ReVEL: Multi-Turn Reflective LLM-Guided Heuristic Evolution via Structured Performance Feedback

Anonymous ACL submission

Abstract

Designing effective heuristics for NP-hard combinatorial optimization problems remains challenging and often requires substantial domain expertise. Recent LLM-guided evolutionary methods have shown promise for automated heuristic generation, but most existing approaches refine heuristics independently or through limited pairwise feedback. We propose **ReVEL**: Multi-Turn **Re**flecti**v**e LLM-Guided Heuristic **E**vo**l**ution via Structured Performance Feedback, a framework for group-wise multi-turn heuristic refinement. ReVEL organizes heuristics into behavior-aware reflective groups, including similarity-driven groups for localized refinement and diversity-driven groups for exploratory search. Within each group, the LLM performs iterative multi-turn refinement using accumulated performance feedback, enabling related heuristics to be jointly analyzed and progressively improved across evolutionary iterations. Experiments on standard combinatorial optimization benchmarks show that ReVEL generally improves optimization performance over existing LLM-guided evolutionary baselines across multiple settings and LLM backbones. Additional analyses suggest that behavior-aware grouping contributes to more consistent refinement trajectories during iterative heuristic evolution.

1 Introduction

Large language models (LLMs) have recently demonstrated strong capabilities in program synthesis, iterative refinement, and feedback-driven code generation (Liao et al., 2026; Gozeten et al., 2026). These advances have motivated growing interest in using LLMs for combinatorial optimization, where heuristics can be progressively improved through repeated interaction with optimization feedback.

Combinatorial optimization problems (COPs), such as the Traveling Salesman Problem (TSP) (Bock et al., 2025) and Bin Packing Problem (BPP) (Aïder et al., 2025), require searching exponentially large discrete spaces in which high-quality solutions typically emerge through sustained iterative refinement.

First, **classical heuristic and meta-heuristic methods** (Almufti and Shaban, 2025; Rondano, 2025) rely heavily on manually designed search operators tailored to problem-specific structures. While effective, these methods depend strongly on expert knowledge and fixed optimization strategies.

Second, **hyper-heuristic methods** (Xu et al., 2025; Vela et al., 2025) attempt to automate heuristic selection and composition. However, optimization remains constrained by predefined heuristic components, limiting the discovery of fundamentally new search behaviors.

Third, recent **LLM-guided evolutionary frameworks** such as EoH (Liu et al., 2024) and ReEvo (Ye et al., 2024) demonstrate that LLMs can directly generate and iteratively refine optimization heuristics using evolutionary feedback. Despite promising results, existing methods primarily rely on reflection over isolated candidates or pairwise comparisons. Consequently, optimization feedback remains fragmented across generations, limiting the model's ability to identify broader behavioral patterns within the evolving heuristic population.

These limitations suggest that the effectiveness of reflective heuristic optimization depends critically on how optimization feedback is organized and exposed to the model during refinement.

Our approach: We propose ReVEL, a reflective evolutionary framework that iteratively refines behavior-aware heuristic groups. Rather than refining heuristics independently, ReVEL organizes heuristics according to shared optimization behavior, enabling the LLM to analyze recurring strengths, common failure modes, and complementary search patterns within each group.

Based on these grouped reflective contexts, ReVEL performs iterative refinement that balances

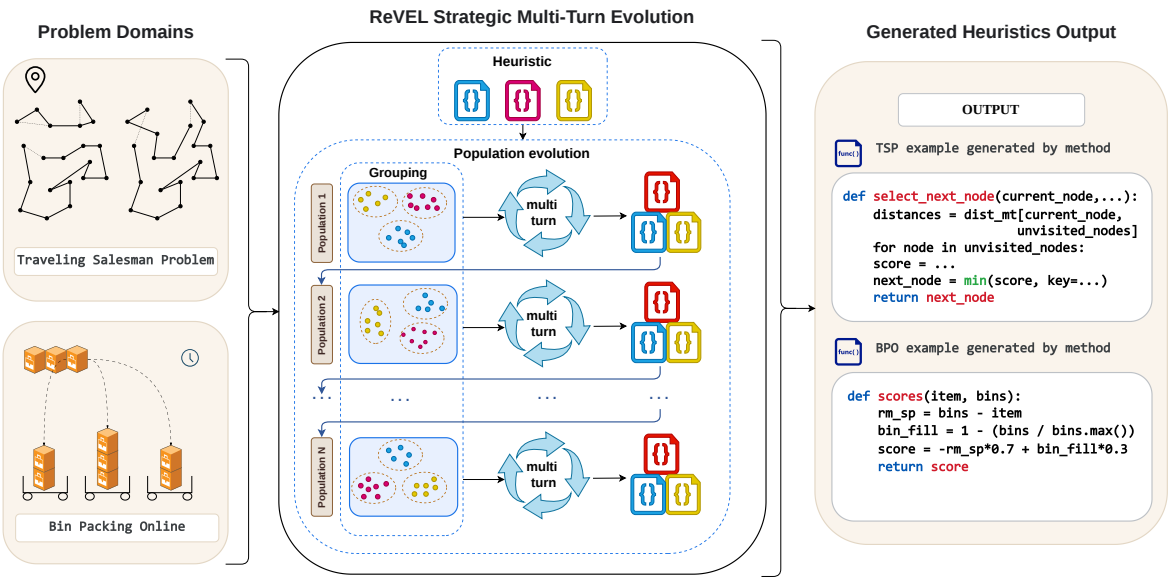

Figure 1: The proposed **REVEL** framework addresses combinatorial optimization problems across diverse domains and problem scales through strategic multi-turn heuristic evolution. Starting from initial candidate search strategies, REVEL organizes heuristic populations into collaborative refinement groups that iteratively perform evaluation, strategic rewriting, and evolutionary adaptation. Through repeated multi-turn interactions, the framework progressively explores and refines increasingly effective search behaviors, enabling the automatic discovery of high-performing heuristics for complex combinatorial optimization problems.

exploratory heuristic modification and exploitative local improvement throughout optimization. This grouped-refinement process produces more coherent, reflective feedback, leading to more stable optimization trajectories and improved long-horizon search performance.

Extensive experiments on TSP and online BPP demonstrate that REVEL consistently outperforms strong LLM-guided evolutionary baselines across multiple optimization settings and LLM backbones. Further analysis shows that behavior-aware grouping substantially improves refinement consistency and optimization stability across search iterations.

Contributions. Our contributions are summarized as follows:

- **Reflective evolutionary optimization.** We introduce REVEL, a reflective evolutionary framework that reformulates LLM-guided heuristic optimization as iterative refinement over grouped heuristic populations.
- **Behavior-aware reflective grouping.** We propose a grouping mechanism that organizes heuristics according to shared optimization behavior, enabling more coherent reflective feedback during heuristic refinement.
- **Iterative grouped refinement.** We develop a multi-turn refinement strategy that balances exploratory heuristic modification and exploitative local improvement through grouped reflective optimization.
- **Comprehensive empirical analysis.** We demonstrate across multiple combinatorial optimization benchmarks that structured reflective grouping substantially improves optimization quality, refinement consistency, and robustness across different LLM backbones.

2 Related work

Automatic heuristic design has been studied through genetic programming and neural combinatorial optimization, which learn heuristics via evolutionary operators or fixed policies over problem instances (Burke et al., 2013; Branke et al., 2015; Kool et al., 2018). More recently, large language models have been used to generate and refine programmatic heuristics within evolutionary frameworks (Romera-Paredes et al., 2023; Liu et al., 2024; van Stein and Bäck, 2025), while multi-turn reflection methods enable iterative self-improvement through feedback-driven updates (Madaan et al., 2023; Shinn et al., 2023).

However, these approaches primarily operate on isolated candidates or pairwise feedback, limiting their ability to model dependencies across search decisions. In contrast, REVEL treats heuristic optimization as a regime-conditioned LLM trajectory process, where search decisions are made over evolving generation history and consolidated post-hoc into high-quality solutions. More related work is presented in Appendix B.

3 Methodology

3.1 Overview

Figure 2 illustrates the architecture of REVEL, a reflective evolutionary framework that integrates LLMs into the optimization loop as population-level reflective refinement framework. Unlike existing LLM-guided evolutionary frameworks that refine heuristics through isolated reflection, REVEL maintains persistent reflective reasoning across optimization iterations as populations evolve.

At a high level, REVEL formulates heuristic evolution as an iterative refinement process guided by accumulated population-level feedback. Starting from an initial heuristic population, the framework repeatedly evaluates collective search behavior, organizes heuristics into behaviorally coherent groups, and performs reflective refinement based on population-level optimization trends rather than isolated candidate performance.

Central to REVEL is a grouped reflective refinement mechanism that balances behavioral coherence with search diversity. By analyzing grouped optimization patterns, the framework reveals recurring strengths, shared failures, and complementary search behaviors that are difficult to infer from isolated heuristic comparisons. Guided by these reflective signals, the LLM iteratively refines existing heuristics or synthesizes new strategies through optimization feedback. Reflective feedback accumulates across generations, enabling progressively more informed refinement. Through this interaction between grouped reflective refinement and iterative optimization feedback, REVEL reformulates heuristic evolution from independent candidate generation into group-aware refinement.

