MULTI-TASK EVOLUTION FOR ZERO-SHOT CROSS-PROBLEM GENERALIZATION USING LLMS

Anonymous authors
Paper under double-blind review

ABSTRACT

Designing effective heuristics for diverse combinatorial optimization problems requires substantial expertise and repeated search. Large language models (LLMs) automate heuristic generation and refinement, but heuristic search typically depends on evaluation feedback from the problem being optimized. Generalizing to new problem definitions using only source-task feedback therefore remains a central challenge. We introduce MECo, an LLM-driven multi-task evolutionary framework for zero-shot cross-problem generalization. MECo maintains task-conditioned heuristic populations and uses a transfer gap based on cross-task population performance to guide their interactions. These interactions enable the transfer and recombination of heuristics. A complementary selection criterion then constructs a compact heuristic set by rewarding each member's additional coverage of source combinations. The selected set is applied to target problems without further search or adaptation. Experiments on 32 problem variants across vehicle routing (VRP) and flexible job-shop scheduling (FJSP) show that MECo achieves the lowest mean costs compared with eight automated heuristic design (AHD) baselines under the same budgets. On out-of-domain problems, it outperforms the strongest baseline in each family. Moreover, integrating the framework of MECo with different AHD methods improves their ID and OOD performance in both families, supporting its effectiveness across different methods.

1 INTRODUCTION

Combinatorial optimization problems (COPs) arise in a wide range of logistics and manufacturing applications (Toth & Vigo, 2014). Changes in operational requirements can introduce new problem variants, often requiring existing heuristics to be adapted or redesigned. This process demands domain expertise and repeated cycles of design and evaluation, making heuristic development a recurring cost. Large language models (LLMs) offer a promising way to reduce this burden through automated heuristic design (AHD) (Liu et al., 2026b). Given a problem description and evaluation feedback, these methods iteratively generate and refine executable heuristics. We distinguish task-specific AHD, which designs heuristics for each problem independently, from cross-task AHD, which jointly develops heuristics for multiple problems. Figure 1 compares these design paradigms alongside our framework.

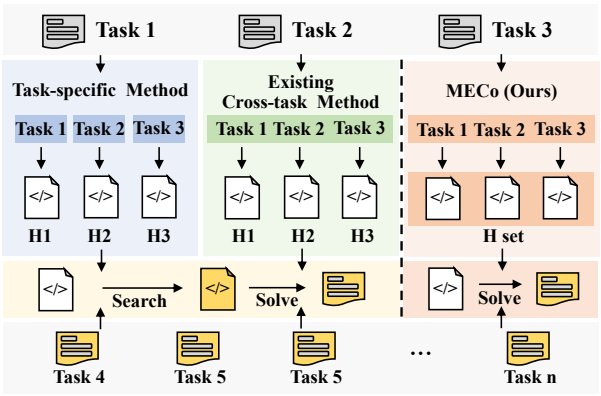

Figure 1: Comparison of heuristic design paradigms. MECo evolves a heuristic set for zero-shot cross-problem generalization.

We define a task as the design of heuristics for a particular optimization problem. Task-specific AHD methods have made substantial progress in automating this process. Population-based evolutionary search combines LLM-driven program generation with fitness-based selection (Liu et al., 2024b;

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.

Romera-Paredes et al., 2024). Reflection further guides program generation using performance feedback (Ye et al., 2024). Other methods balance exploration and exploitation through diversity-oriented evolution with harmony-search refinement (Dat et al., 2025) or Monte Carlo tree search (Zheng et al., 2025). These advances improve heuristic discovery within individual tasks. However, applying the same design framework to another problem typically requires another round of problem-specific search. Without systematic reuse of previously discovered heuristics, even closely related problems can incur repeated design costs.

