INSTANCE CLUSTERING AND SELECTION: REDUCING EVALUATION COST FOR AUTOMATED ALGORITHM DESIGN IN LLM-ASSISTED EVOLUTIONARY SEARCH

Anonymous authors
Paper under double-blind review

ABSTRACT

Candidate evaluation is a computational bottleneck in LLM-assisted evolutionary search (LES) for automated algorithm design, as each generated algorithm must be evaluated on training instances to provide a search signal to guide the evolution. We identify instance redundancy as a key inefficiency and propose Behavior-based Dynamic Instance Clustering and Selection (B-DICS), a feature-free method that represents instances through their behavioral responses to candidate algorithms. B-DICS progressively uncovers latent instance structure during search by clustering instances in the behavior space and selecting representative core instances for efficient evaluation. We further integrate B-DICS into LES to form a co-evolving framework, B-DICS-LES, where emerging algorithms induce more discriminative behavioral signals that refine the instance structure over time. This closed feedback loop reduces evaluations while preserving the fidelity of search guidance and enabling structure-aware algorithm design. Experiments on three algorithm design tasks show that B-DICS-LES achieves state-of-the-art performance while reducing evaluation cost by an average of 29.8%, paving the way for the scalable deployment of the LES paradigm in real-world automated algorithm design.

1 INTRODUCTION

Large language model (LLM)-assisted Evolutionary Search (LES) has recently emerged as a powerful paradigm for automated algorithm design (Liu et al., 2026). By integrating the generative reasoning capabilities of LLMs with the iterative optimization strengths of evolutionary computation, LES enables the automated synthesis of high-performing algorithms across diverse domains (Romera-Paredes et al., 2024; Ye et al., 2024; Dat et al., 2025; van Stein & Bäck, 2024; Shojaee et al., 2025; Mo et al., 2025; Zhou et al., 2025; Hu et al., 2025b).

Despite its promise, a fundamental bottleneck in LES is candidate evaluation, which dominates overall computational cost (Jin, 2011; Hutter et al., 2019). To obtain an algorithm portfolio with target performance, LES typically samples X candidate algorithms and evaluates each of them on M problem instances, resulting in an evaluation complexity of $\mathcal{O}(X \cdot M)$. This makes efficient evaluation a critical factor for scaling LES to more complex tasks.

This raises a critical question: *can we evaluate candidate algorithms on fewer instances (i.e., smaller M) without compromising the effectiveness of the search?* We argue that the answer lies in recognizing and exploiting **instance redundancy**. In practice, not all instances are equally informative (Smith-Miles & Tan, 2012): some induce similar response patterns across candidate algorithms and thus contribute little additional information to guide the search. This suggests that comparable performance could be achieved by focusing on a carefully selected subset of representative instances.

A straightforward solution is to rely on instance features and use clustering-based sampling, as is commonly done in machine learning (Sarwar et al., 2000; Sener & Savarese, 2018). However, this is often impractical in LES settings where tasks are frequently novel or black-box (Belkhir, 2017). In such scenarios, reliable instance features are either unavailable or costly to design, limiting the applicability of feature-based methods (Kerschke et al., 2019).

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.

To overcome these limitations, we propose **Behavior-based Dynamic Instance Clustering and Selection (B-DICS)**, a feature-free approach that learns and continuously refines the latent instance structure *online* during the evolutionary process. Instead of relying on handcrafted, domain-specific features, B-DICS represents each instance through its observed behavioral response across a diverse set of candidate algorithms, which naturally emerge as LES progresses. By clustering instances in this behavioral space, B-DICS groups instances exhibiting similar algorithmic response profiles, thereby uncovering the latent structure of the instance space from an algorithm-centric perspective.

Building on this uncovered structure, B-DICS performs cluster-aware instance selection, selecting representative instances from each cluster to enable reduced yet informative evaluation. As evolution proceeds, newly generated candidate algorithms continuously enrich the behavioral representation, leading to progressively more accurate clustering and more reliable instance selection. In this way, instance clustering and selection are tightly coupled with the search process, forming a closed feedback loop that improves evaluation efficiency while preserving effective search guidance. We refer to this integrated framework as **B-DICS-LES**.

Beyond efficiency, the uncovered instance structure also provides a natural basis for specialized algorithm design. Instances within the same cluster exhibit similar response patterns, enabling the search process to focus on more coherent sub-objectives and facilitating the discovery of specialized algorithms. This is particularly beneficial under heterogeneous instance distributions, where a single monolithic algorithm may be suboptimal. In summary, our contributions are as follows:

(1) **Behavior-driven Instance Reduction:** We propose **B-DICS**, a fully data-driven and low-overhead approach that progressively uncovers latent instance structure from routine LES processes. By grouping instances with similar behavioral patterns, B-DICS enables reliable instance selection for efficient evaluation and supports structure-aware algorithm design.

(2) **A Co-evolving LES Framework:** Building on B-DICS, we introduce a unified framework that tightly integrates instance selection with LES via a co-evolving feedback loop. LES continuously supplies behavioral data for structure learning, while B-DICS provides increasingly accurate structural insights and lightweight instance subsets to guide more efficient and specialized search.

(3) **Extensive Empirical Validation:** We evaluate B-DICS-LES on three algorithm design tasks, showing that it significantly reduces evaluation cost while maintaining—and often improving—the quality of the resulting algorithm portfolios. Additional ablations validate the effectiveness of B-DICS and the benefits of structure-aware specialization.

2 Related Work

LLM-assisted Evolutionary Search. LES employs LLMs as generative operators within the evolutionary process to iteratively design and refine algorithms based on empirical performance feedback (Liu et al., 2023; Romera-Paredes et al., 2024). Its efficiency is governed by two complementary factors. The first is *search efficiency*, which measures the number of candidate algorithms required to achieve a target performance and is closely related to overall resource consumption (van Stein & Bäck, 2024). The second is *evaluation efficiency*, which reflects the number of instances needed to reliably assess each candidate and directly determines computational cost. Prior work has primarily focused on improving search efficiency through enhanced evolutionary mechanisms (Hu & Zhang, 2025; Dat et al., 2025) and prompting strategies (Liu et al., 2024a; Ye et al., 2024; 2025). In contrast, evaluation efficiency remains relatively underexplored, despite often being the dominant computational bottleneck. We address this gap by introducing a structure-aware evaluation paradigm that reduces redundant instance evaluations while preserving effective search guidance.

Instance Selection. Instance selection reduces computational cost by identifying representative subsets (Olvera-López et al., 2010). Classical supervised methods (e.g., ENN, IB3) operate as offline filters requiring explicit features and labels (Wilson, 1972; Aha et al., 1991), limiting their utility in LES where target domains are often novel and reliable features may be unavailable (Mirzasoleiman et al., 2020). Recent studies in benchmark reduction and evolutionary down-sampling explore feature-free instance profiling via empirical performance data (Dietrich et al., 2024; Cenikj et al., 2022; Benjamins et al., 2024). However, they depend on complete offline evaluation matrices or static heuristics, making them ill-suited for the online dynamics of LES. Inspired by performance-driven profiling, we propose B-DICS, which reformulates instance clustering and selection into a feature-

free and dynamically adaptive process tailored for LES. B-DICS infers instance similarity from accumulated algorithmic response patterns observed during search, enabling instance structures to be continuously refined online and fed back into LES. In a concurrent effort, DyCA (Hu et al., 2026) also explores behavior-based dynamic clustering to uncover instance structures, supporting specialized algorithm design and improving tail robustness. B-DICS encompasses a broader framework that not only enables specialization but also harnesses the discovered structures for representative instance selection, improving evaluation efficiency while preserving informative search signals.

3 Behavior-based Dynamic Clustering and Selection for Search

We aim to introduce instance selection to reduce the redundant computational cost incurred when evaluating candidate algorithms on training instances. However, existing selection methods exhibit two fundamental limitations in the context of LES:

- First, they rely heavily on predefined, domain-specific features and labels. The quality of these attributes directly constrains the effectiveness of instance selection, thereby imposing an implicit upper bound on subsequent algorithm design. In novel or poorly understood domains, this dependency renders classical methods impractical.
- Second, instance selection is typically conducted as an offline preprocessing step prior to learning. Any suboptimal selection made at this stage is irreversible, leading to persistent misleading signals throughout the evolutionary design process.

To address these challenges, we propose B-DICS, **a feature-free, online instance clustering and selection module tailored for LES**. B-DICS leverages emergent behavioral signals—the performance responses of instances to candidate algorithms—to progressively uncover the latent instance structure and dynamically select representative subsets during evolution.

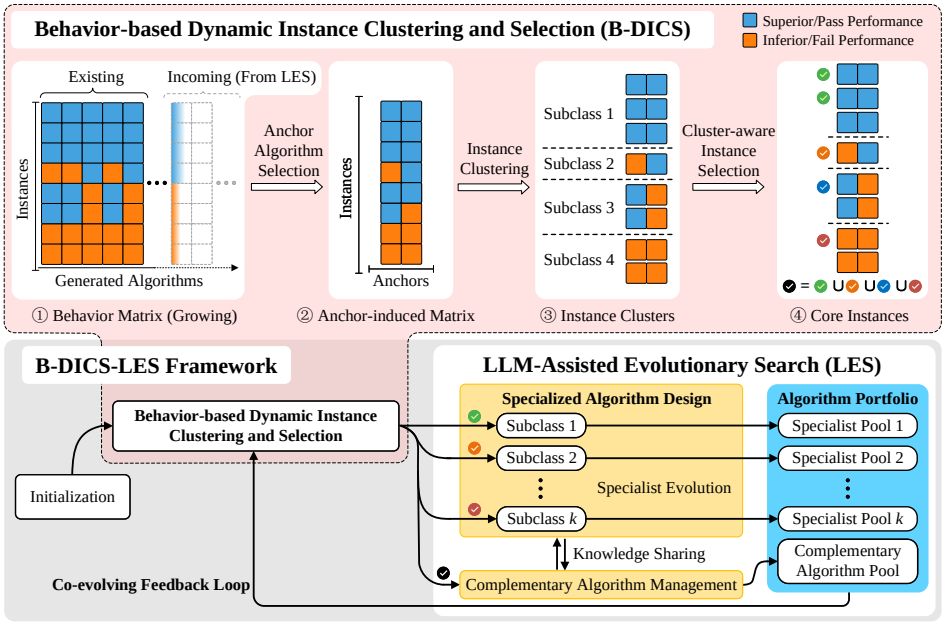


Figure 1: Overview of the B-DICS and its integration with LES for algorithm design. The B-DICS (top) continuously receives newly generated evaluation data from LES, constructs an evolving behavior matrix, dynamically clusters instances into subclasses, and selects core instances. The LES (bottom) can exploit the uncovered instance structure and core instances to guide efficient evolutionary search for building a reliable algorithm portfolio. Both modules are coupled through a co-evolving feedback loop, enabling iterative refinement of instance understanding and algorithm design. For illustration, performance is shown as binary (pass/fail), whereas in practice it is continuous.

At a high level, B-DICS reduces evaluation cost by eliminating behaviorally redundant instances while preserving the algorithmic design targets induced by the original training set. It achieves this by

clustering instances based on their response patterns and performing cluster-aware instance selection. As LES progresses, both the behavioral representation and the resulting structure are continuously refined, forming a co-evolving feedback loop between instance understanding and algorithm design.

3.1 Behavioral representation and anchor-based reduction

We represent each instance through its responses to generated algorithms. Let $\mathcal{I} = \{I_1, \ldots, I_M\}$ denote the set of training instances, and $\mathcal{A} = \{a_1, \ldots, a_n\}$ the archive of algorithms generated up to the current evolutionary stage. B-DICS collects the algorithm–instance evaluation data naturally accumulated throughout the LES process to form a behavior matrix $\mathbf{R} \in \mathbb{R}^{M \times n}$, where $R_{i,j}$ denotes the performance of algorithm a_j on instance I_i.

Each row of $\mathbf{R}$ corresponds to the *response vector* of an instance, capturing how it reacts to a diverse set of algorithms. The central insight of B-DICS is that the intrinsic solvability of an instance is implicitly reflected in its response vector. Consequently, instances that exhibit similar response patterns can be regarded as behaviorally similar and induce consistent algorithmic design targets.

However, directly clustering instances based on $\mathbf{R}$ becomes increasingly impractical as n grows during LES, since the resulting high-dimensional representations degrade clustering quality due to the curse of dimensionality. To address this, we introduce a compact anchor-based representation. Specifically, we select a small subset of *anchor algorithms* $\mathcal{Z} \subset \mathcal{A}$, where $|\mathcal{Z}| \ll n$, and project the behavior matrix onto this subset, yielding an anchor-induced matrix $\bar{\mathbf{R}} \in \mathbb{R}^{M \times |\mathcal{Z}|}$.

To ensure this reduction remains informative, we define the *discriminative power* of an algorithm subset $\mathcal{S}$ as the number of distinct response patterns it induces:

$$\Phi(\mathcal{S}) = |\mathcal{U}(\mathbf{R}_{\cdot,\mathcal{S}})|, \tag{1}$$

where $\mathbf{R}_{\cdot,\mathcal{S}}$ is the sub-matrix formed by the columns in $\mathcal{S}$, and $\mathcal{U}(\cdot)$ returns the set of unique row vectors in a matrix. A higher $\Phi(\mathcal{S})$ indicates a stronger ability to distinguish between instances. Our goal is therefore to identify a compact anchor set that preserves high discriminative power.

3.2 Anchor algorithm selection

Directly computing $\Phi(\mathcal{S})$ from raw continuous performance values leads to degenerate outcomes: even minor numerical variations can yield distinct response patterns, inflating $\Phi(\mathcal{S})$ without reflecting true discriminative capacity. To address this, we first apply a discretization preprocessing step mapping continuous values into a small number of behavioral tiers (details in Appendix C.1.1). This transformation suppresses insignificant numerical differences, enabling $\Phi(\mathcal{S})$ to capture meaningful structural variation. Using the discretized behavior matrix, we adopt a two-phase greedy-pruning strategy to construct the anchor set.

Greedy selection. Starting from an empty set, we iteratively add the algorithm that maximizes the increase in discriminative power until no further improvement can be achieved:

$$a^* = \arg\max_{a \in \mathcal{A} \setminus \mathcal{Z}} |\Phi(\mathcal{Z} \cup \{a\})|. \tag{2}$$

Pruning. To eliminate redundancy introduced during the greedy phase, we subsequently remove any anchor whose exclusion does not reduce discriminative power:

$$\mathcal{Z} \leftarrow \mathcal{Z} \setminus \{a\} \quad \text{if} \quad |\Phi(\mathcal{Z} \setminus \{a\})| = |\Phi(\mathcal{Z})|. \tag{3}$$

This procedure yields a compact yet maximally discriminative set of anchor algorithms, providing a low-dimensional embedding for instance clustering. Empirically, across diverse tasks, the anchor set remains small (typically $|\mathcal{Z}|<15$), placing it within a favorable dimensional regime for clustering.

3.3 Dynamic Instance Clustering and Selection

Given the reduced matrix $\bar{\mathbf{R}}$, we partition the instance space and extract representative core instances.

Instance clustering. Clustering is performed on the rows of $\bar{\mathbf{R}}$ to partition instances (details in Appendix C.2). In principle, any clustering method can be applied. In this work, we adopt k-means

with Euclidean distance due to its simplicity and interpretability, ensuring that the effectiveness of behavior-based clustering arises from the representation itself rather than algorithmic complexity. The number of clusters is determined adaptively via the elbow method (Syakur et al., 2018).

Cluster-aware instance selection. Within each cluster, we extract a subset of *core instances*. Specifically, for each cluster, we first select the instance closest to the centroid as an initial core instance. We then iteratively augment this set by selecting instances that maximize discriminative power with respect to distinguishing candidate algorithms. This procedure follows the same principle as anchor selection, but operates on instances rather than algorithms (details in Appendix C.3). This enables a principled reduction of the instance set: redundant instances within the same behavioral group are removed, while the diversity of distinct behavioral patterns is preserved.

Dynamic recalibration. Since clustering and selection depend on the behavior matrix induced by algorithms, poor initial diversity could bias the uncovered clusters and affect the subsequent search process. B-DICS addresses this by operating continuously rather than as a one-off procedure. As LES progresses, newly generated algorithms introduce additional behavioral distinctions. We incrementally update the anchor set by applying the selection procedure to the union of previously selected anchors and newly generated algorithms, incorporating new information efficiently.

This design preserves previously uncovered structure while allowing refinement as more informative behavioral signals emerge, enabling stable, progressively improved clustering and selection throughout the search. Once the clustering structure stabilizes (e.g., when cluster assignments remain largely unchanged across consecutive updates; see Appendix D.6 for a formal definition), the resulting core instances can reliably serve as a compact surrogate for the full training set for subsequent evaluation.

3.4 Co-evolution with LLM-assisted Evolutionary Search

B-DICS operates alongside the algorithm design process by continuously absorbing algorithm–instance evaluation data, enabling instance clustering and selection to evolve in tandem with algorithm design. This forms a co-evolving system in which instance understanding and algorithm optimization mutually reinforce each other over time.