3.2 Behavior-Aware Heuristic Grouping

Representation. Each heuristic h is evaluated on a benchmark set $\mathcal{I} = \{1, \ldots, m\}$ with objective values $e_i(h)$ on instance i. To characterize optimization behavior across instances, we represent each heuristic using a normalized performance-profile vector:

$$\mathbf{z}(h) = \frac{\mathbf{e}(h) - \mathbf{e}^*}{\mathbf{e}^*}, \tag{1}$$

where $\mathbf{e}(h) = \big(e_1(h), \ldots, e_m(h)\big)^\top$ denotes the performance vector of heuristic h, and $\mathbf{e}^* = \big(e_1^*, \ldots, e_m^*\big)^\top$ contains the best-known objective values within the current heuristic population:

$$e_i^* = \min_{h' \in \mathcal{H}} e_i(h'), \qquad i = 1, \ldots, m. \tag{2}$$

This representation captures the relative behavioral profile of each heuristic and provides a normalized space for population-level comparison.

Structured reflective grouping. The effectiveness of reflective optimization depends critically on how heuristic populations are exposed to the LLM. Reflection over highly similar heuristics enables localized diagnosis of recurring optimization behaviors, while reflection over structurally distinct heuristics broadens the search space explored during refinement. Motivated by this observation, REVEL constructs two complementary reflective contexts: homogeneous groups that preserve behavioral coherence, and heterogeneous groups that promote exploratory diversity.

Homogeneous groups (similarity-driven)

To construct behaviorally coherent groups, we measure similarity between heuristics using both optimization behavior and program structure. Behavioral similarity is computed in the normalized performance space using cosine similarity:

$$\mathrm{sim}_{\mathrm{perf}}(h_i, h_j) = \frac{\mathbf{z}(h_i)^\top \mathbf{z}(h_j)}{\|\mathbf{z}(h_i)\|_2 \|\mathbf{z}(h_j)\|_2}. \tag{3}$$

$$\mathrm{sim}_{\mathrm{code}}(h_i, h_j) = \mathrm{CodeBLEU}(h_i, h_j). \tag{4}$$

(3) captures whether heuristics exhibit similar optimization dynamics across benchmark instances. (4) measures structural similarity between heuristics using CodeBLEU, which incorporates lexical, syntactic, semantic, and data-flow information. The resulting similarity score is defined as:

$$\mathrm{sim}(h_i, h_j) = \alpha\, \mathrm{sim}_{\mathrm{perf}}(h_i, h_j) + \beta\, \mathrm{sim}_{\mathrm{code}}(h_i, h_j), \tag{5}$$

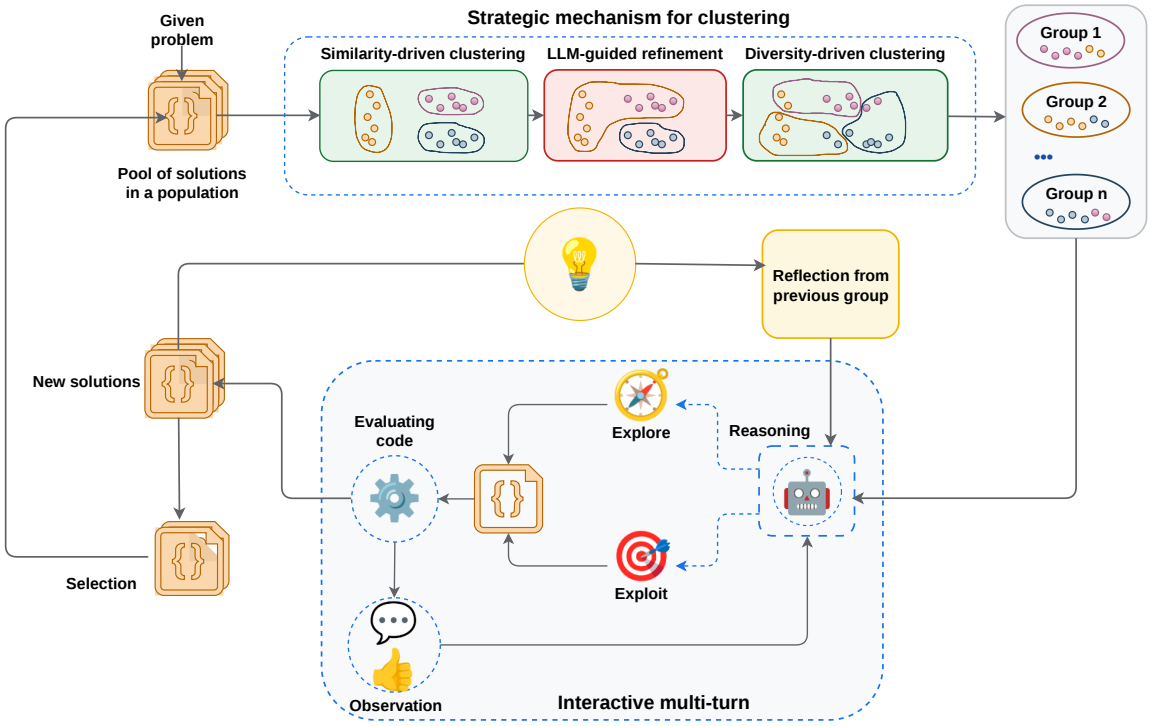

Figure 2: **REVEL: Multi-turn reflective evolutionary optimization through structured population feedback.** Starting from an initial heuristic population, REVEL iteratively organizes heuristics into behaviorally coherent groups, performs reflective refinement over accumulated population feedback, and evolves improved heuristics through population-level reasoning.

where $\alpha, \beta \geq 0$ balance behavioral and structural similarity.

Using these scores, we represent the heuristic population as a weighted similarity graph $G = (H, E, W)$, $W_{ij} = \mathrm{sim}(h_i, h_j)$, where H denotes the heuristic set and W stores pairwise similarities. The similarity matrix is normalized and transformed into a dissimilarity matrix:

$$\tilde{W} = \frac{W}{\max_{p,q} W_{pq}}, \qquad d_{ij} = 1 - \tilde{W}_{ij}. \quad (6)$$

Because reflective quality is highly sensitive to the granularity of grouping, REVEL adopts a two-stage clustering strategy. First, agglomerative clustering with linkage function L produces an intentionally fine-grained over-partition:

$$(C_1, \ldots, C_m) = \mathrm{Agglomerative}_L(D; m), \quad (7)$$

where $m \gg 1$ is selected such that each cluster remains locally coherent:

$$\mathrm{diam}(C_\ell) = \max_{x,y \in C_\ell} d(x, y) \leq \delta, \qquad \delta \ll 1. \quad (8)$$

This over-partition preserves localized behavioral structure while avoiding premature semantic merging. The resulting partition $\mathcal{C}_0 = \{C_1, \ldots, C_m\}$ is subsequently refined by the LLM:

$$\mathcal{C}^{\mathrm{ref}} = \Phi_{\mathrm{LLM}}(\mathcal{C}_0, W), \quad (9)$$

subject to

$$\bigcup_{C \in \mathcal{C}^{\mathrm{ref}}} C = H, \qquad C_p \cap C_q = \emptyset \quad (p \neq q). \quad (10)$$

This refinement stage allows the LLM to globally reorganize clusters according to higher-level semantic and behavioral relationships that static similarity metrics alone may fail to capture.

Heterogeneous groups (diversity-driven)

While homogeneous groups support precise reflective analysis, excessive similarity can narrow exploration and encourage convergence toward locally dominant strategies. To broaden the reflective search space, REVEL additionally constructs heterogeneous groups by combining heuristics from distinct homogeneous clusters.

For a homogeneous cluster $G = \{h_1, \ldots, h_m\}$, We estimate internal diversity using the entropy of pairwise-similarity distributions. Defining normalized affinities:

$$p_{ij} = \frac{\mathrm{sim}(h_i, h_j)}{\sum_{u<v} \mathrm{sim}(h_u, h_v)}, \qquad i < j, \quad (11)$$

The cluster entropy is computed as:

$$\mathcal{H}(G) = -\sum_{i<j} p_{ij} \log p_{ij}. \tag{12}$$

Higher entropy indicates greater structural and behavioral diversity within the cluster. Given homogeneous clusters $\{G_1, \ldots, G_k\}$, Sampling weights are assigned proportionally to entropy:

$$w_i = \frac{\mathcal{H}(G_i)}{\sum_{j=1}^{k} \mathcal{H}(G_j)}. \tag{13}$$

For a target heterogeneous group size L, each cluster contributes $L_i = \lfloor w_i L \rfloor$ heuristics sampled uniformly from G_i. The final heterogeneous group is defined as:

$$G^{\text{het}} = \bigcup_{i=1}^{k} S_i, \qquad S_i \subseteq G_i, \quad |S_i| = L_i. \tag{14}$$

By exposing the LLM to structurally diverse optimization behaviors, heterogeneous grouping expands the reflective context available during refinement and encourages exploration beyond locally coherent heuristic families.

3.3 Multi-turn Reflective Refinement

Given the groups constructed in Section 3.2, REVEL performs iterative multi-turn refinement over grouped heuristics. Refinement operates locally within each group, while selection is performed globally across the population, enabling localized heuristic adaptation without sacrificing population-level diversity.

group-aware multi-turn refinement. At iteration t, each reflective group G_k contains heuristics with related optimization behavior. The LLM receives the grouped heuristics together with structured optimization feedback, including relative performance, recurring failure patterns, prior refinement history, and summarized search observations.

Conditioned on the accumulated search memory M_t, the LLM performs iterative refinement within each group across multiple turns:

$$\tilde{\mathcal{H}}_k^{(r+1)} \sim \mathcal{G}\left(G_k, \tilde{\mathcal{H}}_k^{(r)}, M_t\right), \tag{15}$$

where r denotes the refinement turn and $\mathcal{G}$ denotes the LLM-guided refinement operator.

Unlike isolated single-step refinement, the multi-turn process allows the model to revise heuristics using accumulated group-level feedback across refinement turns.