Cross-task AHD addresses this repetition by jointly developing heuristics for multiple problems. MoH learns reusable heuristic optimizers (Shi et al., 2026), while MTHS jointly evolves general-purpose metaheuristics and task-specific programs (Liu et al., 2026c). However, jointly designing heuristics for multiple problems does not ensure that the resulting programs can be applied directly to new problems without additional search. In practice, operational requirements can be combined in many ways, and each new combination can define a distinct optimization problem. This creates a potentially large family of related design tasks. If each new task requires a fixed search budget, the total design cost grows with the number of tasks. Moreover, evaluation data for new combinations may be unavailable. This motivates designing heuristics using feedback from a limited set of source tasks and applying them directly to new problems without additional heuristic search.

To address this challenge, we propose MECo, an LLM-driven framework that uses Multi-task Evolution through task Combination to enable zero-shot cross-problem generalization. Starting from source tasks, MECo progressively develops new populations for combined task specifications. Their offspring inherit and adapt parent heuristics through evolutionary operators designed for cross-task knowledge transfer. These offspring are evaluated separately on the available datasets, allowing joint feedback to guide heuristic design for combined problems without accessing their instances. To guide interactions among the growing collection of populations, a task transfer gap measures how well the heuristics from one population perform on the sources associated with another. MECo uses this measure to favor compatible parent tasks while exploring new combinations. Finally, it selects a small set from all task populations according to the complementary performance of its members across combinations of sources. Both evolution and set selection rely exclusively on source data, and the selected programs are applied to target problems without further search or adaptation.

Our contributions are summarized as follows.

- We introduce MECo, a multi-task evolutionary framework that develops complementary heuristics for zero-shot cross-problem generalization using source task feedback alone.
- We introduce a population-level transfer gap to quantify empirical task compatibility and guide heuristic transfer between task populations.
- Experiments on 32 variants across vehicle routing (VRP) and flexible job-shop scheduling (FJSP) show lower mean costs on unseen problems than eight AHD baselines under matched proposal budgets. Integrating the framework with three existing AHD methods further improves their performance.

2 PRELIMINARY

2.1 OPTIMIZATION PROBLEMS AND HEURISTIC-DESIGN TASKS

An optimization problem P defines a space of instances $\mathcal{X}_P$. For each instance $x \in \mathcal{X}_P$, let $\mathcal{Y}_P(x)$ be its feasible solution set and $y \in \mathcal{Y}_P(x)$ a feasible solution. The objective function $F_P(x, y)$ assigns a cost to that solution. We use minimization objectives throughout. Therefore, solving an instance means finding

$$y_P^\star(x) \in \arg\min_{y \in \mathcal{Y}_P(x)} F_P(x, y). \tag{1}$$

A heuristic-design task T_P seeks a heuristic for solving instances of P. Let $\mathcal{A}$ denote the heuristic space and $D_{T_P} \subset \mathcal{X}_P$ the evaluation dataset associated with the task. For $h \in \mathcal{A}$, write $\ell(h, x)$ for the objective cost of its solution on x. The single-task AHD objective is to find

$$h_{T_P}^\star \in \arg\min_{h \in \mathcal{A}} \frac{1}{|D_{T_P}|} \sum_{x \in D_{T_P}} \ell(h, x). \tag{2}$$

The optimization variable here is the heuristic, which is evaluated across problem instances. In LLM-based AHD, heuristics are represented as programs proposed and refined by an LLM.

2.2 Source-only design for cross-problem generalization

We consider source tasks $\mathcal{T}_{\text{src}}$ and target tasks $\mathcal{T}_{\text{tar}}$ whose underlying optimization problems differ with $\mathcal{T}_{\text{src}} \cap \mathcal{T}_{\text{tar}} = \varnothing$. During heuristic design, evaluation feedback is available only from D_T for $T \in \mathcal{T}_{\text{src}}$. Target datasets are kept until testing.

The goal is to obtain a set $\mathcal{H} \subseteq \mathcal{A}$ heuristics that performs well across target problems using only the feedback from the source tasks. Within each problem family, the heuristics share a decision representation that permits their application across tasks. Zero-shot evaluation applies the resulting set to target problems without further heuristic search or task-specific adaptation. This setting assesses generalization across out-of-domain problem definitions. Evaluation of new instances of source problems additionally measures the generalization of the in-domain geberakuzatuin.