One key outcome of B-DICS is a set of representative core instances, which replaces the full instance set to significantly reduce evaluation cost. Specifically, it reduces the evaluation complexity from $\mathcal{O}(M)$ to $\mathcal{O}(|\mathcal{I}_{\text{core}}|)$, where $|\mathcal{I}_{\text{core}}| \ll M$. Importantly, since the selected instances preserve the major behavioral patterns of the original instance set, the resulting fitness estimates remain sufficiently informative to guide the evolutionary process.

While the selected core instances alone drastically improve evaluation efficiency, B-DICS reaches its full potential when tightly integrated into LES to enable structure-aware evolutionary search. We propose a unified **B-DICS-LES** framework that leverages uncovered instance clusters to guide fine-grained algorithm design.

The key insight is that instances within the same cluster exhibit similar response patterns across algorithms. Therefore, improvements achieved by an algorithm on one instance are likely to transfer to other instances within the same cluster, indicating a shared optimization direction. Consequently, the uncovered clusters naturally define a set of coherent sub-objectives over the instance space. This allows us to implicitly decompose the overall algorithm design task and encourages the emergence of specialized algorithms tailored to different regions of the instance space.

At a high level, B-DICS-LES operates as an iterative framework consisting of the following stages and mechanisms (see Appendix D for full details):

1. Initialization (warm start): An initial algorithm archive is constructed via repeated prompting of LLMs. The resulting evaluation data provide the initial behavioral evidence to seed B-DICS.

2. B-DICS-assisted evolutionary cycle: Each cycle consists of two tightly coupled steps:

(i) Dynamic instance structure learning and selection: Using the accumulated algorithm–instance evaluation data, B-DICS updates the anchor algorithm set and refines the instance clustering structure. As the clustering stabilizes, B-DICS begins to select representative core instances from each cluster, which are used to replace the full instance set for efficient evaluation. If significant changes in

clustering are detected, the evaluation set is temporarily reverted to the full instance set to avoid introducing biased or unstable fitness signals.

(ii) Structure-aware evolutionary search: Guided by the uncovered instance structure, the search process adopts a divide-and-conquer paradigm for fine-grained algorithm design. The framework maintains distinct algorithm pools tailored to individual clusters, enabling localized search processes that focus on coherent subsets of instances. Within each pool, parent selection and population management are guided solely by the performance on the associated cluster, promoting the emergence of specialized algorithms adapted to specific regions of the instance space.

3. Knowledge sharing and diversity preservation: We introduce a global evaluation and sharing mechanism to mitigate diversity loss caused by excessive specialization in cluster-specific search.

(i) Global cross-evaluation and migration: Every newly generated algorithm is evaluated on the entire active instance set (core instances if selection is active; otherwise, the full set) and made accessible to all specialist pools. This shared protocol ensures each cluster receives its necessary evaluation feedback. Furthermore, algorithms exhibiting strong cross-cluster performance can seamlessly migrate between pools, facilitating the transfer of advantageous traits across the search space.

(ii) Complementary pooling for diversity preservation: Building upon this evaluation mechanism, B-DICS-LES incorporates a complementary pool (inspired by EoH-S (Liu et al., 2025)) as a global hub for enhanced diversity and knowledge sharing. Unlike specialist pools that prioritize local elites, the complementary pool maintains a set of algorithms exhibiting performance complementarity across the entire active set. This retains structurally and behaviorally diverse algorithms that, while not dominating any single cluster, collectively provide broader coverage of the instance space and facilitate smoother knowledge transitions between pools. Finally, if a specialist pool experiences prolonged stagnation, it can escape local optima by selectively importing structurally diverse algorithms from the complementary pool or other clusters.

4. Termination: The process terminates when a predefined stopping criterion is met, such as exhausting the evaluation budget or reaching the LLM inference limit.

The co-evolving process progressively refines both the instance representation and the algorithm portfolio, forming a positive feedback loop where improved algorithms induce more discriminative behavioral signals, which in turn lead to more accurate instance clustering and better-guided search.

4 Experiments

4.1 Experimental Settings

Algorithm design tasks We evaluate B-DICS-LES on three algorithm design tasks. Following recent LES works (Ye et al., 2024; Liu et al., 2025; Zhang et al., 2025), we consider the Capacitated Vehicle Routing Problem (CVRP) and Online Bin Packing (OBP), enabling direct comparison with prior methods. Additionally, we include a Lunar Lander Control (LLC) task (Hu et al., 2025a) to assess generality beyond combinatorial optimization. For CVRP, the task is to synthesize routing heuristics, with performance measured by the optimality gap relative to the LKH-3 (Helsgaun, 2017). For OBP, the task is to design online packing strategies, evaluated by the excess ratio over a theoretical lower bound (Martello & Toth, 1990). For LLC, the task is to generate control policies that map state vectors to thrust actions, where performance is measured by cumulative reward, and scores above 200 indicate successful task completion (Hu et al., 2025a).

Heterogeneous problem instances. To assess both efficiency gains and reliability, we construct highly diverse training and test instance sets for each task. These heterogeneous settings rigorously challenge B-DICS to identify representative core instances without sacrificing overall algorithm design quality. For CVRP and OBP, each training set comprises 240 instances spanning multiple scales and at least three distinct generative distributions; their corresponding test sets contain 120 instances drawn across the same diverse distributions. For LLC, the training set consists of 35 instances covering a wide range of initial states and difficulty levels, paired with a 50-instance test set. Detailed task and instance configurations are provided in Appendix E.

Baselines. We compare B-DICS-LES with several state-of-the-art LES methods, including EoH (Liu et al., 2024a), PartEvo (Romera-Paredes et al., 2024), ReEvo (Ye et al., 2024), InstSpecHH (Zhang

et al., 2025), and EoH-S (Liu et al., 2025). Among them, InstSpecHH adopts a feature-based instance partitioning strategy, where instances are categorized using predefined features and optimized via a divide-and-conquer paradigm with specialized evolution. In contrast, EoH-S aims to construct a set of complementary algorithms rather than a single solution, and has demonstrated strong performance on heterogeneous instance distributions.

Implementation details. All methods employ GPT-4o-mini as the underlying LLM to generate offspring algorithms. For a fair comparison, each method is allocated an identical LLM query budget of 2,000 per task. We evaluate both the quality of the resulting algorithm portfolio and the total number of instance evaluations incurred during the search process, corresponding to performance and evaluation efficiency, respectively. The algorithm pool size is fixed at 10 across all methods to ensure comparable local search intensity. Reported results are averaged over five independent runs on the same training instance sets to mitigate stochastic variability.

Evaluation protocol. EoH-S recommends using its full complementary pool as the final output, where the best-performing algorithm within the pool is selected to report final performance, reflecting the upper-bound performance of the design process (Liu et al., 2025). B-DICS-LES follows the same protocol and reports the performance of its complementary pool. For a fair comparison, EoH, ReEvo, and PartEvo are also evaluated using their final algorithm pools (of size 10). For InstSpecHH, we evaluate the union of the best algorithms across all specialist pools, thereby avoiding potential performance degradation caused by instance-to-specialist matching errors (Zhang et al., 2025).

4.2 Comparative evaluations

Table 1: Performance and evaluation cost comparison across tasks. Arrows (↓ / ↑) indicate whether lower or higher values are preferred. The best results are highlighted in **bold**.

Method	Performance			Instance Evaluations		
	CVRP ↓	OBP ↓	LLC ↑	CVRP	OBP	LLC
EoH	0.1815	0.0276	261.2	480k	480k	70k
ReEvo	0.2102	0.0275	274.3			
PartEvo	0.1981	0.0277	262.7			
InstSpecHH	0.1553	0.0238	270.1			
EoH-S	0.1464	**0.0229**	272.8			
B-DICS-LES	**0.1420**	**0.0229**	**279.9**	**271k (-43.5%)**	**321k (-33.1%)**	**61k (-12.9%)**

Table 1 presents the overall comparison. In terms of algorithm design performance, B-DICS-LES consistently outperforms all standard LES methods and matches or surpasses structure-aware baselines (InstSpecHH, EoH-S), which are specifically designed for heterogeneous instances. Appendix B.2 further validates the stability of these results and reports instance-level statistical significance.

More importantly, these superior results are obtained with substantially fewer instance evaluations. B-DICS-LES reduces evaluation cost by 43.5% on CVRP, 33.1% on OBP, and 12.9% on LLC, corresponding to an average reduction of 29.8%, with the cost of supplementary evaluations fully accounted for. The varying reduction ratios across tasks suggest that instance redundancy is inherently problem- and instance-set-dependent. B-DICS-LES adaptively identifies and exploits such task-specific redundancy without any task-specific prior, while incurring negligible additional computational overhead ($<$0.01% relative to the standard LES process; see Appendix B.3.3 for details).

Overall, B-DICS-LES achieves state-of-the-art algorithm design performance with substantially fewer instance evaluations while incurring negligible computational overhead. Its ability to dynamically eliminate redundancy across diverse tasks—without relying on predefined prior features—demonstrates its broad applicability and efficiency for automated algorithm design.

4.3 Effectiveness of B-DICS

Uncovered instance structure enables specialization. A core claim of B-DICS is that it can organize instances solely from algorithmic responses, without handcrafted features or prior distributional knowledge, and that the resulting structure can effectively guide specialized algorithm design.

To empirically validate this, we visualize the performance rankings of the evolved specialist pools on the OBP training set in Fig. 2. The heatmap exhibits a clear block structure, indicating that different pools have autonomously specialized to dominate distinct regions of the instance space. Specifically, Pool 1 achieves the best rank predominantly on instances drawn from the *Uniform* distribution, Pool 2 shows strong dominance on *Weibull*, and Pool 3 consistently dominates the *Bimodal* distribution.

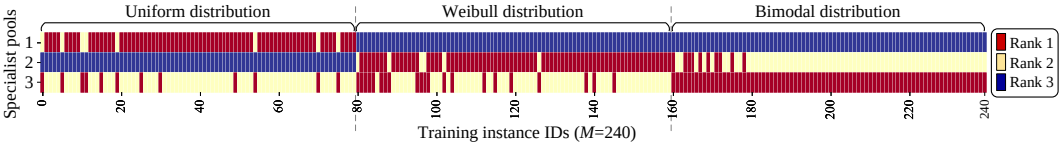


Figure 2: Performance ranks of specialist pools on OBP training instances. The x-axis shows the 240 training instances, grouped by their ground-truth distribution types (Uniform, Weibull, Bimodal), while the y-axis corresponds to the three evolved specialist pools.

Crucially, B-DICS remains entirely agnostic to these ground-truth labels throughout training. The observed specialization arises solely from behavioral signals induced by accumulated evaluation data. This provides strong evidence that behavior-based clustering effectively uncovers coherent sub-objectives, enabling the spontaneous emergence of specialized algorithms without supervision.

Superiority of behavior-driven instance selection. Table 2 compares the quality of the algorithm portfolios derived from different instance selection methods under an identical evolutionary mechanism. We use the final performance guided by the selected core instances as a proxy to evaluate the fidelity of the search signals they provide. To contextualize these results, we set the state-of-the-art baseline, EoH-S (0.1464), as a rigorous benchmark; a performance score below this threshold strongly indicates that the evaluation subset provided high-quality, effective guidance for the search process.

Table 2: Impact of instance selection on end-to-end performance (CVRP).

No.	Selection Strategy	Clustering	Selection Method	Performance
1	Global Selection	N/A	Random	0.1632
2	Cluster-aware Selection	Random	Behavior-driven	0.1644
3	Cluster-aware Selection	Feature-based (Oracle)	Behavior-driven	0.1447
4	Cluster-aware Selection	Behavior-based (Ours)	Behavior-driven	0.1420

- **No. 1 (Random Global Selection):** Randomly sub-sampling the training set yields the worst performance. This indicates that naive reduction destroys the original algorithmic design targets and introduces severe bias, underscoring the vital need to select high-fidelity core instances.
- **No. 2 (Random Clustering + Behavior-driven Selection):** Simulating naive parallel evolution by randomly partitioning instances before selection yields performance remarkably similar to No. 1. This proves that behavior-guided selection fails to recover the original search signals if the underlying structural organization is meaningless.
- **No. 3 (Feature Clustering, Oracle):** Instances are clustered using ground-truth distribution labels available during dataset generation, representing an oracle setting with ideal prior knowledge. Selection based on this oracle structure successfully preserves the search signals, achieving a score (0.1447) that surpasses the EoH-S benchmark.
- **No. 4 (B-DICS, Ours):** Remarkably, B-DICS (0.1420) achieves slightly better performance than the Oracle setting. This demonstrates that purely behavioral signals are not only sufficient to encode instance heterogeneity as effectively as well-established handcrafted features, but they also capture subtle, performance-relevant nuances that rigid semantic labels may overlook. Consequently, B-DICS induces reliable instance clustering and core instance selection for guiding specialization.

In conclusion, B-DICS is a feature-free module that effectively circumvents the limitations of static, prior-dependent instance selection. By uncovering the intrinsic behavioral landscape of instances, it guarantees high-fidelity evaluation signals and paves the way for automated algorithm specialization without domain-specific intervention. Further analyses, including the semantic interpretation of the uncovered instance structures, the dynamic convergence and computational cost-benefit of B-DICS, and t-SNE visualizations of the core instance distributions, are provided in Appendix B.3.

4.4 ABLATION STUDY

To quantify the contribution of individual components within the B-DICS-LES framework, we conduct a comprehensive ablation study on the CVRP task. The results are summarized in Table 3.

Table 3: Ablation studies on CVRP.

No.	Setting	CVRP	Degradation
1	B-DICS-LES (Full)	**0.1420**	-
2	w/o Instance Selection	0.1428	0.56%
3	w/o Anchor	0.1570	10.56%
4	w/ Random Anchor	0.1557	9.65%
5	w/o Dynamic recalibration	0.1640	15.49 %
6	w/o Complementary pool	0.1687	18.80 %

Fidelity of core instances to search signals. Variant 2 disables the selection mechanism entirely, executing cluster-aware evolution using the full instance set. Its performance is nearly identical to the full B-DICS-LES framework (a negligible 0.56% difference). This forcefully demonstrates that the core instances selected by B-DICS preserve the optimization landscape of the entire dataset, achieving maximum computational efficiency without sacrificing target fidelity.

Importance of informative anchor selection. Variant 3 disables the anchor algorithm selection in B-DICS, forcing instance clustering and selection to operate directly on the high-dimensional behavioral response matrix. This results in a 10.56% performance degradation, attributable to two issues: (1) redundant algorithms with similar behaviors distort instance representations, and (2) the high dimensionality hinders accurate similarity estimation, ultimately affecting downstream clustering and algorithm design. Meanwhile, Variant 4 replaces our greedy-pruning strategy with random anchor sampling, resulting in a 9.65% degradation. Although this reduces the dimensionality of the representation, randomly selected anchors fail to preserve the discriminative power needed to distinguish instances effectively. These results validate our anchor selection design, which achieves dimensionality reduction while preserving the informative behavioral signals needed for effective instance discrimination.

Significance of dynamic recalibration. Variant 5 disables the dynamic update mechanism in B-DICS, forcing the system to rely solely on the initial behavioral clustering established during the warm-up phase. The resulting 15.49% performance drop highlights that early-stage evaluation data is insufficient to capture the true, evolving structure of the instances. Continuous, online refinement is crucial as novel algorithm behaviors emerge during the search.

Effectiveness of the complementary pool. Variant 6 disables the global complementary pool, relying exclusively on specialist pools. This leads to a severe 18.80% performance degradation. This confirms that while instance specialization is powerful, a global sharing mechanism is vital. It prevents premature convergence within isolated clusters, facilitates cross-cluster knowledge transfer, and maintains the population diversity necessary for constructing a reliable algorithm portfolio.

5 CONCLUSION

This work addresses the computational bottleneck of candidate evaluation in automated algorithm design using LLM-assisted Evolutionary Search (LES). We introduce Behavior-based Dynamic Instance Clustering and Selection (B-DICS) to identify and remove redundant instances without relying on domain priors, supporting more lightweight yet high-fidelity evaluations. By leveraging naturally accumulated evaluation data during LES, B-DICS progressively uncovers latent instance structures in a completely feature-free manner, enabling the structure-aware evolution of specialist algorithms across the instance space. Extensive experiments demonstrate that B-DICS-LES significantly reduces evaluation costs by up to 29.8% on average, while consistently achieving state-of-the-art performance. Future research will focus on developing B-DICS into a plug-and-play module to facilitate the lightweight and scalable application of the LES paradigm.

