et al., 2024; 2025). These approaches demonstrate the potential of LLMs to reason over complicated combinatorial structures and to generate feasible solutions without explicit algorithmic decomposition and solver utilization.

Despite their promise, LLM-as-optimizer and LLM-as-E2E-solver approaches face inherent limitations when scaling to elaborate or large-scale NP-hard problems, largely due to the reduced coherence and accuracy given extended contextual descriptions (Xu et al., 2024). On the other hand, these approaches often impose extensive pressure on reasoning ability as well as contextual memory of LLMs, especially when historical solutions and feedback are repeatedly encoded into prompts. Moreover, LLM-as-E2E-solver methods require additional training pipelines and encounter difficulties in representing high-precision numerical information, such as coordinates or capacities, in purely linguistic token spaces (Wu et al., 2024). The paradigm of using LLMs as heuristic generators, such as PRS, remains an effective alternative to address these challenges. By automatically discovering standalone (e.g., step-by-step construction) or plug-in (e.g., heuristic matrix) heuristics through AHD, this paradigm exhibits better generalization across problem instances while requiring less training resources.

A.4 Operators in AHD and LLM-based AHD.

Crossover has long been a central operator in evolutionary computation, both in automated heuristic design and in fields other than AHD (O'Neill et al., 2001). In genetic programming and hyper-heuristics, heuristics are typically represented as program trees (e.g., Abstract Syntax Tree (AST)) or rule compositions, and crossover is used to recombine partial structures from multiple parents to generate new heuristics (Langdon & Poli, 2013; Burke et al., 2013). This paradigm has been widely adopted in GP-based hyper-heuristics for combinatorial optimization, including studies on the traveling salesman problem (Duflo et al., 2019) and later extensions that incorporate semantic constraints or explainability into crossover-based heuristic search. Recent LLM-based AHD frameworks like Funsearch and EoH largely inherit this crossover-like philosophy in a semantic or prompt-level form. Funsearch constructs prompts from multiple previously discovered functions and asks the LLM to produce a better offspring, thereby inheriting an operator organization that is closer to parent recombination, while EoH explicitly combines multiple selected parent heuristics to synthesize improved ones. A more recent framework, VRPAgent (Hottung et al., 2025) discovers heuristic operators by re-combining a non-elite with an elite for a biased crossover analogous to ReEvo's. Beyond AHD, LLM-driven evolutionary optimizers such as LMEA (Liu et al., 2024d) and LLaMEA (van Stein & Bäck, 2024) directly instruct LLMs to perform parent selection, crossover, and mutation at the solution level (e.g., a complete tour of TSP), closely mirroring classical genetic algorithms.

In contrast to crossover-centered search, single-parent perturbation has emerged as an alternative way to sustain effective heuristic search. Classical evolution strategies show that meaningful search progress can arise from controlled perturbations of a single parent without explicit recombination, provided that the perturbation strength and adaptation are properly regulated (Bäck et al., 1997). More broadly, this line of research highlights the importance of perturbation control, rather than parent recombination alone, in governing search dynamics (Eiben & Smith, 2015). PRS is conceptually closer to this single-parent perturbation perspective, but it departs from classical mutation-style search in an important way. Rather than treating mutation as a generic mechanism for producing another variant of a parent, PRS explicitly factorizes heuristic transformation into complementary directions before code realization. Expansion, compression, and steering specify whether a parent heuristic should move toward a richer, simpler, or locally refined design, respectively. Thus, the key distinction of PRS lies not merely in avoiding recombination, but in making the direction of heuristic transformation an explicit generation-level search primitive.

A.5 Adaptive Control in EC and LLM-based AHD.

Adapting search strategies based on the evolving state has long been a crucial objective in evolutionary computation. Traditional approaches mainly rely on low-level mechanisms such as Adaptive Operator Selection (AOS) (Fialho, 2010; Tian et al., 2022) and parameter control (Eiben & Smith, 2015; Aleti & Moser, 2016), which adjust mutation rates or operator probabilities in a reactive manner. Such approaches operate through numerical credit assignment mechanisms that bias the search toward recently successful strategies, but they lack the capacity to reason about search dynamics, for example, detecting stagnation or loss of diversity. More recently, LLM-based AHD methods have begun to leverage the reasoning capabilities of large language models to perform higher-level adaptive interventions, such as injecting diversity to escape stagnation (Veerapen et al., 2012; Dat et al., 2025) or steering search using semantic interpretations of search behavior (Karafotias et al., 2014; de Lacerda et al., 2021). This represents a shift from local parameter tuning towards search-state-aware, strategy-level adaptation. Driven by these motivations, PRS combines adaptive operator control with stagnation-aware restart, where operator selection is continuously adjusted according to recent search utility, while prolonged stagnation triggers an elite restart and temporary reheating of exploration.

B Evaluated Problem Settings

B.1 Definition of Tasks

We evaluate the proposed PRS framework on a diverse set of combinatorial optimization problems, covering routing, packing, scheduling, and combinatorial set construction tasks. Specifically, they are TSP, CVRP, OP, MKP, BPP, KP, ASP and FSSP. For BPP, we only test its online setting (online BPP). In this subsection, we will introduce the mathematical formulation and evaluation protocols of these problems in detail.

Traveling Salesman Problem (TSP). Given a set of n cities with pairwise distances, the Traveling Salesman Problem aims to find the minimum-length tour that visits each city exactly once and returns to the starting city.

$$\min_{\pi \in S_n} \sum_{t=1}^{n} d_{\pi_t, \pi_{t+1}}, \quad \pi_{n+1} = \pi_1, \tag{7}$$

where d_{ij} is the Euclidean distance between city i and city j, π_t is the selected node at step t of a certain valid tour (i.e. a Hamiltonian tour over n cities) and S_n is the solution space of the TSP with n cities. We consider the symmetric TSP formulation and evaluate heuristics on both synthetic instances and real-world benchmarks from TSPLib. Following ReEvo, synthetic TSP instances sample nodes uniformly from the unit square $[0, 1]^2$ (Ye et al., 2024).

Capacitated Vehicle Routing Problem (CVRP). CVRP consists of a depot indexed 0 and a set of customers $\{1, \ldots, n\}$, each customer i has a demand q_i (with $q_0 = 0$), and all vehicles have an identical capacity Q. The objective is to design a set of closed routes that serve all customers without exceeding vehicle capacities, while minimizing the total travel cost:

$$\min \sum_{k} \sum_{(i,j) \in \pi_k} d_{ij} \tag{8}$$

subject to

$$\sum_{i \in \pi_k} q_i \leq Q, \quad \forall k. \tag{9}$$

where each sub-route π_k starts and ends at the depot.

Orienteering Problem (OP). Given a graph with nodes $\{0, 1, \ldots, n\}$, where node 0 denotes the depot and is associated with zero reward ($r_0 = 0$), each remaining node i has a reward r_i, and d_{ij}

denotes the travel distance. The goal of OP is to find a route $\pi = (\pi_1, \ldots, \pi_k)$ starting and ending at the depot that maximizes the total reward collected under a travel-length budget B:

$$\max_{\pi} \sum_{i \in \pi} r_i \quad \text{s.t.} \quad \sum_{t=0}^{k} d_{\pi_t, \pi_{t+1}} \leq B \tag{10}$$

where $\pi_0 = \pi_{k+1} = 0$ and each node is visited at most once.

Knapsack Problems (KP and MKP). The Knapsack Problem (KP) aims to select a subset of items to maximize total profit subject to a capacity constraint. Given item values $\{v_i\}$, item weights $\{w_i\}$, and knapsack capacity C, KP aims to

$$\max_{x \in \{0,1\}^n} \sum_{i=1}^{n} v_i x_i \quad \text{s.t.} \quad \sum_{i=1}^{n} w_i x_i \leq C. \tag{11}$$

where x_i is a binary indicator of whether the item i is packed into the knapsack or not. We follow AM (Kool et al., 2019) in generating instances and take $W = 25$ for 100-item and 200-item KP instances and $W = 12.5$ for 50-item ones.

The Multiple Knapsack Problem (MKP) extends this formulation by introducing multiple resource constraints. We evaluate heuristics under a constructive selection setting, where items are considered sequentially. For the Multiple knapsack problem (MKP), we follow the instance generation procedure in ReEvo (Ye et al., 2024). Specifically, item values and weights are independently sampled from a uniform distribution on $[0, 1]$, and the capacity of each knapsack is set proportionally to the total item weights to ensure non-trivial feasible solutions. Specifically for the knapsack problem, `select_next_item` function takes as input the remaining capacity, a numpy array of item values, and a numpy array of item weights and returns the next item to pick. For both KP and MKP, feasibility is enforced at each step, and the total profit of the selected items serves as the objective value.

Online Bin Packing Problem (Online BPP). In the Online Bin Packing Problem, items arrive sequentially and must be assigned to bins immediately upon arrival without knowledge of future items. Given the bin capacity W and an item stream $\{s_t\}$, the objective is to minimize the number of bins used. Formally, upon receiving s_t, the policy must choose a bin index b_t with the highest priority score. We follow Funsearch (Romera-Paredes et al., 2024) and EoH (Liu et al., 2024a) to generate Weibull instances for online BPP.

Flow Shop Scheduling Problems (FSSP). Scheduling problems aim to determine the execution order of operations to optimize the overall schedule length. In the Flow Shop Scheduling Problem (FSSP), n jobs must be processed on m machines, where each job consists of m operations executed in the same predetermined order across all machines. Each machine can process at most one operation at a time, and the objective is to minimize the makespan, i.e., the total schedule length.

Following prior work (Emmons & Vairaktarakis, 2012), we consider a perturbation-based scheduling setting, where the processing order remains consistent during each step, and the machines cannot execute multiple operations simultaneously. During heuristic search, we perform optimization on a set of randomly generated instances D. Each instance in D consists of 50 jobs and between 2 and 20 machines, with processing times sampled independently from a uniform distribution over $[0, 1]$ (Pan et al., 2021). The average makespan over these instances is used as the fitness value for heuristic evaluation.

Admissible Set Problem (ASP). The admissible set problem (ASP) aims to construct the largest possible set of vectors $A \subset \{0, 1, 2\}^n$, where each vector has exactly w non-zero elements. Formally, ASP can be written as

$$\max |A| \tag{12}$$

subject to

$$\sum_{i=1}^{n} \mathbb{I}[a_i \neq 0] = w, \quad \forall a \in A, \tag{13}$$

and

$$\exists i \in \{1, \ldots, n\} \text{ s.t. } \{a_i, b_i, c_i\} \in \big\{\{0, 0, 1\}, \{0, 0, 2\}, \{0, 1, 2\}\big\}, \quad \forall a, b, c \in A,\ a \neq b \neq c. \tag{14}$$

Under the step-by-step constructive framework, the heuristic to be designed is a priority function that assigns a score to each candidate vector according to its suitability for being added to the current admissible set. Higher scores indicate higher priority.

B.2 Evaluated Framework for Tasks

We evaluate PRS on eight NP-hard combinatorial optimization problems (COPs) under three widely used algorithmic frameworks for automatic heuristic design (AHD). Specifically, we consider (i) **step-by-step construction** for TSP, KP, ASP and Online-BPP; (ii) **ACO** for MKP, CVRP and OP; and (iii) **GLS** for TSP and FSSP.

Evaluation metric. For minimization problems (TSP, CVRP, Online-BPP, FSSP), we report the relative optimality gap:

$$\text{Gap}(\%) \;=\; \frac{f(h) - f^\star}{f^\star} \times 100\%, \tag{15}$$

where $f(h)$ is the objective value returned by a heuristic function h and $f^\star$ is the optimal reference value (e.g., best-known or a lower bound). For maximization problems (OP, KP, MKP, ASP), we use

$$\text{Gap}(\%) \;=\; \frac{f^\star - f(h)}{f^\star} \times 100\%. \tag{16}$$

where $f^\star$ is the best-known value or an upper bound.

B.2.1 Step-by-step Constructive Framework

Step-by-step construction is a generic framework for combinatorial optimization that incrementally builds a solution by repeatedly selecting and adding the highest-priority candidate until a complete feasible solution is obtained. Nearest-neighbor is a widely used handcrafted step-by-step constructive heuristic for TSP, which selects the next node solely based on its distance to the current node. Analogously, for the knapsack problem, a greedy heuristic is adopted that prioritizes items according to their value-to-weight ratio. We refer to these two manually designed heuristics as the Greedy Construct (GC) baseline in Table 3 and Table 9.

- **TSP.** Under the step-by-step construction framework, the designed heuristic is a `select_next_node` function that scores all unvisited candidates and greedily appends the best one. For each instance, the heuristic outputs the best next city to select based on the current node, the destination node (starting node), the indices of all unvisited nodes and the symmetric distance matrix.