2.3 Compositional problem families

We instantiate this setting through related problems defined by constraint attributes. Let P_0 be a base problem and $\mathcal{C} = \{c_1, \ldots, c_m\}$ an attribute library. A subset $C \subseteq \mathcal{C}$ specifies a problem P_C, with $P_\varnothing = P_0$, and its design task $T_C = T_{P_C}$. Let $T_j = T_{\{c_j\}}$ denote a single-attribute task. The source and target sets are

$$\begin{aligned} \mathcal{T}_{\text{src}} &= \{T_j \mid 1 \le j \le m\}, \\ \mathcal{T}_{\text{tar}} &= \{T_C \mid C \subseteq \mathcal{C},\ |C| \neq 1\}. \end{aligned} \tag{3}$$

The targets comprise the base problem and combinations of two or more attributes. We call the source tasks roots and write $D_j = D_{T_j}$ when indexing their datasets.

We study VRP and FJSP as two independent problem families under this formulation. Section 4.1 specifies their attributes and evaluation splits. Appendix C gives the complete problem definitions.

3 Method

3.1 Generalization Through Joint Training

Joint source training can yield stronger heuristics for a combined problem than separate source training. We illustrate this phenomenon in VRP, where O, TW, B, and L label constraint attributes. Two parent populations evolve on their respective source datasets. An offspring population inherits their heuristics and evolves using both sources, with the same number of additional LLM-generated heuristic proposals for each branch. Figure 2 compares the best heuristic from each branch on the same combined problems and shows that joint source feedback can improve generalization. This observation motivates exchanging heuristics between task populations.

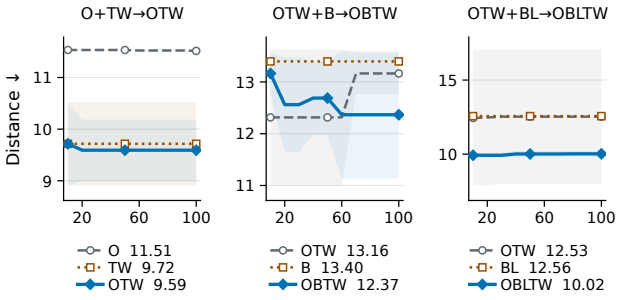

Figure 2: A case of generalization to combined VRP variants under separate and joint source training. Curves show the mean cost of the best heuristic in each population. Bands show sample standard deviations.

3.2 Framework Overview

MECo develops heuristics for new task specifications by bringing together populations from existing tasks. For each explored task T_C, the population $\mathcal{P}_C$ contains heuristics intended for P_C. The

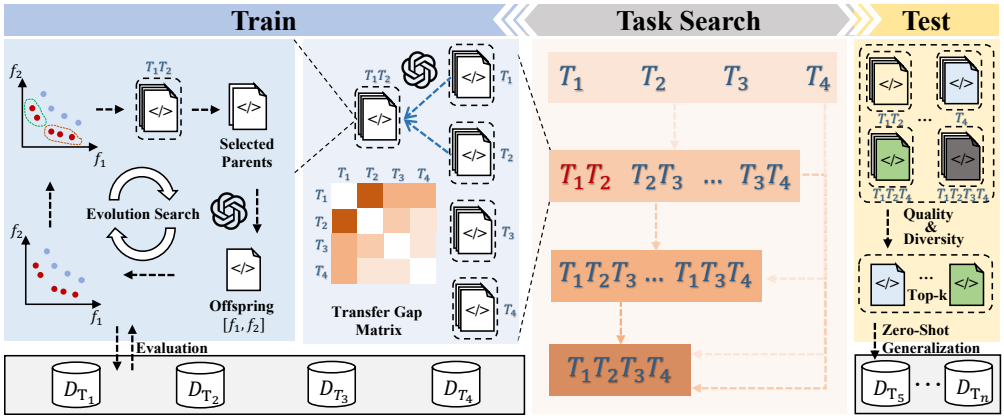

Figure 3: MECo couples transfer-guided task interactions with heuristic evolution. Each node represents a task-conditioned population. Source datasets D_{T_j} guide training and final set selection. The selected heuristics are frozen before zero-shot target evaluation.