AI USE STATEMENT

In this work, generative AI tools were used solely for language polishing. We did not use generative AI tools for literature survey, framework design, code implementation, or experimental analysis and statistical evaluation. It is worth noting that LLMs are an integral component of the algorithm design framework studied in this work. Specifically, LLMs are used within our proposed framework to generate and evolve algorithms for the target algorithm design tasks, consistent with their role in existing LLM-driven automated algorithm design research. This use of LLMs is part of the proposed algorithmic methodology rather than AI assistance in conducting or writing the research. We have reviewed and verified all AI-assisted content and take full responsibility for the final content of this work, including all text, claims, and artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

The necessary technical details of the B-DICS-LES framework are provided in Section 3, with further details given in Appendices C and D. To ensure full reproducibility of our results, the appendix provides all essential experimental details, including a comprehensive description of hyperparameters and experimental configurations (Appendix B.1), the datasets employed (Appendix E), and the complete set of LLM prompts (Appendix F). Furthermore, the source code is included in the supplementary materials to facilitate direct replication of all reported experiments.

REFERENCES

David W Aha, Dennis Kibler, and Marc K Albert. Instance-based learning algorithms. *Machine learning*, 6(1):37–66, 1991.

Nacim Belkhir. *Per instance algorithm configuration for continuous black box optimization*. PhD thesis, Université Paris Saclay (COmUE), 2017.

Carolin Benjamins, Gjorgjina Cenikj, Ana Nikolikj, Aditya Mohan, Tome Eftimov, and Marius Lindauer. Instance selection for dynamic algorithm configuration with reinforcement learning: Improving generalization. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pp. 563–566, 2024.

Gjorgjina Cenikj, Ryan Dieter Lang, Andries Petrus Engelbrecht, Carola Doerr, Peter Korošec, and Tome Eftimov. Selector: selecting a representative benchmark suite for reproducible statistical comparison. In *Proceedings of The Genetic and Evolutionary Computation Conference*, pp. 620–629, 2022.

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.

Maxence Delorme, Manuel Iori, and Silvano Martello. Bin packing and cutting stock problems: Mathematical models and exact algorithms. *European Journal of Operational Research*, 255(1): 1–20, 2016.

Maxence Delorme, Manuel Iori, and Silvano Martello. Bpplib: a library for bin packing and cutting stock problems. *Optimization Letters*, 12(2):235–250, 2018.

Konstantin Dietrich, Diederick Vermetten, Carola Doerr, and Pascal Kerschke. Impact of training instance selection on automated algorithm selection models for numerical black-box optimization. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 1007–1016, 2024.

Pablo A Estévez, Michel Tesmer, Claudio A Perez, and Jacek M Zurada. Normalized mutual information feature selection. *IEEE Transactions on neural networks*, 20(2):189–201, 2009.

Emanuel Falkenauer. A hybrid grouping genetic algorithm for bin packing. *Journal of heuristics*, 2 (1):5–30, 1996.

Soham Gadgil, Yunfeng Xin, and Chengzhe Xu. Solving the lunar lander problem under uncertainty using reinforcement learning. In *2020 SoutheastCon*, volume 2, pp. 1–8. IEEE, 2020.

Ian P Gent and Toby Walsh. Analysis of heuristics for number partitioning. *Computational Intelligence*, 14(3):430–451, 1998.

Keld Helsgaun. An extension of the lin-kernighan-helsgaun tsp solver for constrained traveling salesman and vehicle routing problems. *Roskilde: Roskilde University*, 12:966–980, 2017.

Qinglong Hu and Qingfu Zhang. Partition to evolve: Niching-enhanced evolution with llms for automated algorithm discovery. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025.

Qinglong Hu, Xialiang Tong, Mingxuan Yuan, Fei Liu, Zhichao Lu, and Qingfu Zhang. Multimodal llm-assisted evolutionary search for programmatic control policies, 2025a. URL https://arxiv.org/abs/2508.05433.

Qinglong Hu, Qingfu Zhang, Fei Liu, Xialiang Tong, Kun Mao, and Mingxuan Yuan. Beyond average performance: Dynamic instance clustering and specialized algorithm design in llm-assisted evolutionary search. *arXiv preprint arXiv:2608.03129*, 2026.

Shengran Hu, Cong Lu, and Jeff Clune. Automated design of agentic systems. In *The Thirteenth International Conference on Learning Representations*, 2025b. URL https://openreview.net/forum?id=t9U3LW7JVX.

Frank Hutter, Lars Kotthoff, and Joaquin Vanschoren. *Automated machine learning: methods, systems, challenges*. Springer, 2019.

Yaochu Jin. Surrogate-assisted evolutionary computation: Recent advances and future challenges. *Swarm and Evolutionary Computation*, 1(2):61–70, 2011.

Pascal Kerschke, Holger H Hoos, Frank Neumann, and Heike Trautmann. Automated algorithm selection: Survey and perspectives. *Evolutionary computation*, 27(1):3–45, 2019.

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

Fei Liu, Xialiang Tong, 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 International Conference on Machine Learning*, pp. 32201–32223, 2024a.

Fei Liu, Rui Zhang, Zhuoliang Xie, Rui Sun, Kai Li, Qinglong Hu, Ping Guo, Xi Lin, Xialiang Tong, Mingxuan Yuan, et al. Llm4ad: A platform for algorithm design with large language model. *arXiv preprint arXiv:2412.17287*, 2024b.

Fei Liu, Yilu Liu, Qingfu Zhang, Xialiang Tong, and Mingxuan Yuan. Eoh-s: Evolution of heuristic set using llms for automated heuristic design, 2025. URL https://arxiv.org/abs/2508.03082.

Fei Liu, Yiming Yao, Ping Guo, Zhiyuan Yang, Xi Lin, Zhe Zhao, Xialiang Tong, Kun Mao, Zhichao Lu, Zhenkun Wang, Mingxuan Yuan, and Qingfu Zhang. A systematic survey on large language models for algorithm design. *ACM Comput. Surv.*, January 2026. ISSN 0360-0300. doi: 10.1145/3787585. URL https://doi.org/10.1145/3787585. Just Accepted.

Silvano Martello and Paolo Toth. Lower bounds and reduction procedures for the bin packing problem. *Discrete Applied Mathematics*, 28(1):59–70, 1990.

Baharan Mirzasoleiman, Jeff Bilmes, and Jure Leskovec. Coresets for data-efficient training of machine learning models. In *International Conference on Machine Learning*, pp. 6950–6960. PMLR, 2020.

Shibing Mo, Kai Wu, Qixuan Gao, Xiangyi Teng, and Jing Liu. Autosgnn: automatic propagation mechanism discovery for spectral graph neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pp. 19493–19502, 2025.

J Arturo Olvera-López, J Ariel Carrasco-Ochoa, J Francisco Martínez-Trinidad, and Josef Kittler. A review of instance selection methods. *Artificial Intelligence Review*, 34(2):133–143, 2010.

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

Jorge M Santos and Mark Embrechts. On the use of the adjusted rand index as a metric for evaluating supervised classification. In *International conference on artificial neural networks*, pp. 175–184. Springer, 2009.

Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. Analysis of recommendation algorithms for e-commerce. In *Proceedings of the 2nd ACM Conference on Electronic Commerce*, EC '00, pp. 158–167, New York, NY, USA, 2000. Association for Computing Machinery. ISBN 1581132727. doi: 10.1145/352871.352887. URL https://doi.org/10.1145/352871.352887.

Armin Scholl, Robert Klein, and Christian Jürgens. Bison: A fast hybrid procedure for exactly solving the one-dimensional bin packing problem. *Computers & Operations Research*, 24(7): 627–645, 1997.

Petra Schwerin and Gerhard Wäscher. The bin-packing problem: A problem generator and some numerical experiments with ffd packing and mtp. *International transactions in operational research*, 4(5-6):377–389, 1997.

Ozan Sener and Silvio Savarese. Active learning for convolutional neural networks: A core-set approach, 2018. URL https://arxiv.org/abs/1708.00489.

Parshin Shojaee, Kazem Meidani, Shashank Gupta, Amir Barati Farimani, and Chandan K. Reddy. LLM-SR: Scientific equation discovery via programming with large language models. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=m2nmp8P5in.

Kate Smith-Miles and Thomas T Tan. Measuring algorithm footprints in instance space. In *2012 IEEE congress on evolutionary computation*, pp. 1–8. IEEE, 2012.

Marius M Solomon. Algorithms for the vehicle routing and scheduling problems with time window constraints. *Operations research*, 35(2):254–265, 1987.

Muhammad Ali Syakur, B Khusnul Khotimah, Eka MS Rochman, and Budi Dwi Satoto. Integration k-means clustering method and elbow method for identification of the best customer profile cluster. In *IOP conference series: materials science and engineering*, volume 336, pp. 012017. IOP Publishing, 2018.

Eduardo Uchoa, Diego Pecin, Artur Pessoa, Marcus Poggi, Thibaut Vidal, and Anand Subramanian. New benchmark instances for the capacitated vehicle routing problem. *European Journal of Operational Research*, 257(3):845–858, 2017.

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

Dennis L. Wilson. Asymptotic properties of nearest neighbor rules using edited data. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-2(3):408–421, 1972. doi: 10.1109/TSMC.1972.4309137.

Shunyu Yao, Fei Liu, Xi Lin, Zhichao Lu, Zhenkun Wang, and Qingfu Zhang. Multi-objective evolution of heuristic using large language model. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pp. 27144–27152, 2025.

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. *Advances in Neural Information Processing Systems*, 37:43571–43608, 2024.

Huigen Ye, Hua Xu, An Yan, and Yaoyang Cheng. Large language model-driven large neighborhood search for large-scale milp problems. In *Proceedings of the International Conference on Machine Learning*, 2025.

Shaofeng Zhang, Shengcai Liu, Ning Lu, Jiahao Wu, Ji Liu, Yew-Soon Ong, and Ke Tang. Llm-driven instance-specific heuristic generation and selection. *arXiv preprint arXiv:2506.00490*, 2025.

Xun Zhou, Xingyu Wu, Liang Feng, Zhichao Lu, and Kay Chen Tan. Design principle transfer in neural architecture search via large language models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pp. 23000–23008, 2025.

APPENDIX

Contents

A Limitations, discussion, and future work . . . 15

B Extended experimental results . . . 16

B.1 Detailed experimental setup . . . 16

B.2 Empirical stability and instance-level statistical significance . . . 17

B.3 Extended analysis of B-DICS . . . 17

B.3.1 Semantic interpretation of the uncovered instance structures . . . 18

B.3.2 Dynamic convergence of B-DICS . . . 19

B.3.3 Computational cost-benefit of B-DICS (Wall-clock Time Analysis) . . . 20

B.3.4 Validation of search guidance fidelity via visualization and ranking analysis 21

B.4 Sensitivity Analysis of Hyperparameters . . . 23

B.4.1 Recalibration Interval . . . 23

B.4.2 Acceleration Threshold . . . 23

B.4.3 Natural-Break Discretization . . . 24

B.5 Generalization results on public benchmarks . . . 25

C Methodology Details of B-DICS . . . 26

C.1 Anchor algorithm selection . . . 27

C.1.1 Preprocessing: natural break discretization . . . 28

C.1.2 Anchor selection strategy: greedy search with pruning . . . 28

C.2 Instance clustering via anchor-induced representations . . . 29

C.3 Instance selection via anchor-induced representations . . . 29

C.4 Dynamic recalibration during the LES process . . . 30

D Detailed Description of B-DICS-LES. . . . 31

D.1 Framework Overview and Lifecycle . . . 31

D.2 Algorithm Generation and LLM Operators . . . 32

D.3 Specialist Pool Re-initialization and Warm-start . . . 32

D.4 Evaluation Strategy and Global Registration . . . 33

D.5 Pool Management and Synergistic Dynamics . . . 33

D.6 Stability Criteria for Evaluation Acceleration . . . 33

E Task descriptions and heterogeneous instance set construction . . . 33

E.1 Capacitated Vehicle Routing Problem (CVRP) . . . 34

E.2 Online Bin Packing (OBP) . . . 35

E.3 Lunar Lander Control (LLC) 35

F Prompts used in experiments 36

F.1 Task-specific prompts 36

F.2 Prompt templates for evolutionary operators 37

G Licenses for used assets 38

A LIMITATIONS, DISCUSSION, AND FUTURE WORK

Limitations of response-based representation. B-DICS represents instances through their behavioral responses to candidate algorithms, providing a domain-agnostic mechanism for uncovering latent instance structures and enabling instance selection. However, this design implicitly relies on the quality of the response metric. In particular, the response of an algorithm on the same instance across multiple runs should be both *stable* and *accurate*. Otherwise, noisy or inconsistent responses may propagate through the B-DICS pipeline, distort the inferred instance structure, and ultimately degrade the effectiveness of downstream clustering, selection, and structure-aware search.

A representative counterexample is the use of algorithm runtime as a response metric. In practice, runtime is often highly sensitive to external factors such as hardware parallelism, system load, and resource contention. As a result, the same algorithm evaluated on the same instance may exhibit substantial variability across runs, leading to unreliable behavioral signals.

Therefore, when applying B-DICS, it is important to select response metrics that exhibit strong *consistency* and *robustness* to environmental variations. Suitable choices include solution quality, optimality gap, normalized objective values, and constraint violation measures. Such metrics are widely available in combinatorial optimization, planning, and scheduling domains, suggesting that the applicability of B-DICS remains broad despite this requirement.

Discussion: Why not surrogate models? Surrogate-assisted evolutionary computation is a standard approach for reducing evaluation cost in continuous optimization. However, its application to automated algorithm design in LES settings faces fundamental challenges.

First, LES operates in a highly discrete and non-smooth search space of programmatic code. Unlike continuous variables, program representations are extremely brittle: even a minor syntactic modification (e.g., changing a comparison operator) can lead to drastic semantic changes, including incorrect outputs or non-terminating execution. This violates the smoothness assumptions typically required by surrogate models and makes accurate fitness approximation difficult.

Second, representing LLM-generated programs as inputs to surrogate models (e.g., via abstract syntax trees or learned embeddings) introduces significant representation gaps and often fails to capture dynamic execution behaviors that are critical to performance.

Third, evolutionary processes are known to exploit imperfections in surrogate models, leading to *surrogate deception*, where candidate solutions achieve high predicted fitness but perform poorly under true evaluation. This issue is particularly severe in program synthesis, where adversarial code patterns can easily emerge.

In contrast, B-DICS-LES preserves the use of ground-truth execution while reducing evaluation cost through instance selection. By evaluating candidate algorithms on a dynamically maintained, representative subset of real instances, the framework avoids the need for surrogate approximation altogether, thereby eliminating representation mismatch and mitigating surrogate deception. At the same time, it retains the ability to detect invalid or unstable algorithm behaviors during actual execution.

Future work: Plug-and-play instance selection for LES. In this work, B-DICS-LES is presented as a tightly integrated framework that jointly performs instance selection and structure-aware evolutionary search. Beyond accelerating evaluation through core instances, it leverages the discovered instance structure to guide specialized algorithm design and improve portfolio quality.

However, the core mechanism of B-DICS is fundamentally lightweight and modular. It relies only on algorithm–instance evaluation data naturally generated during the LES process to construct behavioral representations and perform instance selection, without requiring domain-specific features or task-dependent modeling assumptions. As such, it can be naturally incorporated into other LES frameworks as a plug-and-play module with minimal modification, provided that evaluation records are available and the instance set is accessible for update.

In such a setting, B-DICS can be executed periodically to absorb newly generated algorithms, update the latent instance structure, and maintain a compact set of representative core instances. Once the instance structure becomes sufficiently stable, these core instances can replace the full training set for candidate evaluation, enabling substantial computational savings with minimal changes to the original pipeline.

More broadly, B-DICS suggests a general paradigm of *behavior-driven instance reduction*, where instance structure is inferred from algorithmic responses rather than predefined features. This perspective may extend beyond LES to other optimization settings involving repeated evaluation over large instance sets, such as evolutionary computation, AutoML, and hyperparameter optimization. Exploring this broader applicability is a promising direction for future work.

B Extended experimental results

B.1 Detailed experimental setup

This subsection reports the detailed experimental configuration and hyperparameter settings used in all experiments. All methods were implemented in Python and executed on a single CPU (Intel i7-13700) with 64 GB of RAM. Table 4 summarizes the shared hyperparameters for B-DICS-LES and all baseline methods. Unless otherwise specified, identical settings were used across methods to control for confounding factors and ensure a fair comparison.

Table 4: Shared hyperparameter settings for all LES methods.

Parameter Description	**Value**
Pool size	10
LLM version used in the evolutionary operators	GPT-4o-mini
LLM sampling temperature	1
Maximum LLM queries B_{total}	2,000
B-DICS recalibration interval b	Every 200 LLM queries

InstSpecHH relies on predefined features to cluster the instances. To simulate an idealized feature setting and avoid confounding effects, we provide InstSpecHH with ground-truth cluster labels. In contrast, all other methods operate in a strictly feature-free setting.

The algorithm pool size is set to 10 for single-pool methods (EoH, ReEvo, EoH-S). For multi-pool methods (PartEvo, InstSpecHH, and B-DICS-LES), each sub-pool also contains 10 individuals, ensuring comparable local search intensity.

For each task, B-DICS-LES is first used to initialize the algorithm pool to a size of 10. This initialized pool is then reused as the starting seed for every run of all LES methods. By fixing the initial pool across methods and runs, we explicitly eliminate the influence of initialization quality on the observed performance differences, allowing the comparison to focus solely on the effectiveness of the evolutionary and search strategies.

