BeNav: Behavior-Space Navigation for Automatic Heuristic Design

Anonymous authors
Paper under double-blind review

Abstract

Large language models (LLMs) have recently enabled automatic heuristic design by generating and evolving optimization algorithms. However, existing methods mainly search directly in the algorithm space, leaving the direction of exploration largely implicit. This often leads to inefficient search and premature stagnation. In this work, we propose **BeNav**: a behavior-space navigation approach for automatic heuristic design. We represent each algorithm in an interpretable behavior space characterized by search style, information usage, uncertainty, and adaptivity. The behavior space is then used to explicitly navigate algorithm exploration: **behavioral composition** guides the generation of algorithms toward intermediate behaviors between two parent behaviors, while **behavioral diversion** drives the search toward under-explored behavioral regions. These mechanisms are integrated with local exploitation of the current best-performing algorithm to balance exploitation and exploration. Extensive experiments demonstrate that **BeNav** enables more effective search and consistently outperforms state-of-the-art LLM-driven automatic heuristic design methods.

1 Introduction

Designing effective optimization algorithms has traditionally relied on extensive human expertise, requiring researchers to manually develop, modify, and adapt algorithmic strategies for different problems. Recent advances in large language models (LLMs) have opened a new paradigm for automating this process. Algorithm Evolution using Large Language Model (AEL) demonstrated that LLMs can generate and evolve optimization algorithms, while EoH and FunSearch showed that LLM-based program generation coupled with systematic evaluation can discover competitive solutions to challenging mathematical and algorithmic problems (Liu et al., 2023; Romera-Paredes et al., 2024). Subsequent studies have established evolutionary search as an effective paradigm for LLM-driven automatic heuristic design (LLM-AHD), with applications extending to heuristic optimization, hyper-heuristics, scientific discovery, and automated generation of Bayesian optimization algorithms (Liu et al., 2024; Stein & Bäck, 2025; Ye et al., 2024; Zhong et al., 2025; Novikov et al., 2025; Li et al., 2026; Laskaris et al., 2026). These advances demonstrate the potential of LLMs as open-ended generators of algorithmic innovation.

As LLMs become increasingly capable of generating effective algorithms, the challenge is shifting from *how to generate algorithms* to *how to guide their search*. Most existing LLM-AHD methods explore algorithms through iterative generation, evaluation, and selection in the algorithm space. Recent studies have incorporated experience, code structure, prior knowledge, population diversity, and predicted performance to make this search more informed (Ye et al., 2024; Chen et al., 2026; Wu et al., 2026; van Stein et al., 2025a; van Stein et al.; Huang et al., 2026; Dat et al., 2025; Liu et al., 2026; Wang et al., 2026). Although these approaches provide increasingly informative guidance signals, they mainly determine *how candidate algorithms are generated, modified, or selected*. They do not explicitly specify *where in the space of algorithmic behaviors the search should proceed*. This leaves the trajectory of algorithm exploration largely implicit and may result in redundant exploration or premature stagnation.

Algorithm behavior provides a natural basis for making this search direction explicit. Recent studies have shown that the behavior of LLM-generated algorithms contains meaningful information beyond source code and final performance. Behaviour Space Analysis characterizes generated meta-

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.

heuristics using behavioral measures to analyze their differences, while BehaveSim captures algorithm similarity through problem-solving trajectories and highlights the role of behavioral diversity in LLM-AHD (van Stein et al., 2026; Zhang & Lu, 2026). These studies demonstrate that algorithm behavior provides a meaningful representation of automatically generated algorithms. However, behavior is primarily used as a *descriptor, analysis signal, or diversity criterion*, rather than as a mechanism for directing search. Existing studies mainly answer *where generated algorithms are in behavior space*, whereas effective search should also determine *where the next algorithm should be*. This motivates us to turn algorithm behavior from a descriptive property into an explicit coordinate system for search navigation.

Based on this perspective, we propose **BeNav**, a behavior-space navigation approach for LLM-driven automatic heuristic design. **BeNav** represents each candidate algorithm using an interpretable behavioral dimension characterized by four complementary aspects: **search style**, **information usage**, **uncertainty**, and **adaptivity**. This behavioral space provides a semantic dimension system for directing exploration while preserving the open-ended nature of the algorithm space. **BeNav** combines behavior-driven exploration with direct local exploitation of the best-performing algorithm. **Behavioral composition** combines the behavioral dimensions of promising algorithms to define intermediate behavioral targets, which guide the LLM toward generating algorithms with desired behaviors. When the search stagnates, **behavioral diversion** identifies sparse behavioral directions from the current high-performing algorithms and guides the search toward under-explored behavioral regions. Thus, the LLM determines **how** to realize an algorithm, while behavior space determines **where** to search. This separation provides explicit control over the search trajectory without restricting the generative flexibility of LLMs. In summary, our contributions are summarized as follows:

- We introduce **behavior-space navigation** as a new perspective for LLM-driven automatic heuristic design, turning algorithm behavior from a descriptive property into an explicit mechanism for steering algorithm search.
- We develop an interpretable **behavioral dimension system** that characterizes algorithms from four complementary aspects: search style, information usage, uncertainty, and adaptivity.
- We propose **behavioral composition** and **behavioral diversion** to guide LLM generation toward specified and under-explored behavioral regions, while retaining direct local exploitation of the current best-performing algorithm.
- Extensive experiments demonstrate that **BeNav** improves search effectiveness and consistently outperforms state-of-the-art LLM-AHD methods across different algorithm design tasks.

2 Related Work

LLM-Driven automatic heuristic design LLMs have recently emerged as powerful tools for automatic heuristic design, enabling the generation and evolution of executable algorithms with limited human intervention. Early studies established the feasibility of this paradigm. Algorithm Evolution using Large Language Model (AEL) demonstrated the use of LLMs to generate and evolve optimization algorithms (Liu et al., 2023), while FunSearch combined LLM-based program generation with systematic evaluation to discover competitive solutions for mathematical and algorithmic problems (Romera-Paredes et al., 2024). Building on these ideas, Evolution of Heuristics (EoH) integrates the evolution of heuristic ideas and executable implementations into an iterative search process (Liu et al., 2024), and LLaMEA treats LLMs as generative operators within an evolutionary algorithm for automatically constructing metaheuristics (Stein & Bäck, 2025). ReEvo further incorporates reflective feedback into evolutionary search to improve the quality of generated heuristics (Ye et al., 2024). The scope of LLM-AHD has continued to expand beyond conventional heuristic design. LLMOA applies LLMs to hyper-heuristic optimization (Zhong et al., 2025), while AlphaEvolve demonstrates the potential of LLM-based evolutionary search for broader scientific and algorithmic discovery (Novikov et al., 2025). More recent studies have explored the automatic generation of Bayesian optimization algorithms, including LLaMEA-BO and LLM-driven multi-objective Bayesian optimization (Li et al., 2026; Laskaris et al., 2026).

Search Guidance for LLM-AHD A growing body of work has investigated how additional information can be used to guide LLM-driven algorithm search. One direction exploits **experience and feedback** accumulated during the search process. ReEvo uses reflective feedback from previously generated algorithms to guide subsequent evolution (Ye et al., 2024). HiFo-Prompt incorporates hindsight and foresight derived from population dynamics to adapt the generation process (Chen et al., 2026), while RefineEvo introduces planning and bidirectional experience to reuse both successful insights and negative lessons during heuristic evolution (Wu et al., 2026). These methods primarily provide *generation-level guidance*, helping the LLM determine how to modify or refine existing algorithms. Another direction exploits **algorithmic structure and prior knowledge**. Code Evolution Graphs analyze the structural evolution of generated algorithms and provide insights into how algorithmic components change during the search (van Stein et al., 2025a). LLaMEA-SAGE further extracts structural and complexity-related information from generated code and converts informative signals into natural-language guidance for subsequent mutations (van Stein et al.). Strong Priors demonstrates that high-quality existing algorithms can serve as useful priors for LLM-based algorithm generation (Huang et al., 2026). LLaMEA-HPO instead separates algorithmic structure generation from hyper-parameter optimization, reducing the search burden placed on LLM-based generation (van Stein et al., 2025c). These approaches exploit information about existing algorithms or their structures to constrain or bias the search toward potentially promising designs. A third direction exploits **population information and predicted performance**. HSEvo explicitly considers diversity and convergence when evolving LLM-generated heuristics (Dat et al., 2025), while EoH-S evolves complementary sets of heuristics to promote diversity and improve generalization (Liu et al., 2026). DGS introduces surrogate models to predict the outcomes of different parent–operator combinations and uses these predictions to select the next generation action (Wang et al., 2026). BLADE further provides a systematic benchmark for evaluating LLM-AHD methods from multiple perspectives, including search efficiency, generalization, and information exploitation (van Stein et al., 2025b). In addition, a systematic study shows that the evolutionary search strategy itself has a substantial impact on automatic heuristic design (Zhang et al., 2024).