- **KP.** In step-by-step construction for KP, the heuristic scores the remaining items given (v_i, w_i) and the remaining capacity, then selects the next item to pack.

- **Online BPP.** We evaluate heuristics under this strict online setting, where the heuristic determines the bin assignment for each incoming item based solely on current bin states and the current item's size.

- **ASP.** In step-by-step construction for ASP, the heuristic scores each candidate vector given the vector itself, the problem dimension n, and the weight parameter w, and then selects high-priority vectors to incrementally build the admissible set while preserving feasibility.

B.2.2 GLS-based Problems

Guided Local Search (GLS). For GLS-based evaluation, we embed the LLM-designed heuristic into a standard GLS loop (e.g., maintaining a penalty-based edge weights matrix). We use two local operators (i.e., Swap and Relocate) for perturbation and local improvements, and evaluate after each perturbation-local search cycle.

TSP under GLS. We apply GLS to TSP by optimizing an edge-weight matrix that penalizes frequently used edges. Let c_{ij} be the edge cost between node i and node j and let $p_{ij} \in \mathbb{Z}_{\geq 0}$ be a penalty. The augmented edge cost takes the typical form:

$$c'_{ij} \;=\; c_{ij} + \lambda\, p_{ij}, \tag{17}$$

where λ controls penalty strength. The AHD target is improving the penalty (utility) heuristic (e.g., how to update p_{ij}). We apply LLM-based AHD methods to design the core heuristic for generating knowledge matrices within the knowledge-guided local search (KGLS) framework (Arnold & Sörensen, 2019). For TSP, KGLS maintains a current solution together with the best solution encountered so far. At each iteration, the current solution is first perturbed according to the learned knowledge matrix, followed by the local search using 2-opt and relocate operators (Sengupta et al., 2019; Tuononen, 2022). We use scale-wise GLS budgets summarized in Table 5.

Flow Shop Scheduling Problem (FSSP) under GLS. We evaluate FSSP within a GLS-style perturbation and local improvement procedure, where the evolved heuristic controls (i) which operations to penalize (or prioritize) and (ii) which parts of the schedule to perturb.

B.2.3 ACO-BASED PROBLEMS

For ACO-based problems, including CVRP and OP, we adopt a standard ant colony optimization (ACO) framework that mimics the nature of ants to find routes between food sources and the colony. In this setting, the LLM-designed component is the heuristic information function, which defines the edge desirability matrix η used by ants during solution construction.

At each construction step, an ant located at node i selects the next node j according to the transition probability

$$P_{ij} \;=\; \frac{\tau_{ij}^{\alpha}\, \eta_{ij}^{\beta}}{\sum_{k \in \mathcal{A}(i)} \tau_{ik}^{\alpha}\, \eta_{ik}^{\beta}}, \tag{18}$$

where τ_{ij} denotes the pheromone value, η_{ij} is the heuristic information produced by the evolved heuristics, α and β are fixed hyperparameters, and $\mathcal{A}(i)$ denotes the set of feasible next nodes under the problem-specific constraints (Dorigo, 1992; Dorigo et al., 2006).

The heuristic matrix functions under ACO framework have different semantics for CVRP, OP, and MKP:

- **CVRP.** Under the ACO framework, ants sequentially construct routes by selecting the next customer based on pheromone trails and LLM-generated heuristic scores.
- **OP.** Within the ACO framework, ants incrementally build candidate paths by choosing the next site according to the pheromone information and LLM-based heuristics that balance reward and travel budget.
- **MKP.** For MKP, the ACO framework is applied in a black-box manner, which ants stochastically sample item selections guided by pheromone information and LLM-generated heuristic scores that indicate the desirability of including each item in the solution.

The pheromone update rule and other ACO hyperparameters are kept identical across all methods to ensure a fair comparison. Only the heuristic information function η is replaced by the LLM-designed heuristics generated by PRS or the baseline AHD.

C Complete Algorithm

Algorithm 1 provides the complete pseudo-code for the PRS framework. The full framework includes an initialization stage (up to line 6) and an iterative stage consisting of a BFS-based search process with multi-directional perturbation operators and stagnation-aware mechanisms.

Algorithm 1 PRS: Perturbation–Rewrite Search via BFS and Adaptive Control

Input: Validation dataset D; evaluation environment g; heuristic pool size N; initial number of sampled heuristics N_{init}; maximum evaluation steps T; root selection via ϵ-greedy; BFS parameters: depth d, beam size κ, frontier size fs, softmax temperature T_{sel}; UCB operator selection with exploration coefficients c_0 and c_T; stagnation trigger threshold $\mathcal{C}$ and reheating horizon $\Delta = 0.1T$; perturbation operators Φ_θ and rewrite operator Ψ_θ.
Get $g_0 = g(h_{\text{seed}})$ where h_{seed} is the seed heuristic function and set $t \leftarrow 1$
Initialize candidate set $\mathcal{P} \leftarrow \{h_{\text{seed}}\} \cup \{h_1, \ldots, h_{N_{\text{init}}-1}\}$ using h_{seed} and an initialization prompt
Evaluate all newly generated $h \in \mathcal{P}$ using g
$h^\star \leftarrow \arg\max_{h \in \mathcal{P}} g(h)$; $t \leftarrow |\mathcal{P}|$
$\mathcal{P} \leftarrow \pi_{\text{greedy}}(\mathcal{P}, N)$; initialize operator reward window $\mathcal{H}$; $t' \leftarrow -\infty$
while $t < T$ **do**
  $c_t^{\text{base}} \leftarrow c_0 + (c_T - c_0)\frac{t}{T}$
  if $t - t' < \Delta$ **then**
    $\hat{t} \leftarrow \text{clip}\Big(\frac{t-t'}{\Delta}, 0, 1\Big)$
    $c_t \leftarrow c_t^{\text{base}} + (c_0 - c_t^{\text{base}})(1 - \hat{t})$ // operator reheating
  else
    $c_t \leftarrow c_t^{\text{base}}$
  end if
  $h_{\text{root}} \leftarrow \text{EpsilonGreedy}(\mathcal{P}, \epsilon)$
  $\mathcal{F} \leftarrow \{h_{\text{root}}\}$ // BFS frontier (layer 0)
  $\mathcal{S} \leftarrow \emptyset$
  for $\ell = 1$ **to** d **do**
    $\mathcal{C}_\ell \leftarrow \emptyset$
    for each parent $h \in \mathcal{F}$ **do**
      Select $\Phi_\theta \in \{\Phi_\theta^{\text{expn}}, \Phi_\theta^{\text{comp}}, \Phi_\theta^{\text{steer}}\}$ using UCB with coefficient c_t
      $\tilde{z} \leftarrow \Phi_\theta(z_h \mid \mathcal{C})$ // perturbation
      $\{h^{(1)}, \ldots, h^{(\kappa)}\} \sim \Psi_\theta(h, \tilde{z}, \Phi_\theta \mid \mathcal{C})$ // rewrite
      Evaluate all $h^{(j)}$ using g; $t \leftarrow t + \kappa$
      Update $\mathcal{H}$ using the best offspring improvement; $\{\mathcal{C}_\ell, \mathcal{S}\} \leftarrow \{\mathcal{C}_\ell, \mathcal{S}\} \cup \{h^{(j)}\}_{j:\text{valid}}$
      if $t \geq T$ **then**
        break
      end if
    end for
    if $t \geq T$ or $\mathcal{C}_\ell = \emptyset$ **then**
      break
    end if
    $\mathcal{F} \sim \text{Sample}(\mathcal{C}_\ell, p(h_i))$ // According to Eq. (4)
  end for
  $\mathcal{P} \leftarrow \pi_{\text{sel}}(\mathcal{P} \cup \mathcal{F})$ // preserve best and randomly retain remaining candidates
  Update $h^\star$ using all evaluated offspring in $\mathcal{S}$
  if no global-best improvement for $\mathcal{C}$ evaluations **then**
    $\mathcal{P} \leftarrow \{h^\star\}$; $t' \leftarrow t$; clear $\mathcal{H}$ // Elite Pool Reset + Operator Reheating
  end if
end while
return Best valid heuristic $h^\star$ found

D Detailed Methodology

D.1 PRS Prompt

PRS uses a shared prompt library across all problem settings. It contains two system prompts that regulate the behavior of the LLM as an idea generator and as a code generator, respectively, as well as seven workflow prompt templates. These templates are used for: (i) describing the seed heuristic as a natural-language idea, (ii) generating initial heuristic ideas, (iii) applying the Expansion operator, (iv) applying the Compression operator, (v) applying the Steering operator, (vi) realizing an initial idea as executable code, and (vii) rewriting a parent implementation to realize a target idea.

Besides those common prompts, each problem setting contains two types of problem-specific prompts: function contract which dictates the format (including the names and types of inputs and outputs) and robustness constraints of designed heuristics, and a task description prompt (i.e. problem description). In this section, we describe their meaning and prompt engineering in detail, where we highlight the uniqueness of the prompt design in red colors. For problem-specific prompt components, we take the problem setting Knapsack Problem in step-by-step constructive framework as an example.

- **Task Description and Function Contract:** While the seed heuristics are only used once in the heuristic pool initialization prompt, the task description and function contract are employed not only during initialization, but also throughout the perturbation and rewrite stages of the evolutionary process. The robustness constraints in function contract do not add inductive bias that explicitly improves the performance of heuristics. Instead, they ensures the validity and consistency of the designed heuristics (e.g., the return must be of the same shape as the distance matrix). Below, we provide examples of the task description and function contract.

Task Description Prompt

Solving 0-1 Knapsack Problem (KP) with constructive heuristics. The goal is to incrementally build a feasible solution by selecting items one by one, so as to maximize the total value under a weight (capacity) constraint. Each item can be selected at most once. At each step, the heuristic observes the current remaining capacity and the full list of item weights and values, and decides which item to select next. The selection is irreversible and directly affects future decisions.

Function Contract Prompt

You must implement the function:
def select_next_item(remaining_capacity, weights, values):
    return next_item

inputs:
-remaining_capacity: int
-weights: np.ndarray
-values: np.ndarray
outputs:
-next_item: int

Robustness constraints:
- The select_next_item function takes as input the remaining capacity, a numpy array of item values, and a numpy array of item weights, and returns the next item to pick.

- **System Prompts:** PRS uses two system prompts to separate semantic heuristic reasoning from executable code generation. The *system idea prompt* instructs the LLM to reason at the level of decision principles, interactions, and hyperparameters that affect behaviors, and to produce a concrete natural language heuristic idea. The *system code prompt* instructs the LLM to translate a supplied heuristic idea into a complete executable implementation. This separation allows the three directional search primitives to operate directly on explicit heuristic ideas, while code realization is guided by the resulting target idea and the required function interface.

System Idea Prompt

You are an expert in optimization heuristics and algorithm design. Your task is to reason at the level of heuristic ideas, decision principles, interactions, and search logic. Do NOT output Python code.
Describe the heuristic idea concretely enough that another expert can implement it without guessing important decision logic.
Clearly explain what information or signals are used, how they affect decisions, how different mechanisms interact, and when the behavior changes.
For every behavior-affecting hyperparameter, threshold, schedule, or fallback, state its role and concrete value or rule. Do not leave implementation-critical parameter choices implicit.
Focus on algorithmic and heuristic logic rather than Python syntax, variable names, code length, or implementation layout.
Avoid cosmetic wording changes and parameter-only tweaks unless they meaningfully change the decision mechanism.
Return only the heuristic idea description.

System Code Prompt

You are an expert in optimization heuristics and Python implementation. Your task is to translate the supplied heuristic idea into a complete executable Python heuristic.
Return only Python code wrapped in ‘python ... ‘.
Write all necessary imports inside the function when needed.
Do not output explanations, comments, or docstrings.
Do NOT use type annotations in the function signature.

- **Initialization Prompts:** PRS initializes the heuristic pool in two stages. The initial idea prompt generates a distinct heuristic idea from the seed idea, and the initial realization prompt translates it into executable code using the seed implementation as context.

Initial Idea Prompt

Task:
`{task_desc}`

Function Format:
`{contract}`

Seed Heuristic Idea:
`{seed_idea}`

Propose one distinct heuristic idea for the same task. The proposal must remain compatible with the required function format while introducing a **meaningful algorithmic change** rather than cosmetic rewriting.

Description Requirement:
Describe the proposed heuristic concretely. Explain its main decision principle, what information it uses, how that information influences decisions, how different mechanisms interact, and any conditions under which its behavior changes.