Due to the high computational cost of LLM-assisted evolutionary search, we follow prior work (Liu et al., 2025; Yao et al., 2025) and report results from five independent runs to ensure robustness and reliability. To rigorously validate our findings despite the limited number of macro-runs, we provide detailed stability and instance-level statistical analyses in the following subsection.

B.2 Empirical stability and instance-level statistical significance

To evaluate the stability and reproducibility of the reported results, we first analyze performance variability. Table 5 reports the mean performance together with 95% confidence intervals, computed as Mean $\pm$ $1.96 \cdot$ SEM, where SEM denotes the standard error of the mean across the five runs.

Table 5: Performance comparison across five independent runs. Results are reported as $Mean \pm 1.96 \cdot SEM$ (95% confidence interval). Lower values indicate better performance for CVRP and OBP, while higher values are better for LLC. Best results are shown in bold.

Method	CVRP (↓)	OBP (↓)	LLC (↑)
EoH	0.1815 ± 0.0051	0.0276 ± 0.0003	261.21 ± 8.63
ReEvo	0.2102 ± 0.0286	0.0275 ± 0.0002	274.32 ± 7.35
PartEvo	0.1981 ± 0.0149	0.0277 ± 0.0004	262.74 ± 13.48
InstSpecHH	0.1553 ± 0.0141	0.0238 ± 0.0004	270.08 ± 3.94
EoH-S	0.1464 ± 0.0067	0.0229 ± 0.0004	272.79 ± 4.25
B-DICS-LES	$\mathbf{0.1420 \pm 0.0044}$	$\mathbf{0.0229 \pm 0.0003}$	$\mathbf{279.87 \pm 6.66}$

Across all four tasks, B-DICS-LES consistently achieves the best average performance. Moreover, the variability across independent runs remains low, indicating stable optimization behavior despite the inherent stochasticity of LLM generative processes. This suggests that the observed performance gains are systematic rather than driven by isolated favorable runs.

To obtain a more fine-grained and statistically powerful comparison, we further conduct instance-level paired Wilcoxon signed-rank tests comparing B-DICS-LES against two state-of-the-art LES baselines (InstSpecHH and EoH-S). Specifically, we construct paired samples by averaging the performance metric of each testing instance across the independent runs. The non-parametric Wilcoxon test is then applied to these paired instance-level mean scores to evaluate whether the median difference between algorithms is statistically significant. This allows us to assess performance differences across a large number of paired observations, providing strong statistical evidence at the instance level.

Table 6: Statistical significance of B-DICS-LES compared to the strongest baselines (InstSpecHH and EoH-S) using the instance-level paired Wilcoxon signed-rank test. * denotes $p < 0.05$, ** denotes $p < 0.01$, *** denotes $p < 0.001$, and n.s. denotes not significant ($p \geq 0.05$).

Task	vs. InstSpecHH (p-value)	vs. EoH-S (p-value)
CVRP	6.6×10^{-4} (***)	0.21 (n.s.)
OBP	0.167 (n.s.)	0.369 (n.s.)
LLC	1.8×10^{-5} (***)	6.8×10^{-5} (***)

Crucially, while Table 6 indicates no statistically significant performance difference between B-DICS-LES and EoH-S on CVRP and OBP ($p \geq 0.05$), this outcome is highly favorable in our context. It explicitly establishes that B-DICS-LES successfully preserves the upper-bound search quality of the state of the art. When coupled with the substantial reduction in instance evaluations (up to 43.5%, as previously shown in Table 1), these results robustly validate our core claim: B-DICS effectively eliminates redundant evaluations without compromising the effectiveness or reliability of the search process. Conversely, on tasks like LLC, and compared to InstSpecHH across most domains, B-DICS-LES demonstrates statistically significant superiority, further highlighting the benefits of its dynamic, behavior-driven instance clustering.

Overall, these results provide empirical evidence that the advantages reported in the main paper are statistically meaningful and reproducible under independent runs, despite the inherent stochasticity of LLM-assisted evolutionary search.

B.3 Extended analysis of B-DICS

As discussed in the main text, B-DICS is a fully data-driven, feature-free module that progressively uncovers the latent structure of the instance space. To provide a deeper understanding of its behavior

and advantages, this section presents an extended analysis focusing on (1) the semantic meaningfulness of the uncovered clusters, (2) the dynamic convergence of the online recalibration, (3) the overall computational cost-benefit, and (4) a validation of its search guidance fidelity.

B.3.1 Semantic interpretation of the uncovered instance structures

B-DICS operates without predefined features by relying solely on algorithmic response behaviors. This subsection provides a post-hoc analysis to explicitly interpret the uncovered latent structures and elucidate why they effectively facilitate the construction of a reliable algorithm portfolio. As observed in Table 1, B-DICS reduces computational resources by over 30% on both the OBP and CVRP tasks, demonstrating its substantial impact on search efficiency. To understand the root of this effectiveness, we conduct an in-depth analysis of its final clustering outcomes on these two tasks.

We analyze the clustering results produced by B-DICS after the algorithm design process is completed (i.e., after 2,000 LLM queries). Based on the dataset descriptions provided in Appendix E, we summarize the final instance partitions identified by B-DICS and their corresponding semantic interpretations in Table 7.

Table 7: Final B-DICS clustering results and their semantic interpretations. The reported partitions summarize consistent qualitative patterns observed across five independent runs.

Task	Final B-DICS Clusters		Semantic Interpretation
	Cluster ID	Training Instance IDs	
OBP	1	0–80	*Uniform* item size distribution
	2	80–160	*Weibull* item size distribution
	3	160–240	*Bi-modal* item size distribution
CVRP	1	0–60 & 160–220	Mixture of clustered and uniform distributions
	2	80–160	Heavy-demand instances (all scales)
	3	60–80	Clustered customers under high vehicle capacity
	4	220–240	Uniform customers under high vehicle capacity

Several notable observations can be drawn from these results.

- **OBP.** B-DICS partitions the 240 instances primarily according to item size distributions. Overall, these partitions are consistent with expert knowledge in bin packing, where item size characteristics are widely regarded as the dominant factor governing instance difficulty (Falkenauer, 1996; Gent & Walsh, 1998; Schwerin & Wäscher, 1997).
- **CVRP.** B-DICS roughly separates the 240 training instances into four clusters, successfully disentangling multiple sources of instance heterogeneity without any explicit feature inputs. Most notably, it captures the interaction between vehicle capacity and customer topology—a well-known phenomenon in vehicle routing (Solomon, 1987; Uchoa et al., 2017). For instances with large vehicle capacities (which induce long routes), algorithmic behavior is highly sensitive to macro-topologies, prompting B-DICS to strictly separate clustered distributions (Cluster 3) from uniform ones (Cluster 4). Conversely, under tighter capacity constraints (which restrict vehicles to short, localized routes), algorithms become relatively insensitive to global topologies, leading B-DICS to naturally group mixed topologies together (Cluster 1). Furthermore, it explicitly isolates heavy-demand instances (Cluster 2) as a distinct group. This implicit alignment with advanced OR insights confirms that B-DICS characterizes the true problem landscape purely through algorithmic behavioral feedback.

Crucially, these structured and interpretable partitions are obtained without any prior knowledge of instance distributions or handcrafted features. Instead, B-DICS relies entirely on accumulated evaluation data, analyzing instance response patterns to select anchors and dynamically cluster instances. This provides strong empirical evidence that B-DICS can uncover latent instance heterogeneity with genuine semantic meaning, rather than producing arbitrary or purely statistical artifacts.

More broadly, these results suggest that B-DICS is capable of revealing instance heterogeneity **from scratch**. In domains where expert knowledge is limited or unavailable, the final clustering produced by B-DICS can serve as a valuable analytical tool, offering insights into which instance

characteristics most strongly influence algorithmic performance. In this sense, B-DICS not only supports efficient evaluation and specialized algorithm design but also has the potential to assist practitioners in constructing or refining domain-specific understanding for previously underexplored problem settings.

B.3.2 Dynamic convergence of B-DICS

We investigate how B-DICS converges from an initially uninformative partition to the final clustering described in Appendix B.3.1. To quantify this process, we employ two standard clustering similarity metrics: Adjusted Rand Index (ARI) (Santos & Embrechts, 2009) and Normalized Mutual Information (NMI) (Estévez et al., 2009).

ARI measures the agreement between two clustering results and ranges from -1 to 1, where 1 indicates identical partitions. Values above 0.8 generally suggest strong similarity. NMI also measures clustering similarity but is less sensitive to the number of clusters, with values ranging from 0 to 1.

Using these metrics, we track four quantities throughout the recalibration process:

- **Step-ARI**: the ARI between the clustering results of two consecutive recalibration steps.
- **Step-NMI**: the NMI between the clustering results of two consecutive recalibration steps.
- **Ref-ARI**: the ARI between the current clustering and the final clustering (used as a reference).
- **Ref-NMI**: the NMI between the current clustering and the final clustering.

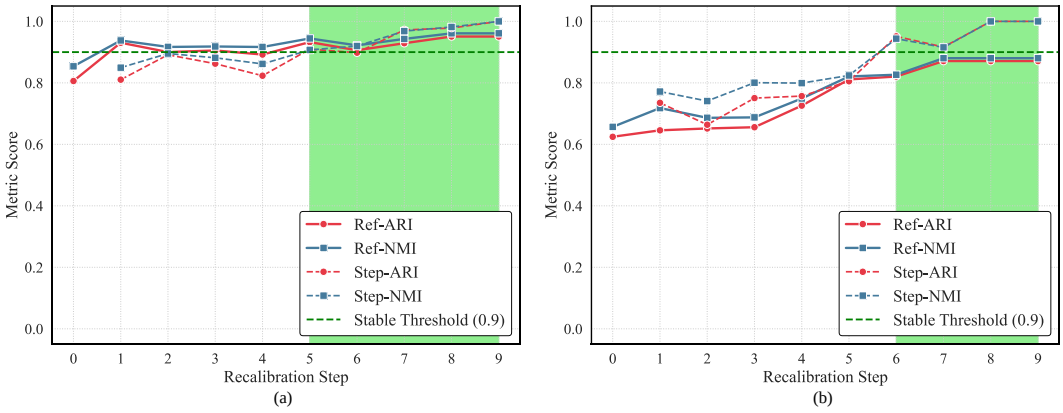


Figure 3: Convergence behavior of B-DICS during the recalibration process on (a) OBP and (b) CVRP. The green dashed line represents the stability threshold (0.9) that triggers the accelerated evaluation using core instances. The green shaded area indicates the phases where the search process operates under this accelerated evaluation mode.

Step-ARI and Step-NMI characterize the self-adjustment behavior of B-DICS. As shown in Figure 3, during the early recalibration steps (e.g., steps 0 to 4), both metrics exhibit minor fluctuations, particularly in the more complex CVRP task. This is because the algorithm portfolio expands during the search, producing diverse behavioral responses that continuously shape and refine the discovered instance structure. As the search progresses, both metrics stabilize and exceed the predefined stability threshold of 0.9. By step 5 for OBP and step 6 for CVRP, this stability triggers the selection of core instances, initiating the accelerated evaluation phase (indicated by the green shaded areas). This pattern indicates that B-DICS achieves robust local stability without prematurely collapsing into a static, suboptimal state. Instead, it remains sensitive to newly acquired algorithmic information, allowing freshly discovered anchor algorithms to induce subtle yet meaningful refinements in instance relationships. These observations empirically support our claim that B-DICS progressively reveals instance structure through iterative interaction with the evolving algorithm portfolio.

Ref-ARI and Ref-NMI provide a global view of convergence. As depicted in the figures, both reference metrics show a generally increasing trend. Even in the initial steps, the initial warm-up

phase provides a reasonable baseline, helping the clustering structure to roughly approximate the general topology of the final partition. As more algorithmic feedback is accumulated, the system does not stubbornly cling to this initial state; rather, the continuous influx of diverse behavioral data gradually sharpens and validates this topology. Consequently, subsequent recalibration steps primarily refine cluster boundaries rather than requiring drastic alterations to the global partition. This behavior highlights both (i) the effectiveness of the initial warm-up in avoiding poor local optima, and (ii) the adaptive convergence of B-DICS as a co-evolving component of the algorithm design process.

Taken together, these results demonstrate that the clustering structure of B-DICS emerges jointly with the evolution of candidate algorithms. It exhibits continuous local adaptability and strong global convergence throughout the design process, effectively underpinning its ability to support safe and efficient evaluation in heterogeneous settings.

B.3.3 Computational cost-benefit of B-DICS (Wall-clock Time Analysis)

Beyond its effectiveness, B-DICS is explicitly designed to be computationally lightweight. Its overhead arises primarily from basic matrix computations and array operations on the behavior matrix, resulting in a marginal cost outside the core algorithm design process. To quantify this computational cost-benefit, we profile the wall-clock time consumed by different components of the B-DICS-LES framework. Table 8 provides a detailed breakdown of the execution time on the CVRP task.

Table 8: Wall-clock time breakdown of B-DICS-LES on the CVRP task (average over 5 runs). The overhead of B-DICS modules is negligible compared to the massive reduction in evaluation time.

Process Component	**Wall-clock Time (s)**	**Percentage**
LLM Generation (Prompting & Inference)	**4043.24**	**30.65%**
Instance Evaluation	**9147.17**	**69.34%**
– Routine evaluations in LES	*7791.89*	*59.07%*
– Complementary evaluations for anchor updates	*1355.28*	*10.27%*
B-DICS Overhead (Total)	**1.296**	**0.01%**
– Anchor algorithm update	*0.656*	*0.0050%*
– K-means clustering recalibration	*0.494*	*0.0037%*
– Core instance selection	*0.041*	*0.0003%*
– Specialist pool reorganization	*0.117*	*0.0009%*
Total B-DICS-LES Execution Time	**13191.71**	**100.0%**

As shown in Table 8, the operations for updating anchor algorithms, recalibrating instance clusters, greedily selecting core instances, and reorganizing specialist pools are computationally inexpensive. They require barely over a single second across the entire search trajectory—even when cumulatively executing 10 full B-DICS recalibration cycles over the 2,000 LLM generation queries. This total cost (a mere 0.01% of the overall execution time) is infinitesimal compared to the dominant bottlenecks of LES: LLM-driven offspring generation and, crucially, algorithm evaluations on the training instances.

Crucially, the computational overhead of B-DICS does not scale with the intrinsic evaluation cost of the target design task. Its algorithmic complexity is tied exclusively to the dimensions of the anchor-induced behavior matrix. Consequently, as the target task becomes more computationally demanding to evaluate (e.g., requiring complex environmental simulations), the relative overhead of B-DICS shrinks even further. In other words, the more expensive the fitness evaluations, the higher the return on investment (ROI). Given that B-DICS saves approximately 30% to 40% of the total instance evaluations on tasks like CVRP (which translates to saving thousands of seconds of heavy routing simulations), its computational cost-benefit ratio is overwhelmingly positive.

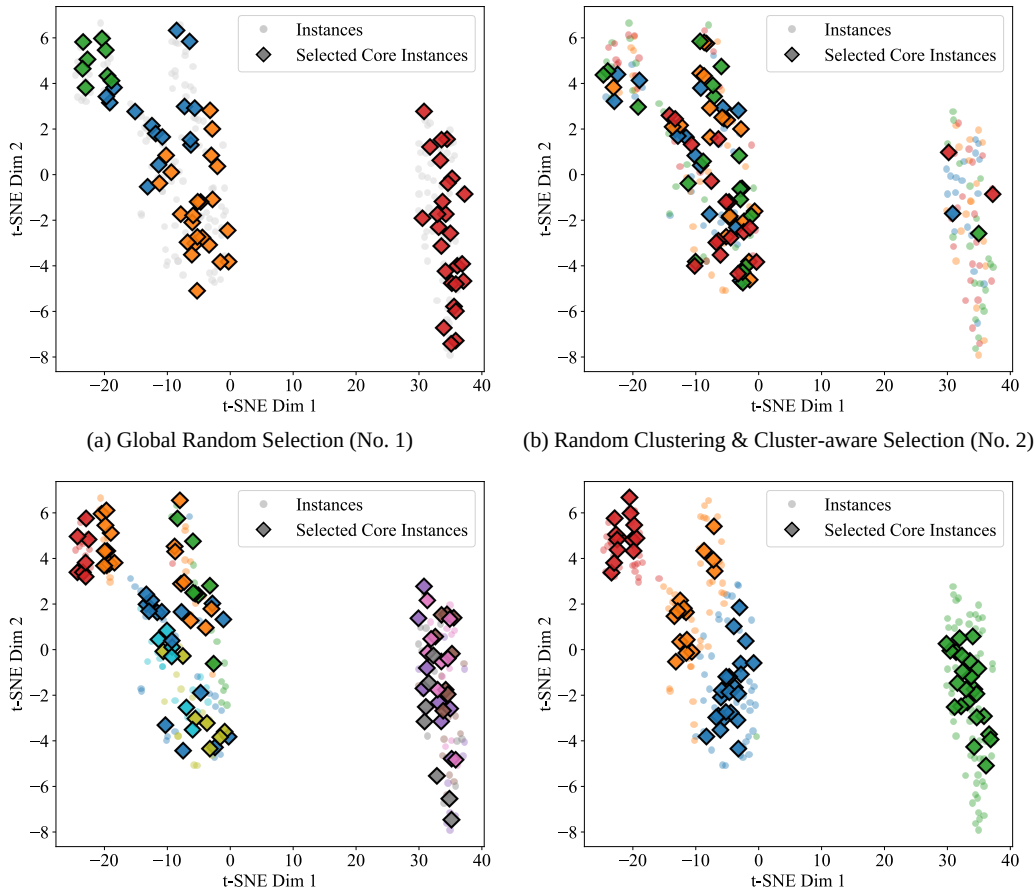
Furthermore, as observed in the main text, the exact evaluation savings rate of B-DICS naturally varies across tasks, depending on the inherent redundancy and heterogeneity of the training instances. However, owing to its domain-agnostic (feature-free) nature combined with its extreme computational efficiency, B-DICS operates as a highly practical *fail-safe* mechanism suitable for diverse problem settings. In a worst-case scenario where a dataset lacks structural redundancy and no redundant

instances can be filtered, integrating B-DICS introduces merely a negligible computational penalty. Conversely, even in such non-redundant cases, the latent structural insights it dynamically uncovers from scratch remain highly valuable for guiding the construction of specialized algorithm portfolios. Thus, integrating B-DICS consistently benefits the overall algorithm design process—either through direct evaluation acceleration or structural knowledge discovery—while imposing minimal computational burden.