Behavioral Representation and Analysis in LLM-AHD Algorithm behavior has long been studied in meta-heuristics to characterize search dynamics beyond final solution quality. Early studies mainly described behavior through concepts such as exploration, exploitation, intensification, diversification, and convergence. Hayward and Engelbrecht further proposed a set of search behavior characteristics to quantitatively describe meta-heuristic search dynamics, which was later extended to 20 behavioral characteristics for comparing the behavioral similarity of different algorithms (Hayward & Engelbrecht, 2023; 2025). Search trajectory networks provide another data-driven representation by modeling the trajectories generated during algorithm execution, enabling the analysis and comparison of search dynamics across different meta-heuristic paradigms (Ochoa et al., 2021). Recent work has extended behavioral analysis to LLM-driven algorithm discovery. Behaviour Space Analysis characterizes LLM-generated algorithms using dynamic measures such as exploration, exploitation, convergence, and stagnation, while BehaveSim represents algorithms through problem-solving trajectories to measure behavioral similarity and promote behavioral diversity in LLM-AHD (van Stein et al., 2026; Zhang & Lu, 2026). These studies demonstrate that behavioral representations provide useful information for understanding, comparing, and diversifying automatically generated algorithms. However, existing approaches primarily characterize *observed behavior* from algorithm execution, rather than constructing an explicit and interpretable behavioral coordinate system that can be specified before generation to guide the search for new algorithms.

3 METHODOLOGY

3.1 OVERVIEW

We formulate LLM-based automatic heuristic design as an iterative evolutionary search process, where candidate algorithms are repeatedly generated, evaluated, and evolved. Existing methods improve this process by leveraging prompts, reflections, historical experience, and other search information to generate new algorithms. However, generating a new algorithm does not necessarily imply moving to a new region of the algorithm search space. In particular, successive generations may remain behaviorally similar even when their implementations differ, causing the search to repeatedly refine similar behaviors without explicitly exploring alternative behavioral regions.

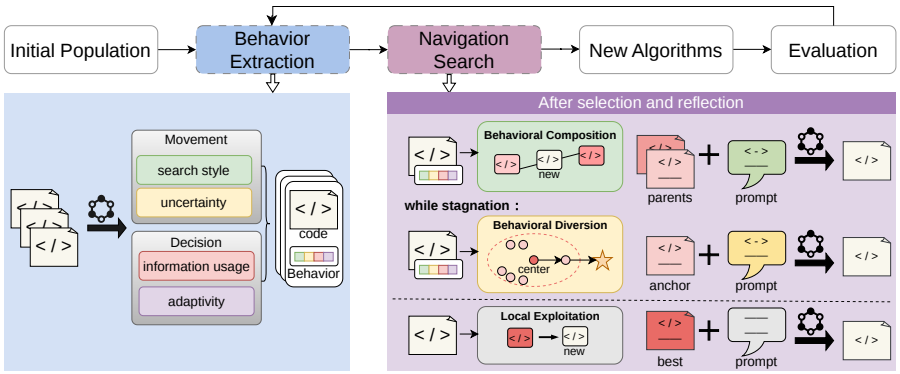


Figure 1: Overall framework of **BeNav**. Candidate algorithms are first mapped into the behavioral space through **behavior extraction**. Based on the current population, **BeNav** generates new algorithms through **Navigation Search**, which consists of three complementary mechanisms: **behavioral composition** between promising algorithms, **behavioral diversion** toward under-explored behavioral regions, and **local exploitation** of the current best-performing algorithms. The generated algorithms are then evaluated and incorporated into the population for the next iteration.

To address this limitation, we propose **BeNav**, a behavior-space navigation approach for LLM-based automatic heuristic design. As illustrated in Figure 1, **BeNav** first maps candidate algorithms into a quantitative behavioral space through **behavior extraction**, and then uses **Navigation Search** to determine the direction of subsequent algorithm generation. The resulting algorithms are evaluated on the training problems and stored in the archive, whose performance and behavioral information are used to guide the search in the next iteration. In this way, **BeNav** forms an iterative loop that continuously links algorithm evaluation with behavioral navigation. Overall, **BeNav** consists of two key components: **Behavior Extraction**, which constructs the behavioral representation of candidate algorithms, and **Navigation Search**, which uses this representation to guide the generation of new algorithms. The two components are described in detail below.

3.2 Behavior Extraction

Behavioral Space To navigate the search over algorithms, **BeNav** represents each candidate algorithm by its behavioral characteristics rather than by its source code or performance alone. For each candidate algorithm A_i, we define a behavioral coordinate

$$\mathbf{B}_i = [b_{i,1}, b_{i,2}, \ldots, b_{i,n}] \in [0,1]^n, \tag{1}$$

where each dimension corresponds to an interpretable characteristic of algorithmic search behavior. In this work, we use four dimensions to construct the behavioral space. The coordinate does not aim to reconstruct the complete algorithmic mechanism. Instead, it provides a compact representation of *how an algorithm searches and makes decisions*, allowing algorithms with different implementations to occupy similar behavioral positions when they exhibit similar search characteristics.An overview of this behavior extraction process is illustrated in Figure 2.

We organize these dimensions into two complementary aspects: **Decision** and **Movement**. **Decision** describes what information drives search decisions and how the strategy adapts, whereas **Movement** describes how these decisions translate into search movements. This decomposition provides a structured view of both the decision-making basis and the resulting search dynamics.

Behavioral Dimensions Under this decomposition, the four behavioral dimensions are defined as follows:

- **Decision:** *Information Usage* measures the information exploited when making search decisions, ranging from fixed heuristic rules to information derived from the current search state, historical experience, or previously observed outcomes. *Adaptivity* measures the extent to which the search strategy changes in response to such information and search feedback, ranging from relatively fixed strategies to dynamically adjusted ones.

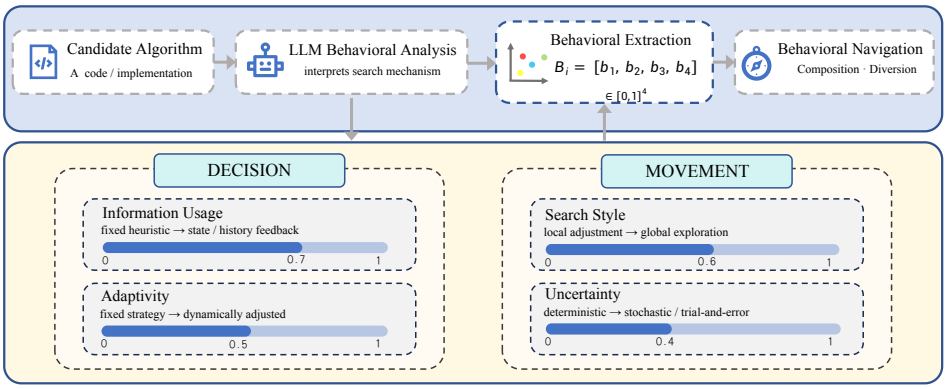


Figure 2: Behavior extraction in **BeNav**. A candidate algorithm is first analyzed by the LLM to identify its underlying search mechanisms, which are then mapped into a four-dimensional behavioral representation $\mathbf{B}_i = [b_1, b_2, b_3, b_4]$. The four behavioral dimensions characterize two complementary aspects of algorithmic behavior: **Movement**, including search style and uncertainty, and **Decision**, including information usage and adaptivity. Each dimension is represented by a continuous coordinate in $[0, 1]$, providing a compact behavioral description that is subsequently used to support behavioral navigation.

- **Movement:** *Search Style* characterizes the range of search movements induced by the algorithm, ranging from predominantly local adjustments to broader and more global exploration. *Uncertainty* characterizes the determinacy of these movements and the extent to which stochastic sampling, perturbation, and trial-and-error mechanisms affect the search process.