task description guides what these heuristics should solve, and evaluations on its constituent sources guide their selection. Search begins by evolving one population for each source task $T_1, \ldots, T_m$. Each subsequent expansion chooses two existing populations and evolves offspring for their combined task. The resulting population can participate in later expansions.

Figure 3 presents the two search levels. Task-level search determines which populations interact using their measured transfer gaps. Within the chosen task, program-level evolution generates and selects heuristics. The source datasets D_{T_j} remain fixed throughout both levels. After search, MECo selects a small set $\mathcal{H}$ whose members perform well on different combinations of sources, then freezes it for target evaluation. In the figure, T_iT_j abbreviates the combined task $T_{\{c_i,c_j\}}$.

3.3 Transfer-Guided Evolution

Source feedback and task quality. We evaluate heuristics through normalized source costs and average them over the sources associated with each task,

$$f_j(h) = \frac{|D_{T_j}|^{-1} \sum_{x \in D_{T_j}} \ell(h, x)}{b_j}, \qquad J_C(h) = \frac{1}{|C|} \sum_{j \mid c_j \in C} f_j(h), \tag{4}$$

where $f_j(h)$ is the normalized cost on source task T_j, $J_C(h)$ measures the quality of h for task T_C, and $b_j > 0$ is the mean cost of a simple heuristic on D_{T_j}, fixed throughout evolution.

The vector $\mathbf{f}(h) = (f_1(h), \ldots, f_m(h))$ records a heuristic's performance across sources. A source population is evaluated on its own dataset. Offspring for a combined task are evaluated on all sources, with only the constituent entries contributing to J_C. For example, an O+TW population uses the average of its normalized O and TW costs for local quality.

Choosing interacting tasks. Given these source scores, MECo chooses two populations to develop a combined task. A pair $\mathcal{P}_A, \mathcal{P}_B$ is eligible when $C = A \cup B$ contains attributes contributed by both parents, so $A \subsetneq C$ and $B \subsetneq C$. Thus a combined task such as O+TW can later interact with L to develop O+TW+L. Different parent pairs can reach the same task, and their populations are merged after its inner evolution search.

Among eligible pairs, transfer gap measures the change in average population quality on the destination task. We define the transferred and native scores as

$$s(A \to B) = \frac{1}{|\mathcal{P}_A|} \sum_{h \in \mathcal{P}_A} J_B(h), \qquad s(B \to B) = \frac{1}{|\mathcal{P}_B|} \sum_{h \in \mathcal{P}_B} J_B(h).$$

Both scores use the complete datasets of the sources associated with T_B. The directional and symmetric transfer gaps are

$$g(A \to B) = \max\left\{0,\ \frac{s(A \to B) - s(B \to B)}{\max\{|s(B \to B)|, \epsilon\}}\right\}, \tag{5}$$

$$G(A, B) = \frac{g(A \to B) + g(B \to A)}{2}. \tag{6}$$

$G(A, B)$ averages the relative transfer losses in both directions.

MECo first prioritizes pairs leading to unexplored combined tasks and then favors a smaller $G(A, B)$. Each task pair is retired after a successful expansion, while either task remains available to interact with other tasks. Different pairs can therefore contribute to the same combined task.

Evolving Task-Conditioned Heuristics. Once the parent tasks are chosen, MECo uses their best heuristics as the initial parents for T_C. The LLM receives the parent program or programs required by its operator, their task descriptions and source scores, and the combined-task description. Later generations choose parents from the local population and evaluate each proposed heuristic on the source datasets.