Exploration and exploitation. To balance structural exploration and localized heuristic improvement, REVEL performs adaptive exploration–exploitation refinement conditioned on multi-turn optimization trajectories. Rather than relying on predefined switching rules, refinement behavior evolves according to population-level search signals accumulated across successive refinement turns, including performance progression, refinement history, structural diversity, and recurring failure patterns across heuristic groups.

When optimization trajectories indicate stagnation or structurally redundant search behavior, REVEL shifts toward *exploration*, encouraging broader structural rewrites and diversification of heuristic families. Conversely, when trajectories exhibit stable improvement within high-performing groups, refinement increasingly shifts toward *exploitation*, emphasizing localized heuristic modifications and parameter adaptation. This multi-turn adaptive exploration–exploitation refinement process enables REVEL to progressively transition from coarse-grained structural exploration toward increasingly localized optimization around promising heuristic regions while maintaining refinement diversity throughout evolutionary search.

Search memory. To maintain refinement continuity across iterations, REVEL maintains compressed search memory that accumulates grouped optimization feedback across evolutionary iterations. The memory captures recurring successes, failures, explored search directions, and refinement outcomes throughout the optimization process.

Population update. Refined heuristics generated across all groups are evaluated and merged with the current population. The next generation is formed using elitist selection:

$$\mathcal{H}_{t+1} = \text{TopK}(\mathcal{H}_t \cup \tilde{\mathcal{H}}_t), \tag{16}$$

where $\tilde{\mathcal{H}}_t$ denotes all heuristics generated through group-aware multi-turn refinement.

By combining intra-group refinement with global evolutionary selection, REVEL maintains diverse search trajectories while progressively improving promising heuristic families across optimization iterations.

4 Experiments

4.1 Experimental Setup

We evaluate REVEL through 3 research questions:

Table 1: Analysis of Reasoning Categories, their frequency, and examples.

Reasoning Category	Description	Freq	Example
Paradigm shift	Completely changes the algorithm family.	16	Given that the highest-performing known approach is "Stabilized Harmonic–Arctanh" . . . I should explore this proven algorithmic family rather than tuning simpler approaches.
Heuristic modification	Same algorithm family and pipeline structure, but decision and scoring logic is substantially rewritten.	89	Given the significant performance gap and clear indication that Worst-Fit works better, we should focus on refining this approach . . .
Hyperparameter tuning	Only numeric changes (weights, thresholds, constants) with formulas and pipeline unchanged.	75	Given the regression, we should revert to the simpler version and make minimal adjustments to core parameters . . .

RQ1. [Effectiveness] How effectively does REVEL improve heuristic optimization over existing evolutionary and LLM-guided methods?

RQ2. [Heuristic Evolution] How do behavior-aware grouping and multi-turn refinement shape heuristic evolution during optimization?

RQ3. [Ablation Study] How does each component of REVEL contribute to overall performance?

Benchmarks and baselines. We evaluate REVEL on two representative combinatorial optimization benchmarks: the Traveling Salesman Problem (TSP) and online Bin Packing Problem (BPP). Additional results on CVRP and TSPLib benchmarks are provided in the Appendix.

We compare against recent LLM-based optimization frameworks, including EoH (Liu et al., 2024) and ReEvo (Ye et al., 2024), as well as classical heuristic baselines.

Implementation details. DeepSeek-V3 is used as the default backbone for heuristic generation and reflective refinement. Performance is evaluated using the optimality gap with respect to the best-known solution value, averaged across runs.

4.2 Results

RQ1. [Effectiveness]

Online bin packing. Table 2 shows that REVEL generally outperforms both handcrafted heuristics and prior LLM-based optimization methods across most BPP settings. The largest gains appear under tighter capacities and longer streams, where long-horizon search becomes increasingly difficult.

For example, under capacity $C = 100$ and stream length 1k, REVEL reduces the optimality gap from 5.32% (First Fit) and 3.78% (ReEvo) to 2.34%. At $C = 300$ and length 10k, REVEL further achieves the best overall result with a gap of only 0.16%. These improvements indicate that reflective refinement is particularly effective in challenging online optimization regimes requiring sustained adaptive search.

Table 2: Online bin packing results. Fraction of excess bins to lower bound (lower is better).

Capacity	Size	First Fit	Best Fit	EoH	ReEvo	ReVEL (ours)
100	1k	5.32%	4.87%	3.03%	3.78%	**2.34%**
	5k	4.40%	4.08%	2.15%	**0.80%**	1.13%
	10k	4.44%	4.09%	**0.33%**	**0.33%**	0.59%
300	1k	1.34%	1.19%	0.60%	1.04%	**0.30%**
	5k	0.93%	0.84%	0.63%	0.27%	**0.24%**
	10k	0.92%	0.86%	0.58%	0.19%	**0.16%**
500	1k	0.25%	0.25%	0.25%	0.25%	**0.25%**
	5k	0.50%	0.50%	0.50%	0.50%	**0.50%**
	10k	0.50%	0.47%	0.47%	0.47%	**0.45%**

Traveling salesperson problem. Table 3 reports TSP results across instance sizes ranging from 10 to 200 nodes. REVEL achieves the best performance on most scales and substantially improves over EoH on larger instances. In particular, on TSP200, REVEL reduces the optimality gap from 15.58% to 11.46%.

Table 3: Traveling Salesman Problem Result. Optimality gaps relative to the best-known solutions.

Heuristic	TSP10 (%)	TSP20 (%)	TSP50 (%)	TSP100 (%)	TSP200 (%)
EoH	3.52	9.33	10.24	11.39	15.58
ReEvo	4.22	6.74	11.63	**11.01**	15.58
ReVEL (ours)	**2.11**	**6.74**	**9.20**	12.64	**11.46**

Although ReEvo remains competitive on selected settings such as TSP100, the performance gap generally widens as problem complexity increases, suggesting that iterative reflective refinement improves scalability in high-dimensional

heuristic search spaces.

Across both benchmarks, the results indicate that group-aware reflective refinement improves search robustness across optimization problems with substantially different combinatorial structures.

Table 4: Online Bin Packing: Method Across Models

Heuristic	Model	Gap(%)
ReVEL	deepseek-v3-0324	1.13
ReVEL	kimi-k2-instruct	1.95
ReVEL	qwen3-coder-480b-35a	1.41
ReVEL	glm-4.5	1.20

Effect of LLM backbone. We instantiate REVEL using multiple backbones, including Kimi-K2, DeepSeek-V3, Qwen3-Coder, and GLM-4.5.

As shown in Table 4, REVEL maintains stable performance across all tested backbones, with final optimality gaps ranging from 1.13% to 1.95%. The relatively small variation suggests that the observed improvements primarily arise from the reflective evolutionary workflow rather than dependence on a specific underlying model.

RQ2. [Heuristic Evolution].

Effect of grouping strategies. We evaluate how different grouping strategies influence heuristic refinement on TSP50 under four settings: no grouping, random grouping, pair-wise grouping, and the proposed behavior-aware grouping. Let G_t denote the population optimality gap at iteration t. We define the step-wise improvement as $\Delta_t = G_{t-1} - G_t$, and measure refinement consistency as the fraction of iterations yielding improvement.

Table 5: Performance of Grouping Strategies

Setting	Gap (%)	Δ_t		Cons. (%)
		Mean	Var.	
Random Grouping	15.97	0.86	1.79	45
Without Grouping	17.60	0.72	1.48	40
Pair-wise Grouping	15.55	0.92	1.67	45
Our Method	9.20	1.61	2.84	65

Table 5 shows that behavior-aware grouping consistently improves both optimization quality and refinement consistency. Compared with no grouping, REVEL reduces the final optimality gap from 17.60% to 9.20%, while increasing refinement consistency from 40% to 65%. Moreover, the average step-wise improvement nearly doubles compared with the strongest baseline (1.61 vs. 0.92).

These results suggest that heuristic refinement is highly sensitive to the organization of group-wise feedback. Without behavior-aware grouping, optimization feedback becomes less coherent across behaviorally incompatible heuristics, leading to weaker and less stable updates. In contrast, grouping heuristics according to shared optimization behavior produces more coherent refinement signals and more stable multi-turn optimization.

Evolution of Reflective Refinement Strategies

Beyond final optimization performance, we analyze how reflective reasoning strategies evolve during evolutionary search. At each iteration, the LLM first generates a reasoning trace to interpret previous optimization outcomes and determine whether to explore alternative heuristic strategies or refine existing ones. Rather than treating these traces as isolated qualitative examples, we study them as indicators of refinement behavior during optimization. We categorize reflective reasoning into three operational refinement modes: (1) *structural rewrites*, which introduce substantial changes to the overall heuristic strategy; (2) *heuristic modifications*, which refine local decision logic while preserving the core search structure; and (3) *parameter tunning*, which tune numerical configurations without altering the underlying heuristic behavior. To improve consistency and scalability, we combine expert-defined operational criteria with LLM-assisted classification, followed by expert verification on disagreement and uncertain cases.

The resulting distribution is dominated by heuristic modifications (**89**) and parameter adjustments (**75**), while structural rewrites occur relatively infrequently (**16**). More importantly, we observe a progressive shift in reasoning behavior across generations: early optimization stages contain more structural exploration, whereas later stages increasingly favor localized heuristic refinement. This suggests that grouped reflective feedback enables the optimization process to gradually transition from broad exploratory search toward stable exploitation of promising heuristic families, rather than repeatedly restarting search from scratch.