Together, these four dimensions capture *what information drives decisions, how the strategy adapts, how broadly the search moves, and how deterministically it moves*. For each candidate algorithm, an LLM analyzes its implementation and underlying search mechanisms along these dimensions and assigns a normalized score in $[0, 1]$, yielding its behavioral coordinate $\mathbf{B}_i$. The coordinate is stored together with the algorithm and its evaluation result and is subsequently used for behavioral navigation.

Behavioral Coordinate for Navigation The key role of this representation is to transform qualitative search characteristics into an explicit coordinate that can be manipulated during algorithm generation. Unlike execution-based behavioral descriptors, which are obtained by observing an algorithm's search trajectory, our representation is inferred from its underlying algorithmic mechanisms and can therefore be obtained before evaluating a newly generated algorithm. More importantly, the behavioral coordinate is not merely a descriptor of existing algorithms; it can serve as an explicit behavioral target for generating new ones. Specifically, behavioral coordinates can be combined to define intermediate behavioral targets or used to identify under-explored directions in the current population, providing the basis for the **Behavioral Composition** and **Behavioral Diversion** mechanisms introduced in the next section.

3.3 Navigation Search

BeNav introduces behavioral navigation to guide evolutionary search by explicitly controlling how generated algorithms move across and within behavioral regions. Rather than relying solely on fitness to determine the direction of search, BeNav incorporates behavioral information into the generation process to support different forms of search movement. This navigation is realized through three complementary mechanisms: **Behavioral Composition**, which combines existing behavioral patterns to transition between known behavioral regions; **Behavioral Diversion**, which expands the search from promising behavioral regions toward nearby under-explored behaviors; and **Local Exploitation**, which locally refines the current best-performing algorithm using its effective search strategy and related algorithmic mechanisms. Together, these mechanisms support behavioral transition, controlled behavioral expansion, and local refinement within promising regions.

Behavioral Composition At each evolutionary iteration, BeNav applies Behavioral Composition to generate new algorithms by combining two parent algorithms with different fitness levels. Specifically, two parent algorithms A_a and A_b are selected, where A_a denotes the better-performing parent and A_b denotes the other parent. Let $\mathbf{B}_a$ and $\mathbf{B}_b$ denote their behavioral representations, respectively. The source code and behavioral representations of the two parents are provided to the LLM as references. Based on their behavioral representations, the prompt guides the LLM to generate an offspring whose behavior lies between those of the two parents, which can be conceptually expressed as

$$\mathbf{B}_o \approx \alpha \mathbf{B}_a + (1-\alpha)\mathbf{B}_b, \qquad \alpha \in [0,1], \tag{2}$$

where $\mathbf{B}_o$ denotes the behavioral representation of the generated offspring. This equation describes the intended search target of Behavioral Composition rather than an explicitly computed target, since α is not predefined or optimized during the search.

The behavioral representations of the two parents provide the LLM with information about the desired behavioral direction, while their algorithmic implementations provide concrete mechanisms that can be transferred and combined. The LLM treats one parent as the main structure while selectively incorporating useful mechanisms from the other parent, generating a new algorithm that integrates complementary search characteristics. This allows the offspring to inherit useful properties from both parents without simply reproducing either existing algorithm. Through this process, Behavioral Composition constructs new search behaviors from previously discovered ones, enabling transitions between behavioral regions while retaining useful algorithmic structures.

Behavioral Diversion When the search becomes stagnant, BeNav applies Behavioral Diversion to expand the search beyond the current promising behavioral region. Diversion first selects a high-quality behavioral region as the starting point and then identifies promising frontier positions around this region toward under-explored areas. The selected frontier position is subsequently used to guide the generation of new algorithms, encouraging the search to move beyond the currently explored behavioral region.

Let $\mathcal{P}$ denote the pool of behavioral records accumulated from evaluated algorithms, and let $\mathbf{B}_i$ denote the behavioral representation of the i-th record in $\mathcal{P}$. We select the top-k_e records according to fitness to form an elite collection $\mathcal{E}$, and compute their behavioral centroid as

$$\mathbf{B}_c = \frac{1}{|\mathcal{E}|} \sum_{\mathbf{B}_i \in \mathcal{E}} \mathbf{B}_i. \tag{3}$$

We then select a frontier candidate $\mathbf{B}_f$ within a predefined neighborhood of the elite centroid. Specifically, for each $\mathbf{B}_i$ in $\mathcal{P}$ that falls within this neighborhood, we evaluate it by combining three factors: its local sparsity, measured as the average distance to its k_n nearest records; its normalized distance from the elite centroid; and the quality of the corresponding algorithm. This selection favors behavioral positions that remain close to high-quality regions while being relatively under-explored and located toward their boundaries.

Starting from the selected frontier candidate, BeNav expands the search outward from the elite region. The expansion direction is determined by the vector from the elite centroid to the frontier candidate:

$$\mathbf{d} = \frac{\mathbf{B}_f - \mathbf{B}_c}{\|\mathbf{B}_f - \mathbf{B}_c\|_2}. \tag{4}$$

A target behavioral position is then constructed through a controlled step along this direction with a stochastic perturbation:

$$\mathbf{B}_{\text{target}} = \text{clip}\left(\mathbf{B}_f + \delta \mathbf{d} + \boldsymbol{\epsilon},\, 0,\, 1\right), \tag{5}$$

where δ denotes the expansion step and $\boldsymbol{\epsilon}$ denotes the perturbation. Both quantities are adapted to the behavioral sparsity of the current search: the expansion step increases in sparser regions, whereas the perturbation magnitude increases in denser regions.

The target behavioral position specifies where the search should move in behavioral space, rather than prescribing a specific algorithmic implementation. Since a behavioral position alone does not provide the LLM with a concrete algorithmic starting point, BeNav retrieves an existing feasible algorithm whose behavioral representation is closest to $\mathbf{B}_{\text{target}}$ as a reference. The LLM is then

Table 1: Performance comparison on MKP-ACO and CVRP-ACO. The results are averaged over three independent runs. The best results are highlighted in bold.

	MKP-ACO			CVRP-ACO		
Method	$n = 100 \uparrow$	$n = 300 \uparrow$	$n = 500 \uparrow$	$n = 20 \downarrow$	$n = 50 \downarrow$	$n = 100 \downarrow$
EoH	23.086±0.12	61.062±0.81	101.152±1.61	**4.820±0.08**	9.500±0.22	16.998±0.58
ReEvo	23.098±0.12	61.258±0.65	101.691±1.24	5.355±0.24	9.809±0.14	16.874±0.25
HSEvo	23.227±0.01	61.932±0.15	102.830±0.17	5.196±0.21	9.750±0.29	16.949±0.40
MCTS	23.202±0.09	61.604±0.94	102.109±1.93	5.537±0.18	9.885±0.12	16.775±0.80
HiFo-Prompt	23.097±0.08	61.388±0.66	102.085±1.40	4.840±0.08	9.374±0.11	16.450±0.07
BeNav	**23.229±0.02**	**62.067±0.20**	**103.082±0.48**	4.830±0.03	**9.295±0.07**	**16.282±0.09**

provided with both the target behavioral position and the reference algorithm, and is instructed to generate new algorithms that move from the reference toward the target. In this way, Behavioral Diversion enables controlled exploration toward alternative behavioral regions while remaining grounded in existing feasible algorithms, rather than encouraging unconstrained behavioral novelty.

Local Exploitation At each evolutionary iteration, BeNav applies Local Exploitation to further refine the current best-performing algorithm. The current best algorithm serves as the primary reference for generation. To provide additional refinement information, BeNav retrieves the most behaviorally similar algorithm from the library based on the behavioral representation of the current best algorithm. Specifically, let $\mathbf{B}_e$ denote the behavioral representation of the current best-performing algorithm. The supplementary reference algorithm is selected as

$$A^* = \arg \min_{A_j \in \mathcal{L}^-} \left(1 - \frac{\mathbf{B}_e^\top \mathbf{B}_j}{\|\mathbf{B}_e\|_2 \|\mathbf{B}_j\|_2}\right), \tag{6}$$

where $\mathbf{B}_j$ denotes the behavioral representation of candidate algorithm A_j, and $\mathcal{L}^-$ denotes the library after excluding the current best-performing algorithm. The semantic description of A^* is then provided to the LLM together with the code of the current best algorithm.