Four prompt-based operators guide this evolution through three forms of crossover and one guided mutation: **(1)** c_1 performs mechanism composition, combining compatible decision rules from two parents. **(2)** c_2 performs conflict-resolving crossover, integrating conflicting parent rules through conditional or hierarchical decisions. **(3)** c_3 performs exploratory crossover, using two parents with contrasting source scores to propose a distinct decision principle for an under-covered source dimension. **(4)** m_1 performs single-parent mutation, retaining the parent's backbone while modifying a decision mechanism to improve its source performance.

The local population preserves candidates with different source strengths. Candidates are ranked by nondominated fronts on the constituent dimensions of $\mathbf{f}(h)$, with crowding distance favoring less crowded score vectors within each front (Deb et al., 2002). Comparing their decisions on the same construction states further prioritizes distinct behaviors (Zhang & Lu, 2026).

After the evolutionary search, successful heuristics from that expansion are selected by J_C and behavioral priority to form the retained population. If T_C already exists, its previous and new populations are merged using the local survival rule. The updated population then becomes available for future task interactions. Repeating these expansions produces candidate heuristics for progressively richer task specifications.

3.4 Heuristic Set Selection for Zero-Shot Evaluation

After evolution, we collect representatives from all task populations, complete their source evaluations, and filter repeated behaviors to obtain a candidate pool $\mathcal{E}$. Following the principle of complementary heuristic selection (Xu et al., 2010), we choose K members to cover combinations of sources. To direct selection toward generalization beyond the source tasks, the coverage objective ranges over combined task specifications $\mathcal{C}_{\geq 2} = \{C \subseteq \mathcal{C} \mid |C| \geq 2\}$. Candidates from source populations remain eligible. For a nonempty set $\mathcal{H}$, define

$$J_{\mathrm{comp}}(\mathcal{H}) = \frac{1}{|\mathcal{C}_{\geq 2}|} \sum_{C \in \mathcal{C}_{\geq 2}} \min_{h \in \mathcal{H}} J_C(h). \tag{7}$$

For each combination C, we first compute every member's mean cost on the constituent sources, $J_C(h)$, and retain the lowest value in the set. We then average these values across combinations. The objective rewards their complementary strengths.

We begin with the heuristic of lowest mean normalized source cost. Each additional member is chosen by how much it reduces the current set score,

$$h^\star \in \arg\max_{h \in \mathcal{E} \setminus \mathcal{H}} \left[J_{\mathrm{comp}}(\mathcal{H}) - J_{\mathrm{comp}}(\mathcal{H} \cup \{h\})\right] \tag{8}$$

until $|\mathcal{H}| = K$. Ties in marginal gain favor the candidate with lower mean normalized source cost. We use $K = 4$, freeze the selected programs and set membership, and evaluate all members on each target problem without further search or adaptation.

4 EXPERIMENTS

4.1 EXPERIMENTAL SETUP

Problem setup. We study vehicle routing problem (VRP) and flexible job-shop scheduling problem (FJSP). The base VRP is the capacitated vehicle routing problem (CVRP), which serves every customer using depot-to-depot routes subject to vehicle capacity and minimizes total travel distance. The base FJSP assigns each operation to an eligible machine and sequences operations to minimize makespan, subject to within-job precedence and machine availability. For VRP, the designed heuristic selects the next customer during route construction. For FJSP, it selects the next operation and its machine from feasible assignments.

We construct 16 variants per family by activating subsets of 4 optional constraints. VRP uses open routes (O), backhauls (B), time windows (TW), and route-length limits (L), following multi-task VRP formulations (Liu et al., 2024a). FJSP uses release dates (R), assembly precedence (A), sequence-dependent setups (S), and transport lags (T) (Reijnen et al., 2026; Lin et al., 2022; Shen et al., 2018; Berterottière et al., 2024). Only the four single-constraint tasks provide training datasets. Testing on larger, unseen instances of these four problems measures in-domain (ID) generalization. Testing on the 11 combined-constraint problems and the base problem measures out-of-domain (OOD) generalization. Appendix C specifies the constraint semantics.