The description should be sufficiently specific to guide a separate implementation step without requiring the implementer to invent missing heuristic logic.

Output Requirement:
Do not discuss Python syntax or implementation layout. Do not output code.
Return only the proposed heuristic idea description.

Initial Realization Prompt

Function Name:
`{func_name}`

Task:
`{task_desc}`

Function Format:
`{contract}`

Target Heuristic Idea:
`{target_idea}`

Seed Implementation for Context:
`{seed_code}`

Implement the TARGET HEURISTIC IDEA as a complete executable heuristic. The target idea is the semantic specification. Use the seed implementation only as low-level implementation context when useful.

Implementation Requirement:
Faithfully realize the stated decision principles, information signals, interactions, conditions, and adaptive behavior. Do not invent additional heuristic mechanisms that are not implied by the target idea.
When low-level implementation details are unspecified, choose simple implementations that preserve the intended decision logic.

Output Requirement:
Return only valid Python code wrapped in "`` ```python ... ``` ``".

- **Multi-directional Idea Operators:** During BFS search, PRS applies one of three idea operators to transform each parent idea into a target idea. Expansion enriches the heuristic with new decision mechanisms, Compression removes or consolidates unnecessary mechanisms, and Steering improves the current strategy through localized modifications. The target idea is subsequently realized by a separate code generation prompt.

Expansion Operator Prompt

Task:
`{task_desc}`

Current Heuristic Idea:
`{parent_idea}`

Perform IDEA EXPANSION. Develop the current heuristic into a richer algorithmic idea while preserving its useful backbone. Introduce meaningful new decision logic, such as additional information signals, interactions between existing mechanisms, context-dependent rules, or adaptive behavior.

Expansion Requirement:
Do not simply append independent terms. Explain specifically how every newly introduced mechanism changes the heuristic's decisions and how it interacts with the retained mechanisms.

The resulting description must be concrete enough to guide implementation directly: state what information is used, how it influences decisions, how mechanisms are combined, and under what situations the behavior changes.

Output Requirement:
Do not increase complexity merely through extra wording, constants, parameters, duplicated terms, or implementation details.
Do not output code. Return only the expanded heuristic idea description.

Compression Operator Prompt

Task:
`{task_desc}`

Current Heuristic Idea:
`{parent_idea}`

Perform IDEA COMPRESSION. Simplify the current heuristic at the level of algorithmic reasoning while preserving its most useful core principle. Remove, merge, replace, or decouple mechanisms that are redundant, weak, overly specific, or unnecessarily interacting.

Compression Requirement:
When removing or merging a mechanism, explain how the remaining mechanisms should compensate for its role so that the resulting heuristic remains complete and directly implementable.

The resulting description must still specify a complete decision rule: what information is retained, how it influences decisions, how the remaining mechanisms interact, and when behavior changes.

Output Requirement:
Compression must reduce conceptual burden rather than merely shorten the description, delete syntax, remove lines, or simplify implementation details.
Do not output code. Return only the compressed heuristic idea description.

Steering Operator Prompt

Task:
`{task_desc}`

Current Heuristic Idea:
`{parent_idea}`

Perform IDEA STEERING. Preserve the core heuristic strategy and its overall conceptual structure, but improve it through localized modifications. Exploit the strengths of the current idea rather than redirecting it toward a fundamentally different heuristic family.

Steering Requirement:
Focus on fine-grained changes such as adjusting how existing signals are weighted or combined, modifying thresholds, schedules, or parameter settings, refining conditional logic or activation conditions, changing local interactions between existing mechanisms, or strengthening one specific part without altering the overall strategy.

Parameter-only changes are allowed only when they induce a meaningful behavioral change. Avoid purely cosmetic rewriting.

Output Requirement:
Describe the locally steered heuristic concretely enough for direct implementation: state what local mechanism changes, how it affects decisions, and when it is active.
Do not output code. Return only the steered heuristic idea description.

- **Heuristic Rewrite Prompt:** After an idea operator generates a target idea, PRS rewrites the parent implementation to realize that idea as executable code. The prompt receives the target idea, the selected operation, and the parent implementation. The target idea serves as the semantic specification of the offspring, while the operation specifies the intended type of change. The rewrite may modify the parent code as needed, but it must preserve parent mechanisms only when they remain consistent with the target idea.

Heuristic Rewrite Prompt

Function Name:
`{func_name}`

Task:
`{task_desc}`

Function Format:
`{contract}`

Target Heuristic Idea:
`{target_idea}`

Operation that Produced the Target Idea:
`{operation}`

Parent Implementation:
`{parent_code}`

Rewrite the parent implementation into a complete executable heuristic that faithfully realizes the TARGET HEURISTIC IDEA.

The target idea is the semantic specification of the new heuristic; the operation states the intended kind of change.

Rewrite Requirement:
You may add, remove, replace, or reorganize any logic necessary to realize the target idea completely. Preserve parent mechanisms only when they remain consistent with the target idea. Do not introduce unrelated mechanisms.

Faithfully instantiate the decision principles, information signals, interactions, conditional behavior, and adaptive mechanisms described in the target idea. Do not introduce additional heuristic mechanisms that are not implied by the target idea.

When low-level implementation details are unspecified, choose the simplest implementation consistent with the target idea.

Output Requirement:
Return only valid Python code wrapped in “‘`python ...` “‘.

D.2 GENERATED HEURISTICS

This subsection provides example outputs of PRS-designed heuristics. The problem settings include a step-by-step constructive framework for the online bin packing problem and the ant colony optimization framework for the orienteering problem.

Evolved Heuristic #1: priority function for OBPP with step-by-step constructive framework

```
def priority(item: float, bins_remain_cap: np.ndarray) -> np.ndarray:
    """
    This heuristic scores feasible bins using residual capacity, balanced
     utilization, and protection against premature bin exhaustion. It favors tight
     placements and moderate post-placement fill levels while giving additional
     preference to near-exact fits and already utilized bins.
    """
    import numpy as np
    eps = 1e-9
    tight_threshold = item * 0.1
    remaining = bins_remain_cap - item
    capacity_scale = bins_remain_cap.max() if bins_remain_cap.size > 0 else 1.0
    normalized_remaining = remaining / capacity_scale
    post_fill = 1 - normalized_remaining
    desired_fill = 0.5

    # Favor small residual capacity and balanced post-placement utilization
    residual_score = 1 / (remaining + eps)
    fill_gap = post_fill - desired_fill
    balance_score = np.exp(-20 * (fill_gap ** 2))

    # Discourage opening or repeatedly using nearly empty bins
    unused_penalty = np.exp(-10 * (normalized_remaining > (1 - item /
     capacity_scale)) * normalized_remaining)

    # Reward tight placement, near-exact fit, and previously utilized bins
    tight_bonus = np.where(
        remaining <= tight_threshold,
        1.5 + (tight_threshold - remaining) / tight_threshold,
        1.0
    )
    utilization = 1 - (bins_remain_cap / capacity_scale)
    utilization_bonus = 1 + 0.5 * utilization
    exact_match = (remaining < eps * 10).astype(float)
    exact_bonus = 1 + 0.2 * exact_match

    scores = residual_score * balance_score * unused_penalty * tight_bonus *
     utilization_bonus * exact_bonus
    return scores
```

Evolved Heuristic #2: Heuristic Matrix for OP with ACO framework

```
def heuristics(prize: "np.ndarray", distance: "np.ndarray", maxlen: float) ->
     "np.ndarray":
    """
    Edge-scoring heuristic that integrates reward, distance, and local
     neighborhood structure under a route-length constraint. It favors profitable,
     nearby, and well-connected transitions while suppressing moves that are
     difficult to complete feasibly.
    """
```

Evolved Heuristic #2 (cont.)

```
import numpy as np
    eps = 1e-9
    n_nodes = prize.shape[0]
    depot_dist = distance[:, 0]
    prize_scale = np.max(prize[1:]) if n_nodes > 1 else 1.0
    scaled_prize = prize / (prize_scale + eps)
    inverse_dist = np.zeros_like(distance)
    valid_dist = (distance > 0)
    inverse_dist[valid_dist] = 1.0 / (distance[valid_dist] + eps)

    prize_power, dist_power, density_power, connect_power = 7.0, 5.0, 4.0, 3.0
    depot_weight, reward_threshold = 2.2, 0.25
    local_radius = maxlen / 4.0
    high_reward = (scaled_prize >= reward_threshold)
    local_mask = (distance <= local_radius) & (distance > 0)
    local_mask = local_mask & high_reward[None, :]
    local_density = local_mask.sum(axis=1)
    density_scale = np.max(local_density) if np.max(local_density) > 0 else 1.0
    normalized_density = local_density / density_scale
    density_bonus = 1.0 + normalized_density  # reward nodes near many high-prize
     neighbors

    connect_radius = maxlen / 2.0
    connect_mask = (distance <= connect_radius) & (distance > 0) &
     high_reward[None, :]
    node_connectivity = connect_mask.sum(axis=1)
    connectivity_scale = np.max(node_connectivity) if np.max(node_connectivity) >
     0 else 1.0
    normalized_connectivity = node_connectivity / connectivity_scale
    weak_node_penalty = np.where(normalized_connectivity < 0.3, 0.4, 1.0)
    connectivity_penalty = weak_node_penalty[:, None] * weak_node_penalty[None, :]
      # downweight weakly connected nodes

    return_length = distance + depot_dist[None, :]
    normalized_depot_dist = depot_dist / (maxlen + eps)
    route_depth = (return_length + depot_dist[:, None]) / (maxlen + eps)
    route_depth = np.clip(1.0 - route_depth, 0.0, 1.0)
    infeasible_edges = return_length > maxlen  # forbid edges that cannot return
     to depot within maxlen

    depot_affinity = 1.0 - normalized_depot_dist
    depot_affinity = np.clip(depot_affinity, 0.0, 1.0)
    depot_bonus = 1.0 + depot_weight * ((depot_affinity[:, None] +
     depot_affinity[None, :]) / 2.0)

    edge_prize = (scaled_prize[:, None] + scaled_prize[None, :]) / 2.0
    ring_index = np.floor(normalized_depot_dist * 3).astype(int)
    ring_distance = np.abs(ring_index[:, None] - ring_index[None, :])
    ring_penalty = np.where(ring_distance > 1, 0.5, 1.0)

    structure_score = ((density_bonus[:, None] * density_bonus[None, :]) **
     density_power * (connectivity_penalty ** connect_power))

    heuristic_matrix = ((edge_prize ** prize_power) * (inverse_dist ** dist_power)
     * structure_score * route_depth * depot_bonus * ring_penalty)
    heuristic_matrix[infeasible_edges] = 0.0
    np.fill_diagonal(heuristic_matrix, 0.0)
    heuristic_matrix = np.nan_to_num(heuristic_matrix, nan=eps, posinf=eps,
     neginf=eps)
    heuristic_matrix = np.maximum(heuristic_matrix, 0.0)
    return heuristic_matrix
```

Evolved Heuristic #3: Black-Box Heuristic (Edge Scoring) for OP with ACO framework

```
def heuristics(
    node_attr: np.ndarray, edge_attr: np.ndarray, node_constraint: float
) -> np.ndarray:
    """
    This heuristic ranks edges using endpoint attributes and inverse edge cost,
    with a soft penalty when the associated node-based constraint is exceeded.
    """
    n_nodes = len(node_attr)
    score_matrix = np.zeros((n_nodes, n_nodes))
    for i in range(n_nodes):
        for j in range(n_nodes):
            if edge_attr[i, j] > 0:
                combined_cost = np.sum(edge_attr[i]) + np.sum(edge_attr[:, j])
                base_score = (node_attr[i] + node_attr[j]) / edge_attr[i, j]
                if combined_cost > node_constraint:
                    score_matrix[i, j] = base_score * 0.5
                else:
                    score_matrix[i, j] = base_score
            else:
                score_matrix[i, j] = -1
    score_matrix = np.clip(score_matrix, -1, None)
    score_matrix = (score_matrix - np.mean(score_matrix)) / (np.std(score_matrix)
    + 1e-10)
    return score_matrix
```

E Details of Evaluation Configurations

Hyperparameters for PRS. To initialize the heuristic pool, we set the number of initially sampled heuristics to 30, aligning with ReEvo (Ye et al., 2024). The softmax temperature is fixed to $T_{sel} = 1.0$ for all tasks. The base temperature of the LLMs for both perturbation and rewrite is 1.0. During initialization, the LLM temperature are further increased by 0.2, to enhance diversity. Our code will be released upon acceptance.