Figure 3 further reveals staged refinement behavior across optimization iterations. Structural rewrites occur primarily during early search stages, when the heuristic space remains broadly unexplored, but continue to appear intermittently in later generations. In contrast, intermediate and later optimization stages are increasingly dominated by heuristic modifications and parameter adjustments,

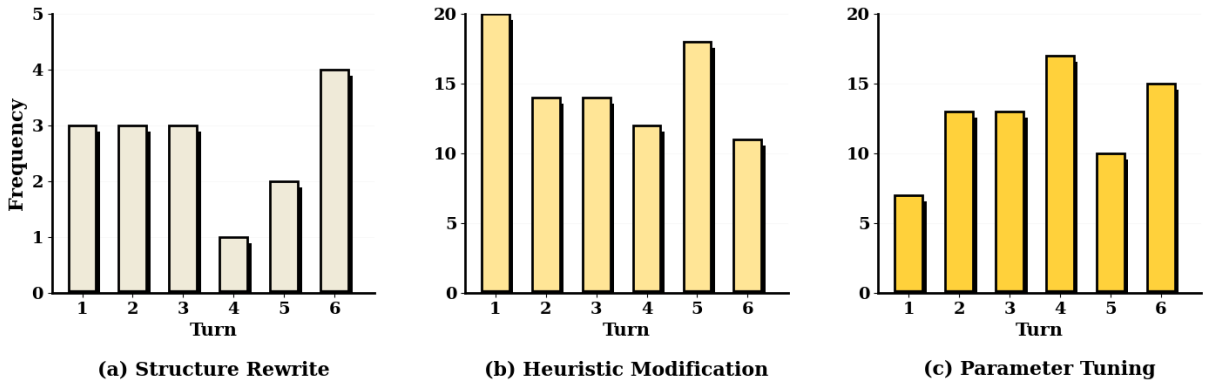

Figure 3: Distribution of refinement operation types across optimization iterations.

indicating a gradual shift from coarse-grained exploration toward more localized refinement around high-performing heuristic regions.

These observations suggest that behavior-aware grouping supports a balanced exploration–exploitation process throughout optimization. Early grouped refinement encourages broader structural exploration over heuristics with similar behavior, while later refinement increasingly concentrates on adaptation and tuning within promising heuristic families. Notably, the continued reappearance of structural rewrites in later generations suggests that reflective grouping may preserve exploratory search behavior even after refinement becomes predominantly exploitative, reducing premature convergence toward narrow heuristic regions.

RQ3. [Ablation Study].

Table 6: Ablation study on TSP50.

Setting	Gap (%)
Single-turn Refinement	17.18
Best-of-N Sampling	16.68
Random Grouping	15.97
Without Grouping	17.60
REVEL (Full)	**9.20**

Table 6 shows that both iterative refinement and behavior-aware grouping are critical to REVEL. Removing either component nearly doubles the final optimality gap (**9.20%** $\rightarrow$ **17.18–17.60%**).

Best-of-N sampling achieves only limited improvement (**16.68%**), indicating that performance gains do not arise from increased sampling alone, but from sustained reflective refinement across iterations. Similarly, random grouping improves over no grouping but remains worse than the proposed method, indicating that reflective quality strongly depends on a coherent grouping structure. **Overall,** the results suggest that REVEL derives its effectiveness from the interaction between long-horizon reflective refinement and behavior-aware grouping, which together produce substantially more effective evolutionary search.

Effect of similarity balancing.

Table 7: Effect of similarity balancing coefficient α on TSP50.

α	Gap (%)
0.2	15.79
0.4	14.06
0.5	**9.20**
0.6	15.97
0.8	12.50

Table 7 studies the effect of similarity balancing coefficient α. Performance is maximized at $\alpha = 0.5$, whereas emphasizing either behavioral or structural similarity alone results in substantial degradation. This suggests that effective reflective grouping requires alignment between optimization behavior and program structure, enabling more coherent reflective feedback and stronger optimization performance.

5 Conclusion

We presented REVEL, a reflective evolutionary framework for LLM-guided heuristic optimization based on grouped iterative refinement. By combining behavior-aware grouping with accumulated reflective feedback, REVEL supports more effective refinement across optimization iterations. Experiments on TSP and online BPP show that REVEL consistently achieves competitive or superior performance over existing LLM-guided evolutionary baselines across benchmarks and LLM backbones. Further analysis suggests that grouped reflective feedback helps maintain a balanced transition from structural exploration to localized heuristic refinement throughout optimization.

6 Limitations

Limitations. While structured performance feedback enables informed multi-turn reflection, it does not guarantee uniformly beneficial refinements. In some cases, group-level feedback may obscure fine-grained behavioral differences, leading the LLM to propose ineffective or redundant heuristic modifications. Although such proposals are filtered by evolutionary selection, this can reduce sample efficiency in certain stages of the search.

External Validity. Although the evaluation does not cover the full diversity of combinatorial optimization settings, REVEL was evaluated across several structurally distinct and widely studied classes of combinatorial optimization problems (COPs), including TSP, CVRP, online Bin Packing, and TSPLib benchmarks. These tasks span routing, packing, and large-scale benchmark optimization scenarios with varying search characteristics and solution structures. Furthermore, REVEL was tested using multiple large language model backbones, including `deepseek-v3-0324`, `kimi-k2-instruct`, `qwen3-coder`, `480b-35a`, and `glm-4.5` (Table 4), spanning both code-oriented and general-purpose instruction-tuned models. Nevertheless, optimization performance may still vary under alternative model architectures, prompting strategies, or domain-specialized LLMs.

Internal Validity. The evolutionary process involves inherent stochasticity arising from initialization, variation operators, and LLM sampling. To control variance, all experiments were conducted with fixed random seeds, repeated across multiple independent runs.

Construct Validity. LLM behavior is sensitive to prompt formulation, and variations in lexical framing or structural emphasis may affect reflective reasoning. To ensure consistency, all prompts followed fixed templates with explicit structural constraints, as detailed in the Appendix.

References

Méziane Aïder, Sabrin Boulebene, and Mhand Hifi. 2025. An adaptative multi-objective scatter search for solving the dynamic bin packing problem. *Journal of Heuristics*, 31(1):5.

Saman M Almufti and Awaz Ahmed Shaban. 2025. Comparative analysis of metaheuristic algorithms for solving the travelling salesman problems'. *International Journal of Scientific World*, 11(2):26–30.

Irwan Bello, Hieu Pham, Quoc V. Le, Mohammad Norouzi, and Samy Bengio. 2017. Neural combinatorial optimization with reinforcement learning. *Preprint*, arXiv:1611.09940.

Stefan Bock, Stefan Bomsdorf, Nils Boysen, and Michael Schneider. 2025. A survey on the traveling salesman problem and its variants in a warehousing context. *European Journal of Operational Research*, 322(1):1–14.

Juergen Branke, Su Nguyen, Christoph Pickardt, and Mengjie Zhang. 2015. Automated design of production scheduling heuristics: A review. *IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION,*, 20.

Edmund Burke, Michel Gendreau, Matthew Hyde, Graham Kendall, Gabriela Ochoa, Ender Özcan, and Rong Qu. 2013. Hyper-heuristics: A survey of the state of the art. *Journal of the Operational Research Society*, 64:1695–1724.

Xinyun Chen, Maxwell Lin, Nathanael Schärli, and Denny Zhou. 2024. Teaching large language models to self-debug. In *The Twelfth International Conference on Learning Representations*.

Minyan Chi, Wei Pang, Xuan Wu, Peng Zhao, Yuan-Shu Li, Tianfang Wang, Junjie Qian, Yubin Xiao, Liupu Wang, and You Zhou. 2026. A generalized neural solver based on llm-guided heuristic evoluation framework for solving diverse variants of vehicle routing problems. *Expert Systems with Applications*, 296:128876.

Pham Vu Tuan Dat, Long Doan, and Huynh Thi Thanh Binh. 2025. 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, pages 26931–26938. https://github.com/datphamvn/HSEvo.

Halil Alperen Gozeten, Muhammed Emrullah Ildiz, Xuechen Zhang, Hrayr Harutyunyan, Ankit Singh Rawat, and Samet Oymak. 2026. Continuous chain of thought enables parallel exploration and reasoning. In *The Fourteenth International Conference on Learning Representations*.

Qi Huang, Furong Ye, Ananta Shahane, Thomas Bäck, and Niki van Stein. 2026. From heuristic selection to automated algorithm design: Llms benefit from strong priors. *arXiv preprint arXiv:2603.02792*.

Wouter Kool, Herke van Hoof, and Max Welling. 2018. Attention, learn to solve routing problems! In *International Conference on Learning Representations*.

Yujia Li, David Choi, Junyoung Chung, Nate Kushman, Julian Schrittwieser, Rémi Leblond, Tom Eccles, James Keeling, Felix Gimeno, Agustin Dal Lago, Thomas Hubert, Peter Choy, Cyprien de Masson d'Autume, Igor Babuschkin, Xinyun Chen, Po-Sen Huang, Johannes Welbl, Sven Gowal,

Alexey Cherepanov, and 7 others. 2022. Competition-level code generation with alphacode. *Science*, 378(6624):1092–1097.

Junfeng Liao, Qizhou Wang, Shanshan Ye, Xin Yu, Ling Chen, and Zhen Fang. 2026. Explainable LLM unlearning through reasoning. In *The Fourteenth International Conference on Learning Representations*.

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

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. 2023. Self-refine: Iterative refinement with self-feedback. In *Thirty-seventh Conference on Neural Information Processing Systems*.

Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D Manning, Stefano Ermon, and Chelsea Finn. 2023. Direct preference optimization: Your language model is secretly a reward model. In *Thirty-seventh Conference on Neural Information Processing Systems*.

Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Matej Balog, M. Kumar, Emilien Dupont, Francisco Ruiz, Jordan Ellenberg, Pengming Wang, Omar Fawzi, Pushmeet Kohli, and Alhussein Fawzi. 2023. Mathematical discoveries from program search with large language models. *Nature*, 625.

Federico Rondano. 2025. *Heuristic Algorithms for the Min-Max Traveling Salesman Problem*. Ph.D. thesis, Politecnico di Torino.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. 2024. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *Preprint*, arXiv:2402.03300.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik R Narasimhan, and Shunyu Yao. 2023. Reflexion: language agents with verbal reinforcement learning. In *Thirty-seventh Conference on Neural Information Processing Systems*.

Niki van Stein and Thomas Bäck. 2025. Llamea: A large language model evolutionary algorithm for automatically generating metaheuristics. *Preprint*, arXiv:2405.20132.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.

Alonso Vela, Gerardo Humberto Valencia-Rivera, Jorge M Cruz-Duarte, José Carlos Ortiz-Bayliss, and Ivan Amaya. 2025. Hyper-heuristics and scheduling problems: strategies, application areas, and performance metrics. *IEEE Access*.

Song Xu, Xiangyue Ou, Kannan Govindan, Mingzhou Chen, and Wenting Yang. 2025. An adaptive genetic hyper-heuristic algorithm for a two-echelon vehicle routing problem with dual-customer satisfaction in community group-buying. *Transportation Research Part E: Logistics and Transportation Review*, 194:103874.

Haoran Ye, Jiarui Wang, Zhiguang Cao, Federico Berto, Chuanbo Hua, Haeyeon Kim, Jinkyoo Park, and Guojie Song. 2024. Reevo: Large language models as hyper-heuristics with reflective evolution. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*.

Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, and 1 others. 2023. A survey of large language models. *arXiv preprint arXiv:2303.18223*, 1(2).

A Appendix

A.1 Additional Related Work

A.1.1 Automatic Heuristic Design

Traditional heuristic design relies heavily on manually engineered rules tailored to problem-specific structures, limiting scalability and domain generalization (Burke et al., 2013). Early attempts to automate this process employed Genetic Programming (GP) and hyper-heuristics to evolve heuristics from primitive operators or predefined pools (Branke et al., 2015). More recently, Neural Combinatorial Optimization (NCO) approaches leverage Deep Reinforcement Learning (DRL) to learn constructive optimization policies directly from problem instances (Bello et al., 2017; Kool et al., 2018).

Although these approaches reduce manual engineering effort, they typically require substantial training costs and remain sensitive to problem representations, reward design, and distributional shifts. More importantly, they primarily optimize fixed policy structures rather than enabling iterative reasoning and adaptive heuristic refinement through feedback.

A.1.2 Large Language Models for Evolutionary Search

Large Language Models (LLMs) (Vaswani et al., 2017; Zhao et al., 2023) have recently emerged as powerful tools for code generation and algorithmic search, as they can reason over program semantics rather than relying solely on syntactic perturbations. This capability enables LLMs to participate directly in heuristic synthesis, variation, and refinement within evolutionary optimization frameworks (Chi et al., 2026; Huang et al., 2026). Compared to earlier systems such as AlphaCode (Li et al., 2022), which largely rely on large-scale sampling and post-hoc filtering, recent LLM-guided frameworks increasingly emphasize iterative refinement guided by evaluator feedback.

FunSearch (Romera-Paredes et al., 2023) pioneered the integration of LLMs with Evolutionary Computation (EC) by coupling program generation with evaluator-guided search for combinatorial optimization. Subsequent approaches explored tighter integration between LLMs and ECs. EoH (Liu et al., 2024) co-evolves executable heuristics and natural-language reasoning traces, while LLaMEA (van Stein and Bäck, 2025) replaces conventional genetic operators with semantic LLM-driven mutation and crossover. Despite promising results, these frameworks primarily emphasize heuristic generation and variation, while reflective feedback remains localized to isolated candidates or pairwise comparisons.

A.1.3 Multi-Turn Reflection

Beyond standard evolutionary optimization, recent work has explored multi-turn reflection mechanisms for improving LLM outputs through iterative self-correction. Frameworks such as Self-Refine (Madaan et al., 2023) and Reflexion (Shinn et al., 2023) demonstrate that LLMs can iteratively critique and refine their own outputs using feedback generated during interaction. In code generation, Self-Debugging (Chen et al., 2024) further extends this paradigm by leveraging execution feedback, such as unit test results and runtime traces, to correct implementation errors autonomously.

Within heuristic optimization, ReEvo (Ye et al., 2024) adapts reflective prompting to evolutionary search by generating verbal critiques from performance differences between heuristic pairs and subsequently refining the corresponding code. HSEvo (Dat et al., 2025) further considers population diversity during heuristic evolution. However, existing reflective optimization frameworks still reason primarily over isolated heuristics or local pairwise relations, lacking mechanisms for reasoning over optimization populations as coherent, evolving systems. Consequently, they remain limited in their ability to accumulate population-level optimization knowledge and coordinate long-horizon exploratory and exploitative search dynamics. Collectively, existing approaches either emphasize heuristic generation without structured reflection or reflection without population-level optimization reasoning. In contrast, REVEL formulates heuristic evolution as iterative reasoning over evolving heuristic populations, guided by structured reflective feedback.

B Detail of benchmark dataset

Baseline setting. To ensure a fair comparison, we align the setup based on the shared evolutionary paradigm of both methods by standardizing the population size and number of generations, which determine selection pressure and search depth. Concretely, we use comparable configurations (a population size of 10 and 20 evolutionary generations), ensuring similar evolutionary dynamics while retaining original mechanisms and default hyperparameters to avoid tuning bias; note that exact

LLM call counts may differ due to implementation-specific factors, so we focus on aligning the high-level evolutionary structure.

B.1 Traveling Salesman Problem (TSP)

B.1.1 Problem Definition

The Traveling Salesman Problem (TSP) consists of finding the shortest possible tour that visits each city exactly once and returns to the starting city. Each instance is defined by a set of N cities with two-dimensional coordinates

$$\mathbf{x}_i = (x_i, y_i), \quad i = 1, \ldots, N,$$

where $x_i, y_i \in [0, 1]$.

The objective is to find a permutation π of $\{1, \ldots, N\}$ that minimizes the total tour length

$$L(\pi) = \sum_{k=1}^{N-1} d(\pi_k, \pi_{k+1}) + d(\pi_N, \pi_1).$$

B.1.2 Evaluation Metrics

The distance between cities i and j is the Euclidean distance $d(i, j) = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$. We evaluate solution quality using the **Optimality Gap** relative to the OR-Tools baseline (L_{OR}):

$$\text{Gap } (\%) = \frac{L(\pi) - L_{OR}}{L_{OR}} \times 100\%$$

where $L(\pi)$ represents the total tour length. We report the average gap across all instances; a lower gap indicates higher proximity to the best-known solution.

B.1.3 Problem Sizes and Instances

We consider multiple TSP problem sizes in order to evaluate scalability across small, medium, and large instances. Table 8 summarizes all problem sizes used in our experiments.

Table 8: TSP problem sizes

Dataset	Number of Cities
TSP10	10
TSP20	20
TSP50	50
TSP100	100
TSP200	200

B.2 Online Bin Packing Problem

B.2.1 Problem Definition

In the online one-dimensional bin packing problem, items arrive sequentially. Each item must be assigned immediately to a bin of C capacity. If no existing bin has sufficient remaining capacity, a new bin is opened.

Given a sequence of item sizes $s_1, s_2, \ldots, s_T$ with $s_i \in (0, 1]$, the objective is to minimize the total number of bins used. Reordering or moving previously packed items is not allowed.

B.2.2 Evaluation Metrics

For each instance, items are processed strictly in their arrival order (Online Bin Packing). Each item is either assigned to an existing bin with sufficient remaining capacity or placed into a newly opened bin. Once an item is placed, it cannot be moved or reassigned.

Let B denote the total number of bins used by an algorithm for a given instance I. We compare the algorithm's performance against the standard **volume lower bound**, defined as:

$$LB(I) = \left\lceil \frac{\sum_{i=1}^{T} s_i}{C} \right\rceil$$

where s_i is the size of item i, T is the sequence length, and C is the bin capacity.

The efficiency is measured using the **Fraction of Excess Bins**, expressed as a percentage:

$$\text{Fraction of excess bins}(I)(\%) = \left(\frac{B - LB(I)}{LB(I)}\right) \times 100\%$$

This metric quantifies the relative overhead compared to the theoretical minimum. A value of 0% indicates an optimal solution (matching the lower bound), while higher percentages reflect lower packing efficiency. Final results are reported as the average percentage across all test instances.

B.2.3 Problem Sizes and Instances

To evaluate performance across different scales, we construct datasets with varying bin capacities and number of items. Table 9 lists all configurations, where each dataset is defined by a pair (T, C) of number of items T and bin capacity C.