Baselines. We compare eight AHD methods and two classical construction rules per family, organized into three groups.

(1) Task-specific AHD. EoH (Liu et al., 2024b), ReEvo (Ye et al., 2024), MCTS-AHD (Zheng et al., 2025), EoH-S (Liu et al., 2026a), MEoH (Yao et al., 2025), and HSEvo (Dat et al., 2025) perform heuristic search for a given design task.

(2) Cross-task AHD. MoH co-evolves heuristics and optimizers (Shi et al., 2026), while EMO-STA combines shared search with task-specific adaptation (Gozeten et al., 2026). All methods use the same four source tasks and problem interfaces.

(3) Classical heuristics. VRP uses nearest-feasible construction and savings-priority construction (Clarke & Wright, 2012). FJSP uses earliest-completion-time (ECT) dispatch and most-work-remaining dispatch with ECT machine assignment (MWKR-ECT), following established dispatching principles (Lei et al., 2022). Appendix D.3 gives the baseline settings and rule definitions.

Search and data settings. All AHD searches use GPT-4o-mini with a budget of 2,000 new algorithms per family and select 4 best heuristics for final testing. Following feasibility-masked construction (Liu et al., 2024a), common task-specific masks enforce constraints for all methods. Table 1 summarizes the data. Size counts customers for VRP and operations for FJSP. Computation uses Intel Xeon Gold 6348 CPUs at 2.60 GHz. Appendix D.2 details data construction.

Table 1: Data per family. Instances are counted per variant.

Split	Variants	Instances	Size
Training	4	10	50
ID test	4	1,000	100
OOD test	12	1,000	100

Evaluation. For the test dataset $D_{T_C}^{\text{test}}$ of task T_C, we report

$$\text{Best4}(C) = \min_{h \in \mathcal{H}} \frac{1}{|D_{T_C}^{\text{test}}|} \sum_{x \in D_{T_C}^{\text{test}}} \ell(h, x). \tag{9}$$

Best4 is the lowest whole-dataset mean cost among the four selected heuristics. Unless stated otherwise, we repeat each AHD experiment three times and report mean and sample standard deviation of the resulting Best4 scores, averaged over all 16, 4 ID, or 12 OOD variants. Deterministic classical rules are evaluated once. Relative gap is the percentage excess over the best observed method mean on the same variant. Gap and Rank average these gaps and per-variant ranks over all 16 variants. Bold and underlined means indicate first and second place, including ties. Green rows highlight the corresponding best methods, and $\#k$ denotes the number of variants.

Table 2: Performance comparison across the two problem families.

(a) VRP distance ↓

Method	All (#16)	ID (#4)	OOD (#12)	Rank (#16)	Gap (%) (#16)
Nearest-feasible	23.53	23.04	23.69	7.66	7.74
Savings-priority	24.51	22.48	25.19	7.75	14.64
EoH	23.16 ±0.80	22.14 ±0.08	23.49 ±1.04	7.12	7.52
ReEvo	23.20 ±0.55	22.41 ±0.46	23.46 ±0.60	8.78	7.58
MCTS-AHD	22.54 ±0.17	22.10 ±0.05	22.69 ±0.21	5.81	4.76
EoH-S	22.75 ±0.10	22.18 ±0.12	22.94 ±0.12	6.94	5.64
MEoH	22.94 ±0.03	22.53 ±0.19	23.08 ±0.08	6.22	5.83
HSEvo	22.47 ±0.27	22.12 ±0.11	22.58 ±0.33	3.97	4.34
EMO-STA	23.01 ±0.50	22.46 ±0.68	23.20 ±0.45	5.28	5.02
MoH	22.39 ±0.62	21.88 ±0.80	22.55 ±0.57	4.16	2.82
MECo	**21.98** ±0.50	**21.60** ±0.70	**22.11** ±0.43	**2.31**	**1.40**