For adaptive operator control, we set the initial and final UCB exploration coefficients to $c_0 = 2.0$ and $c_T = 0.5$, respectively, with a sliding reward window of 50 operator selections. The stagnation threshold and the post-restart operator reheating horizon are set to $C = 0.1T$ and $\Delta = 0.1T$, respectively. We set an evaluation timeout of 60s for each heuristic in PRS following MCTS-AHD (Zheng et al., 2025). For FSSP-GLS, this timeout is applied per instance within each heuristic evaluation.

Use of Large Language Models. For most experiments, we use GPT-5-nano-2025-08-08 under its default reasoning mode to ensure consistency. In some problem settings, we additionally adopt GPT-4.1-nano to examine the adaptability of PRS. We further evaluate both closed-source and open-source LLMs to demonstrate the applicability and flexibility of our framework, with additional results reported in the Appendix F.9. In designing heuristics under the ACO framework for CVRP, PRS uses on average 3.09M input tokens and 1.55M output tokens per run, with 1353 LLM queries and a total wall-clock time of 6h. Detailed runtime, token, and query-cost comparisons are reported in Appendix G.

Generation of Validation Sets D and Testing Setups. We follow the procedures in FunSearch (Romera-Paredes et al., 2024), EoH (Liu et al., 2024a), ReEvo (Ye et al., 2024) and MCTS-AHD to construct all validation datasets D and testing instances. In particular, MCTS-AHD employs a varying-scale validation dataset D for online BPP to facilitate the discovery of heuristics with improved generalization performance on a broader scale of testing instances, following the settings of ELG (Gao et al., 2024). The complete evaluation configurations are summarized as follows:

- **ACO Framework**. We adopt the configurations (i.e. number of ants and iterations) in ReEvo (Ye et al., 2024). **(a)** For CVRP (both black- and white-box settings), PRS is validated on a

dataset of 10 instances with problem size 50 and tested on three sets of 64 instances each at size 50, 100, and 200. During both validation and testing, the number of ants and the number of iterations are fixed to 30 and 100, respectively. **(b)** For OP (both black- and white-box), the size of the dataset D is 5 and the validation problem size is also 50. As for evolved heuristics for OP, they are tested with three sets of 64 instances each at size 50, 100, and 200. The number of ants is 20 while the number of iterations is 50 for both validation and testing. **(c)** For black-box MKP, the validation dataset consists of 5 instances of problem size 100. The testing set consists of two sets of 64 instances each at sizes 100 and 200. The number of ants is 10 and the number of iterations is 50 and 100 for validation and testing, respectively.

- **Step-by-step Construction Framework**. **(a)** For KP, PRS is validated on a dataset D of 64 instances with problem size 100. Upon testing, the evolved heuristic is tested on three sets of 1,000 instances each at size 50, 100, 200. The capacity of size 50 is 25 while the capacity of size 100 and 200 is 50. **(b)** For Online BPP, PRS generates a dataset D consisting of 4 instances: a 1,000-item instance with $W = 100$, a 1,000-item instance with $W = 500$, a 5,000-item instance with $W = 100$, and a 5,000-item instance with $W = 500$. The number of problem instances for each test set in Table 2 is 5. **(c)** For TSP, we use a validation set of 64 TSP instances with $N = 50$ nodes and three test sets of 1,000 instances each at $N = 50$, 100, and 200. **(d)** For ASP, PRS uses a validation set D consisting of a single instance with $(n, w) = (15, 10)$. Upon testing, the evolved heuristic is evaluated on four settings, namely $(12, 7)$, $(15, 10)$, $(21, 15)$, and $(24, 17)$, with one instance for each setting.

- **GLS Framework. (a)** For FSSP, we follow EoH to generate D: a dataset of 64 instances is randomly generated with a fixed number of 50 jobs and the number of machines is randomly chosen from $U[2, 20]$. Details of the testing instances are provided in Appendix F.5. The maximum number of iterations is 1,000. The per instance timeout is 60s. **(b)** For TSP, we follow ReEvo to generate D of 64 instances of problem size 200 and keep the same iteration limits and perturbation moves, which are 1200 and 30, respectively. We evaluated the heuristics on a wide range of instances as shown in Table 10. The details of the hyperparameters chosen for GLS are provided in Table 5.

Hyperparameters for Baselines. We use the default hyperparameters reported in their papers to run other baseline AHD methods. However, to ensure a fair comparison, we set the maximum evaluation steps for ReEvo and HSEvo to $T = 1000$ in our reported results to allow the convergence of the heuristic search. In EoH, the number of generations is fixed at 20, with population sizes of 20 for online BPP and 10 for TSP and FSSP, corresponding to 810 evaluations for TSP and FSSP and 1620 evaluations for online BPP. Furthermore, they observed that the search of FSSP's heuristic can converge within about 810 evaluations. MCTS-AHD uses $T = 2000$ for online BPP and $T = 1000$ for all remaining tasks and follows EoH in the maximum evaluation steps. Therefore, we adopt the same maximum number of evaluation steps as MCTS-AHD T for PRS. Specifically, we use $T = 2000$ for OBPP, while keeping $T = 1000$ for the remaining problems.

Table 5: GLS parameter settings for different TSP problem scales. The number of perturbation moves controls the strength of solution diversification, while the iteration limit determines the overall local search budget. Heuristics for TSP under GLS found by all LLM-based AHD frameworks are tested under the same configurations here.

Problem Size	Perturbation Moves	Iteration Limit
TSP20	5	73
TSP50	40	300
TSP100	40	1800
TSP200	40	1000
TSP500	60	250
TSP1000	80	100

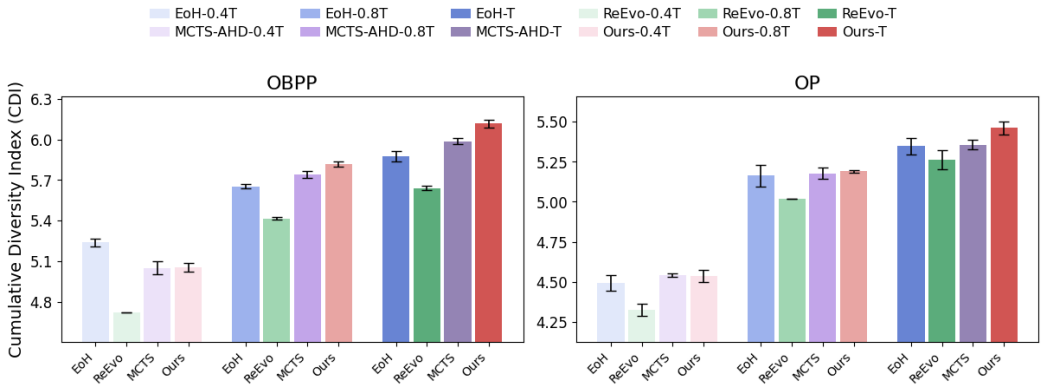


Figure 4: Diversity analysis (CDI) of EoH, ReEvo, MCTS-AHD, and PRS on OBPP-Constructive and OP-ACO at different search stages ($0.4T$, $0.8T$, and T). Error bars indicate standard deviation over three independent runs.

F More Experimental Results

F.1 Search Dynamic Analysis

F.1.1 Diversity Analysis

Diversity in evolutionary search provides valuable insights into the exploration strength and the convergence performance of AHD algorithms (Pires et al., 2014). A substantial body of optimization research has investigated the impact of estimating and maintaining diversity within the evolutionary search (Pires et al., 2019; Wang & Chen, 2012; Dat et al., 2025). The evolution curves alone in Section 5 are not sufficient to quantify the diversity within the heuristic search process of our PRS framework. Therefore, we adopt Cumulative Diversity Index (CDI) to quantify diversity, which is the same metric used in previous work, HSEvo (Dat et al., 2025). In the context of heuristic search, CDI reflects how widely heuristic candidates are dispersed in the search space, serving as an implicit signal of system energy, or diversity (Jost, 2006).

To align with the approaches in calculating CDI in HSEvo, we utilized the cumulative archive of the embeddings of the top-20 candidates in every 80 evaluation steps to measure CDI for three other AHD baselines (i.e. EoH, ReEvo, MCTS-AHD) and our PRS framework. CDI is calculated by constructing a Minimum Spanning Tree (MST) of all embeddings in the archive A with edge lengths measured by Euclidean distance. The MST is then used to assess heuristic diversity. Let d_i denote the length of the i-th edge in the minimum spanning tree (MST) constructed over the archive A, where $i \in \{1, 2, \ldots, |A| - 1\}$. The cumulative diversity is computed using the Shannon entropy as

$$D_{\text{CDI}} = -\sum_{i=1}^{|A|-1} p_i \log p_i. \tag{19}$$

where the corresponding probability distribution of distance d_i is defined as

$$p_i = \frac{d_i}{\sum_{j=1}^{|A|-1} d_j}, \quad i = 1, \ldots, |A| - 1. \tag{20}$$

A higher CDI reveals a more diverse dispersion of heuristic structures in the search space, and thus suggests more robust exploration dynamics. We mainly consider three stages to demonstrate the diversity of search in different phases of heuristic search: $\frac{2}{5}T$, $\frac{4}{5}T$, and T within the maximum evaluation steps T, corresponding to the early, middle, and final stages of heuristic search ($T = 1000$ for OP and $T = 2000$ for OBPP).

As shown in Figure 4, all four AHD frameworks exhibit ascending CDI values as the evaluation steps grow, indicating that their cumulative heuristic archives become progressively more diverse during search. Across both OBPP and OP, PRS consistently achieves the highest or near-highest CDI at the middle and final stages and attains the largest final stage diversity among all compared methods.

Compared to other baselines, PRS's diversity growth is particularly pronounced from $0.8T$ to T, suggesting that PRS maintains active exploration even in the late stage of the search rather than rapidly collapsing into a narrow set of heuristic structures. This delayed convergence indicates a stronger capability to avoid premature convergence and preserve diverse heuristic candidates throughout the search process.

F.1.2 IDEA-SPACE DIVERSITY ANALYSIS

CDI measures diversity in the learned representation space of realized heuristic implementations, while mean nearest-neighbor distance (MNND) measures diversity in the semantic idea space before code realization. This distinction is important for PRS because its directional perturbation primitives operate directly on natural language heuristic ideas. Distance-based criteria have long been used to characterize dispersion and space-filling properties in multidimensional spaces (Johnson et al., 1990; Morris & Mitchell, 1995), while mean nearest-neighbor distance provides a direct measure of local packing and redundancy (Layman et al., 2007). In the context of AHD, MNND characterizes how semantically distinct each retained heuristic idea is from its closest alternative in the archive: a larger value indicates fewer locally redundant ideas and broader coverage of distinct conceptual neighborhoods in the heuristic design space. For each distinct heuristic idea z_i in the cumulative archive, we obtain an L2-normalized sentence embedding $\mathbf{e}_i$ using `sentence-transformers/all-mpnet-base-v2`. The nearest-neighbor semantic distance of z_i is defined using cosine distance, and MNND is computed as

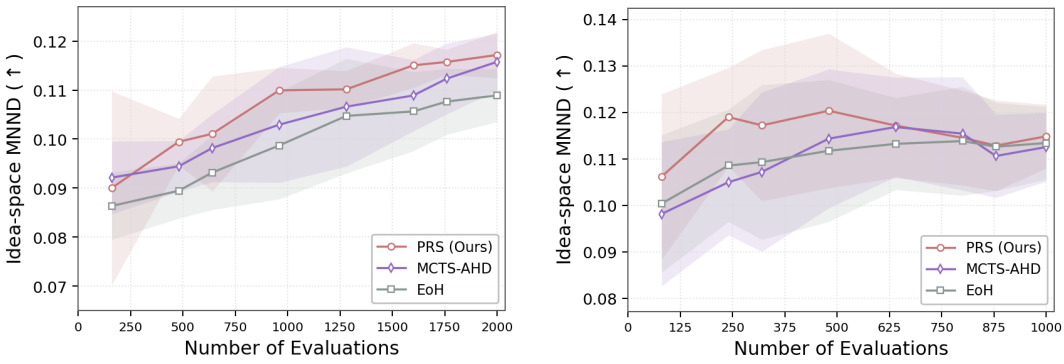
$$D_{\mathrm{MNND}} = \frac{1}{N}\sum_{i=1}^{N}\min_{j\neq i}\left(1-\frac{\mathbf{e}_i^{\top}\mathbf{e}_j}{\|\mathbf{e}_i\|_2\|\mathbf{e}_j\|_2}\right), \tag{21}$$

where N denotes the number of distinct ideas in the archive. By focusing on each idea's nearest semantic neighbor, MNND reflects local redundancy and sustained exploration across distinct regions of the heuristic idea space.