Given these references, the LLM performs targeted modifications to the current best algorithm while preserving its overall search strategy. The modifications focus on refining existing mechanisms or introducing lightweight local optimization components, rather than rewriting the entire algorithm. The generated algorithms are subsequently evaluated and incorporated into the population for the next stage of evolutionary search.

4 EXPERIMENTS

4.1 EXPERIMENTAL SETTINGS

We evaluate BeNav on four automatic heuristic design tasks, covering different optimization problems and algorithmic frameworks: MKP with Ant Colony Optimization (ACO), CVRP with ACO, TSP with Guided Local Search (GLS), and TSP with a constructive heuristic. Detailed descriptions of the optimization problems, benchmark instances, and algorithmic frameworks are provided in Appendix A.

We compare BeNav with five representative LLM-based automatic heuristic design methods, including EoH(Liu et al., 2024), ReEvo(Ye et al., 2024), HSEvo(Dat et al., 2025), MCTS(Zheng et al., 2025), and HiFo-Prompt(Chen et al., 2026). All methods are evaluated under the same experimental setting and use the locally deployed Qwen3-Coder-30B as the underlying LLM. We set the population size to 10 and allow at most 500 algorithm evaluations for each independent run. For each task and method, we conduct three independent runs to reduce the influence of stochastic variation. The best-performing algorithm obtained during each run is selected for subsequent evaluation on the test instances.

Table 2: Performance comparison on TSP with constructive heuristic. The results are averaged over three independent runs. The best results are highlighted in bold.

	TSP-Constructive		
Method	$n = 20 \downarrow$	$n = 50 \downarrow$	$n = 100 \downarrow$
EoH	4.1449±0.0203	6.2174±0.0297	8.6412±0.0356
ReEvo	4.1418±0.0402	6.1900±0.0384	8.6376±0.0385
HSEvo	4.1516±0.0166	6.2341±0.0205	8.6507±0.0828
MCTS	4.2251±0.0681	6.3717±0.1354	8.9464±0.2609
HiFo-Prompt	4.2169±0.0510	6.3284±0.1055	8.7419±0.1307
BeNav	**4.1119±0.0192**	**6.1507±0.0401**	**8.6305±0.0459**

Table 3: Performance comparison on TSP with guided local search (GLS). The results are averaged over three independent runs. The best results are highlighted in bold.

	TSP-GLS			
Method	$n = 100 \downarrow$	$n = 200 \downarrow$	$n = 300 \downarrow$	$n = 500 \downarrow$
EoH	7.8084±0.0004	10.7255±0.0042	13.0169±0.0082	16.6739±0.0097
ReEvo	7.8085±0.0002	10.7319±0.0118	13.0326±0.0257	16.6992±0.0531
HSEvo	7.8084±0.0001	10.7351±0.0075	13.0462±0.0267	16.7212±0.0474
MCTS	7.8102±0.0025	10.7363±0.0128	13.0348±0.0216	16.7106±0.0399
HiFo-Prompt	7.8248±0.0125	10.7809±0.0506	13.1496±0.0759	16.9869±0.1032
BeNav	**7.8083±0.0001**	**10.7253±0.0026**	**13.0147±0.0040**	**16.6689±0.0060**

4.2 Overall Performance

We evaluate BeNav on four automatic heuristic design tasks covering different optimization problems and heuristic frameworks. The evaluation aims to examine whether BeNav can consistently discover high-quality algorithms compared with existing LLM-based heuristic design methods. Overall, BeNav achieves competitive or superior performance across the evaluated benchmarks.

On the ACO-based tasks, BeNav achieves the best performance across all three problem scales on MKP-ACO, outperforming the strongest baseline on each instance size, as reported in Table 1. On CVRP-ACO, BeNav achieves the best performance on the medium- and large-scale instances, while remaining competitive with the baselines on the small-scale instance. The performance gap in favor of BeNav also becomes more pronounced as the instance size increases.

Beyond the ACO-based tasks, BeNav achieves strong performance under both the constructive heuristic and GLS frameworks for TSP. On TSP-Constructive, BeNav consistently obtains lower objective values than the compared methods across all three problem sizes, as shown in Table 2, demonstrating that the effectiveness of behavioral navigation is not limited to ACO-based heuristic design. On TSP-GLS, BeNav also achieves the best average results across all four evaluated scales, with relatively small standard deviations over the three independent runs, as shown in Table 3, indicating consistent performance across different test-instance sizes.

To further examine whether the search moves beyond a fixed behavioral region, we cluster the behavioral representations of generated algorithms using DBSCAN and track the proportion of algorithms belonging to each behavioral cluster over the search process, with the best individual in each stage marked as the elite. As shown in Figure 3, the population gradually shifts among different behavioral regions, and the best individuals observed during the search are associated with different regions at different stages. This provides behavioral evidence that BeNav can move the search toward alternative behavioral regions rather than repeatedly remaining in the same region. The corresponding behavioral clustering results for the baseline methods are provided in Appendix B.3.

4.3 Ablation Study

We investigate the effects of behavioral representation and navigation components, with the results summarized in Table 4. For behavioral representation, the four-dimensional variant performs best

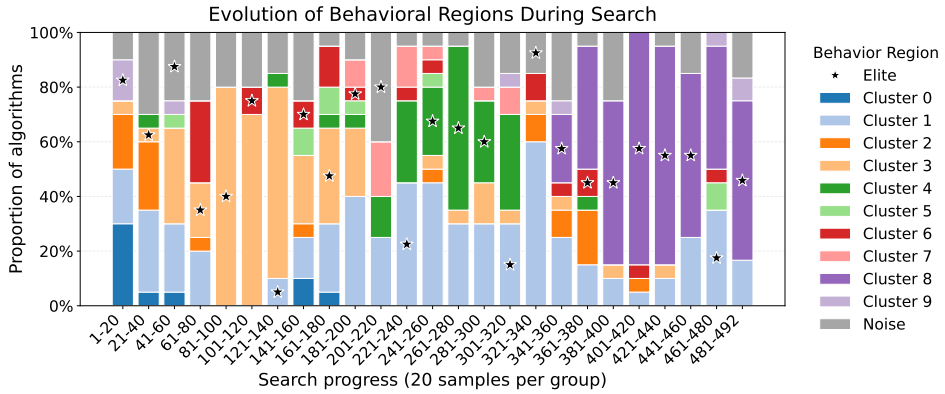


Figure 3: Evolution of behavioral clusters during the BeNav search on CVRP-ACO.

Table 4: Ablation study of behavioral representation and navigation components in BeNav on MKP-ACO. The upper part evaluates different behavioral representations, while the lower part examines individual navigation components.

Method	MKP-ACO n=100↑	n=300↑	n=500↑
	Behavioral Representation		
2D Behavioral Representation (-SS,-IU)	23.192±0.07	61.750±0.54	102.417±1.07
3D Behavioral Representation (-AD)	23.221±0.02	62.036±0.30	**103.118±0.74**
	Navigation Components		
w/o Composition	23.210±0.08	61.608±1.06	102.134±2.19
w/o Exploitation	23.142±0.08	60.963±0.27	100.966±0.38
w/o Diversion	23.192±0.06	61.526±0.61	102.169±1.06
BeNav	**23.229±0.02**	**62.067±0.20**	103.082±0.48

on small- and medium-scale instances, while some three-dimensional variants perform better on larger instances. The complete results of the two- and three-dimensional variants are provided in Appendix B.1. These results suggest that higher-dimensional representations provide finer-grained behavioral information for more precise navigation, whereas fewer dimensions may offer greater flexibility under distribution changes. Thus, overly detailed behavioral representations may potentially limit generalization. Meanwhile, the three- and four-dimensional variants generally outperform the two-dimensional variant, indicating that excessively reducing the representation may provide insufficient information for effective navigation. These observations motivate our use of four behavioral dimensions.

We next examine the three navigation components by removing Behavioral Composition, Local Exploitation, and Behavioral Diversion individually. Removing any component degrades performance on at least one scale, with Local Exploitation showing the most consistent impact across the three scales. Behavioral Composition also improves performance across scales, while Behavioral Diversion is particularly beneficial on medium- and large-scale instances. These results support the complementary roles of the three components in local refinement, behavioral combination, and expansion toward alternative behavioral regions.