B.3.4 VALIDATION OF SEARCH GUIDANCE FIDELITY VIA VISUALIZATION AND RANKING ANALYSIS

To comprehensively demonstrate how B-DICS achieves substantial evaluation efficiency without sacrificing evolutionary search guidance, we provide both a qualitative topological visualization and a quantitative ranking analysis.

Qualitative visualization of core instances We project the high-dimensional behavioral space of CVRP instances into a 2D plane using t-SNE. Figure 4 illustrates the spatial distribution of core instances selected by different strategies evaluated in Table 2.



(a) Global Random Selection (No. 1)

(b) Random Clustering & Cluster-aware Selection (No. 2)

(c) Feature Clustering & Cluster-aware Selection (No. 3, Oracle)

(d) B-DICS (No. 4)

Figure 4: t-SNE visualization of the instance behavioral space under different selection strategies. Colors indicate structural partitions, and diamond markers denote selected core instances.

As shown in Figure 4, Global Random Selection, Oracle-based Cluster-aware Selection, and B-DICS all achieve broad coverage of the behavioral space at a macro level. However, their underlying selection mechanisms differ substantially.

Global Random Selection operates without structural awareness. While it preserves overall density, it may underrepresent boundary or low-density regions that can carry important discriminative signals,

thereby subtly shifting the effective design target. Random Clustering attempts to introduce structure, but its arbitrary partitions are not aligned with the space's intrinsic geometry, leading to suboptimal coverage.

In contrast, both Oracle clustering and B-DICS leverage global structure. The Oracle approach enforces uniform coverage over predefined feature-based clusters, while B-DICS derives clusters directly from emergent behavioral signals. Notably, B-DICS organizes the space into fewer behaviorally meaningful groups, leading to a *target-concentrated* selection pattern. This avoids over-allocating evaluation budget to behaviorally redundant regions while still maintaining global coverage, thereby promoting more efficient and focused evolutionary search, ultimately yielding a superior algorithm portfolio under the same LLM query budget.

Quantitative verification via ranking fidelity We further conduct a quantitative analysis to verify that the core instances selected by B-DICS provide high-fidelity search signals. In evolutionary frameworks such as LES, selection is driven by relative fitness comparisons rather than absolute performance values. Therefore, the ability of a reduced subset (surrogate) to preserve the true relative performance ranking of candidate algorithms—referred to as *ranking fidelity*—is critical for maintaining effective selection pressure. In particular, correctly identifying top-performing algorithms is essential, as these individuals are preferentially selected as parents and directly influence the subsequent search trajectory.

We collect a diverse pool of $N = 567$ candidate algorithms generated during the CVRP search process and evaluate them on the full training set (240 instances) to obtain ground-truth rankings. We then compare these to the proxy rankings obtained on the reduced instance subsets. To provide a comprehensive evaluation, we report Spearman's rank correlation coefficient (ρ) to assess global ranking consistency, along with **Top-1** and **Top-10 overlap** rates to measure the preservation of elite solutions. The results are summarized in Table 9.

Table 9: Ranking fidelity of different selection strategies ($N = 567$ reference algorithms). Spearman's ρ measures global ranking consistency, while Top-K overlap measures the subset's ability to correctly identify the highest-performing algorithms.

No.	Selection Strategy	Spearman's ρ	Top-1 Overlap	Top-10 Overlap
1	Global Selection (Random)	0.9881	0.00	0.70
2	Cluster-aware Selection (Random)	0.9566	0.00	0.30
3	Cluster-aware Selection (Oracle)	0.9840	1.00	0.60
4	**Cluster-aware Selection (B-DICS)**	0.9879	1.00	0.90

Global Random Selection achieves a high global correlation ($\rho = 0.9881$), indicating that it preserves the average ranking structure. However, it fails to identify the true best algorithm (Top-1 = 0.00). This suggests that although most mid-performing algorithms are correctly ordered, some critical instances that provide strong selection pressure are missing, making the subset unreliable for distinguishing top-tier candidates.

In contrast, B-DICS maintains a similarly high correlation (0.9879) while correctly identifying the Top-1 algorithm and achieving the highest Top-10 overlap (0.90). This indicates that B-DICS not only preserves global ranking consistency but also retains the most informative instances for differentiating elite solutions, which is crucial for effective evolutionary selection.

Interestingly, B-DICS also outperforms the Oracle strategy in Top-10 overlap. While the Oracle method enforces uniform coverage over predefined feature-based clusters, such partitions are not necessarily aligned with performance-relevant variations in the search space. In contrast, B-DICS constructs clusters directly from observed behavioral feedback, allowing it to emphasize regions that are more discriminative with respect to algorithm performance. Consequently, the selected subset is better aligned with the underlying selection dynamics of the evolutionary process.

Ultimately, these results corroborate our claim that B-DICS is not merely dropping random instances to save time; instead, it actively constructs a representative subset that effectively preserves the original selection pressures and algorithmic design targets.

Summary In summary, this extended analysis confirms that B-DICS is a high-return, low-cost enhancement for automated algorithm design. By mapping the instance space purely from naturally accumulated behavioral feedback, it simultaneously delivers interpretable structural insights, significant computational savings, and high-fidelity evolutionary guidance. Given its negligible computational overhead and domain-agnostic formulation, B-DICS functions as a highly practical module that can be seamlessly used in any algorithm design task, offering substantial performance and efficiency gains.

B.4 Sensitivity Analysis of Hyperparameters

This section examines the sensitivity of B-DICS-LES to its three primary hyperparameter groups: the recalibration interval b, the acceleration threshold θ_{stable}, and the natural-break discretization parameters $(\eta_{\text{range}}, \eta_{\text{gap}})$. Unless stated otherwise, all analyses are conducted on the CVRP task across five independent runs. Notably, our default configuration ($b = 200$, $\theta_{\text{stable}} = 0.9$, $\eta_{\text{range}} = 0.15$, $\eta_{\text{gap}} = 2.0$) was adopted across all benchmarks without explicit tuning, yet it consistently delivers competitive performance—demonstrating the system's strong out-of-the-box robustness.

B.4.1 Recalibration Interval

The recalibration interval b controls how frequently B-DICS integrates newly generated behavioral data to update anchor algorithms, refresh instance clusters, reconstruct core sets, and reassess whether to trigger accelerated evaluation. We evaluate B-DICS-LES under $b \in \{100, 200, 300\}$ and report both the final algorithm performance and evaluation cost savings.

Table 10: Sensitivity to the recalibration interval b on CVRP (five runs).

b	**Final performance**	**Evaluation saving (%)**
100	0.1445	41.9
200 (default)	0.1420	43.7
300	0.1452	41.3

As shown in Table 10, both the final design performance and evaluation savings remain virtually invariant across the three settings, forming a broad performance plateau around the default setting $b = 200$. In stark contrast, completely disabling dynamic recalibration (i.e., $b \to \infty$, as evaluated in Configuration No. 5 of Table 3) results in a substantial 15.49% performance drop. Together, these findings indicate that the primary benefit of B-DICS lies in periodically absorbing the behavioral distinctions brought by newly discovered algorithms, rather than in fine-tuning the exact frequency of these updates. Consequently, B-DICS-LES exhibits high robustness to b, rendering $b = 200$ a reliable default choice for unseen tasks.

B.4.2 Acceleration Threshold

The acceleration threshold θ_{stable} (default: 0.9) governs when B-DICS switches from full-set evaluation to accelerated core-set evaluation: the accelerated mode engages once the ARI between consecutive clusterings reaches θ_{stable} for the required number of consecutive recalibrations. A lower threshold engages the acceleration earlier and saves more evaluation budget, but risks acting on an immature instance structure; a higher threshold is more conservative. With $\theta_{\text{stable}} = 1.0$, acceleration engages only under perfect clustering stability, effectively reducing to the full-set regime. We train B-DICS-LES with $\theta_{\text{stable}} \in \{0.7, 0.8, 0.9, 1.0\}$ and report the final algorithm design performance and the evaluation saving.

As shown in Table 11, the acceleration threshold governs a clear trade-off between evaluation saving and design quality. Lowering θ_{stable} engages the accelerated mode earlier and raises the saving from 43.7% to 47.2%, at the cost of a mild performance degradation (from 0.1420 to 0.1471, i.e., 3.6%). Conversely, $\theta_{\text{stable}} = 1.0$ suppresses acceleration almost entirely (the residual 10.6% saving stems from the occasional recalibrations with perfectly stable clusterings) and yields 0.1430—slightly *worse* than the default, which is consistent with the ablation results (configuration No. 2 in Table 3) showing that core-set guidance is at least harmless, and in fact mildly beneficial. Overall, the final performance varies by less than 4% across the entire threshold range, while the default $\theta_{\text{stable}} = 0.9$

Table 11: Sensitivity to the acceleration threshold θ_{stable} on CVRP (five runs).

θ_{stable}	**Final performance**	**Evaluation saving (%)**
0.7	0.1471	47.2
0.8	0.1457	45.9
0.9 (default)	0.1420	43.7
1.0	0.1430	10.6

attains the best performance together with a substantial saving, confirming that it is a well-balanced and robust choice.

B.4.3 Natural-Break Discretization

The natural-break discretization (Appendix C.1.1) introduces two hyperparameters—the range ratio η_{range} and the gap multiplier η_{gap}—that jointly control the granularity of behavioral tiers. To evaluate sensitivity, we perform a grid search over $\eta_{\text{range}} \in \{0.05, 0.10, 0.15, 0.20, 0.30, 0.40\}$ and $\eta_{\text{gap}} \in \{1.0, 1.5, 2.0, 2.5, 3.0, 4.0\}$ (36 combinations) around the default configuration $(0.15, 2.0)$, sweeping both parameters jointly.

Evaluating all 36 hyperparameter combinations end-to-end with LES would be computationally prohibitive and introduce confounding stochastic noise. Fortunately, these discretization parameters operate strictly within B-DICS, meaning their only avenue of influence on the downstream pipeline is through B-DICS's outputs (i.e., instance clusterings and core sets). If these module outputs remain stable across parameter choices, variations in η_{range} and η_{gap} will not propagate to the downstream evolutionary search. This allows us to evaluate parameter sensitivity in a decoupled manner. Specifically, we re-run B-DICS across all grid configurations on recorded historical trajectory data from five independent CVRP runs to measure the stability of the resulting clusterings and core sets.

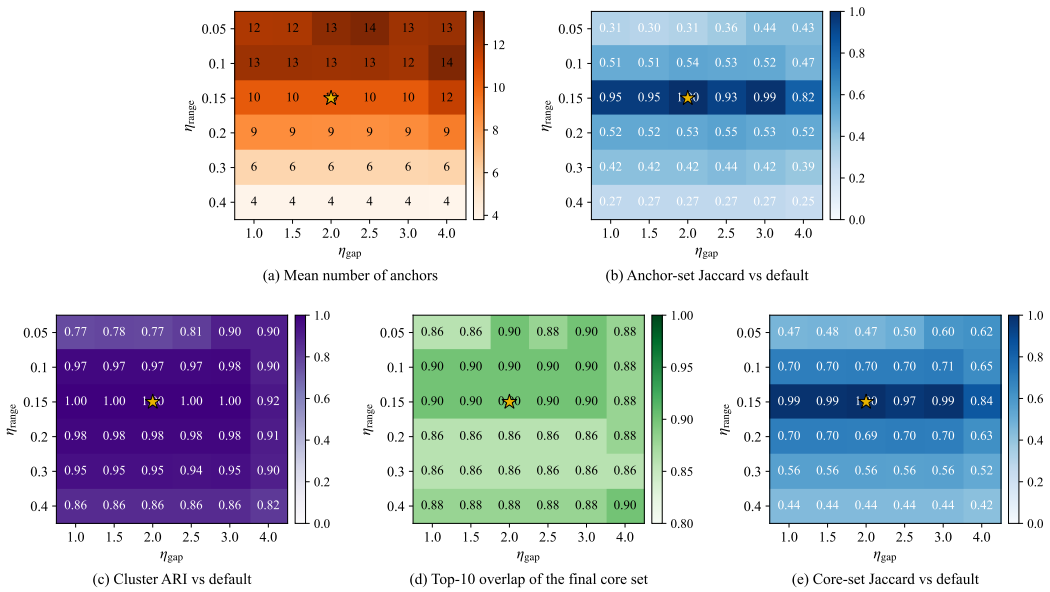


Figure 5: Sensitivity of B-DICS to the natural-break discretization hyperparameters on CVRP, averaged over five runs.

Figure 5 reports five quantities for each grid point, averaged over the five runs: (a) the mean number of selected anchors; (b) the Jaccard similarity between the anchor set and that of the default configuration; (c) the ARI comparing the resulting instance clustering to that of the default configuration, i.e., how similar the clustering under each configuration is to the default one; (d) the overlap between the

top-10 algorithms ranked by the *final* core set (the last recalibration of each run) and those ranked by the full set; and (e) the Jaccard similarity between the core set and that of the default configuration. The default configuration $(0.15, 2.0)$ is marked with a star.

The discretization hyperparameters perform their expected mechanical function: as thresholds increase, the mean anchor count decreases smoothly from 13.6 to 3.8 (panel (a)), driving corresponding shifts in anchor composition (panel (b)). Yet downstream-critical metrics remain remarkably stable around the default configuration. First, panel (c) shows high cluster ARI across neighboring configurations, indicating that interchangeable anchor sets induce highly consistent instance clusterings. Second, panel (d) demonstrates that top-10 ranking overlap stays within $[0.86, 0.90]$ across the entire grid (0.90 ± 0.06 at default), confirming that the final core set reliably preserves the ranking signal of the full set. Together, panels (c), (d), and (e) reveal a stable plateau around $\eta_{\text{range}} = 0.15$ with $\eta_{\text{gap}} \in [1.0, 3.0]$.

Most importantly, our default configuration was adopted directly across three distinct tasks without explicit hyperparameter tuning, yielding consistently strong performance. This confirms its reliability as a robust default for future tasks, supported by our sensitivity analysis showing that its behavior is representative of a broad parameter region.

B.5 Generalization results on public benchmarks

A key concern with instance selection is that evaluating candidates on a smaller subset might bias the design targets, leading to overfitting and degraded generalization. To examine this, we evaluate the generalization ability of the algorithm portfolios learned by B-DICS-LES and the compared LES baselines on two widely used public benchmark suites—CVRPLib (Uchoa et al., 2017) and BPPLib (Delorme et al., 2018)—which are entirely unseen during the LES process. These benchmarks exhibit substantial structural and distributional heterogeneity and thus provide a rigorous test for assessing generalization beyond the synthetic training distributions. The benchmark datasets considered in this study are summarized below.

- **BPPLib.** We select representative benchmark sets from BPPLib (Delorme et al., 2018), as summarized in Table 12. This collection comprises over 700 instances, with the number of items ranging from 100 to 1,002. To ensure consistency across instances, all bin capacities are normalized to 100.
- **CVRPLib.** We consider seven widely used benchmark sets (A, B, E, F, M, P, and X) from CVRPLib (Uchoa et al., 2017). The characteristics of these sets are summarized in Table 13. All coordinates are normalized to the range $[0, 1]^2$ using the maximum spatial extent to maintain scale consistency.

Table 12: Summary of BPPLib benchmark sets.

Benchmark Set	Number of Instances	Capacity	Number of Items
Schwerin_1 (Scholl et al., 1997)	100	1k	100
Schwerin_2 (Scholl et al., 1997)	100	1k	120
AugmentedIRUP (Delorme et al., 2016)	250	{2.5K–80K}	{201–1002}
AugmentedNonIRUP (Delorme et al., 2016)	250	{2.5K–80K}	{201–1002}

Table 13: Summary of CVRPLib benchmark sets.