We follow the same archive construction protocol as in the CDI analysis, adding the top-20 valid candidates every 80 evaluations to a cumulative archive. Exact duplicate ideas are counted only once for MNND to avoid bias from repeated code realizations. The same treatment is applied to all compared methods. We evaluate PRS against EoH and MCTS-AHD, both of which explicitly maintain natural language heuristic descriptions. To minimize differences caused merely by the wording or style of the thought-generation prompts, we additionally standardize the natural language idea-generation prompt across the three methods while retaining their original search mechanisms. Thus, the comparison primarily reflects differences in how the respective search frameworks explore heuristic ideas rather than differences in description style. Results are averaged over three independent runs, with shaded regions denoting standard deviation.

As shown in Figure 5, PRS generally explores less locally redundant regions of the idea space than EoH and MCTS-AHD across both tasks. On OBPP-Constructive, although MCTS-AHD shows a slightly larger MNND at the earliest checkpoint, PRS subsequently maintains the highest mean MNND and finishes at approximately 0.117, compared with 0.116 for MCTS-AHD and 0.109 for EoH. On OP-ACO, PRS shows a clearer advantage during the early and middle stages, reaching an MNND of approximately 0.120, while EoH and MCTS-AHD remain lower; the gap narrows toward the end, but PRS still retains the highest final mean MNND. These results indicate that the directional perturbation primitives allow PRS to reach distinct semantic neighborhoods earlier and sustain broader idea space exploration throughout the search.

Together with the CDI results, the MNND analysis provides two complementary views of search diversity. CDI shows that PRS maintains broad dispersion among the realized heuristic implementations, whereas MNND shows that this diversity is already present at the level of semantic heuristic ideas. Their consistent trends suggest that the broader exploration of PRS is not merely a consequence of superficial variation in code realization. Instead, the explicit expansion, compression, and steering primitives promote sustained exploration of distinct regions of the heuristic design space before these ideas are instantiated as executable heuristics.



(a) OBPP-Constructive (b) OP-ACO

Figure 5: Idea-space diversity measured by mean nearest-neighbor distance (MNND) for PRS, MCTS-AHD, and EoH. Higher values indicate that retained heuristic ideas remain farther from their closest semantic neighbors.

F.1.3 Operator Contribution Analysis

In this section, we further analyze how the three directional perturbation operators complement each other during search and how their contributions vary across different tasks and search stages. We measure the cumulative share of best-so-far improvements triggered by each operator across three representative settings: FSSP-GLS, KP-Constructive, and CVRP-ACO, based on five independent runs for each setting (Figure 6).

1) *FSSP-GLS.* In FSSP-GLS, compression is the dominant contributor, accounting for approximately 46% of the cumulative improvements, and its contribution continues to increase throughout the search trajectory. Expansion contributes about 32%, with most improvements occurring during the earlier and middle stages, while steering accounts for the remaining 22%. This pattern suggests that both enriching and simplifying heuristic structures are useful in FSSP-GLS, with compression playing a particularly persistent role throughout the search.

2) *KP-Constructive.* In KP-Constructive, the pattern changes substantially: expansion becomes the largest contributor at approximately 42%, while steering also contributes strongly at around 37%. Compression accounts for a smaller share of approximately 21%, with most of its improvements occurring relatively early in the search. The comparable contributions of expansion and steering indicate that KP benefits not only from exploring richer heuristic structures, but also from fine-grained exploitation around promising regions.

3) *CVRP-ACO.* In CVRP-ACO, compression again becomes the dominant contributor, accounting for approximately 47% of cumulative improvements, followed by steering at around 37%, whereas expansion contributes about 16%. Compression accumulates improvements across the search trajectory, while steering also remains active across multiple stages. This result shows that effective search in CVRP-ACO relies more strongly on structural simplification and local refinement than on continued structural expansion.

Stage-wise UCB behavior. To further examine the stage-wise behavior of adaptive operator selection, Table 6 reports the stage-wise behavior of the UCB controller. For each operator, we consider its selection rate, the probability of producing at least one improving beam child, and the mean best-child relative improvement. The results show that contribution is not determined by selection frequency alone. Across tasks and search stages, operators with similar selection rates can exhibit markedly different improvement probabilities and improvement magnitudes, indicating that the three perturbation directions provide distinct search utility even when they receive comparable amounts of search effort.

The stage-wise statistics also reveal clear task-dependent differences. In CVRP-ACO, compression maintains consistently strong improvement rates across later stages, which is consistent with its large cumulative contribution. In FSSP-GLS, compression frequently achieves high improvement rates, while steering often produces larger best-child improvements in several stages. For OBPP-

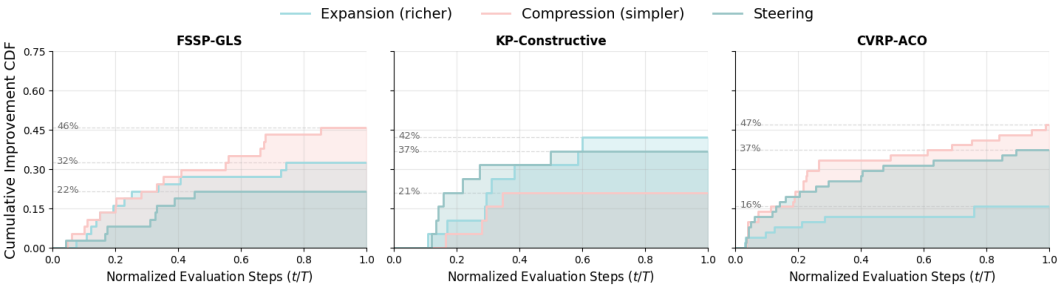


Figure 6: Operator contribution statistics. Each curve represents the cumulative share of best-so-far improvements triggered by each perturbation operator across all runs.

Table 6: Stage-wise behavior of UCB-based operator selection. For each operator, we report the selection rate (Sel.), any-improvement rate (Imp.), and mean best-child relative improvement (Gain), all in %. Statistics are aggregated over five runs.

Task	Evaluation stage	Expansion			Compression			Steering		
		Sel.	Imp.	Gain	Sel.	Imp.	Gain	Sel.	Imp.	Gain
OBPP-Constructive	0–249	32.42	35.21	3.241	34.25	26.67	5.881	33.33	41.10	4.849
	250–499	32.13	23.75	1.314	34.14	12.94	0.678	33.73	16.67	2.947
	500–749	34.15	20.24	3.541	32.11	16.46	2.415	33.74	22.89	1.958
	750–999	31.43	16.88	2.632	31.02	10.53	4.006	**37.55**	**23.91**	**7.667**
CVRP-ACO	0–249	31.36	**66.67**	3.947	34.09	**66.67**	4.059	**34.55**	53.95	**6.751**
	250–499	32.40	51.85	**2.867**	**34.40**	**70.93**	2.171	33.20	45.78	2.513
	500–749	30.49	52.00	2.220	**36.59**	**65.56**	**3.554**	32.93	43.21	3.106
	750–999	33.73	54.12	3.244	**34.92**	**68.18**	**3.872**	31.35	60.76	3.269
FSSP-GLS	0–249	33.78	44.00	3.329	31.98	**74.65**	4.735	34.23	59.21	**5.476**
	250–499	31.98	56.96	3.557	**34.01**	**61.90**	3.236	**34.01**	51.19	**4.500**
	500–749	32.11	46.84	1.522	**34.55**	47.06	1.984	33.33	**47.56**	**3.667**
	750–999	**35.34**	44.32	3.309	31.33	**64.10**	1.388	33.33	46.99	**3.499**

Constructive, the relative utility of the operators changes more noticeably over time, with steering becoming particularly effective in the final stage. Together with Figure 6, these results show that operator effectiveness varies across both tasks and search stages, supporting adaptive allocation among expansion, compression, and steering rather than a fixed operator preference.

F.1.4 Operator-wise Generalization

The preceding operator contribution analysis shows that expansion, compression, and steering contribute differently across tasks and search stages. We further examine whether the strongest candidates produced by different perturbation operators exhibit different in-distribution (ID) and out-of-distribution (OOD) generalization behavior.

For each independent run, we collect the valid children generated by each perturbation operator and rank them according to validation performance. Since AHD search ultimately prioritizes and retains high-performing candidates, we focus on the strongest outcomes produced by each perturbation direction and evaluate the top five children from each operator on both ID test instances and larger OOD instances.

Table 7 shows a consistent qualitative difference between the strongest outcomes produced by the three perturbation directions. On KP-Constructive, expansion achieves the best ID performance, whereas compression achieves the best OOD performance. Steering remains competitive across both evaluation regimes.

Table 7: Generalization performance of the top-five children generated by each perturbation operator. Children are selected according to validation performance within each run. Lower gap is better.

Task	Operator	ID gap ↓	OOD gap ↓
KP-Constructive	Expansion	$\mathbf{0.03\% \pm 0.01\%}$	$0.04\% \pm 0.02\%$
	Compression	$0.06\% \pm 0.02\%$	$\mathbf{0.03\% \pm 0.00\%}$
	Steering	$0.05\% \pm 0.02\%$	$0.05\% \pm 0.03\%$
CVRP-ACO	Expansion	$\mathbf{0.89\% \pm 0.53\%}$	$6.31\% \pm 1.54\%$
	Compression	$1.45\% \pm 0.71\%$	$\mathbf{5.94\% \pm 1.02\%}$
	Steering	$1.26\% \pm 0.43\%$	$6.52\% \pm 0.63\%$

A similar pattern appears on CVRP-ACO, where expansion again performs best on ID instances while compression achieves the strongest performance on larger OOD instances. Steering provides competitive intermediate performance without consistently favoring either regime.

These results do not imply that expansion is generally preferable for ID performance or that compression always improves OOD generalization. Rather, they suggest that different perturbation directions can produce strong heuristics under different evaluation conditions.

F.2 Comparison on Black-box Settings

We further evaluate PRS under black-box evaluation scenarios, which are inherently harder for heuristic search: candidate programs must remain executable under strict interfaces, and improvements must be discovered purely through outcome-driven trial-and-error, making robust diversification essential. Table 8 shows that PRS remains highly competitive or consistently exceeds other AHD frameworks on black-box tasks. These results suggest that PRS can sustain effective exploration even when supervision is extremely sparse, supporting robust heuristic search in opaque environments where objective evaluations are costly.

F.3 TSP Step-by-step Constructive Framework

For TSP, PRS is responsible for designing a function to select the next node to visit based on the node coordinates, the starting point, the distance matrix, and all unvisited nodes. The experiments in Table 9 test the heuristics on four different scales, where PRS consistently achieves the best performance among the compared LLM-based AHD methods under both LLM settings. For each method, we set a 1200s total time limit to ensure a manageable testing time budget and avoid excessive validation cost.

Discussion. Across both LLM settings, PRS consistently outperforms all compared LLM-based AHD methods on TSP50, TSP100, TSP200, and TSP500. In particular, PRS improves upon MCTS-AHD across all four problem sizes under both GPT-4.1-nano and GPT-5-nano, while the advantage remains

Table 8: Performance of **ACO-based heuristics on black-box CVRP, MKP, and OP**. We follow the settings of ReEvo (Ye et al., 2024) in heuristic evolution and run each LLM-based method three times for average performance.

Task	CVRP		MKP		OP	
N=	N=50	N=100	N=100	N=200	N=50	N=100
Method	Obj.↓	Obj.↓	Obj.↑	Obj.↑	Obj.↑	Obj.↑
ACO	18.282	30.030	18.969	38.766	13.570	24.149
DeepACO	8.888	14.932	23.091	41.988	19.867	36.392
LLM-based AHD (GPT-5-nano)						
EoH	9.360	16.700	22.518	40.568	14.776	28.627
ReEvo	9.403	16.693	23.192	42.148	14.717	28.603
MCTS-AHD	9.551	17.070	23.185	42.111	**14.815**	28.448
Ours	**9.351**	**16.627**	**23.207**	**42.174**	14.789	**28.898**

Table 9: Performance of step-by-step construction heuristics on TSP is averaged over three runs. Bold denotes the best non-optimal objective value at each problem size. For each LLM setting, the best LLM-based AHD result in each column is shaded gray. Gaps are the relative difference to the optimal (or near-optimal) objective value obtained by LKH3.