5 Conclusion

We introduced **BeNav**, a behavior-guided evolutionary framework for LLM-driven automatic heuristic design that explicitly represents and navigates algorithm behaviors during search. Extensive experiments across diverse optimization problems and heuristic frameworks demonstrate the effectiveness of behavioral navigation, while ablation studies verify the complementary roles of its key components. These results highlight the potential of explicit behavioral guidance for organizing and directing the search process in LLM-driven automatic heuristic design.

AI USE STATEMENT

In this work, we used generative AI tools to assist with research ideation and execution, including brainstorming and refining research ideas, discussing methodological and experimental designs, assisting with implementation and debugging of experimental code, and analyzing and interpreting experimental results. We also used generative AI tools for retrieval and discovery of related work and for improving the clarity and readability of the manuscript. We did not use generative AI tools to generate synthetic datasets, formulate or prove mathematical claims, or replace human judgment in evaluating the validity of the proposed method and experimental results.

All AI-assisted work was reviewed and verified by the authors. AI-assisted code was checked and tested by the authors, and research ideas, methodological decisions, experimental results, and scientific claims were independently examined by the authors. The authors take full responsibility for the final content of this work, including all claims, code, and other artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

We provide detailed descriptions of the BeNav framework, behavioral representation, and navigation components in Sections 3. The experimental settings and evaluation protocol are described in Section 4, while detailed benchmark descriptions and additional experimental results are provided in the Appendix A and Appendix B. The main performance results are averaged over three independent runs under a consistent evaluation budget. The implementation details and supplementary experimental results provided in the manuscript and Appendix are intended to facilitate reproduction of the reported experiments.

REFERENCES

Chentong Chen, Mengyuan Zhong, Ye Fan, Jialong Shi, and Jianyong Sun. Hifo-prompt: Prompting with hindsight and foresight for llm-based automatic heuristic design. In *International Conference on Learning Representations*, 2026.

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

Lauren Hayward and Andries P. Engelbrecht. How to tell a fish from a bee: Constructing metaheuristic search behaviour characteristics. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pp. 1562–1569, 2023. doi: 10.1145/3583133.3596338.

Lauren Hayward and Andries P. Engelbrecht. Determining metaheuristic similarity using behavioral analysis. *IEEE Transactions on Evolutionary Computation*, 29(1):262–274, 2025. doi: 10.1109/TEVC.2023.3346672.

Qi Huang, Furong Ye, Ananta Shahane, Thomas Bäck, and Niki van Stein. From heuristic selection to automated algorithm design: Llms benefit from strong priors. *arXiv preprint arXiv:2603.02792*, 2026. doi: 10.48550/arXiv.2603.02792.

Georgios Laskaris, Reuben Brasher, Niki van Stein, Elena Raponi, Thomas Bäck, and Florian Neukart. Llm-driven evolutionary generation of multi-objective bayesian optimization algorithms. *arXiv preprint arXiv:2607.08791*, 2026. doi: 10.48550/arXiv.2607.08791.

Wenhu Li, Niki van Stein, Thomas Bäck, and Elena Raponi. Llamea-bo: A large language model evolutionary algorithm for automatically generating bayesian optimization algorithms. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 909–918. Association for Computing Machinery, 2026. doi: 10.1145/3795095.3805059.

Fei Liu, Xialiang Tong, Mingxuan Yuan, and Qingfu Zhang. Algorithm evolution using large language model. *arXiv preprint arXiv:2311.15249*, 2023. doi: 10.48550/arXiv.2311.15249.

Fei Liu, Tong Xialiang, Mingxuan Yuan, Xi Lin, Fu Luo, Zhenkun Wang, Zhichao Lu, and Qingfu Zhang. Evolution of heuristics: Towards efficient automatic algorithm design using large language model. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 32201–32223. PMLR, 2024.

Fei Liu, Yilu Liu, Qingfu Zhang, Xialiang Tong, and Mingxuan Yuan. Eoh-s: Evolution of heuristic set using llms for automated heuristic design. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pp. 37090–37098, 2026. doi: 10.1609/aaai.v40i43.41038.

Alexander Novikov, Ngân Vũ, Marvin Eisenberger, Emilien Dupont, Po-Sen Huang, Adam Zsolt Wagner, Sergey Shirobokov, Borislav Kozlovskii, Francisco J. R. Ruiz, Abbas Mehrabian, M. Pawan Kumar, Abigail See, Swarat Chaudhuri, George Holland, Alex Davies, Sebastian Nowozin, Pushmeet Kohli, and Matej Balog. Alphaevolve: A coding agent for scientific and algorithmic discovery. *arXiv preprint arXiv:2506.13131*, 2025. doi: 10.48550/arXiv.2506.13131.

Gabriela Ochoa, Katherine M. Malan, and Christian Blum. Search trajectory networks: A tool for analysing and visualising the behaviour of metaheuristics. *Applied Soft Computing*, 109:107492, 2021. doi: 10.1016/j.asoc.2021.107492.

Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Matej Balog, M. Pawan Kumar, Emilien Dupont, Francisco J. R. Ruiz, Jordan S. Ellenberg, Pengming Wang, Omar Fawzi, Pushmeet Kohli, and Alhussein Fawzi. Mathematical discoveries from program search with large language models. *Nature*, 625(7995):468–475, 2024. doi: 10.1038/s41586-023-06924-6.

Niki van Stein and Thomas Bäck. Llamea: A large language model evolutionary algorithm for automatically generating metaheuristics. *IEEE Transactions on Evolutionary Computation*, 29 (2):331–345, 2025. doi: 10.1109/TEVC.2024.3497793.

Niki van Stein, Anna V. Kononova, Lars Kotthoff, and Thomas Bäck. Llamea-sage: Guiding automated algorithm design with structural feedback from explainable ai.

Niki van Stein, Anna V. Kononova, Lars Kotthoff, and Thomas Bäck. Code evolution graphs: Understanding large language model driven design of algorithms. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 943–951. Association for Computing Machinery, 2025a. doi: 10.1145/3712256.3726328.

Niki van Stein, Anna V. Kononova, Haoran Yin, and Thomas Bäck. Blade: Benchmark suite for llm-driven automated design and evolution of iterative optimisation heuristics. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, GECCO '25 Companion, pp. 2336–2344, New York, NY, USA, 2025b. Association for Computing Machinery. doi: 10.1145/3712255.3734347.

Niki van Stein, Diederick Vermetten, and Thomas Bäck. In-the-loop hyper-parameter optimization for llm-based automated design of heuristics. *ACM Transactions on Evolutionary Learning and Optimization*, 2025c. doi: 10.1145/3731567.

Niki van Stein, Haoran Yin, Anna V. Kononova, Thomas Bäck, and Gabriela Ochoa. Behaviour space analysis of llm-driven meta-heuristic discovery. In *Computational Intelligence*, volume 2828 of *Communications in Computer and Information Science*, pp. 367–385. Springer, 2026. doi: 10.1007/978-3-032-15635-8_23.

Yuhan Wang, Chaoda Peng, Xingyu Wu, Sheng-Hao Wu, and Zhi-Hui Zhan. How to guide llm generation: Dual-surrogate guided search for automated heuristic design. *arXiv preprint arXiv:2607.13911*, 2026. doi: 10.48550/arXiv.2607.13911.

Yang Wu, Junran Pan, Yifan Zhang, Ning Xu, Fanshuo Zeng, and Jian Cheng. Refineevo: Planning-guided heuristic evolution with bidirectional experience. In *Forty-third International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=fZ6iuiHDqf.

Haoran Ye, Jiarui Wang, Zhiguang Cao, Federico Berto, Chuanbo Hua, Haeyeon Kim, Jinkyoo Park, and Guojie Song. Reevo: Large language models as hyper-heuristics with reflective evolution. In *Advances in Neural Information Processing Systems*, volume 37, pp. 43571–43608. Curran Associates, Inc., 2024.

Rui Zhang and Zhichao Lu. Rethinking code similarity for automated algorithm design with llms. In *The Fourteenth International Conference on Learning Representations*, 2026.

Rui Zhang, Fei Liu, Xi Lin, Zhenkun Wang, Zhichao Lu, and Qingfu Zhang. Understanding the importance of evolutionary search in automated heuristic design with large language models. In *Parallel Problem Solving from Nature – PPSN XVIII*, volume 15149 of *Lecture Notes in Computer Science*, pp. 185–202. Springer, 2024. doi: 10.1007/978-3-031-70068-2_12.