(b) FJSP makespan ↓

Method	All (#16)	ID (#4)	OOD (#12)	Rank (#16)	Gap (%) (#16)
ECT	1486.8	1301.4	1548.6	8.44	5.88
MWKR-ECT	1751.7	1436.3	1856.9	11.00	23.64
EoH	1466.8 ±33.5	1247.0 ±37.6	1540.1 ±32.2	7.50	4.00
ReEvo	1446.0 ±29.7	1229.2 ±12.0	1518.2 ±35.8	5.25	2.54
MCTS-AHD	1456.8 ±47.4	1239.7 ±47.9	1529.2 ±48.1	6.56	3.35
EoH-S	1460.2 ±45.7	1241.7 ±45.3	1533.1 ±47.3	6.91	3.58
MEoH	1453.0 ±36.7	1237.2 ±26.2	1525.0 ±42.1	6.00	3.06
HSEvo	1443.3 ±23.3	1225.2 ± 5.1	1516.0 ±29.5	4.81	2.32
EMO-STA	1447.6 ±32.4	1255.7 ±38.2	1511.5 ±31.1	5.81	2.93
MoH	1421.4 ±12.2	1223.5 ±13.0	1487.3 ±11.9	2.59	0.95
MECo	**1408.7** ± 1.1	**1211.3** ± 3.5	**1474.4** ± 1.1	**1.12**	**0.02**

4.2 Cross-problem generalization

As shown in Table 2, MECo achieves the lowest mean costs on both the source and unseen problems in each family. Its advantage extends across the all-variant, ID, and OOD aggregates, together with the lowest average rank and relative gap. Under the shared source-only protocol, MECo therefore achieves better aggregate performance on unseen problems than the compared AHD methods.

Specifically, on VRP, MoH is the strongest baseline by mean cost in all three groups. MECo lowers its ID cost from 21.88 to 21.60 and its OOD cost from 22.55 to 22.11. Therefore, the improvement extends beyond the four problems supplying training feedback. In addition, MECo also has an average rank of 2.31, compared with 3.97 for the next-best method by rank, HSEvo. Similarly, on FJSP, MECo lowers the OOD mean from 1487.3 for MoH to 1474.4, alongside a lower ID mean of 1211.3. Its OOD standard deviation is also smaller, 1.1 compared with 11.9 for MoH, indicating less variation across the repeated searches in this study. The average rank of 1.12 and mean relative gap of 0.02% place MECo close to the best observed method across the 16 variants.

As shown in Figure 4, both task-specific and cross-task baselines have higher OOD aggregate costs in both families. Together, these observations support the advantage of MECo on cross-problem generalization across VRP and FJSP. Appendix E.1 provides more detailed results.

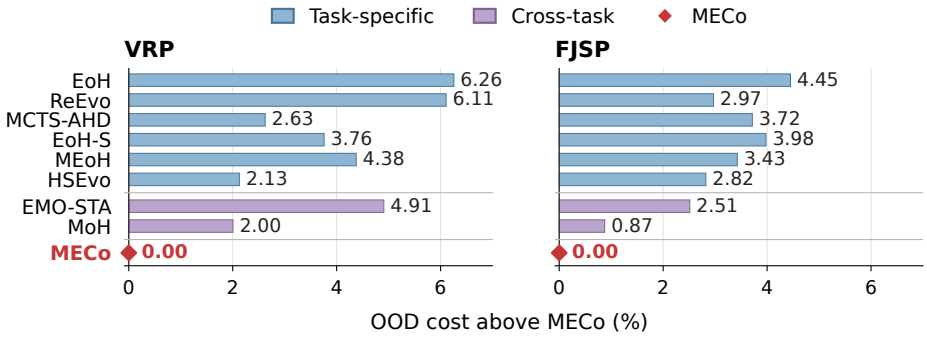

Figure 4: OOD mean cost relative to MECo, computed as $100(\bar{f}/\bar{f}_{\mathrm{MECo}} - 1)$.