Travelling Salesman Problem (TSP)								
Method	TSP 50		TSP 100		TSP 200		TSP 500	
	Obj.↓	Gap↓	Obj.↓	Gap↓	Obj.↓	Gap↓	Obj.↓	Gap↓
LKH3 (Lin & Kernighan, 1973)	5.680	–	7.762	–	10.715	–	16.524	–
POMO	**5.697**	**0.30%**	**8.001**	**3.08%**	12.897	20.37%	20.190	22.18%
GC (Nearest Neighbour)	6.944	22.26%	9.755	25.67%	13.502	26.02%	20.938	26.71%
LLM-based AHD GPT-4.1-nano								
FunSearch	6.583	15.90%	9.140	17.75%	12.798	19.44%	19.985	20.94%
EoH	6.345	11.71%	8.862	14.17%	12.428	15.99%	19.568	18.42%
MCTS-AHD	6.337	11.57%	8.865	14.21%	12.419	15.91%	19.382	17.29%
PRS (Ours)	6.299	10.90%	8.797	13.33%	12.268	14.50%	19.092	15.54%
LLM-based AHD GPT-5-nano								
FunSearch	6.444	13.46%	9.104	17.28%	12.874	20.15%	20.021	21.16%
EoH	6.377	12.27%	8.755	12.80%	12.563	17.25%	19.874	20.27%
MCTS-AHD	6.299	10.89%	8.694	12.00%	12.364	15.39%	18.825	13.92%
PRS (Ours)	6.215	9.42%	8.568	10.38%	**12.161**	**13.50%**	**18.553**	**12.28%**

evident as the problem scale increases. The results also show that PRS benefits consistently from the stronger GPT-5-nano backbone, achieving lower gaps across all four test sizes. Overall, these results demonstrate that the proposed multi-directional search process remains effective across different LLM backbones and generalizes well from the search setting to substantially larger TSP instances.

F.4 TSP Heuristic under Guided Local Search Framework

To evaluate the effectiveness of PRS in designing penalty heuristics within the guided local search (GLS) framework, we follow the general experimental protocol of Ye et al. (Ye et al., 2024). Specifically, PRS is employed to generate the key penalty heuristic used to construct the knowledge matrices in KGLS (Arnold & Sörensen, 2019), which enables adaptive refinement of the original manually designed heuristic.

All heuristic generation and evaluation procedures are conducted using GPT-5-nano as the backbone LLM. We evaluate the resulting heuristics on different problem scales of TSP, and compare against representative LLM-based AHD baselines, the original KGLS with handcrafted penalty functions, as well as two neural combinatorial optimization (NCO) methods, namely NeuralOpt (Ma et al., 2021) and GNNGLS (Hudson et al., 2022). NeuralOpt is an NCO method that learns to improve the solutions while GNNGLS is primarily based on GLS framework. The GLS hyperparameters are scaled with problem size to balance exploration and computational cost. Specifically, the number of perturbation moves and the iteration limit are adaptive to problem sizes, as summarized in Table 5. This adaptive budgeting ensures fair comparison across problem scales while avoiding excessive local search on large instances.

As shown in Table 10, PRS consistently improves the performance of KGLS across different problem scales. In particular, PRS achieves lower optimality gaps on TSP100 and TSP200 compared to other LLM-based AHD methods, while maintaining competitive or lowest runtime. Notably, our approach consistently reaches zero gaps on small-scale problems (TSP 50-100) and achieves the lowest gap of 1.649% on TSP 500 and the second lowest on TSP 200. Together with Table 9, these findings further demonstrate that our multi-directional perturbation–rewrite framework is effective and adaptable across different heuristic formulations and problem scales.

Pareto analysis of runtime and solution quality. Beyond reporting the final best-performing heuristic in Table 10, we further analyze the quality–efficiency tradeoff of the heuristics discovered

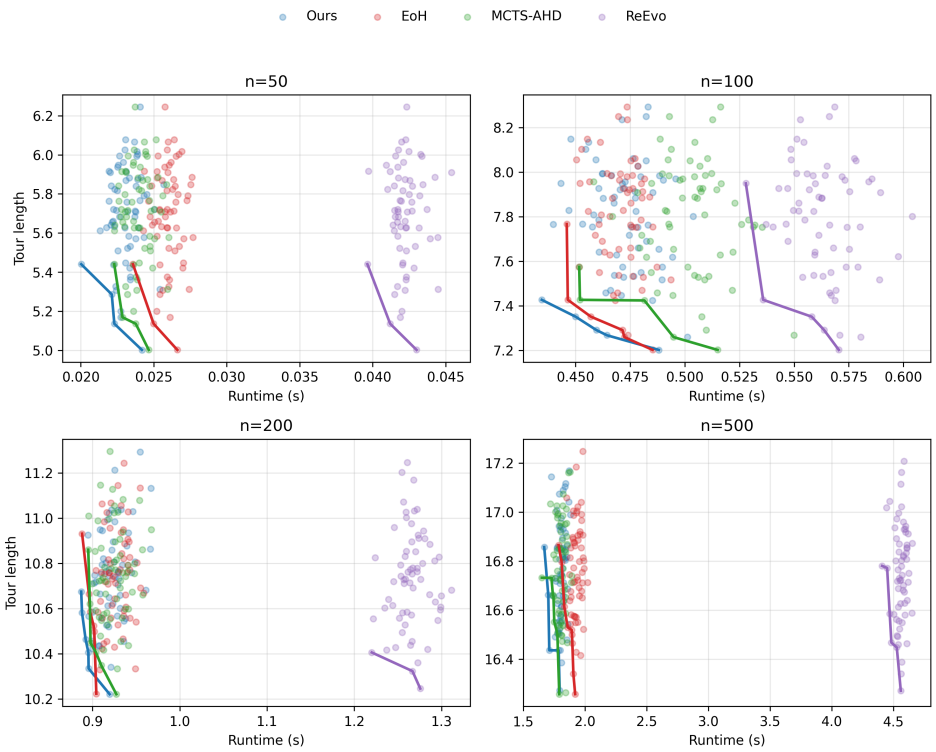


Figure 7: Pareto analysis of runtime and solution quality on TSP under the GLS framework. Each scatter point corresponds to one heuristic candidate evaluated during search, where the x-axis denotes runtime per instance and the y-axis denotes tour length. Results are shown for four problem sizes ($n = 50, 100, 200, 500$). The connected frontier for each method highlights the best observed quality–efficiency trade-off among its discovered heuristics. Lower tour length and runtime are preferred.

Table 10: Performance of heuristics on TSP benchmarks with guided local search framework. For a fair comparison, we report the result of the best performing heuristic over three independent runs following ReEvo (Ye et al., 2024). Gap (%) denotes the relative optimality gap to the best-known solution, and Time (s) denotes the average runtime per instance. * denotes results are drawn from ReEvo (Ye et al., 2024).

Method	TSP50		TSP100		TSP200		TSP500	
	Gap(%)↓	Time(s)	Gap(%)↓	Time(s)	Gap(%)↓	Time(s)	Gap(%)↓	Time(s)
NeuOpt*	0.000	1.32	0.027	2.67	0.403	4.81	–	–
GNNGLS*	0.052	3.83	0.705	6.78	3.522	9.92	–	–
EoH	**0.000**	0.03	0.390	0.47	**0.196**	0.90	1.667	1.93
ReEvo	**0.000**	0.04	0.025	0.55	0.338	1.26	1.703	4.56
MCTS-AHD	**0.000**	0.02	**0.000**	0.47	0.270	0.90	1.680	1.82
Ours	**0.000**	0.02	**0.000**	0.46	0.212	0.90	**1.649**	1.81

by different AHD frameworks on TSP under the GLS framework. Figure 7 visualizes the runtime and tour length of candidate heuristics discovered during search at four problem scales.

Across all scales, PRS shows a consistently favorable Pareto frontier. At smaller scales ($n = 50$ and $n = 100$), the heuristics discovered by PRS achieve lower tour lengths while remaining among the fastest candidates. At larger scales ($n = 200$ and $n = 500$), this advantage becomes more evident: PRS maintains a frontier that is competitive or better in solution quality while avoiding the large runtime overhead observed in some baselines, especially ReEvo. This indicates that PRS is able to discover heuristics with a stronger balance between effectiveness and computational efficiency, rather than improving quality merely through more expensive heuristic structures.

The results also provide additional evidence for the discussion in Appendix F.3. PRS tends to discover candidates that remain closer to the efficient part of the quality–runtime tradeoff frontier, especially on medium- and large-scale TSP instances.

F.5 Flow Shop Scheduling Problem under Guided Local Search Framework

We adopt the same guided local search (GLS) framework equipped with two neighborhood operators, namely *Swap* and *Relocate* as EoH. Within this framework, PRS is employed to automatically design a heuristic strategy that jointly governs the update of the execution-time matrix and the selection of jobs to be perturbed during the search process. The heuristics generated by PRS are compared against a diverse set of non-AHD methods, including both classical heuristics and learning-based approaches. Specifically, we consider the following competitors: (i) *GUPTA* (Gupta, 1971), a classical heuristic for the flow-shop scheduling problem (FSSP); (ii) *CDS* (Campbell et al., 1970), another well-established heuristic for FSSP; (iii) *NEH* (Nawaz et al., 1983), which is widely regarded as an effective heuristic for flow-shop scheduling; (iv) *NEHFF* (Fernandez-Viagas & Framinan, 2014), an enhanced variant of NEH; (v) *PFSPNet* (Pan et al., 2021), a recently proposed deep learning-based solver for flow-shop scheduling; (vi) *Local Search (LS)*, which applies the same neighborhood operators as PRS without heuristic adaptation; (vii) *ILS1* (Stützle, 1998), a classical iterated local search method for FSSP; and (viii) *ILS2*, which follows the same overall framework as our method but relies on manually designed heuristic rules. We take EoH as the LLM-based AHD baseline. All methods are evaluated on the widely used Taillard (Taillard, 1993) benchmark instances. The number of jobs ranges from 50 to 200, while the number of machines varies between 5 and 20. Each reported value is the average relative gap over 10 instances in the corresponding test set.

As shown in Table 11, PRS achieves lower gaps across Taillard instances than EoH in most test sets (e.g., 0.27% vs 0.62% on j100_m20), outperforming it on average while also demonstrating superior performance against non-AHD baselines, including NEH-based heuristics and iterated local search methods.

F.6 Results on TSPLib instances

We conduct experiments on the well-known real-world TSP benchmark TSPLib (Reinelt, 1991) to evaluate the effectiveness of the step-by-step constructive heuristics of PRS in generalizing to real-world TSP instances. We consider TSPLib instances with up to 500 cities following MCTS-AHD (Zheng et al., 2025) and report the optimality gap with respect to the best-known solutions.

We compare PRS against manually designed heuristics, including Greedy, as well as the GP-based AHD method GHPP (Duflo et al., 2019). For LLM-based AHD baselines, we include the best-performing heuristics of EoH and ReEvo. All LLM-designed heuristics are executed three times with different starting nodes and averaged for fair comparison. The results of the instances reported in GHPP within the problem scale $[0, 500]$ are reported in Table 12. As shown in Table 12, the heuristic generated by PRS achieves a lower overall optimality gap than manually designed heuristics and the

Table 11: Results on Taillard (Taillard, 1993) instance sets. The results of non-LLM-based methods are cited from EoH (Liu et al., 2024a). Each entry reports the average relative gap (%) to the known upper bound in Taillard (lower is better). The best result in each row is highlighted in **bold**. Additionally, among all the LLM-based AHD methods, the best value is shaded.

Test Set	GUPTA	CDS	NEH	NEHFF	PFSPNet	LS	ILS1	ILS2	EoH	Ours
j50_m5	12.23	6.98	0.57	0.40	0.51	0.32	0.15	0.32	**0.00**	**0.00**
j50_m10	20.11	12.72	3.47	3.62	3.48	3.33	1.47	0.29	0.21	**0.17**
j50_m20	22.78	15.03	5.48	5.10	5.05	4.67	2.13	**0.34**	0.72	0.83
j100_m5	5.98	5.10	0.39	0.31	0.31	0.28	0.20	0.38	-0.04	**-0.05**
j100_m10	15.03	9.36	2.07	1.88	1.72	1.38	0.77	0.34	0.22	**0.14**
j100_m20	21.00	13.55	3.58	3.73	3.56	3.51	2.27	0.43	0.62	**0.27**
j200_m10	11.59	7.22	0.98	0.70	0.82	0.87	0.74	0.54	0.18	**0.18**
j200_m20	18.09	11.89	2.90	2.52	2.49	2.53	2.26	0.59	0.65	**0.55**
Average	15.85	10.23	2.43	2.28	2.24	2.11	1.25	0.40	0.32	**0.26**

GP-based AHD baseline, and leads all methods in 8 of the 15 instances. Collectively, these results demonstrate the competitive performance of PRS for designing TSP-Constructive heuristics that generalize well to an out-of-distribution benchmark.