Zhi Zheng, Zhuoliang Xie, Zhenkun Wang, and Bryan Hooi. Monte carlo tree search for comprehensive exploration in LLM-based automatic heuristic design. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267, pp. 78338–78373. PMLR, 2025. URL `https://proceedings.mlr.press/v267/zheng25o.html`.

Rui Zhong, Abdelazim G. Hussien, Jun Yu, and Masaharu Munetomo. Llmoa: A novel large language model assisted hyper-heuristic optimization algorithm. *Advanced Engineering Informatics*, 64:103042, 2025. doi: 10.1016/j.aei.2024.103042.

A Experiments Details

Benchmarks We evaluate BeNav on four combinatorial optimization (CO) problems spanning three distinct algorithmic paradigms:

- MKP-ACO — the Multi-dimensional Knapsack Problem solved via Ant Colony Optimization. A learned heuristic produces pheromone-weighting coefficients and the objective is to maximize total profit ($n = 100$ items, $m = 5$ constraints; 50 ACO iterations, 10 ants).
- CVRP-ACO — the Capacitated Vehicle Routing Problem under ACO. The heuristic guides ant decisions subject to a vehicle capacity of 50 (training size $n = 50$; 100 ACO iterations, 30 ants).
- TSP-GLS — the Traveling Salesman Problem tackled by Guided Local Search, where the learned heuristic defines a penalty-augmented cost matrix (training size $n = 100$; 30 perturbation moves, 1200 search iterations).
- TSP-Constructive — a learned `select_next_node` policy that greedily constructs a tour node-by-node (training size $n = 50$).

During evolution, each candidate heuristic is evaluated on the training split, and its fitness is the mean objective over all training instances (MKP-ACO: 5 instances; CVRP-ACO and TSP-GLS: 10; TSP-Constructive: 64). All problem instances are randomly generated under standard protocols: for MKP-ACO, item prizes and weights are drawn uniformly in $[0, 1]$ with 5 capacity constraints; for CVRP-ACO, customer locations are sampled uniformly in $[0, 1]^2$ with integer demands in $\{1, \ldots, 9\}$ and a central depot at $[0.5, 0.5]$; for both TSP variants, node coordinates are sampled uniformly in $[0, 1]^2$.

Final results are reported on held-out evaluation datasets of 64 instances per problem size, across multiple sizes to assess both solution quality and size generalization: MKP-ACO on $\{100, 300, 500\}$, CVRP-ACO on $\{20, 50, 100\}$, TSP-GLS on $\{100, 200, 300, 500\}$, and TSP-Constructive on $\{20, 50, 100\}$.

Implementation Details The generator and reflector share a single Qwen3-Coder-30B model served locally via Ollama at a sampling temperature of 1.0; the same LLM performs strategy-unit extraction.

Baselines We compare against five representative LLM-based automatic heuristic design methods: EoH (Liu et al., 2024), ReEvo (Ye et al., 2024), HSEvo (Dat et al., 2025), MCTS (Zheng et al., 2025), and HiFo-Prompt (Chen et al., 2026). For fairness, all baselines share the same Qwen3-Coder-30B backbone, the same problem formulations and datasets, and the same 500-FE evaluation budget.

Evaluation Protocol Following prior LLM-driven heuristic evolution work, we adopt a train/generalization protocol: heuristics are evolved solely on the training split, and the best-of-evolution heuristic is then evaluated, without any further tuning, on the held-out 64-instance-per-size datasets described above. Each configuration is independently run 3 times with different random seeds; we report the mean and standard deviation across runs.

B Additional Experimental Results

This section provides additional experimental results that complement the main experiments. We report detailed results on behavioral dimensionality, evaluations with different backbone LLMs, and behavioral clustering results for BeNav and the baseline methods.

B.1 Additional Results on Behavioral Dimensions

Table 5: Ablation study of behavioral dimensions on MKP-ACO. SS, IU, UN, and AD denote Search Style, Information Usage, Uncertainty, and Adaptivity, respectively.

	MKP-ACO		
Method	$n = 100 \uparrow$	$n = 300 \uparrow$	$n = 500 \uparrow$
2D ($-$SS, $-$IU)	23.192±0.07	61.750±0.54	102.417±1.07
2D ($-$SS, $-$UN)	23.160±0.10	61.440±1.16	101.976±2.27
2D ($-$SS, $-$AD)	23.111±0.13	61.227±1.28	101.546±2.48
2D ($-$IU, $-$UN)	23.215±0.07	61.766±0.51	102.471±0.98
2D ($-$IU, $-$AD)	23.077±0.03	60.287±0.39	99.796±0.85
2D ($-$UN, $-$AD)	23.082±0.19	60.750±1.73	100.679±3.09
3D ($-$SS)	23.113±0.03	61.129±0.38	101.293±0.72
3D ($-$IU)	23.182±0.03	61.964±0.35	102.741±1.26
3D ($-$UN)	23.198±0.03	61.714±0.36	102.426±0.86
3D ($-$AD)	23.221±0.02	62.036±0.30	**103.118±0.74**
BeNav	**23.229±0.02**	**62.067±0.20**	103.082±0.48

The complete results of the behavioral dimension ablation on MKP-ACO are reported in Table 5. We evaluate all six possible two-dimensional combinations and four three-dimensional variants, providing a more detailed comparison than the aggregated results presented in the main paper. The results show that the effect of removing different behavioral dimensions varies across problem scales. In particular, the four-dimensional representation generally provides strong performance on small- and medium-scale instances, while some three-dimensional variants achieve slightly better results on the largest instances. These results provide additional evidence for the trade-off between behavioral granularity and the flexibility of the resulting behavioral representation.

B.2 Results with Different Backbone LLMs

Table 6: Performance comparison of BeNav with different backbone LLMs on MKP-ACO. The results are averaged over three independent runs.

	MKP-ACO		
Method	$n = 100 \uparrow$	$n = 300 \uparrow$	$n = 500 \uparrow$
BeNav (GPT-OSS-20B)	23.123±0.11	61.034±0.70	101.116±1.42
BeNav (Llama-3-8B)	23.153±0.09	61.205±0.83	101.597±1.54
BeNav (DeepSeek-R1-8B)	23.113±0.19	60.817±1.55	100.842±2.99
BeNav (Qwen3-Coder-30B)	**23.229±0.02**	**62.067±0.20**	**103.082±0.48**

To examine the performance of BeNav with different backbone LLMs, we additionally evaluate the framework using several alternative models on MKP-ACO. The corresponding results are reported in Table 6. BeNav maintains effective performance across the evaluated backbone models, although

the absolute performance varies between models. These results provide an additional robustness check of the proposed behavioral navigation framework under different LLM backbones.

B.3 Behavioral Clustering Results

To further examine the behavioral evolution of different methods, we apply the same behavioral clustering procedure to the algorithms generated by BeNav and the baseline methods on CVRP-ACO. Figure 4 presents the corresponding behavioral clustering results for BeNav, ReEvo, HSEvo, and EoH. The same behavioral representation and clustering procedure are used across methods to ensure a consistent comparison. The results show that the composition of behavioral regions changes throughout the search process, with different behavioral regions becoming dominant at different stages. For BeNav, the generated algorithms progressively shift among different behavioral regions rather than remaining concentrated in a fixed region. In comparison, the baseline methods exhibit different patterns of behavioral concentration and transition. These results provide additional behavioral evidence for the navigation mechanism of BeNav and complement the behavioral evolution analysis presented in the main paper.