Table 9: Online Bin Packing problem sizes

Dataset	Number of Items (T)	Capacity (C)
BPP-1K-100	1K	100
BPP-1K-300	1K	300
BPP-1K-500	1K	500
BPP-5K-100	5K	100
BPP-5K-300	5K	300
BPP-5K-500	5K	500
BPP-10K-100	10K	100
BPP-10K-300	10K	300
BPP-10K-500	10K	500

C Preliminary

C.1 Evolutionary Algorithms

Formally, let $\mathcal{H}$ denote the space of all feasible heuristics. The goal of an Evolutionary Algorithm (EA) is to identify an optimal heuristic $h^* \in \mathcal{H}$ that maximizes a fitness function $\mathcal{F} : \mathcal{H} \to \mathbb{R}$:

$$h^* = \arg\max_{h \in \mathcal{H}} \mathcal{F}(h) \tag{17}$$

The algorithm maintains a population state at generation t, denoted as $\mathcal{P}^{(t)} = \{h_1^{(t)}, h_2^{(t)}, \ldots, h_N^{(t)}\}$, where N is the population size. The evolutionary process describes the transition $\mathcal{P}^{(t)} \to \mathcal{P}^{(t+1)}$ via a stochastic mapping $\mathcal{T}$:

$$\mathcal{P}^{(t+1)} = \mathcal{T}(\mathcal{P}^{(t)}; \mathcal{S}, \mathcal{V}) \tag{18}$$

where the transition relies on two primary operators:

1. **Fitness Evaluation:** Every individual $h_i^{(t)}$ is evaluated to obtain a quality score $s_i = \mathcal{F}(h_i^{(t)})$. This score typically drives the selection pressure.

2. **Selection Operator ($\mathcal{S}$):** This operator selects a subset of parents $\mathcal{P}_{parents} \subset \mathcal{P}^{(t)}$ based on their fitness. A common formulation is tournament selection, where a parent is chosen via:

$$h_{parent} = \arg\max_{h \in \mathcal{T}_{sub}} \mathcal{F}(h) \tag{19}$$

$$\mathcal{T}_{sub} \sim \text{Uniform}(\mathcal{P}^{(t)}, k) \tag{20}$$

where $\mathcal{T}_{sub}$ is a random subset of size k.

3. **Variation Operator ($\mathcal{V}$):** In standard EAs, this involves Crossover ($\mathcal{C} : \mathcal{H} \times \mathcal{H} \to \mathcal{H}$) and Mutation ($\mathcal{M} : \mathcal{H} \to \mathcal{H}$).

   In the context of **LLM-driven optimization**, the variation operator is re-parameterized as a conditional generation task using a Large Language Model π_θ. Given a prompt template I and parent heuristics, the offspring h_{new} is sampled from the model's distribution:

$$h_{new} \sim \pi_\theta(\cdot \mid I, \mathcal{P}_{parents}) \tag{21}$$

   Here, the LLM acts as a semantic mutation operator, leveraging its pre-trained knowledge to propose meaningful code transformations rather than random bit-flips.

Finally, the next generation $\mathcal{P}^{(t+1)}$ is formed by selecting the best candidates from the union of the current population and the newly generated offspring (elitism), ensuring monotonic improvement in the best-found solution over time.

C.2 From Pairwise to group-aware Estimation

A central challenge in guiding LLM-based evolution is effectively estimating the relative quality of generated solutions to provide constructive feedback. Our methodology is directly motivated by the recent shift in preference optimization—specifically, the transition from pairwise comparisons to group-relative evaluations.

Limitations of Pairwise Comparison (DPO): Direct Preference Optimization (DPO) (Rafailov et al., 2023) aligns models by maximizing the likelihood margin between a winning sample y_w and a losing sample y_l. While effective for general alignment, this pairwise paradigm is suboptimal for evolutionary search. Evaluating a population of size N via pairwise comparisons restricts the model's view to local differences, ignoring the global context of how a solution stands against the entire population distribution.

The group-aware Shift (GRPO): Group Relative Policy Optimization (GRPO) (Shao et al., 2024) addresses this by introducing a *group-aware* perspective. Instead of isolating pairs, GRPO evaluates a candidate o_i relative to a group of outputs $\{o_1, \ldots, o_G\}$ generated from the same prompt, using the group mean as a dynamic baseline:

$$A_i = \frac{f(o_i) - \text{mean}(\{f(o_j)\}_{j=1}^G)}{\text{std}(\{f(o_j)\}_{j=1}^G) + \epsilon} \tag{22}$$

Adoption in Our Approach: Inspired by this specific advancement, we adopt the *grouping mechanism* of GRPO as the core principle for our reflection module. We hypothesize that for an LLM to

effectively critique and improve heuristics, it must analyze solutions not in isolation or pairs, but as a cluster.

C.3 Prompt

Online Bin Packing Prompt Formulation

```
TASK SUMMARY

You are an AI assistant whose job is to
     iteratively produce and refine
     Python heuristic implementations for
     the Bin Packing Online Problem.
You will be given an existing heuristic
     (or helper functions). Use
     multi-turn reasoning: at each turn
     you must reflect, then either
     **explore** a new heuristic family
     or **exploit** (refine) the last
     submitted heuristic, and finally
     receive an observation/feedback from
     the environment.

---

### FUNCTION CONTRACT (must be strictly
     respected)
- Language: Python only. Only standard
     library and numpy allowed (if
     already used by provided code).
- Required signature:
    def score(item, bins)
- Input arguments:
    - item: int # size of current item
    - bins : Numpy arrays # the rest
        capacities of feasible bins,
        which are larger than the item
        size.
- Return: scores (Numpy array)
- Correctness rules:
    - 'item' is of type int
    - 'bins' and 'scores' are both Numpy
        arrays.
```

Travelling Salesman Problem Prompt Formulation

```
TASK SUMMARY

You are an AI assistant whose job is to
     iteratively produce and refine
     Python heuristic implementations for
     the Travelling salesman problem.
Given a set of nodes with their
     coordinates, \
you need to find the shortest route that
     visits each node once and returns to
     the starting node. \
The task can be solved step-by-step by
     starting from the current node and
     iteratively choosing the next node. \
You will be given an existing heuristic,
     Let use multi-turn reasoning: at
     each turn you must reflect, then
     either **explore** a new heuristic
     family or **exploit** the last
     submitted heuristic, and finally
     receive an observation/feedback from
     the environment.

### FUNCTION CONTRACT (must be strictly
     respected)
- Language: Python only. Only standard
     library and numpy allowed (if
     already used by provided code).
- Required signature:
    def select_next_node(current_node,
        destination_node,
        unvisited_nodes, distance_matrix)
- Input arguments: This function should
     accept 4 inputs: 'current_node',
     'destination_node',
     'unvisited_nodes', 'distance_matrix'
- Output: The function should return 1
     output: 'next_node'
- Correctness rules: 'current_node',
     'destination_node', 'next_node', and
     'unvisited_nodes' are node IDs.
     'distance_matrix' is the distance
     matrix of nodes. All are Numpy
     arrays.
Do not give additional explanations.
```

Prompt Template for the THINK Step

```
ALWAYS REMEMBER THAT, LOWER fitness
     score = BETTER solution.
First, based on the evaluation result
     from <observation> or GROUP
     REFLECTION,
you should do some critical reasoning
     about the previous approach(s) ABOUT:
+ its logical algorithm
+ its heuristic
     components/hyperparamters/features
     specifically. Then, think about the
     affect of these
     parameters/hyperparamters to the
     fitness score result (in detailed).

Then, you can:

1. Explore a totally new approach, to
     make some experiments to get
     information.
OR
2. Focus on the behaviour of the
     heuristic features/components from
     the fitness result to tune them and
     get better result from the test
     evaluation.

You are ONLY allowed to do reasoning,
     NOT to generate code.
Note that, your reasoning should be very
     BRIEF but STILL critical and
     concise, focus on analyzing the
     heuristic components/features.
At the last of your response, there must
```

```
    be one of the tags <explore> or
    <exploit>, which indicate your
    decision.
```

Prompt Template for the Exploration Phase

```
Now, BASED solely on your REASONING,
    generate EXACTLY ONE solution for
    exploring.
Your output MUST be exactly the SAME as
    the following format:
<explore>
<algorithm>
# clear and complete algorithm
    description of the proposed
    heuristic.
</algorithm>
<code>
# the completely new Python function
    implementation for the algorithm in
    <algorithm> : `score(...)` (only
    code inside `<code>`).
</code>
</explore>
OUTPUT RULE:
Always output exactly one <explore>
    block containing both <algorithm>
    and <code>, nothing else.
```

Prompt Template for the Exploitation Phase

```
Now, BASED solely on your REASONING,
    generate EXACTLY ONE solution for
    exploiting.
Your output MUST be exactly the SAME as
    the following format:
<exploit>
<algorithm>
# Clear algorithm description of the
    improvements you're making to the
    selected algorithm
</algorithm>
<code>
# Complete and concise Python function
    implementation with your
    refinements: `score(...)`
</code>
</exploit>
OUTPUT RULE:
Always output exactly one <exploit>
    block containing both <algorithm>
    and <code>, nothing else.
```

LLM Prompt Template for Heuristic Population Clustering

```
You are given a list of candidate
    solutions for a heuristic problem.
Each candidate is a JSON object with
    fields:
- "id": unique identifier
- "code": the code itself
- "score": numeric performance value

Task: Group candidates that likely have
    the same behavior
1. Compare candidates by, code
    structure, and score values.
2. Group candidates into clusters of
    similar behavior, idea, or
    performance.
3. For each group, output:
   - Group label (integer ID).
   - List of member IDs.
   - Shared characteristics that define
       this group (algorithm style,
       coding pattern, performance
       range).
   - Distinctive differences within the
       group (if any).
4. If a candidate does not fit any
    existing group, assign it as its own
    group.
Also consider performance similarity:
- Candidates with very different scores
    (e.g., difference > 0.1 on a [01]
    scale or > 10% relative difference)
    should not be grouped, even if code
    is similar.
- Candidates with similar algorithmic
    ideas and close scores should be
    grouped together.

Output format:
Return ONLY a JSON object:

### example
{
  "groups": [
    {
      "id": 1,
      "members": ["1", "7", "9"],
      "shared_characteristics": "...",
      "differences": "...",
      "score_range": {"min": 0.72, "max":
          0.75}
    },
    {
      "id": 2,
      "members": ["3", "4"],
      "shared_characteristics": "...",
      "differences": "...",
      "score_range": {"min": 0.55, "max":
          0.58}
    },
    ...
  ]
}
###

CANDIDATE SOLUTIONS:\
```

D Additional Result

D.1 Reasoning Behavior

As shown in Fig. 4, the Reflective LLM-Guided Heuristic Evolution via Structured Performance Feedback process demonstrates how score dynamics and strategic adaptation interact to drive progress beyond local optima. At the outset, when the system observed a fitness of 0.042, considerably worse than the best-known score of 0.031, it recognized stagnation and initiated an exploratory shift. This led to the generation of Adaptive Resonance Packing (ARP), a structurally novel heuristic that improved the score to 0.036. Although this gain was modest, the introduction of ARP provided a fertile ground for subsequent refinements.