Table 12: Results of the GP-based method GHPP, and LLM-based AHD methods on designing step-by-step constructive heuristics for TSP on TSPLib instances. We report the optimality gap (%) of each instance. The results of GHPP-best, Christofides (Christofides, 2022), Greedy (Brecklinghaus & Hougardy, 2015), Nearest insertion and Nearest-greedy (Rosenkrantz et al., 1977) are taken from (Duflo et al., 2019). LLM-based AHD methods (EoH, ReEvo, and PRS) are each run three times with different starting nodes and averaged. The leading LLM-designed heuristic on each instance is shaded, and the overall best heuristic is shown in bold.

Instance	Christofides	Greedy	Nearest insertion	Nearest-greedy	GHPP-best	EoH	ReEvo	Ours
bier127	13.03	19.50	23.05	23.25	15.64	19.60	10.79	**8.31**
d493	**9.52**	16.68	20.39	24.00	15.58	17.02	13.43	13.15
ch130	10.09	28.40	19.21	25.66	14.77	12.47	**9.40**	12.24
eil51	15.18	13.03	16.14	31.96	10.20	18.51	6.47	15.20
kroA150	13.44	20.24	19.09	26.08	15.59	24.61	11.62	**9.86**
kroB100	**9.82**	16.59	21.53	26.26	14.06	13.46	12.20	10.56
kroC100	9.08	12.94	24.25	25.76	16.22	16.85	15.88	**8.16**
lin318	13.80	18.75	24.44	25.78	14.30	23.02	16.63	**10.95**
pr226	14.17	21.44	28.02	24.65	15.50	12.44	18.02	**10.00**
pr264	11.28	11.89	34.28	17.87	24.00	21.50	16.78	**10.97**
pr299	**11.23**	31.42	25.05	31.42	18.24	20.95	20.63	15.06
rat99	9.43	22.30	21.05	21.79	14.09	21.01	12.41	**9.18**
ts225	5.67	**5.38**	19.93	16.82	7.71	17.55	6.56	11.33
fl417	15.57	12.64	25.52	32.42	22.72	24.13	19.15	**10.59**
pr439	**11.16**	20.08	24.67	27.36	21.36	26.83	19.25	14.52

F.7 Statistical Significance

In this section, we provide a more rigorous statistical significance analysis on three representative scenarios: CVRP under the ACO framework, KP under the step-by-step constructive framework, and TSP with GLS at $N = 200$. We conduct 10 independent runs for CVRP-ACO and KP-Constructive, and 5 independent runs for TSP-GLS. For each setting, we report the best objective score for each run, together with their average and standard deviation. We then perform one-sided t-tests to compare PRS against the state-of-the-art baseline MCTS-AHD. The results in Table 13 indicate that the observed gains of PRS are statistically significant over MCTS-AHD across all three settings. Moreover, the standard deviations of PRS remain comparable to those of MCTS-AHD, suggesting that its improvements are not obtained at the cost of substantially increased instability.

Table 13: Statistical significance analysis of PRS over multiple independent runs. The reported p-values are from one-sided t-tests comparing PRS against MCTS-AHD. For TSP-GLS, the reported objective values denote gap (%).

Method	run1	run2	run3	run4	run5	run6	run7	run8	run9	run10	avg	std	p-value
CVRP-ACO (↓)													
MCTS-AHD	8.842	8.957	8.623	8.993	9.012	8.790	9.103	8.929	9.231	9.186	8.967	0.184	–
PRS	8.563	8.588	8.751	8.785	8.989	8.949	8.835	8.943	9.012	9.058	**8.847**	0.174	0.0331
KP-Constructive (↑)													
MCTS-AHD	40.249	40.255	40.261	40.257	40.258	40.252	40.258	40.254	40.260	40.252	40.256	0.004	–
PRS	40.262	40.258	40.263	40.258	40.266	40.259	40.262	40.258	40.263	40.261	**40.261**	0.003	0.001
TSP-GLS (↓)													
MCTS-AHD	0.185	0.216	0.238	0.208	0.198	–	–	–	–	–	0.209	0.020	–
PRS	0.171	0.193	0.189	0.153	0.211	–	–	–	–	–	**0.183**	0.022	0.0457

F.8 SEARCH STRATEGY VARIANTS

In this section, we examine whether the effectiveness of PRS is critically dependent on the use of BFS. To this end, we replace the BFS-based search in PRS with two alternative strategies, namely CoT-SC (Wang et al., 2022) and MCTS, while keeping the perturbation–rewrite operator unchanged. Table 14 shows that PRS-BFS achieves the best overall performance on both KP and CVRP. However, the margins are moderate, and both PRS-MCTS and PRS-CoT-SC remain competitive. This suggests that BFS is a strong default choice in our framework, likely because it provides a favorable balance between exploration coverage, candidate utilization, and search efficiency, but the effectiveness of PRS does not rely solely on BFS.

Table 14: Comparison of search strategy variants within the PRS framework. Results are reported as mean $\pm$ standard deviation over three independent runs. For KP, higher is better; for CVRP, lower is better. The maximum number of evaluated heuristics is fixed to $T = 1000$ for all variants.

Variations	**KP** ($n = 100$) $\uparrow$	**CVRP** ($n = 50$) $\downarrow$
PRS-BFS	**40.2618**$\pm$0.0145	**8.9583**$\pm$0.0726
PRS-CoT-SC	40.2532$\pm$0.0418	9.3159$\pm$0.0586
PRS-MCTS	40.2584$\pm$0.0175	9.0728$\pm$0.0960

F.9 EXPERIMENTS WITH DIFFERENT LLMS

Tables 15-16 further break down the performance of heuristics generated by different LLM backbones on OBPP and KP. We benchmark PRS on LLM models other than GPT, including a closed-source model Claude-3.7-sonnet and the open-source models, namely DeepSeek-v3 and Llama-3.1-70B-Instruct.

On OBPP, among closed-source models, GPT-5-nano consistently achieves the lowest average optimality gaps and smaller variance across both test settings, indicating a more stable heuristic generation compared to GPT-4o-mini and Claude-3.7-sonnet. Among open-source models, DeepSeek-v3 performs competitively and achieves the best result on the OOD larger-scale setting (5k_500), while Llama-3.1-70B-Instruct exhibits slightly larger gaps. Overall, closed-source models tend to produce more stable heuristics on in-the-distribution instances, whereas strong open-source models can still achieve competitive performance on the OOD setting.

Table 15: Benchmarking PRS on different LLMs on OBPP (average gap (%) to the lower bound). Best result among all methods of each problem size is highlighted in bold while the best result among all LLM-based results of each scale is shaded.

OBPP				
N=	1k_100		5k_500	
Method	Gap (%) ↓	std (%)	Gap (%) ↓	std (%)
Best Fit	4.77%	–	0.55%	–
First Fit	5.02%	–	0.55%	–
Closed Source LLMs				
GPT-4.1-nano	2.58%	0.723	0.32%	0.125
GPT-5-nano	**1.82%**	**0.123**	0.35%	0.117
Claude-3.7-sonnet	2.25%	0.489	0.39%	0.178
Open Source LLMs				
DeepSeek-v3	3.40%	0.456	**0.23%**	**0.164**
Llama-3.1-70B-Instruct	4.03%	0.361	0.40%	0.124

On KP, all evaluated LLMs are capable of generating high-performing heuristics, which substantially outperform the constructive neural baseline POMO on both $N = 100$ and $N = 200$. In particular, GPT-5-nano and Claude-3.7-sonnet consistently achieve the smallest optimality gaps across both problem sizes, while open-source models such as DeepSeek-v3 and Llama-3.1-70B-Instruct remain slightly behind. These results suggest that the proposed PRS is robust to the choice of LLM backbones, while stronger models primarily contribute to improved stability and marginal gains in solution quality.

Table 16: Benchmarking PRS on different LLMs on KP (average objective and average gap (%) to optimal). Best result among all methods of each problem size is highlighted in bold while the best result among all LLM-based results of each scale is shaded.

KP				
N=	N=100		N=200	
Method	Obj. ↑	Gap (%) ↓	Obj. ↑	Gap (%) ↓
Optimal	40.271	–	57.448	–
POMO	39.676	1.48%	57.271	0.31%
Closed Source LLMs				
GPT-4.1-nano	40.234	0.09%	57.405	0.07%
GPT-5-nano	**40.262**	**0.02%**	**57.436**	**0.01%**
Claude-3.7-sonnet	**40.263**	**0.02%**	57.435	0.02%
Open Source LLMs				
DeepSeek-v3	40.232	0.10%	57.398	0.09%
Llama-3.1-70B-Instruct	40.233	0.09%	57.400	0.08%

G COST ANALYSIS

This section reports the computational cost of different AHD frameworks. Specifically, for each framework, we measure (i) total wall-clock time, (ii) input and output token consumption, and (iii) the number of LLM queries. These metrics complement solution quality by characterizing the practical efficiency and cost profile of different frameworks under the same evaluation budget.

Table 17: Cost comparison across different AHD frameworks.

Method	Metric	OBPP-Constructive	KP-Constructive	CVRP-ACO	OP-ACO
ReEvo	Time	8h	-	4h	5h
	Input Tokens	2.94M	-	3.30M	2.82M
	Output Tokens	1.02M	-	1.65M	1.68M
	#LLM Queries	2869	-	1433	1410
EoH	Time	10h	9h	6h	6h
	Input Tokens	2.40M	1.80M	1.83M	2.94M
	Output Tokens	0.90M	0.75M	1.44M	1.26M
	#LLM Queries	2082	1054	1045	1063
MCTS-AHD	Time	26h	19h	12h	10h
	Input Tokens	3.09M	2.10M	3.06M	3.36M
	Output Tokens	1.05M	1.11M	2.16M	2.10M
	#LLM Queries	4012	2010	2004	2012
PRS	Time	10h	5h	6h	5h
	Input Tokens	5.00M	3.15M	3.09M	3.12M
	Output Tokens	2.86M	1.87M	1.55M	1.60M
	#LLM Queries	2706	1353	1353	1353

LLM queries and token consumption. PRS consistently uses fewer LLM queries than MCTS-AHD across all four tasks, requiring 2706 versus 4012 queries on OBPP, and 1353 versus 2010, 2004, and 2012 queries on KP, CVRP, and OP, respectively. This reduction is approximately 32-33% across the evaluated settings. Compared with ReEvo and EoH, however, PRS does not uniformly use fewer queries: its query cost is lower than ReEvo on the tasks where both methods are reported, but can exceed that of EoH.

Token consumption follows a different pattern. PRS incurs higher token usage than the lightweight baselines in several settings, reflecting its idea-generation and idea-guided rewrite stages. Nevertheless, its token usage is lower than that of MCTS-AHD on CVRP and OP, while its query count remains substantially lower than MCTS-AHD across all tasks.

Runtime Cost and efficiency. In terms of wall-clock time, PRS is comparable to lightweight evolutionary methods and significantly faster than search-heavy approaches such as MCTS-AHD across evaluated settings, as shown in Table 17. The total computational cost is primarily governed by the maximum number of evaluations T and the per-heuristic evaluation time. While T is consistent for all LLM-based AHD methods, the per-heuristic evaluation time of PRS remains under control, leading to a competitive total runtime among LLM-based AHD methods. This shows that PRS stands as a practical, cost-efficient AHD framework with strong exploration capability.

Runtime factors. The overall runtime is primarily dominated by heuristic evaluation rather than LLM inference, and thus scales with the complexity of the heuristics rather than the branching search structure. Heuristics with expensive operations, nested loops, or inefficient data structures can substantially increase per-evaluation runtime, even under the same evaluation budget. As discussed in Appendix F.3, PRS tends to discover heuristics with favorable runtime profiles, particularly on larger problem scales, as evidenced by the Pareto analysis of solution quality and runtime in Fig. 7.

Search granularity. PRS allocates its LLM budget through bounded BFS expansion. For each selected parent, one idea operator produces a target idea, which is then realized through multiple code rewrites with a fixed beam size. This organization concentrates several candidate realizations around a single idea-level proposal, rather than treating every LLM call as an independent search iteration. As shown in Table 17, this design yields substantially fewer total LLM queries than MCTS-AHD, while retaining a competitive end-to-end runtime.

H SENSITIVITY ANALYSIS

We analyze the sensitivity of PRS to its main search-control hyperparameters on KP and OBPP under the step-by-step constructive framework. BFS beam size κ and tree depth d jointly control the local expansion breadth and depth; the stagnation and reheating ratio ($C/T = \Delta/T$) determines the interval for triggering stagnation-aware restart and the subsequent operator reheating horizon; the softmax temperature T_{sel} governs the selection pressure on the frontier; and the UCB window size $|\mathcal{H}|$ controls the amount of recent operator feedback used for adaptive operator selection. For each setting, we report the average optimality gap and token cost in Table 18.