Benchmark Set	Number of Instances	Instance Size
Set A	27	31–79
Set B	23	30–77
Set E	11	22–101
Set F	3	44–134
Set M	5	100–199
Set P	23	15–100
Set X	43	100–300

Table 14 summarizes the performance of B-DICS-LES and the three main baselines (EoH, InstSpecHH, and EoH-S) on the two benchmark suites, averaged over the corresponding benchmark sets. B-DICS-LES consistently achieves the best results across both suites. Notably, all compared methods are trained on the same original training sets, yet B-DICS-LES generalizes best despite operating on a significantly reduced evaluation budget during training. This confirms that B-DICS preserves the essential structure of the instance space without distorting the underlying optimization objective. In fact, by dynamically filtering out redundant and potentially noisy instances, B-DICS-LES mitigates the risk of overfitting to specific training samples, yielding a more robust algorithm portfolio under distributional shifts. We next report the detailed per-set results to support these aggregate observations.

Table 14: Performance comparison on standard public benchmarks.

Methods	**CVRPLib** (Uchoa et al., 2017)	**BPPLib** (Delorme et al., 2018)
EoH	0.4016	0.1126
InstSpecHH	0.3061	0.1121
EoH-S	0.2610	0.1111
B-DICS-LES	**0.2427**	**0.1107**

Table 15: Detailed results on public benchmarks.

Benchmarks	**InstSpecHH**	**EoH-S**	**B-DICS-LES**
CVRPLib A	0.2860	0.2524	**0.2257**
CVRPLib B	0.2588	0.1943	**0.1853**
CVRPLib E	0.3208	0.2724	**0.2365**
CVRPLib F	0.4738	0.4237	**0.4114**
CVRPLib M	0.3881	0.3234	**0.2984**
CVRPLib P	0.2337	0.1967	**0.1775**
CVRPLib X	0.1817	0.1641	**0.1642**
BPPLib Sch 1	0.1521	0.1528	**0.1523**
BPPLib Sch 2	0.1414	0.1420	**0.1413**
BPPLib IRUP	0.0770	0.0745	**0.0740**
BPPLib NonIRUP	0.0782	0.0752	**0.0751**

For all LES methods, the algorithm portfolios trained on synthetic datasets are directly applied to these benchmarks without any fine-tuning or adaptation. This evaluation protocol isolates each method's intrinsic generalization capability and enables a fair comparison in realistic settings.

As shown in Table 15, B-DICS-LES consistently outperforms all baselines across all evaluated benchmarks. Notably, these improvements are observed across different benchmark families, suggesting that B-DICS-LES does not rely on overfitting to specific structural patterns, nor does it sacrifice performance on particular instance types. When transferred directly to unseen real-world instances without adaptation, B-DICS-LES demonstrates stronger and more stable generalization than existing LES baselines. Overall, these results provide additional empirical support for the effectiveness and robustness of the mechanisms introduced in B-DICS-LES across diverse problem domains.

C Methodology Details of B-DICS

This section provides technical details and necessary clarifications regarding the Behavior-based Dynamic Instance Clustering and Selection (B-DICS) module.

At its core, B-DICS operates on the **emergent** behavior space defined by algorithm-instance evaluation data. It continuously executes three key operations: (1) anchor algorithm selection, (2) instance clustering, and (3) instance selection. As LES progresses, the continuous influx of newly generated algorithms progressively enriches this behavior space, allowing these three operations to become increasingly accurate and reliable over time.

Formally, let $\mathcal{I} = \{I_1, \ldots, I_M\}$ denote the set of training instances, and $\mathcal{A} = \{a_1, \ldots, a_n\}$ denote the archive of algorithms generated up to the current evolutionary stage. B-DICS leverages the evaluation data naturally accumulated throughout the LES process to construct a behavior matrix $\mathbf{R} \in \mathbb{R}^{M \times n}$, where the entry $R_{i,j}$ represents the performance of algorithm a_j evaluated on instance I_i.

Within this matrix, each row $\mathbf{r}_i \in \mathbb{R}^n$ represents the *response vector* of instance I_i, encoding how it reacts to a diverse set of algorithmic strategies. Conversely, each column $\mathbf{p}_j \in \mathbb{R}^M$ corresponds to the *performance vector* of algorithm a_j, detailing its efficacy across the entire instance set.

The central insight of B-DICS relies on the **duality** of this behavioral representation. On one hand, the intrinsic solvability of an instance is implicitly reflected in its response vector. Instances exhibiting similar response patterns can be regarded as behaviorally similar, thus inducing consistent algorithmic design targets. By analyzing these response vectors, B-DICS groups instances with coherent solvability patterns, revealing latent structural heterogeneity in the instance space and providing a principled basis for specialized algorithm design. On the other hand, the functional traits of an algorithm are characterized by its performance vector. Analyzing these column vectors allows B-DICS to identify algorithmic similarities and redundancies.

However, a practical challenge arises as evolution proceeds: the dimensionality of the response vectors (n) grows continuously, rendering direct clustering and distance computations in $\mathbb{R}^n$ increasingly inefficient and susceptible to the curse of dimensionality. To mitigate this, B-DICS introduces a compact set of *anchor algorithms*, $\mathcal{Z} \subset \mathcal{A}$, which projects the high-dimensional matrix $\mathbf{R}$ onto a low-dimensional yet highly discriminative anchor-induced matrix $\bar{\mathbf{R}} \in \mathbb{R}^{M \times |\mathcal{Z}|}$.

In summary, B-DICS exploits $\mathbf{R}$ through a sequential three-step procedure:

(i) Column-wise analysis to select informative anchor algorithms;

(ii) Row-wise analysis of anchor-induced response vectors to uncover latent instance clusters;

(iii) Cluster-aware sampling to identify representative core instances.

Clarification: The Asymmetry in Handling Algorithms and Instances. Given the duality between response vectors and performance vectors, a natural question arises: *Why employ fundamentally different technical routes for algorithm selection (direct redundancy evaluation) versus instance selection (clustering followed by cluster-aware sampling on a reduced matrix)?*

The rationale stems from the fundamental difference between the **static** nature of algorithm features and the **emergent** nature of instance features within the dynamic LES context:

- **Algorithms possess static features.** For any generated algorithm, its feature space (i.e., its performance across the fixed set of M instances) is completely defined and fully observed once evaluated. Consequently, similarities and redundancies between algorithms can be determined definitively, allowing one-time, irreversible decisions during anchor selection.
- **Instances possess emergent features.** In contrast, the feature space of an instance (i.e., its responses to the dynamically growing archive $\mathcal{A}$) continuously expands. Two instances that appear identical early may diverge under newly generated algorithms. Therefore, redundancy assessments must be revisited over time, and direct high-dimensional comparisons become computationally prohibitive, rendering a dynamic online implementation intractable.

To resolve this, B-DICS adopts an asymmetric pipeline: we first exploit the static nature of algorithms to construct a compact, highly informative anchor-induced representation, and then perform dynamic instance clustering and selection within this stabilized space. This asymmetry is therefore not a design choice, but a necessity imposed by the fundamentally different observability structures of algorithms and instances.

C.1 Anchor algorithm selection

The goal of anchor selection is to identify a compact subset of algorithms $\mathcal{Z} \subset \mathcal{A}$ that maximally preserves the diversity of instance response patterns.

A key challenge arises from the heterogeneous difficulty of instances. Instances with low difficulty often exhibit large absolute performance ranges, whereas hard instances may show tightly clustered values, with small numerical differences corresponding to qualitatively different outcomes (e.g., infeasible vs. feasible). As a result, selection strategies based directly on raw values, variances, or Euclidean distances tend to be dominated by easy instances, obscuring critical structural signals from difficult ones.

To avoid this bias, B-DICS emphasizes relative behavioral tiers rather than absolute numerical magnitudes.

C.1.1 PREPROCESSING: NATURAL BREAK DISCRETIZATION

We adopt a Natural Break Discretization scheme that treats each instance row $\mathbf{r}_i$ as a one-dimensional clustering problem. The objective is to map continuous performance values into discrete behavioral tiers defined by significant gaps, thereby normalizing instance contributions when assessing algorithmic discrimination.

Let $v^{(i)}$ denote the sorted response vector for instance I_i. We compute adjacent gaps $g_u = v_{u+1}^{(i)} - v_u^{(i)}$ for $u = 1, \ldots, n-1$. A gap is considered a *significant break* if

$$g_u > \tau_i, \quad \tau_i = \max\left(\eta_{\text{range}} \cdot (v_n^{(i)} - v_1^{(i)}), \quad \eta_{\text{gap}} \cdot \frac{1}{n-1}\sum_{u=1}^{n-1} g_u\right) \tag{4}$$

where $v_n^{(i)} - v_1^{(i)}$ is the response range for instance I_i, and $\frac{1}{n-1}\sum_{u=1}^{n-1} g_u$ is the mean adjacent gap. The threshold τ_i is designed to be instance-adaptive by combining two complementary criteria.

The range-based term ηrange $\cdot R_i$ prevents spurious tier splits when the overall response variation is small, while the gap-based term $\eta_{\text{gap}} \cdot \bar{g}_i$ filters out minor fluctuations that arise from noise or near-ties among algorithms. Taking the maximum of the two ensures that a break is introduced only when it is significant both in absolute and relative terms. In our experiments, we set $\eta_{\text{range}} = 0.15$ and $\eta_{\text{gap}} = 4.0$. These values are chosen to strike a conservative balance between over-segmentation and under-segmentation: they suppress trivial discretization on nearly homogeneous instances while reliably capturing pronounced performance separations when they exist.

Based on the detected breaks, the raw matrix $\mathbf{R}$ is transformed into a discrete matrix $\mathbf{L} \in \mathbb{Z}^{M \times n}$. This discretization ensures that transitions between behavioral tiers contribute comparably across instances, regardless of absolute performance scale, enabling fair and robust comparison of algorithmic discrimination power in subsequent anchor selection operations.

C.1.2 ANCHOR SELECTION STRATEGY: GREEDY SEARCH WITH PRUNING

Using the discrete matrix $\mathbf{L}$, we define the discriminative power of an algorithm subset $\mathcal{S}$ as the number of distinct response patterns it induces. Consistent with the definition in the main text, but applied to our preprocessed discrete matrix, this is formalized as:

$$\Phi(\mathcal{S}) = |\mathcal{U}(\mathbf{L}_{\cdot,\mathcal{S}})|, \tag{5}$$

where $\mathbf{L}_{\cdot,\mathcal{S}}$ is the sub-matrix formed by the columns in $\mathcal{S}$, and $\mathcal{U}(\cdot)$ returns the set of unique row vectors in a matrix. A higher $\Phi(\mathcal{S})$ indicates a stronger ability to distinguish between instances.

Interpretation of discriminative power. The metric $\Phi(\mathcal{S})$ can be interpreted as the number of equivalence classes induced over instances by the algorithm subset $\mathcal{S}$. Specifically, two instances are considered equivalent under $\mathcal{S}$ if they exhibit identical response patterns across all algorithms in $\mathcal{S}$. Therefore, maximizing $\Phi(\mathcal{S})$ corresponds to maximizing the resolution of instance partitioning induced by the selected algorithms.

From this perspective, anchor selection can be viewed as a process of progressively refining the partition of the instance space, where each additional algorithm acts as a new probe that potentially splits existing equivalence classes into finer subgroups.

Our objective is to identify a minimal subset $\mathcal{Z}$ that maximizes $\Phi(\mathcal{Z})$. This is achieved via a two-phase procedure.

Phase 1: greedy forward selection. Starting from $\mathcal{Z}_0 = \emptyset$, at each step t we select the algorithm that maximizes the marginal gain in discriminative power:

$$a^* = \arg \max_{a \in \mathcal{A} \setminus \mathcal{Z}_t} \Phi(\mathcal{Z}_t \cup \{a\}). \tag{6}$$

Each selected anchor acts as a new probe that further refines instance distinctions. The process terminates when no candidate increases the number of unique patterns or a predefined budget is reached.

Phase 2: backward pruning. To eliminate redundancy, we iteratively remove anchors whose exclusion does not reduce the overall discriminative power:

$$\mathcal{Z} \leftarrow \mathcal{Z} \setminus \{a\} \quad \text{if} \quad \Phi(\mathcal{Z} \setminus \{a\}) = \Phi(\mathcal{Z}). \tag{7}$$

This step ensures that the anchor set remains compact without sacrificing resolution.

Once $\mathcal{Z}$ is finalized, clustering is performed on the anchor-induced matrix $\bar{\mathbf{R}}$, yielding instance groups that guide the specialized evolution of the algorithm portfolio.

C.2 Instance clustering via anchor-induced representations

Given the finalized anchor set $\mathcal{Z}$, each instance I_i is represented by its anchor-induced response vector

$$\bar{\mathbf{r}}_i = \left(R_{i,z_1}, \ldots, R_{i,z_{|\mathcal{Z}|}}\right),$$

forming the reduced matrix $\bar{\mathbf{R}} \in \mathbb{R}^{M \times |\mathcal{Z}|}$. This representation captures how instances respond to a compact yet maximally discriminative set of algorithmic probes.

Clustering objective. The goal of clustering is not to group instances by geometric proximity in the original feature space, but to identify *algorithmically equivalent* instance groups—instances that induce similar relative performance patterns across the anchor algorithms. Such instances are expected to benefit from similar specialized algorithmic strategies, providing a principled basis for downstream specialist evolution.

Automatic determination of the number of clusters. The number of clusters K is determined adaptively using an elbow method (Syakur et al., 2018). Specifically, we compute the within-cluster sum of squared distances for increasing values of K and detect the point of diminishing returns. This procedure yields a data-driven estimate of K that balances resolution and robustness, avoiding both over-partitioning and under-segmentation of the instance space.

Clustering method. We apply k-means clustering to the rows of $\bar{\mathbf{R}}$ using Euclidean distance. Although simple, this choice is effective because the anchor selection process explicitly maximizes behavioral discrimination, rendering $\bar{\mathbf{R}}$ low-dimensional, well-conditioned, and suitable for centroid-based partitioning.

C.3 Instance selection via anchor-induced representations

Once the instances are partitioned into K clusters, B-DICS performs cluster-aware instance selection to identify a subset of *core instances*, denoted as $\mathcal{I}_{\text{core}}$. The goal is to dramatically reduce the evaluation budget from $\mathcal{O}(M)$ to $\mathcal{O}(|\mathcal{I}_{\text{core}}|)$ while preserving the behavioral diversity and optimization targets of the original training set.

For each cluster C_k, the selection is executed in an iterative, behavior-driven manner:

Initialization via spatial centrality. We first select the instance located closest to the cluster centroid in the anchor-induced space. This instance serves as the foundational representative of the cluster's most typical behavioral pattern:

$$I_{\text{init}}^{(k)} = \arg \min_{I_i \in C_k} \|\bar{\mathbf{r}}_i - \mu_k\|_2, \tag{8}$$

where $\bar{\mathbf{r}}_i \in \mathbb{R}^{|\mathcal{Z}|}$ is the anchor-induced response vector of instance I_i, and μ_k is the centroid of cluster C_k. The initial core set for this cluster is set to $\mathcal{I}_{\text{core}}^{(k)} = \{I_{\text{init}}^{(k)}\}$.

Augmentation via discriminative power. While the centroid captures the typical behavioral pattern within the cluster, relying on a single instance may discard important boundary cases. Following the dual principle used in anchor selection, we augment $\mathcal{I}_{\text{core}}^{(k)}$ by selecting instances that maximize the discriminative power over candidate algorithms. Intuitively, if an instance causes a fixed set of algorithms to exhibit distinct performance patterns, it indicates that this instance provides additional, non-redundant search signals for the evolutionary process.

Specifically, just as anchor selection adds columns to observe row (instance) diversity, instance selection adds rows to capture column (algorithm) diversity within the cluster. We define the discriminative power of an instance subset $\mathcal{W} \subseteq C_k$ as the number of distinct algorithmic performance profiles it can induce across the anchor set $\mathcal{Z}$:

$$\Psi(\mathcal{W}) = |\mathcal{U}((\bar{\mathbf{R}}_{\mathcal{W},\mathcal{Z}})^\top)|, \tag{9}$$

where $(\bar{\mathbf{R}}_{\mathcal{W},\mathcal{Z}})^\top$ is the transpose of the sub-matrix formed by the rows in $\mathcal{W}$ and columns in $\mathcal{Z}$. We iteratively add an instance $I^* \in C_k \setminus \mathcal{I}_{\text{core}}^{(k)}$ that maximizes the updated discriminative power:

$$I^* = \arg \max_{I_i \in C_k \setminus \mathcal{I}_{\text{core}}^{(k)}} \Psi(\mathcal{I}_{\text{core}}^{(k)} \cup \{I_i\}). \tag{10}$$

This greedy augmentation stops when the core subset can distinguish the anchor algorithms as effectively as the full cluster (i.e., $\Psi(\mathcal{I}_{\text{core}}^{(k)}) = \Psi(C_k)$), or when a predefined cluster-specific evaluation budget is met.