Building on this foundation, the process transitioned into targeted exploitation. By tuning ARP through dynamic bandwidth control, exponential gap weighting, and softmax normalization, the system sought to consolidate and optimize the idea. While the score did not improve beyond 0.036, this stage illustrates the reflective nature of the method: rather than abandoning a promising approach, it strategically invested in fine-tuning, ensuring stability before pursuing further change.

When performance remained unchanged, the system's observation mechanism signaled diminishing returns and prompted a more dramatic innovation. This shift produced Quantum Tunneling Bin Packing (QTBP), a probabilistic mechanism inspired by tunneling to bypass local barriers. Crucially, this step reduced the score to 0.023, surpassing the best-known baseline of 0.031.

Our approach integrates exploration, exploitation, and reflective reasoning: exploration opens new directions, exploitation strengthens promising structures, and reflection decides when radical change is needed. Unlike naive retries, it uses performance signals to avoid stagnation and achieve breakthroughs beyond conventional search.

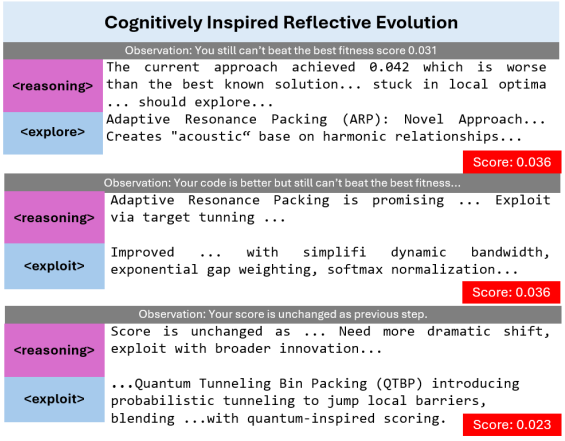

Figure 4: **Qualitative example of reflective reasoning in ReVEL:** Observations guide the LLM to alternate between exploration, exploitation, and innovation, resulting in progressive improvement of heuristic quality.

D.2 Cost efficiency

Table 10: Average API Consumption Costs per Run

Method	API Cost
ReEvo	$0.25
EoH	$0.33
ReVEL (Ours)	**$0.41**

The actual API costs under identical conditions are reported in Table 10. Despite a modest increase in cost compared to the baselines, ReVEL consistently achieves superior performance. Importantly, these gains are practically meaningful: in BPP, they translate to up to 20 fewer bins, directly reducing operational cost. This reflects a stronger cost–quality trade-off, where a small additional budget yields substantially higher-quality solutions.

To further ensure fairness, we extend EoH and ReEvo to match ReVEL's cost budget. The results remain consistent: both baselines plateau coverage early, as they rely on local, instance-level feedback that quickly becomes saturated around suboptimal solutions.

D.3 TSPLib

Evaluation on TSPLib Instances.

D.3.1 Evaluation on TSPLib Instances

To further evaluate the practical effectiveness and robustness of ReVEL, we conducted additional experiments on a diverse set of TSPLib benchmark instances. Unlike the synthetic Euclidean datasets used in previous sections, TSPLib instances exhibit

heterogeneous spatial structures, irregular node distributions, and varying scales. They are widely regarded as representative of practical TSP scenarios.

For each TSPLib instance, ReVEL was executed independently under the same configuration to evolve a heuristic specialized for that instance. Table 9 reports the percentage gap to the best-known TSPLib solutions and compares ReVEL with Nearest Neighbour and EoH.

Overall, ReVEL achieves the best results on 12 out of 18 instances and consistently improves upon the Nearest Neighbour baseline, confirming that the discovered heuristics substantially outperform classical constructive strategies. Compared with EoH, ReVEL shows competitive performance across most instances and provides substantial improvements on several structurally diverse problems, including Berlin52 (6.24% vs. 17.68%), KroC100 (5.88% vs. 19.31%), Rat99 (12.30% vs. 21.07%), and Ts225 (3.73% vs. 8.99%).

Notably, the performance advantage of ReVEL is particularly pronounced on instances with irregular spatial structures, where purely local constructive rules are insufficient. This suggests that the structured reflective feedback in ReVEL enables the LLM to discover heuristics that capture higher-level structural patterns rather than relying solely on greedy proximity decisions.

Across different instance sizes, ReVEL maintains stable performance without instance-specific parameter tuning, indicating that the proposed reflective evolution mechanism scales consistently across problem regimes.

These results demonstrate that ReVEL can reliably guide heuristic discovery across heterogeneous instance structures and problem scales, extending its effectiveness beyond synthetic benchmarks to realistic combinatorial optimization problems.

Table 11: Results on TSPLib instances. The gap (%) to the best-known solution from TSPLib.

Instance	Nearest Neighbour	EoH	Our
Berlin52	19.06	17.68	**6.24**
Bier127	15.27	**8.04**	9.33
Ch130	23.98	**6.81**	10.21
D493	17.20	**14.85**	15.69
Eil51	19.72	**3.19**	3.68
KroA100	26.19	11.06	**6.21**
KroB100	31.68	9.84	**9.65**
KroC100	26.88	19.31	**5.88**
KroD100	26.56	13.21	**10.01**
KroE100	25.01	**11.79**	13.46
Pr124	16.97	16.69	**12.89**
Pr136	24.81	11.95	**6.73**
Pr226	18.52	11.66	**8.44**
Rat99	29.21	21.07	**12.30**
St70	19.33	16.33	**14.98**
Ts225	22.07	8.99	**3.73**
U159	32.93	11.92	**11.43**
Lin105	41.61	**7.53**	21.37

Table 12: Hyperparameters Configuration

Parameter Name	**Value**
Evolution Loop Parameters	
max_turns	6
num_candidates_to_initialize	10
epochs	20
top_k	10
reminder_probability	0.3
Clustering & Cross-over Parameters	
num_clusters	3
num_elements	4
alpha	0.5
groups_per_crossover	1
Evaluator Parameters	
timeout_seconds	70

Explanation of hyperparameters. As shown in Table 12, the hyperparameters are organized into three groups. The *Evolution Loop Parameters* control the overall evolutionary process, including the maximum number of interaction turns within each group (`max_turns`), the initial population size (`num_candidates_to_initialize`), the total number of evolutionary epochs (`epochs`), and the number of top-performing candidates pre-

served after each epoch (`top_k`). The parameter `reminder_probability` defines the probability of proactively injecting a reminder message to the LLM during the interaction loop, reinforcing awareness of the current best-fitness solution.

The *Clustering & Cross-over Parameters* determine how candidates are grouped and recombined. Specifically, `num_clusters` specifies the number of clusters formed from the population, `num_elements` controls how many representative candidates are selected from each cluster for cross-over, and `alpha` balances semantic output similarity and code-level similarity during clustering. The parameter `groups_per_crossover` enforces that each cross-over operation produces exactly one new interaction group.

Finally, the *Evaluator Parameters* define execution constraints during evaluation. The parameter `timeout_seconds` limits the maximum runtime for evaluating each generated candidate, preventing non-terminating executions and ensuring stable evolutionary progress.

D.4 Additional Baseline Results on CVRP

The Capacitated Vehicle Routing Problem (CVRP) is a classical combinatorial optimization problem in which a fleet of vehicles with limited capacity must serve a set of customers with known demands while minimizing the total travel cost. Each vehicle starts and ends at a depot, and customer demands must not exceed vehicle capacity.

To assess the robustness of **ReVEL** beyond TSP and BPP, we conducted additional experiments on the *CVRP* using the same experimental configuration as in the main paper. In this setting, heuristics were evolved independently for each problem size using an *ACO-based backbone*, allowing ReVEL to adapt to different combinatorial structures and heuristic paradigms.

For a strong and meaningful comparison, we evaluated against **ReEvo**, a recent state-of-the-art LLM-guided heuristic evolution framework for this task. Since both methods operate under similar LLM-driven evolutionary settings, this comparison provides a controlled evaluation of the reflective evolution mechanism.

In table 13 ReVEL consistently outperforms ReEvo across all tested scales, with particularly large improvements on **CVRP20 (4.57% vs. 13.70%)** and clear gains on larger instances as well. These results demonstrate that the proposed reflective evolution framework is not limited to a specific problem type or heuristic structure, but can effectively guide heuristic discovery under different optimization paradigms.

Importantly, the CVRP experiments use the same default configuration without problem-specific tuning, indicating that ReVEL generalizes across combinatorial domains with minimal adjustment.

Overall, these additional results provide strong empirical evidence that ReVEL is a general LLM-guided heuristic evolution framework , rather than a method specialized for a single problem class.