Table 18: Sensitivity analysis of PRS on KP and OBPP under the step-by-step constructive framework. Lower gap is better.

Modified Parameter	Value	KP-Constructive		OBPP-Constructive	
		Gap	# Token (M)	Gap	# Token (M)
–	$(\kappa = d = 3),\ \Delta/T = 0.1$ $T_{\text{sel}} = 1.0,\ \lvert\mathcal{H}\rvert = 50$	0.024%	4.25	0.802%	6.07
beam/depth	$(\kappa = d = 4)$	0.026%	3.89	0.835%	5.72
beam/depth	$(\kappa = d = 2)$	0.031%	4.98	0.913%	6.35
Stagnation ratio Δ/T	0.05	0.027%	4.13	0.894%	6.18
Stagnation ratio Δ/T	0.20	0.023%	4.40	0.836%	6.36
Softmax temperature T_{sel}	0.5	0.021%	3.91	0.884%	5.86
Softmax temperature T_{sel}	2.0	0.027%	4.06	0.821%	5.94
UCB window $\lvert\mathcal{H}\rvert$	100	0.025%	4.29	0.793%	6.02

Beam/depth. The beam/depth setting affects the local expansion breadth and depth of the search. Following the BFS convention in ToT (Yao et al., 2023), we adopt a bounded tree search and set $\kappa = d = 3$ as the default for the BFS-based heuristic search. In Table 18, increasing (κ, d) from 3 to 4 yields comparable performance with reduced token cost, suggesting that the default setting already provides sufficient search breadth and that further expansion does not necessarily improve exploration quality within the same evaluation budget T. By contrast, reducing them to 2 leads to a moderate degradation in solution quality together with a noticeable increase in token cost. This indicates that overly small search breadth or depth may hurt search efficiency, while the default setting $(\kappa = d = 3)$ provides a good balance between exploration and cost.

Stagnation ratio. The stagnation ratio controls both the evaluation interval for triggering stagnation-aware restart and the subsequent operator reheating horizon. Changing Δ/T from 0.1 to 0.05 or 0.2 yields only minor changes in both performance and computational cost. This suggests that the adaptive search-control mechanism is robust across a reasonable range of stagnation settings.

Softmax temperature. Varying the selection temperature leads to only limited performance variation across the two tasks. Although different temperatures can achieve slightly better results on individual settings, both $T_{\text{sel}} = 0.5$ and $T_{\text{sel}} = 2.0$ remain competitive with the default $T_{\text{sel}} = 1.0$. Meanwhile, token usage remains relatively stable across these settings, showing that the stochastic selection mechanism does not introduce large cost fluctuations. We therefore retain $T_{\text{sel}} = 1.0$ as a moderate default to balance exploration and exploitation within each frontier in BFS.

UCB window size. Increasing the UCB reward window from $|\mathcal{H}| = 50$ to 100 results in only minor changes in both performance and token cost on KP and OBPP. This indicates that adaptive operator selection is not highly sensitive to the exact amount of recent operator feedback retained in the window. Overall, the sensitivity analysis confirms that PRS is robust to moderate variations in its main search-control hyperparameters.

I EXAMPLE RESPONSE FOR MULTI-DIRECTIONAL PERTURBATIONS

This section presents representative examples showing how parent heuristics are perturbed and rewritten through complementary directional transformations in PRS, as shown in Fig. 8 and 9. For OBPP, lower scores are preferred; for OP, higher scores are better.

Parent Heuristic of OP-ACO (Score: $g = 13.560$)

```
def heuristics(prize, distance, maxlen):
    import numpy as np
    n = len(prize)
    epsilon = 1e-8
    alpha = 1.0
    beta = 2.0
    heuristic_matrix = np.zeros((n, n))

    for i in range(n):
        for j in range(n):
            if i == j or distance[i][j] == 0:
                heuristic_matrix[i][j] = 0.0
            else:
                base_ratio = prize[j] / (distance[i][j] + epsilon)
                scale_factor = 1 + alpha * (
                    maxlen / (distance[i][j] + epsilon))**beta
                heuristic_matrix[i][j] = base_ratio * scale_factor

    row_sums = np.sum(heuristic_matrix, axis=1, keepdims=True)
    row_sums[row_sums == 0] = 1.0
    heuristic_matrix = heuristic_matrix / row_sums
    return heuristic_matrix
```

↓ *Expansion: new reward-aware structures injected*

Offspring Heuristic of OP-ACO (Score: $g = 14.194$)

```
def heuristics(prize, distance, maxlen):
    import numpy as np
    n, epsilon = len(prize), 1e-8
    gamma = delta = 1.0
    median_dist = np.median(distance)

    local_density = np.zeros(n)
    for j in range(n):
        neighbors = [k for k in range(n) if distance[j][k] <= median_dist and j != k]
        local_density[j] = np.mean(prize[neighbors]) if neighbors else prize[j]

    max_density = max(np.max(local_density), 1.0)
    local_density /= max_density

    heuristic_matrix = np.zeros((n, n))
    top_prize_sum = np.sum(prize[np.argsort(-prize)[:3]])
    total_prize = np.sum(prize)
    LAF = 1 + delta * (top_prize_sum / total_prize)

    for i in range(n):
        for j in range(n):
            if i == j or distance[i][j] == 0:
                continue
            ratio = prize[j] / (distance[i][j] + epsilon)
            scale_factor = 1 + (maxlen / (distance[i][j] + epsilon))**2
            density_weight = 1 + gamma * local_density[j]
            heuristic_matrix[i][j] = ratio * scale_factor * density_weight * LAF

    row_sums = np.sum(heuristic_matrix, axis=1, keepdims=True)
    row_sums[row_sums == 0] = 1.0
    heuristic_matrix /= row_sums
    heuristic_matrix[heuristic_matrix < 0] = 0.0
    heuristic_matrix[~np.isfinite(heuristic_matrix)] = 0.0
    return heuristic_matrix
```

Figure 8: Example of an expansion operation on heuristics of OP under the ACO framework. The parent heuristic (top) primarily scores transitions using prize-to-distance preference and a route-length scaling factor. After expansion and LLM-based rewriting, the offspring (bottom) retains this backbone while introducing additional reward-aware structures, including local prize density and global high prize information. These additions enrich the heuristic's decision logic and improve the validation score from $g = 13.560$ to $g = 14.194$ ($+4.68\%$).

Parent Heuristic of OBPP-Constructive (Score: $g = 2.170$)

```
def priority(item, bins_remain_cap):
    import numpy as np
    cap = np.array(bins_remain_cap, copy=True)
    epsilon = 1e-8
    max_cap = np.max(cap)
    variance = np.var(cap) / (np.sum(cap) + epsilon)
    avg_residual = np.mean(cap / (max_cap + epsilon))

    if avg_residual > 0.75:
        threshold_factor = 1.5
    elif avg_residual < 0.3:
        threshold_factor = 3.0
    else:
        threshold_factor = 2.0

    diff = cap - item
    fit_threshold = threshold_factor * item
    near_fit_mask = diff <= fit_threshold
    diff_range = (np.max(diff) - np.min(diff)) / (item + epsilon)
    disparity_factor = 1 / (diff_range + epsilon)

    neighbor_penalty = np.ones_like(cap)
    for i in range(len(cap) - 1):
        gap = np.abs(cap[i] - cap[i + 1])
        if gap > 0.1 * item:
            penalty = 1 - gap / (max_cap + epsilon)
            neighbor_penalty[i] *= penalty
            neighbor_penalty[i + 1] *= penalty

    bonus = np.zeros_like(cap)
    if np.sum(diff <= 1e-3) < 2:
        bonus[diff <= 1.2 * item] = 5.0
    else:
        bonus[near_fit_mask] = 1.0

    scores = (1 / (diff + epsilon) + bonus) * disparity_factor
    scores *= (1 - variance) * neighbor_penalty
    return scores
```

↓ *Compression: simplify adaptive fit and global weighting*

Offspring Heuristic of OBPP-Constructive (Score: $g = 1.846$)

```
def priority(item, bins_remain_cap):
    import numpy as np
    cap = np.array(bins_remain_cap, copy=True)
    epsilon = 1e-8
    max_cap = np.max(cap)
    avg_residual = np.mean(cap / (max_cap + epsilon))

    if avg_residual > 0.75:
        tight_weight = 2.0
    elif avg_residual < 0.3:
        tight_weight = 0.5
    else:
        tight_weight = 1.0

    diff = cap - item
    near_fit_mask = diff <= 1.5 * item
    bonus = np.zeros_like(cap)
    bonus[diff <= (1.2 * item if np.sum(near_fit_mask) < 2 else 1.5 * item)] = (
        3.0 if np.sum(near_fit_mask) < 2 else 1.0
    )

    disparity_factor = 1 / (
        (np.max(diff) - np.min(diff)) / (item + epsilon) + epsilon
    )
    neighbor_penalty = np.ones_like(cap)
    for i in range(len(cap) - 1):
        gap = np.abs(cap[i] - cap[i + 1])
        if gap > 0.1 * item:
            penalty = 1 - gap / (max_cap + epsilon)
            neighbor_penalty[i:i + 2] *= penalty

    scores = (1 / (diff + epsilon) + bonus) * disparity_factor
    scores *= tight_weight * neighbor_penalty
    return scores
```

Figure 9: Example of a compression operation on heuristics of OBPP under the step-by-step constructive framework. The parent heuristic (top) combines adaptive fit thresholds, variance-based global weighting, disparity control, and neighborhood smoothness. After compression and LLM-based rewriting, the offspring (bottom) retains the core fit, disparity, and smoothness mechanisms while simplifying the adaptive threshold and global weighting logic. The resulting heuristic improves the validation score from $g = 2.170$ to $g = 1.846$.

J LIMITATIONS

A current limitation of PRS is that its adaptive exploration controller still relies on predefined scheduling rules and stagnation thresholds. Although the UCB-based operator selection adapts to recent search outcomes, the overall exploration schedule and restart behavior are manually specified, which may limit the flexibility of search control across different tasks and search stages.

Another limitation is the cost of heuristic evaluation. PRS currently realizes each perturbed heuristic idea into executable code before evaluating its performance, even though not every proposed idea is necessarily worth implementation and expensive downstream evaluation. This issue can become particularly important when heuristic evaluation involves large problem instances or computationally costly optimization environments.

In future work, we aim to address these limitations by (1) developing more automatic and LLM-driven control mechanisms, where operator selection, exploration scheduling, and stagnation response are determined autonomously by the LLM agent rather than through predefined rules, and (2) introducing surrogate models, such as learned world models, to estimate the potential utility of perturbed heuristic ideas before realization. Such models could prioritize promising ideas and filter less useful ones, reducing unnecessary implementation and evaluation costs and improving the overall efficiency of heuristic search.

K BASELINES & LICENSES

Optimal Solutions. For the results of TSP in Table 9 and Table 10, we obtain reference solutions using the Lin–Kernighan heuristic (Lin & Kernighan, 1973) implemented in the `elkai` solver, which is a widely used Python implementation of LK-based TSP solvers. For problem sizes up to $N = 315$, the solver returns proven optimal solutions, while for larger instances it provides high-quality approximate solutions. We report optimality gaps with respect to these reference solutions, following common practice in prior TSP literature. For KP instances, we use Google OR-Tools for the optimal value.

Other Baselines Implemented. We implement POMO (Kwon et al., 2020) and DeepACO (Ye et al., 2023) as representative baselines. To ensure a fair comparison, data augmentation in POMO is disabled in all experiments. For KGLS, we adopt the ACO implementation provided in DeepACO and set the heuristic matrix to the distance matrix. The licenses and URLs of baselines, models, and software are summarized in Table 19.

Table 19: External codebases and datasets used in this work.

Resource	License / Availability	URL
LKH3	Academic research use	`http://webhotel4.ruc.dk/~keld/research/LKH-3/`
OR-Tools	MIT License	`https://developers.google.com/optimization/pack/knapsack`
POMO	Available online	`https://github.com/yd-kwon/POMO/tree/master`
DeepACO	MIT License	`https://github.com/henry-yeh/DeepACO`
FunSearch	Apache License	`https://github.com/google-deepmind/funsearch`
EoH	MIT License	`https://github.com/FeiLiu36/EoH/tree/main`
ReEvo	MIT License	`https://github.com/ai4co/reevo`
HSEvo	Available online	`https://github.com/datphamvn/HSEvo`
MCTS-AHD	MIT License	`https://github.com/zz1358m/MCTS-AHD-master`
TSPLIB	Non-commercial use	`http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95`