The final selected subset is the union of core instances across all clusters: $\mathcal{I}_{\text{core}} = \bigcup_{k=1}^{K} \mathcal{I}_{\text{core}}^{(k)}$. This principled reduction eliminates behaviorally redundant instances while retaining the critical structural signals needed to guide specialized algorithm design.

C.4 Dynamic recalibration during the LES process

B-DICS is designed to evolve jointly with the LES process. As new algorithms are discovered, novel behavioral distinctions may emerge, necessitating periodic updates of anchor algorithms, instance clusters, and the selected core instances.

To maintain scalability, we adopt an **incremental anchor update strategy**. Let $\mathcal{Z}_{t-1}$ be the anchor set obtained in the previous cycle, and let $\mathcal{P}_{\text{new}}$ denote the newly generated candidate algorithms. The effective candidate pool at cycle t is defined as

$$\mathcal{S}_t = \mathcal{Z}_{t-1} \cup \mathcal{P}_{\text{new}}. \tag{11}$$

The greedy selection and pruning procedure described above is then applied solely to $\mathcal{S}_t$.

This design yields three key benefits for anchor updating:

1. **Computational Efficiency:** The candidate space is significantly reduced, ensuring scalability as the algorithm archive grows.
2. **Continuity:** Previously selected anchors are retained unless newly discovered algorithms provide strictly stronger or complementary discriminative capability.
3. **Monotonic Discriminative Power:** Since $\mathcal{Z}_{t-1} \subset \mathcal{S}_t$, the resulting anchor set $\mathcal{Z}_t$ satisfies $|\Phi(\mathcal{Z}_t)| \geq |\Phi(\mathcal{Z}_{t-1})|$, guaranteeing non-decreasing resolution of instance heterogeneity and stable refinement of instance clusters.

Co-evolving Instance Selection. The dynamic update of anchors directly influences the clustering structure. If the inclusion of new anchors causes significant shifts in the cluster assignments, the previously selected $\mathcal{I}_{\text{core}}$ may no longer accurately represent the instance space. To prevent biased or unstable fitness evaluations, B-DICS temporarily suspends instance selection and reverts to evaluating algorithms on the full instance set M. Once the clustering structure stabilizes (e.g., assignments remain unchanged across consecutive updates), B-DICS recalculates and extracts a new set of core instances $\mathcal{I}_{\text{core}}$, seamlessly re-enabling efficient evaluation without misleading the evolutionary search.

D DETAILED DESCRIPTION OF B-DICS-LES

In this section, we provide an exhaustive description of the proposed B-DICS-LES framework, detailing its architectural components, the co-evolving data flow, and the formal implementation procedures.

D.1 FRAMEWORK OVERVIEW AND LIFECYCLE

Algorithm 1: B-DICS-LES Framework

Input: Training instances $\mathcal{I}$, total LLM query budget B_{total}, cycle LLM query budget b
Output: Algorithm Portfolio $\overline{\mathcal{A}} = \mathcal{P}_{\text{comp}} \cup \bigcup \mathcal{P}_k$

```
/* Phase 1: Warm-up                                                    */
```
Initialize active instances $\mathcal{I}_{\text{active}} \leftarrow \mathcal{I}$, complementary pool $\mathcal{P}_{\text{comp}} \leftarrow \emptyset$;
while *warm-up budget not exhausted* **do**
  Generate algorithm a via **Initialization Operator** ; `// cf. Appendix D.2`
  Evaluate a on $\mathcal{I}_{\text{active}}$;
  Register a to complementary pool $\mathcal{P}_{\text{comp}}$;
  Update $\mathcal{P}_{\text{comp}}$ via **CPM** ; `// cf. Appendix D.5`

```
/* Phase 2: Structure Learning and Co-evolutionary Search              */
```
Initialize anchor algorithm set $\mathcal{Z} \leftarrow \emptyset$, Specialist Pools $\{\mathcal{P}_k\} \leftarrow \emptyset$;
while *total budget* B_{total} *not exhausted* **do**
  `// Step 1: Structure Learning (B-DICS module)`
  Set candidate anchors $\mathcal{S} \leftarrow \mathcal{Z} \cup \mathcal{P}_{\text{comp}} \cup \bigcup \mathcal{P}_k$;
  Construct behavior matrix $\mathbf{R}$ by ensuring full-set evaluations for $\mathcal{S}$; `// cf. Appendix C`
  $\mathcal{Z} \leftarrow$ **AnchorSelection**$(\mathcal{S}, \mathbf{R})$; `// cf. Appendix C.1`
  $\{\mathcal{C}_k\}_{k=1}^{K} \leftarrow$ **ClusterInstances**$(\bar{\mathbf{R}})$ using anchor-induced representations ; `// cf. Appendix C.2`
  if *clustering structure is* ***stable*** *;* `// cf. Appendix D.6`
  then
    $\mathcal{I}_{\text{core}} \leftarrow$ **SelectCoreInstances**$(\{\mathcal{C}_k\}_{k=1}^{K}, \bar{\mathbf{R}})$;
    Set $\mathcal{I}_{\text{active}} \leftarrow \mathcal{I}_{\text{core}}$; `// Enable accelerated evaluation`
  else
    Set $\mathcal{I}_{\text{active}} \leftarrow \mathcal{I}$; `// Revert to full evaluation`
    `// Re-initialize pools from current active individuals`
    $\mathcal{P}_{temp} \leftarrow \mathcal{P}_{\text{comp}} \cup \bigcup \mathcal{P}_k$;
    Re-initialize each $\mathcal{P}_k$ by selecting N best individuals from $\mathcal{P}_{temp}$ for cluster $\mathcal{C}_k$;
  `// Step 2: Structure-aware Evolutionary Search Cycle`
  Set cycle query counter $q \leftarrow 0$;
  while $q < b$ **do**
    for *each pool* $\mathcal{P} \in \{\mathcal{P}_1, \ldots, \mathcal{P}_K\} \cup \{\mathcal{P}_{comp}\}$ **do**
      if $q \geq b$ **then**
        break;
      Generate algorithm a from $\mathcal{P}$ via corresponding operator;
      Evaluate a on current $\mathcal{I}_{\text{active}}$;
      Global Registration: Update all $\{\mathcal{P}_k\}$ and $\mathcal{P}_{\text{comp}}$ with a ; `// cf. Appendix D.4`
      $q \leftarrow q + 1$;
  `// Step 3: Supplementary Evaluation for Structural Recalibration`
  Supplement evaluations on $\mathcal{I} \setminus \mathcal{I}_{\text{active}}$ for all algorithms in $\mathcal{P}_{\text{comp}} \cup \bigcup \mathcal{P}_k$;
return $\mathcal{P}_{comp} \cup \bigcup \mathcal{P}_k$

B-DICS-LES is designed to autonomously uncover the latent structure of training instances and exploit this structure to facilitate both efficient evaluation and specialized algorithm design. The framework operates as a co-evolving loop that alternates between two primary phases:

- **Structure Learning:** Based on the growing behavior matrix $\mathbf{R}$, this phase updates the anchor set, clusters instances, and selects core instances to refine the system's understanding of the instance space.
- **Structure-aware Evolutionary Search:** This phase utilizes the learned structure to guide LLM-assisted search through specialized and complementary algorithm pools, functioning as the engine for algorithm design.

The framework maintains an active instance set $\mathcal{I}_{\text{active}}$, a **Complementary Pool** ($\mathcal{P}_{\text{comp}}$), and multiple **Cluster-specific Specialist Pools** ($\{\mathcal{P}_k\}_{k=1}^{K}$). Algorithm 1 formalizes the integration of these components. For clarity, we define an *Evolutionary Cycle* as the interval between two B-DICS recalibrations, during which a fixed budget (b) of LLM queries is consumed. To ensure conciseness, we assume that every generated algorithm a is automatically registered into a **Global Archive** $\mathcal{A}$, which serves as a persistent database of all discovered solutions.

D.2 Algorithm Generation and LLM Operators

B-DICS-LES leverages LLMs as stochastic search operators. To maximize search efficiency, we employ distinct prompting strategies tailored to the specific search dynamics of each pool.

Initialization Operator (Warm-up) The resolving power of the B-DICS module depends on the behavioral diversity of algorithms. During warm-up, we prompt the LLM to generate algorithms using diverse algorithmic paradigms. Previously generated successful codes are provided as in-context examples to encourage the LLM to explore novel "response patterns" rather than minor syntax variations.

Operators in Specialist Pools Specialist pools focus on local exploitation within a cluster $\mathcal{C}_k$.

- **Specialized Mutation:** Refines a single parent from $\mathcal{P}_k$ by identifying and fixing bottlenecks specific to the cluster's instances.
- **Inter-cluster Crossover:** Combines a parent from $\mathcal{P}_k$ with a high-performing parent from a different specialist pool $\mathcal{P}_j$. This allows specialists to adapt successful general techniques to their specific niche.

Operators in Complementary Pools The goal of the complementary pool is to maximize instance-wise coverage.

- **Complementary Crossover:** Selects parents with high instance-wise complementarity (i.e., those that excel on disjoint subsets of $\mathcal{I}$), prompting the LLM to synthesize their distinct technical strengths into a single, robust unified solution.
- **Task-specific Mutation:** Refines a single parent by identifying and addressing algorithmic bottlenecks specific to the overarching design task rather than local instance features.

D.3 Specialist Pool Re-initialization and Warm-start

A crucial re-alignment occurs whenever B-DICS detects a significant structural shift or at the start of the first co-evolutionary cycle. To "warm-start" the specialist pools, we aggregate current active individuals $\mathcal{P}_{temp} = \mathcal{P}_{\text{comp}} \cup \bigcup \mathcal{P}_k$. For each new cluster $\mathcal{C}_k$, we calculate the average performance of candidates in $\mathcal{P}_{temp}$ specifically on instances $I_i \in \mathcal{C}_k$ and use fitness-proportionate selection to populate $\mathcal{P}_k$ up to the pool size N. This ensures that every specialized search begins from the most promising known "seeds" for that region of the instance space.

D.4 Evaluation Strategy and Global Registration

Accelerated vs. Full Evaluation Algorithms are evaluated on $\mathcal{I}_{\text{active}}$. When clustering is unstable, $\mathcal{I}_{\text{active}} = \mathcal{I}$ to ensure robust structure learning. Upon stabilization, the transition to $\mathcal{I}_{\text{core}}$ reduces evaluation complexity from $\mathcal{O}(|\mathcal{I}|)$ to $\mathcal{O}(|\mathcal{I}_{\text{core}}|)$, significantly accelerating the search cycle.

Global Registration and Knowledge Sharing Every candidate algorithm a, regardless of its originating pool, is registered globally. This ensures that a is considered for inclusion in all $\{\mathcal{P}_k\}$ and $\mathcal{P}_{\text{comp}}$, thereby participating in subsequent pool management and evolution. This facilitates *cross-pool utilization*, where an algorithm generated for Pool A might inadvertently solve a bottleneck in Pool B, maximizing the utility of every LLM query and preventing the discarding of "incidental experts."

Supplementary Evaluation for Structural Recalibration. To maintain the precision and resolution of the behavior matrix $\mathbf{R}$, algorithms surviving an evolutionary cycle in $\mathcal{I}_{\text{active}}$ undergo supplementary evaluation on the omitted instances $\mathcal{I} \setminus \mathcal{I}_{\text{core}}$. This step ensures that the B-DICS module in the subsequent iteration has access to full-set data.

The maximum computational cost for this step is $N \times (K+1)$ instance evaluations per cycle (typically much less due to overlaps between pools). Given that $N \times (K+1) \ll b \times |\mathcal{I}|$, this strategy achieves a significant evaluation saving while preserving high structural resolution for the B-DICS module.

D.5 Pool Management and Synergistic Dynamics

Pool management strategies serve as the core driving force guiding the evolutionary direction and represent the fundamental distinction between specialist and complementary pools.

Specialist Management $\mathcal{P}_k$ is governed by strict local elitism. It maintains a population of algorithms that achieve the highest mean performance on instances within cluster $\mathcal{C}_k$, driving the search toward extreme specialization.

Complementary Pool Management (CPM) $\mathcal{P}_{\text{comp}}$ utilizes the CPM principle (Liu et al., 2025) to promote instance-wise complementarity. Rather than selecting based on average performance, CPM favors algorithms that offer a high **Complementary Performance Index (CPI)**. Given a pool P_t and a candidate a, the marginal CPI contribution is:

$$\Delta\text{CPI}(a \mid P_t) = \sum_{I_m \in \mathcal{I}} \max\left(f_m^{(}P_t) - f_m(a), 0\right), \tag{12}$$

where $f_m^{(}P_t)$ is the best score achieved on instance I_m by the current pool. By greedily selecting algorithms with the largest ΔCPI, CPM ensures that the complementary pool remains robust across diverse instance types.

Synergy This architecture creates a "trunk-and-branch" dynamic. The complementary pool evolves robust, general-purpose algorithmic "trunks," while specialist pools explore highly optimized "branches." Specialist pools naturally provide diverse material for the complementary pool, which in turn generates breakthrough combinations that can be re-specialized, ensuring a balance between global robustness and local excellence.

D.6 Stability Criteria for Evaluation Acceleration

We quantify structural stability using the Adjusted Rand Index (ARI) to compare cluster assignments across cycles. Acceleration (transitioning to $\mathcal{I}_{\text{core}}$) is triggered when $\text{ARI} > 0.9$ for one consecutive recalibration cycle. This guardrail prevents the search from being misguided by biased or premature instance subsets.

E Task descriptions and heterogeneous instance set construction

We evaluate B-DICS-LES across three representative automated algorithm design tasks spanning routing, packing, and online decision-making problems. Each task is constructed with explicitly

heterogeneous instance distributions to assess the framework's ability to discover latent instance structures and to support bias-aware specialized evolution.

E.1 CAPACITATED VEHICLE ROUTING PROBLEM (CVRP)

Problem definition. The CVRP generalizes the TSP to a multi-vehicle setting with explicit capacity constraints. Given a depot v_0 and a set of customers $\{v_1, \ldots, v_n\}$, each customer v_i is associated with a location $\mathbf{x}_i \in [0, 1]^2$ and an integer demand $d_i \in \mathbb{Z}^+$, where $d_0 = 0$. Each vehicle has a fixed capacity $Q \in \mathbb{Z}^+$, and the travel cost between any two locations is defined as the Euclidean distance $c_{ij} = \|\mathbf{x}_i - \mathbf{x}_j\|_2$.

The objective is to find a set of routes $\mathcal{R} = \{r_1, \ldots, r_m\}$ such that each route starts and ends at the depot v_0, every customer is served exactly once, and the total demand along each route satisfies the capacity constraint $\sum_{v_i \in r_k} d_i \leq Q$. The goal is to minimize the total travel distance of all routes.

Designed algorithm. B-DICS-LES aims to design a constructive routing heuristic. The target algorithm incrementally constructs routes by selecting the next customer to serve at each step, conditioned on the current vehicle location, the set of unserved customers, and the remaining vehicle capacity. The performance of the designed algorithms is evaluated using the optimality gap relative to LKH-3.

Instance generation protocol. To induce diverse logistical scenarios and heterogeneous algorithmic responses, we generate CVRP instances with varying spatial and demand characteristics:

- **Uniform.** Customer coordinates are sampled uniformly from $[0, 1]^2$, with demands drawn from a uniform distribution $U[1, 9]$. This setting serves as a balanced baseline.
- **Clustered.** Customers are generated around 3 to 6 randomly sampled centroids using a Gaussian mixture model with standard deviation $\sigma = 0.06$. This configuration simulates geographically concentrated delivery regions.
- **Heavy Demand.** Customer coordinates are sampled uniformly, while demands are drawn from $U[0.4Q, 0.7Q]$. In this regime, vehicles can typically serve only one or two customers per route, shifting the optimization emphasis from routing efficiency to capacity-aware packing.

Dataset composition. By jointly varying the problem scale (number of customers n), capacity constraints (Q), and instance distributions, we construct a total of 12 CVRP datasets. For each dataset, 20 instances are used for training and 10 instances for testing. The detailed specifications of all datasets are summarized in Table 16.

Table 16: Specification of the heterogeneous CVRP datasets. The benchmark covers varying node scales and capacity constraints.