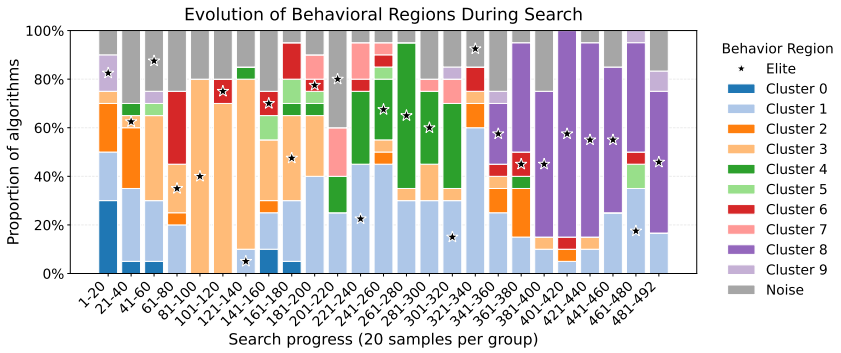
C Prompts Details

C.1 Prompt Design Rationale

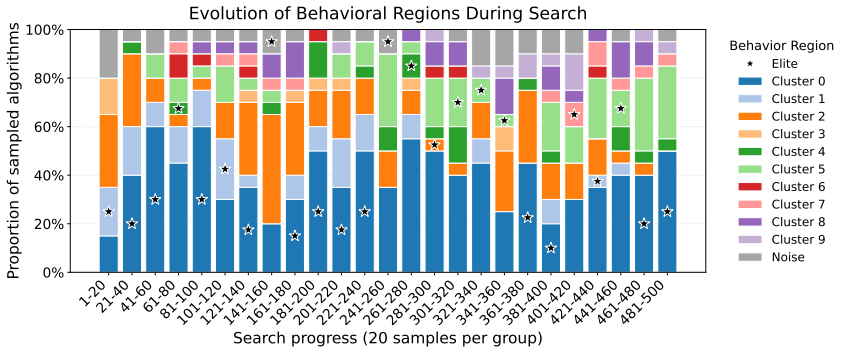
The prompts used by BeNav are designed according to the range of behavioral change required by different navigation mechanisms. BeNav treats both the algorithmic implementation and its behavioral representation as complementary sources of information for LLM-guided generation. Specifically, BeNav first maps each generated algorithm into a quantitative behavioral space, where its coordinates describe the current position of its search behavior. Based on these representations and existing algorithms, BeNav determines different navigation directions according to the behavioral relationships among candidate algorithms, corresponding to local refinement, intermediate behavioral transition, and longer-range expansion. The resulting navigation information, together with relevant algorithmic implementations, is provided to the LLM to generate new algorithms.

Different navigation mechanisms therefore use different forms of behavioral information. Behavioral Composition provides both parent algorithms and their behavioral representations, as the desired transition is defined by the relationship between two existing algorithms. For Behavioral Diversion, the intended behavioral change is larger, and the target region may be far from the provided reference algorithm, making the reference algorithm less informative for determining how the search behavior should change. Moreover, the coarse-grained $[0, 1]$ interpretation used during behavior extraction may provide insufficient semantic resolution for such a large behavioral transition. Therefore, Behavioral Diversion further provides finer-grained qualitative ranges for the target behavioral dimensions to complement the information provided by the reference algorithm. In contrast, Local Exploitation intentionally does not provide a target behavior coordinate. For local refinement, the parent algorithm itself provides the most direct and concrete reference for the current search behavior. Introducing an explicit behavioral coordinate could unnecessarily restrict the LLM's ability to make effective local modifications. Thus, different navigation mechanisms use behavioral information according to their intended range of behavioral change, while the parent implementation serves as the primary behavioral reference for local exploitation.

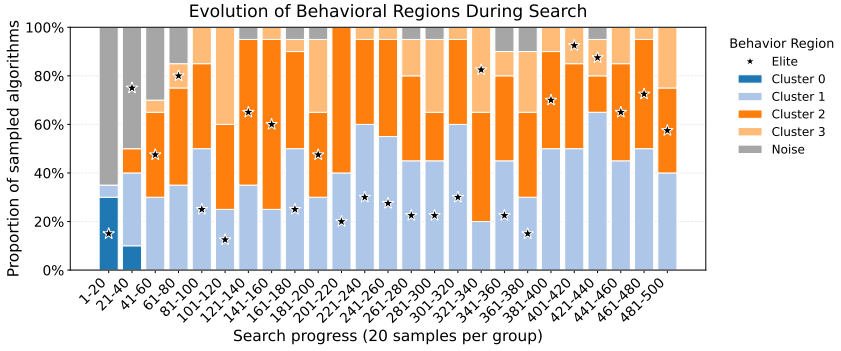
In addition to the navigation-specific information, some prompts also incorporate search reflections as supplementary information. The reflection mechanism is adopted from ReEvo (Ye et al., 2024), where reflections summarize useful observations from previously evaluated algorithms and search iterations. BeNav retains this mechanism as an auxiliary source of search experience rather than as an independent navigation component. The reflection is generated after the selection stage and provided to the corresponding generation prompt through the `reflection` field. It complements the behavioral information by providing potentially useful lessons from previous search experience, while the navigation mechanisms determine the intended direction of behavioral change. The specific use of these information sources is detailed in the following prompt descriptions.



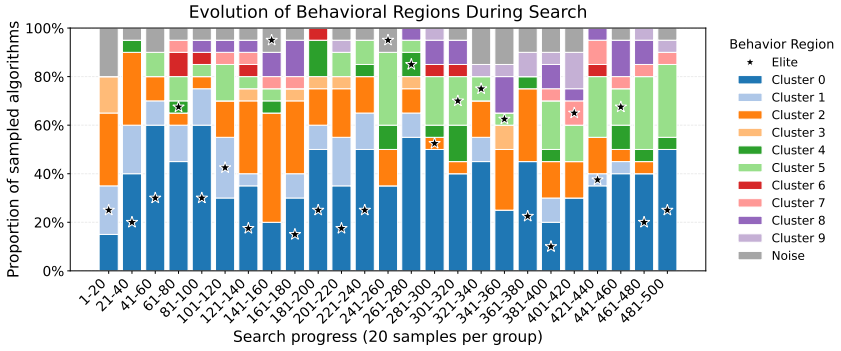
(a) BeNav.



(b) ReEvo.



(c) HSEvo.



(d) EoH.

Figure 4: Behavioral clustering results on CVRP-ACO.

C.2 Prompt for Behavior Extraction

Purpose The Behavior Extraction Prompt converts a generated heuristic implementation into a structured behavioral representation that can subsequently be used by BeNav for behavioral navigation. Given the source code of a heuristic, the prompt asks the LLM to produce a short semantic description and a four-dimensional behavior vector.

Behavioral Information The four dimensions correspond to Search Style, Information Usage, Uncertainty, and Adaptivity. Each dimension is represented as a continuous coordinate in $[0, 1]$, allowing algorithms with similar search characteristics to occupy nearby positions while preserving behavioral differences along multiple dimensions. The semantic description provides a compact textual representation of the algorithmic strategy, while the behavior vector provides a quantitative representation for subsequent behavioral comparison and navigation.

Conservative Scoring The prompt explicitly requires the LLM to infer behavioral scores only from mechanisms observable in the source code. It prohibits inventing mechanisms that are not supported by the implementation and instructs the LLM to use intermediate values when the behavioral evidence is ambiguous. This conservative scoring rule is intended to reduce subjective or unsupported behavioral assignments.

Output Representation The prompt also enforces a fixed JSON schema containing the semantic description and exactly four behavioral coordinates. This ensures that every generated algorithm can be converted into a consistent representation for the subsequent Navigation process.

Behavior Extraction

You are an expert at understanding heuristic algorithms from code.
Given a Python function implementation (a heuristic), extract a Strategy Unit.
Hard constraints:

- Output MUST be a single JSON object.
- Output ONLY JSON. No markdown, no backticks, no extra text.
- JSON schema:

```
{
  "semantic_text": string, # 1-2 short sentences, plain English
  "behavior_vector": [s1, s2, s3, s4] # exactly 4 floats, each in [0,1]
}
```

Behavior dimensions (each score is a continuous coordinate in $[0, 1]$):

1. **Search Style**: 0 = Local exploitation $\rightarrow 1$ = Global exploration
2. **Information Usage**: 0 = Handcrafted heuristic $\rightarrow 1$ = Learned/experience/pattern-based policy
3. **Uncertainty**: 0 = Deterministic $\rightarrow 1$ = Stochastic/sampling/trial-and-error
4. **Adaptivity**: 0 = Fixed strategy $\rightarrow 1$ = Adaptive/self-modifying/feedback-driven

Scoring MUST be explicit and conservative:

- Only use cues present in the code.
- Do NOT invent mechanisms.
- If unclear, choose a mid value (around 0.5).

Now extract from the code below:
{source_code}

C.3 Behavioral Composition Prompt

Purpose The Behavioral Composition Prompt is designed for intermediate-range behavioral transitions, where the search moves from existing behavioral regions toward a region that combines characteristics of two previously discovered algorithms.