Table 13: Baseline OR-Tools Results for the Capacitated Vehicle Routing Problem.

Heuristic	CVRP20 (%)	CVRP50 (%)	CVRP100 (%)
ReEvo	13.70	19.44	18.06
ReVEL (ours)	**4.57**	**17.68**	**14.88**

E Cost–Performance Trade-off and Reflective Improvement Analysis

To better understand the efficiency and mechanism-level benefits of multi-turn reflection, we analyze both the computational cost and the performance trajectories of the generated heuristics. Figures 5 and 6 report the optimization trajectories, along with the accumulated evaluation cost, for representative instances of the Traveling Salesman Problem (TSP20 and TSP50) and the Bin Packing Problem (BPP300–2000 items).

The results demonstrate a clear advantage in the cost–performance trade-off achieved by multi-turn reflection. On TSP50, ReVEL achieves a solution gap of approximately 9% with a total cost of $0.68\$$. In contrast, single-turn reflection achieves only about 17% gap at $0.15\$$, and the no-reflection baseline attains approximately 18% gap at $0.37\$$. Similar behavior is observed on TSP20, where ReVEL consistently produces lower solution gaps while maintaining comparable computational cost.

The advantage of multi-turn reflection becomes even more pronounced in more challenging scenarios. On BPP instances with 300–2000 items, both single-turn reflection and the no-reflection baseline fail to generate any heuristic that improves upon the initial performance, even after the same number of evolutionary steps. In contrast, ReVEL continues to produce progressively improved heuristics under identical evaluation settings.

Although ReVEL requires a moderately higher computational cost, the improvement in solution

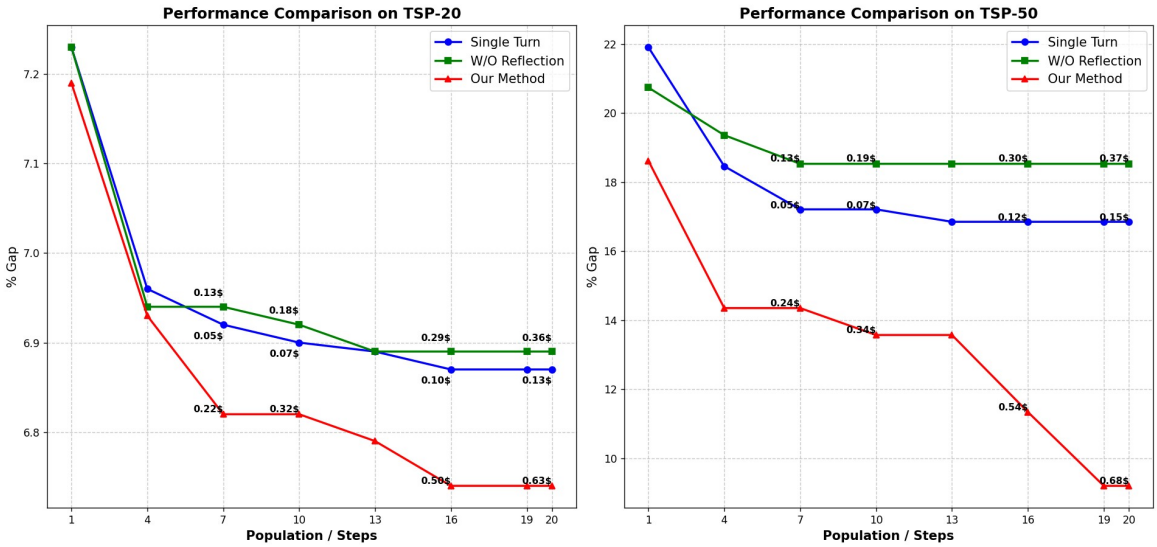

Figure 5: Performance comparison of Single Turn, W/O Reflection, and our method on TSP-20 and TSP-50.

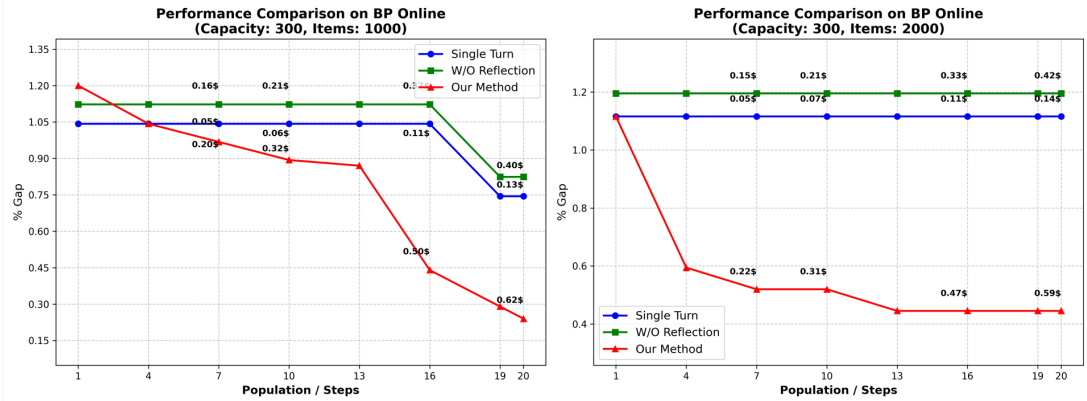

Figure 6: Performance comparison on online bin packing with capacity 300 and 1000/2000 items.

quality is substantial. The additional cost mainly arises from two components:

- **Pre-generation reflection**, which enables the language model to interpret structured performance feedback before heuristic synthesis;
- **Multi-turn refinement**, which allows heuristics to be iteratively improved rather than generated independently.

These mechanisms enable more effective exploration of the heuristic space and improve the utilization efficiency of the evaluation budget.

The trajectory analysis also provides evidence that multi-turn reflection leads to progressive improvement of heuristic quality. Both single-turn reflection and the no-reflection baseline exhibit rapid early improvement, followed by clear performance plateaus, even when given the same computational budget as ReVEL. This behavior suggests limited reasoning depth and restricted capability to refine heuristic logic.

In contrast, ReVEL demonstrates steady and sustained improvement throughout the reflective process. As structured feedback accumulates across reflection turns, the language model can refine heuristic strategies consistently and coherently. This results in a sequence of heuristics with progressively improved performance rather than a collection of largely independent candidates.

Overall, the results indicate that multi-turn reflection is not merely an additional computational expense but a mechanism that enables structured iterative improvement of heuristic programs. The proposed framework achieves a favorable cost–performance trade-off by producing substantially better heuristics with only a modest increase in computation while providing clear empirical evidence of progressive heuristic refinement.

E.1 Robustness to Prompt Variations

We evaluate the sensitivity of REVEL to prompt formulation by testing whether its performance depends on lexical phrasing or is primarily governed by its iterative mechanism. The framework uses a fixed meta-template across all experiments, where only a minimal task specification (input/output definition) is provided, while reasoning directives remain unchanged.

To assess robustness, we conduct experiments on TSP50 using three substantially different prompt variants: (i) the original prompt used in the paper, (ii) a more detailed reasoning-oriented prompt, and (iii) a concise minimal prompt. Each configuration is executed with three independent runs to reduce stochastic variance.

The results show that prompt variations have a limited impact on final performance. While longer prompts provide slightly better initial solutions (17.9% gap vs. 18.2% and 18.4% for the original and concise variants), all configurations converge to nearly identical final performance after full optimization (9.2% gap). This indicates that prompt differences primarily affect early exploration behavior but do not alter the eventual optimization outcome.

These findings suggest that REVEL 's performance is predominantly driven by its iterative reflection-and-selection mechanism rather than prompt engineering. The consistent convergence across substantially different prompt formulations indicates that the feedback structure and trajectory-level refinement process govern the optimization dynamics.

Algorithm 1 ReVEL: Multi-Turn Reflective Evolution

Require: Instance set $\mathcal{D}$, population size N, generations G, reflection turns T

Ensure: Best heuristic h^*

Initialize population $\mathcal{H}^{(0)} = \{h_1, \ldots, h_N\}$ using LLM
for $g = 1$ to G **do**
  Evaluate each heuristic $h \in \mathcal{H}^{(g-1)}$
  Compute fitness:

$$f(h) = \frac{1}{|\mathcal{D}|} \sum_{x \in \mathcal{D}} \text{score}(h, x)$$

  Compute performance vectors $\mathbf{z}(h)$
  Build similarity clusters $\mathcal{C}$ using $\mathbf{z}(h)$
  Construct reflection groups $\mathcal{G} = \{G_1, \ldots, G_M\}$
  (each G_j mixes similar and heterogeneous heuristics)
  for each group $G_j \in \mathcal{G}$ **do**
    Initialize feedback $\mathcal{F}_j^{(1)}$ from:
    group performance statistics
    representative heuristics
    behavioral differences
    for $t = 1$ to T **do**
      Query LLM using feedback $\mathcal{F}_j^{(t)}$
      Generate candidate heuristics $\tilde{\mathcal{H}}_j^{(t)}$
      Evaluate candidates and update feedback:

$$\mathcal{F}_j^{(t+1)} = \mathcal{F}_j^{(t)} \cup \text{Perf}(\tilde{\mathcal{H}}_j^{(t)})$$

    end for
  end for
  Collect candidates:

$$\tilde{\mathcal{H}} = \bigcup_{j=1}^{M} \bigcup_{t=1}^{T} \tilde{\mathcal{H}}_j^{(t)}$$

  Select next population $\mathcal{H}^{(g)}$ from $\mathcal{H}^{(g-1)} \cup \tilde{\mathcal{H}}$
end for
return

$$h^* = \arg\min_h f(h)$$