Group ID	Config	Distribution	Size (n)	Cap (Q)	Training IDs	Testing IDs
1	Baseline	Clustered	30–60	30–60	$[0, 20)$	$[0, 10)$
2		Heavy	30–60	30–60	$[80, 100)$	$[40, 50)$
3		Uniform	30–60	30–60	$[160, 180)$	$[80, 90)$
4	Scale Up	Clustered	80–120	30–60	$[20, 40)$	$[10, 20)$
5		Heavy	80–120	30–60	$[100, 120)$	$[50, 60)$
6		Uniform	80–120	30–60	$[180, 200)$	$[90, 100)$
7	Large	Clustered	180–200	30–60	$[40, 60)$	$[20, 30)$
8		Heavy	180–200	30–60	$[120, 140)$	$[60, 70)$
9		Uniform	180–200	30–60	$[200, 220)$	$[100, 110)$
10	High Cap	Clustered	80–120	90–110	$[60, 80)$	$[30, 40)$
11		Heavy	80–120	90–110	$[140, 160)$	$[70, 80)$
12		Uniform	80–120	90–110	$[220, 240)$	$[110, 120)$

E.2 Online Bin Packing (OBP)

Problem definition. The Online Bin Packing (OBP) problem considers packing a sequence of items into a minimum number of bins, each with a fixed capacity $C \in \mathbb{R}^+$. Items arrive sequentially and must be irrevocably assigned to bins upon arrival, without knowledge of future items.

Formally, let $I = (i_1, i_2, \ldots, i_n)$ denote a sequence of n items, where each item i_j has a size $s_j \in (0, C]$. A bin is a container with capacity C, and a packing is feasible if, for every bin B_k,

$$\sum_{i_j \in B_k} s_j \leq C. \tag{13}$$

The objective is to minimize the total number of bins used:

$$K = |\{B_1, B_2, \ldots, B_K\}|. \tag{14}$$

In the online setting, the following constraints apply: (i) items arrive one-by-one in an unknown order; (ii) each item must be assigned immediately upon arrival; and (iii) previously packed items cannot be moved or reassigned.

Designed algorithm. B-DICS-LES aims to design online packing heuristics that compute a priority score over currently open bins for each incoming item, conditioned on the item size and the residual capacities of existing bins. The item is then assigned to the highest-priority bin, or to a new bin if no feasible bin exists. Algorithm performance is evaluated by the excess ratio over the theoretical lower bound on the minimum number of bins required (Martello & Toth, 1990).

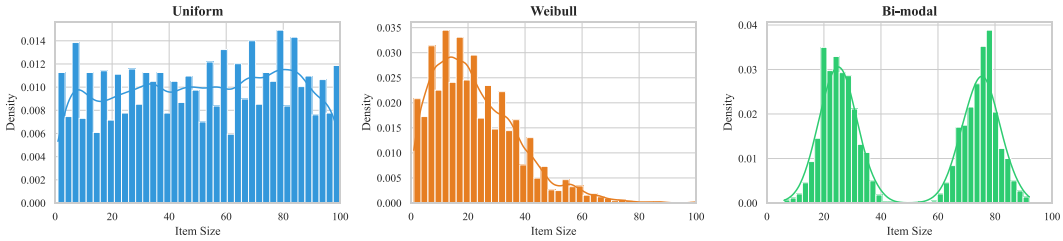


Figure 6: Illustration of the three item size distributions used in the OBP task: **Uniform**, **Weibull**, and **Bi-modal**. These distributions induce qualitatively different online packing behaviors and decision trade-offs.

Instance generation protocol. To induce heterogeneous online decision-making challenges, we generate OBP instances using three distinct item size distributions with a fixed bin capacity $C = 100$:

- **Uniform.** Item sizes are sampled uniformly from $[1, C]$. This serves as a neutral baseline distribution.
- **Weibull.** Item sizes follow a Weibull distribution with shape parameter $k = 1.5$ and scale $\lambda = 0.25C$. This setting mimics long-tailed real-world data, where small items dominate but large items occasionally appear.
- **Bi-modal.** Item sizes are sampled from two distinct modes: small items in $[0.1C, 0.2C]$ and large items in $[0.8C, 0.9C]$. This "dumbbell" distribution challenges algorithms to effectively pair complementary items rather than greedily filling bins.

Figure 6 visualizes these three item size distributions.

Dataset composition. The OBP benchmark consists of 12 datasets covering four distinct sequence length regimes, ranging from short to very long horizons. Consistent with the CVRP tasks, each dataset contains 20 training instances and 10 test instances. Detailed specifications of all datasets are summarized in Table 17.

E.3 Lunar Lander Control (LLC)

Problem definition. The Lunar Lander Control (LLC) task considers a control problem in which an agent must safely land a spacecraft on a designated landing pad under gravity, using limited thrust

Table 17: Specification of the heterogeneous OBP datasets. The benchmark evaluates performance across varying stream lengths and item size distributions.

Group ID	Horizon	Distribution	Length (n)	Training IDs	Testing IDs
1	Short	Uniform	200–500	$[0, 20)$	$[0, 10)$
2		Weibull	200–500	$[80, 100)$	$[40, 50)$
3		Bi-modal	200–500	$[160, 180)$	$[80, 90)$
4	Medium	Uniform	800–1000	$[20, 40)$	$[10, 20)$
5		Weibull	800–1000	$[100, 120)$	$[50, 60)$
6		Bi-modal	800–1000	$[180, 200)$	$[90, 100)$
7	Long	Uniform	1200–1500	$[40, 60)$	$[20, 30)$
8		Weibull	1200–1500	$[120, 140)$	$[60, 70)$
9		Bi-modal	1200–1500	$[200, 220)$	$[100, 110)$
10	Very Long	Uniform	1700–2000	$[60, 80)$	$[30, 40)$
11		Weibull	1700–2000	$[140, 160)$	$[70, 80)$
12		Bi-modal	1700–2000	$[220, 240)$	$[110, 120)$

actions. The environment is adapted from the standard LunarLander benchmark (Gadgil et al., 2020). At each time step t, the lander is described by a continuous state vector $\mathbf{s}_t = (x_t, y_t, \dot{x}_t, \dot{y}_t, \theta_t, \dot{\theta}_t)$, which represents the lander's position, velocity, orientation, and angular velocity. The agent selects control actions corresponding to the main engine thrust and side engine torques, which jointly influence the lander's translational and rotational dynamics. The objective is to control the lander to a smooth touchdown on the landing pad, minimizing fuel consumption while avoiding crashes or excessive impact velocity.

Designed algorithm. B-DICS-LES aims to design reactive control policies that map the current state $\mathbf{s}_t$ to continuous control actions. Rather than learning a neural policy end-to-end, LES searches over programmatic policy representations that compute thrust commands based on interpretable, state-dependent rules. The quality of a candidate controller is evaluated by executing it from a given initial state and measuring the cumulative landing reward, which penalizes crashes, hard landings, and excessive fuel usage while rewarding stable and centered touchdowns. A landing episode with cumulative reward exceeding 200 is regarded as a safe, efficient, and successful landing (Hu et al., 2025a).

Dataset composition. To explicitly introduce heterogeneity in control difficulty, we construct LLC instances by varying the lander's initial state. The training set contains 35 instances covering a wide range of initial velocities, and orientations. These initial conditions induce qualitatively different uncontrolled descent trajectories, as illustrated in Figure 7. Some instances require highly aggressive and early thrust corrections to prevent divergence, while others admit smoother and more fuel-efficient control strategies. The testing set consists of 50 instances whose initial states are fully covered by the training distribution, as visualized in Figure 8.

F Prompts used in experiments

This section documents all prompt templates used in our experiments, including task-specific prompts and prompts for different evolutionary operators. Our goal is to provide a transparent and reproducible description of the interaction interface between the LLM and the evolutionary search process, while abstracting away task-internal details that are not central to the proposed methodology.

F.1 Task-specific prompts

For each task, we employ a concise task description prompt that specifies the problem setting and the expected form of the algorithmic solution. These prompts serve solely to define the optimization objective and the step-wise decision structure, without injecting any domain-specific heuristics or solution strategies. The task-specific prompts used for CVRP, OBP, and LLC are shown in Figures 9–11, respectively.

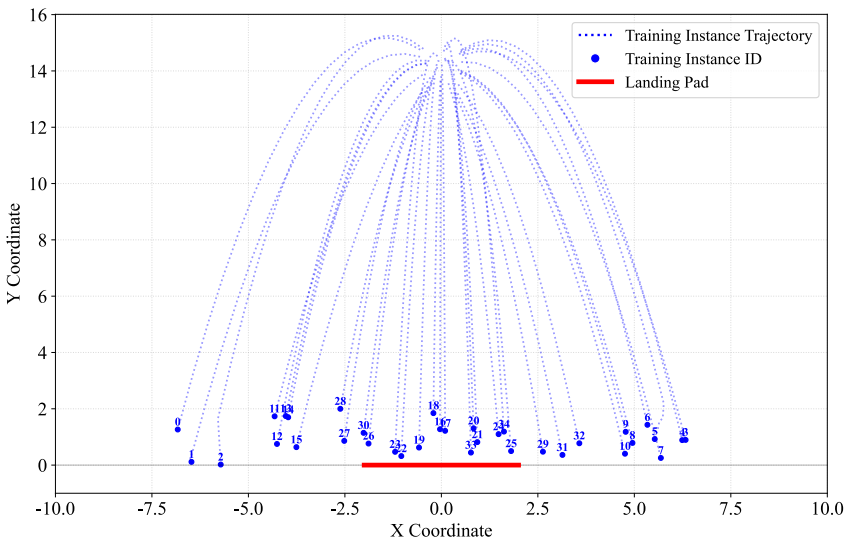


Figure 7: Uncontrolled descent trajectories of the 35 training instances in the LLC task. Different initial states lead to distinct descent patterns, reflecting varying levels of control difficulty and necessitating control policies with different degrees of aggressiveness.

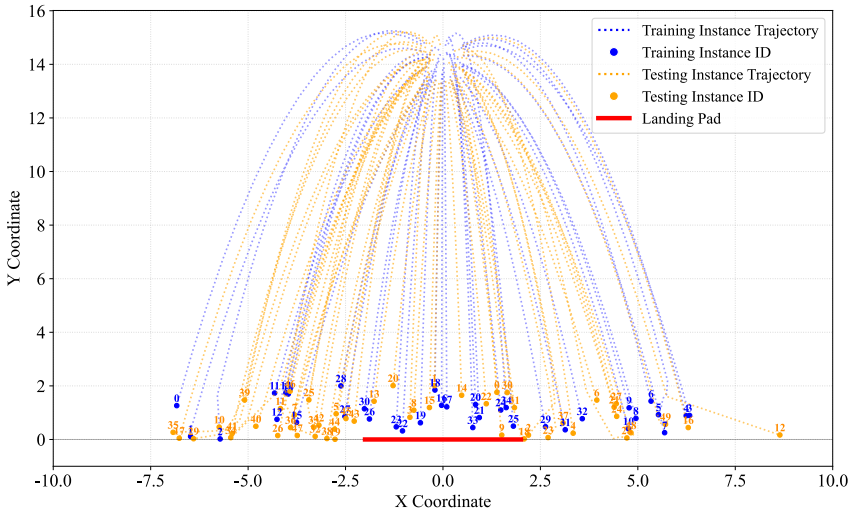


Figure 8: Initial state distribution of the 50 testing instances in the LLC task. All testing states are contained within the coverage of the training distribution, ensuring evaluation under interpolative yet heterogeneous control conditions.

> Given a set of customers and a fleet of vehicles with limited capacity, the task is to design a novel algorithm to select the next node in each step, with the objective of minimizing the total cost.

Figure 9: Task description prompt used for the CVRP.

F.2 Prompt templates for evolutionary operators

In LES methods, different evolutionary operators are implemented through distinct prompt templates that guide the LLM toward specific search behaviors. To ensure fairness and isolate the contribution of the proposed mechanism rather than prompt engineering, B-DICS-LES adopts exactly

Implement a function that returns the priority with which we want to add an item to each bin.

Figure 10: Task description prompt used for the OBP.

Implement a novel heuristic strategy function that guides the lander in selecting actions step-by-step to achieve a safe landing. At each step, an appropriate action could be chosen based on the lander's current state and previous state, with the objective of reaching the target location in as few steps as possible. A 'safe landing' is defined as a touchdown with low vertical speed, upright orientation, and both angular velocity and angle close to zero, and both legs in contact with the ground.

Figure 11: Task description prompt used for the LLC task.

the same operator prompts as EoH-S for the evolution of the complementary pool, including the Complementary-aware Search (CS) operator and the Local Search (LS) operator (Liu et al., 2025), shown in Figures 13 and 14.

For the specialist pools, we design two operators: a mutation operator and a crossover operator. The mutation operator is identical to the LS operator used in the complementary pool. The crossover operator selects one parent from the current pool and another from a different pool, and encourages comparative analysis between their behaviors to enhance specialization toward the current pool. The corresponding prompt templates for these two operators are shown in Figures 14 and 15, respectively.

The initialization prompt used to generate the initial pool is shown in Figure 12. Across all prompt templates, black text denotes fixed instructions, red placeholders indicate task-specific components, and blue placeholders represent algorithmic content that evolves throughout the evolutionary process.

"[Task Description Placeholder]"
Based on your expertise, please create a novel and efficient algorithm to solve this problem.
So far, experts have proposed **"[Value Placeholder]"** algorithms. Their high-level ideas are summarized below. The No. 1 algorithm and the corresponding code are:
"[Algorithm Description Placeholder]"
"[Code Placeholder]"
The No. 2 algorithm and the corresponding code are:
"[Algorithm Description Placeholder]"
"[Code Placeholder]"
...
Your goal is to design a new algorithm whose core idea differs from *all existing ones* by at least 30%.
Please design a new algorithm following the instructions below:
1. Describe your concept for the new algorithm and its main steps in as few words as possible while ensuring clarity, and enclose it **exactly** within the markers '≪' and '≫', like this: ≪Your idea here≫.
2. Implement your proposed algorithm using the following Python function template:
"[Code Template Placeholder]"

Figure 12: Prompt template for Initialization.

G LICENSES FOR USED ASSETS

Since Funsearch is not officially open-source, we conducted our experiments using the version of Funsearch provided on the LLM4AD platform.

"[Task Description Placeholder]"
I have "[Value Placeholder]" existing algorithms with their codes as follows:
The No. 1 algorithm and the corresponding code are:
"[Algorithm Description Placeholder]"
"[Code Placeholder]"
The No. 2 algorithm and the corresponding code are:
"[Algorithm Description Placeholder]"
"[Code Placeholder]"
These algorithms are effective for solving different instance distributions. Please help me create a new algorithm that is different from the given ones.
1. First, describe your new algorithm and main steps in one concise sentence. Enclose your sentence **exactly** within the markers '≪' and '≫' like this: ≪Your idea here≫.
2. Next, implement the algorithm in the following Python function:
"[Code Template Placeholder]"

Figure 13: Prompt template for the CS operator.

"[Task Description Placeholder]"
I have one algorithm with its code as follows.
Algorithm description:
"[Algorithm Description Placeholder]"
Code:
"[Code Placeholder]"
Please assist me in creating an improved version of the algorithm provided.
1. First, describe your new algorithm and main steps in one sentence. Enclose your sentence **exactly** within the markers '≪' and '≫' like this: ≪Your idea here≫.
2. Next, implement the following Python function:
"[Code Template Placeholder]"

Figure 14: Prompt template for the LS operator.

Table 18: Asset Usage and License Information

Type	Asset	License	Usage
Code	EoH (Liu et al., 2024a)	MIT License	Evaluation
	ReEvo (Ye et al., 2024)	MIT License	Evaluation
	PartEvo (Hu & Zhang, 2025)	MIT License	Evaluation
	EoH-S (Liu et al., 2025)	MIT License	Evaluation
Dataset	LLM4AD (Liu et al., 2024b)	MIT License	Testing

Your task is: **"[Task Description Placeholder]"**
There are **"[Value Placeholder]"** existing algorithms with their codes as follows:
The No. 1 algorithm and the corresponding code are:
"[Algorithm Description Placeholder]"
"[Code Placeholder]"
The No. 2 algorithm and the corresponding code are:
"[Algorithm Description Placeholder]"
"[Code Placeholder]"
...
Empirical testing confirms that these algorithms are complementary: they excel at solving different instance distributions, although these distributions are NOT explicitly given.
Your objective is to perform **code-driven reasoning** to infer these hidden differences and evolve a stronger specialized algorithm. Please follow the steps below.
=== Step 1: Contrastive and Root-Cause Analysis (The "Why") ===
Carefully compare the core concepts and source codes of the provided algorithms.
- Identify the explicit differences between them (e.g., heuristic logic, parameters, or search behaviors).
- Focus on the most critical differences that could lead to different behaviors.
- Based on these differences, deduce *WHY* No. **"[Value Placeholder]"** suits one specific type of instance distribution, while the others suit different distributions.
=== Step 2: Targeted Enhancement (The "How") ===
Based on your understanding, improve **"[Value Placeholder]"** in the direction where it already seems strong.
- Do NOT make it a mediocre generalist.
- Instead, use the insights from your contrastive analysis to aggressively specialize and refine the No. **"[Value Placeholder]"** algorithm, amplifying its inherent strengths so it can better dominate its specific niche.
=== Step 3: Propose and Implement the Enhanced Algorithm ===
1. First, describe your enhanced algorithm's core concept and main steps in concise sentences. Enclose your sentences **exactly** within the markers '¡¡' and '¿¿' like this: ¡¡Your idea here¿¿.
2. Next, implement the enhanced algorithm in the following Python function:
"[Code Template Placeholder]"

Figure 15: Prompt template for the Crossover operator.