Algorithmic References The prompt provides two parent algorithms together with their Strategy Units, which serve as structured behavioral descriptions of the corresponding algorithms. Each Strategy Unit contains a concise semantic description of the algorithm's search strategy and a four-dimensional behavioral vector that characterizes its position along four behavioral dimensions: Search Style, Information Usage, Uncertainty, and Adaptivity. The source code provides the LLM with concrete implementation mechanisms, while the Strategy Units provide a compact description of the search behaviors represented by the two parents. The prompt further distinguishes the two parents by assigning one as the main structural reference and the other as a source of complementary mechanisms.

Behavioral Guidance The prompt specifies that the generated child should behaviorally lie between the two parents rather than simply copying or mechanically merging their code. The behavioral vectors and semantic descriptions of the two Strategy Units therefore provide the LLM with relational information about the desired behavioral transition. This corresponds to the intermediate-range nature of Composition: the target is not an explicitly specified behavioral coordinate, but a behavioral region defined relationally by two existing algorithms.

Generation Constraints To avoid producing incoherent hybrids, the prompt requires the generated algorithm to remain executable and optimization-oriented, preserve a coherent optimization pipeline, and avoid redesigning the algorithm from scratch. It also limits the transferred mechanisms to one or two compatible components and instructs the LLM to simplify conflicting interactions rather than forcing incompatible mechanisms together.

Behavior Composition

You are performing semantic composition in heuristic search.
Task:

- You will be given two parent heuristics and their Strategy Units.
- Generate a NEW heuristic that explores the semantic region between the two parents.

Requirements:

- Output ONLY a Python code block with the full function.
- Keep the function executable and optimization-oriented.
- Preserve a coherent optimization pipeline.
- Do NOT redesign the algorithm from scratch.

Composition guidance:

- Treat one parent as the main structure.
- Borrow 1-2 compatible search mechanisms from the other parent.
- Focus on combining compatible optimization behaviors, not mixing code mechanically.
- The child should behaviorally lie between the two parents, while remaining coherent and feasible.
- Prefer simple integrations over complicated hybrids.
- If two mechanisms conflict, simplify the interaction instead of forcing both.

Function signature: {func_signature}
Parent A code: {parent_a_code}
Parent A unit: {unit_a}
Parent B code: {parent_b_code}
Parent B unit: {unit_b}
Behavior analysis: {reflection}

C.4 BEHAVIORAL DIVERSION PROMPT

Purpose. The Behavioral Diversion Prompt is designed for longer-range behavioral expansion, where the search needs to move beyond currently promising behavioral regions toward under-explored areas.

Target Behavioral Guidance. Unlike Composition, where the desired transition is defined relationally by two existing algorithms, the target in Diversion is an independently specified position in the behavioral space and may be relatively far from the feasible reference algorithm used for generation. For such larger behavioral shifts, the coarse 0–1 behavioral interpretation used during behavior extraction may provide insufficient semantic resolution to clearly characterize the intended target. To address this issue, the prompt additionally provides finer-grained qualitative ranges for each behavioral dimension, associating different numerical intervals with more specific behavioral tendencies. This allows the LLM to interpret the target vector with greater semantic precision when generating algorithms for larger behavioral transitions.

Feasible Algorithmic Anchor. Because a target behavioral position does not directly determine an executable algorithm, the prompt also provides a feasible reference heuristic as a concrete implementation anchor. The reference supplies an executable starting point from which the LLM can introduce behavioral changes toward the target region. Since the target may represent a substantial behavioral shift from the reference, the prompt explicitly instructs the LLM not to copy the reference directly, but to preserve its feasibility and optimization functionality while modifying its search behavior.

Generation Constraints. The prompt requires novelty to arise from meaningful changes in search behavior rather than arbitrary code modifications or excessive randomness. It also requires the generated heuristic to remain executable, optimization-oriented, and free from placeholder logic. These constraints allow Diversion to induce substantial behavioral changes while maintaining a valid algorithmic foundation.

Behavioral Diversion

You are performing semantic diversion in heuristic search. Generate a NEW optimization heuristic guided by the target behavior vector. The vector is only a rough behavioral tendency. Do NOT try to exactly match the coordinates.
Function signature: {func_signature}
Target behavior vector: {target_behavior}
Below is a FEASIBLE reference heuristic (it runs and produces valid outputs). Your task is NOT to copy it. Instead:

- Preserve executability and valid return types.
- Preserve optimization validity (a meaningful heuristic, not placeholder logic).
- Shift its search behavior toward the target behavior region.
- Make it behaviorally different from the reference (this is diversion, not exploitation).

Reference heuristic (feasible anchor): {reference_code}
Behavior dimensions:

1. **Search Style:** 0.0–0.4: local refinement; 0.4–0.7: balanced exploration; 0.7–1.0: aggressive global exploration
2. **Information Usage:** 0.0–0.4: fixed handcrafted rules; 0.4–0.7: mixed heuristic guidance; 0.7–1.0: adaptive/pattern-driven guidance
3. **Uncertainty:** 0.0–0.4: mostly deterministic; 0.4–0.7: moderate randomness; 0.7–1.0: heavy stochastic search
4. **Adaptivity:** 0.0–0.4: static strategy; 0.4–0.7: partially adaptive; 0.7–1.0: strongly feedback-driven

Requirements:

- Output ONLY a Python code block.

- Keep the algorithm executable.
- Keep the algorithm optimization-oriented.
- Novelty should come from search behavior, not random code changes.
- Avoid pathological randomness.
- Prefer plausible heuristic strategies.
- Do NOT output placeholder logic.

C.5 Local Exploitation Prompt

Purpose. The Local Exploitation Prompt is designed for short-range behavioral refinement. At each evolutionary iteration, the current best-performing algorithm is used as the primary reference, and the LLM is instructed to improve its search behavior while remaining close to the existing search strategy.

Algorithmic Reference. The parent implementation is provided directly to the LLM as the main reference. This allows the LLM to access the concrete mechanisms responsible for the current search behavior and perform targeted modifications without reconstructing the algorithm from scratch. In addition, BeNav retrieves an algorithm with a behaviorally similar representation as supplementary refinement information. Its semantic description is passed to the LLM through the `associated_unit` field, providing additional information about a related search strategy without imposing a new behavioral target.

Local Refinement. The prompt explicitly requires the generated algorithm to preserve the parent's overall search philosophy and remain behaviorally close to the parent. The LLM is instructed to introduce one or two concrete exploitation mechanisms, such as adaptive parameter adjustment, progress-aware perturbation, local neighborhood refinement, or lightweight local search. This focuses generation on incremental improvements within the current behavioral region rather than large behavioral transitions.

Generation Constraints. The prompt further requires the generated implementation to remain executable and valid, while prohibiting complete algorithmic rewrites and purely cosmetic modifications. Safety instructions also require the LLM to fall back to the original behavior when a newly introduced mechanism becomes unstable. Together, these constraints maintain the local nature of exploitation while allowing the LLM to improve the quality and robustness of existing search strategies.

Local Exploitation

{user_generator} You are performing semantic exploitation to refine behavior locally.
Task:

- You will be given:
  1. a parent implementation
  2. a behavior-associated Strategy Unit
  3. long-term search reflections
- Produce a refined implementation that strengthens the current search behavior region.

Hard constraints:

- Output MUST be ONLY a Python code block with the full function.
- Keep it executable.
- Do not include explanation text.
- If you introduce new functions, include imports explicitly.
- Do NOT use `cdist`, `distance_matrix`, or any SciPy distance helpers. If you need distances, compute them using numpy only (e.g., broadcasting, vectorized norms).

- Always return a valid solution.

Function signature: {func_signature}
Parent code: {parent_code}
Associated Strategy Unit: {associated_unit}
Long-term search reflections: {long_term_reflection}
Refinement guidance:

- Preserve the parent's overall search philosophy.
- Keep the implementation behaviorally close to the parent.
- Do NOT rewrite the entire algorithm.
- Perform targeted local refinement.

You MUST introduce ONE OR TWO concrete exploitation mechanisms, such as:

- adaptive parameter adjustment
- progress-aware perturbation strength
- local neighborhood refinement
- greedy repair after perturbation
- conditional diversification under stagnation
- elite-guided refinement
- adaptive sampling
- lightweight local search
- robustness improvement for edge cases

Behavior constraints:

- Strengthen exploitation capability.
- Improve robustness and refinement quality.
- Avoid cosmetic edits or simple variable renaming.
- Avoid completely changing the search strategy.

Safety:

- If a new mechanism becomes unstable, fall back to the original parent behavior.
- Prefer simple and reliable improvements over complicated rewrites.