4.3 Framework Effectiveness

We further investigate the effectiveness of MECo's task-search framework across different heuristic search methods. For each baseline X, we construct X-MECo by using its native heuristic generation and population update rules within MECo's task populations. MECo chooses the interacting tasks and transfers their parent heuristics to initialize a combined-task population. The baseline then evolves that population using its own search rules. Thus, EoH-MECo runs EoH within each task population while MECo determines which tasks interact. ReEvo-MECo and HSEvo-MECo follow the same construction. After search, MECo selects the final four heuristics from the task populations.

Table 3: Baseline engines with and without MECo.

	VRP distance ↓			FJSP makespan ↓		
Method	All (#16)	ID (#4)	OOD (#12)	All (#16)	ID (#4)	OOD (#12)
EoH	23.16 ±0.80	22.14 ±0.08	23.49 ±1.04	1466.8 ±33.5	1247.0 ±37.6	1540.1 ±32.2
EoH-MECo	**21.99** ±0.66	**21.66** ±0.83	**22.10** ±0.61	**1432.0** ± 6.5	**1211.1** ±13.3	**1505.7** ± 4.9
ReEvo	23.20 ±0.55	22.41 ±0.46	23.46 ±0.60	1446.0 ±29.7	1229.2 ±12.0	1518.2 ±35.8
ReEvo-MECo	**22.86** ±1.75	**21.17** ±0.70	**23.43** ±2.41	**1415.9** ±28.4	**1207.3** ±18.0	**1485.4** ±32.2
HSEvo	22.47 ±0.27	22.12 ±0.11	22.58 ±0.33	1443.3 ±23.3	1225.2 ± 5.1	1516.0 ±29.5
HSEvo-MECo	**22.14** ±0.13	**21.83** ±0.09	**22.24** ±0.19	**1418.1** ±17.5	**1218.5** ±16.4	**1484.6** ±17.8

As shown in Table 3, all three hybrids achieve lower mean costs than their baselines on both ID and OOD variants in both families. In particular, EoH-MECo reduces mean OOD cost relative to EoH over 5% on VRP and 2% on FJSP. These results support the effectiveness of the framework beyond MECo's population evolution rules.

To assess reliability, we extend the FJSP comparison to ten independent runs per method, comparing EoH and ReEvo with their MECo counterparts. As shown in Figure5, both MECo variants achieve lower mean costs on ID and OOD tasks. All four improvements are statistically significant based on two-sided independent-samples permutation tests with Holm correction across the four comparisons. Appendix E.5 provides the experimental and statistical details.

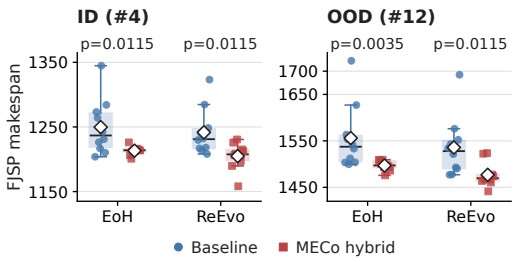

Figure 5: Performance on FJSP for ten runs.

4.4 Component Ablations

We study the importance of task pairing and heuristic evolution operators in MECo. The w/o transfer gap condition replaces task-pair selection with uniform random sampling. The other four conditions

individually remove mechanism composition, conflict resolution, transfer-oriented mutation, or diversification from every stage of evolution. As shown in Table 4, all five conditions have higher mean OOD costs than full MECo in both families. Among these conditions, removing transfer gap guidance yields the highest OOD cost on VRP, while removing diversification yields the highest OOD cost and greatest variability on FJSP. Together, these results suggest that task pairing and the evolution operators contribute to MECo's generalization performance.

Table 4: Component ablations with full MECo.

	VRP distance ↓			FJSP makespan ↓		
Condition	All (#16)	ID (#4)	OOD (#12)	All (#16)	ID (#4)	OOD (#12)
w/o transfer gap	22.35 ±0.05	22.16 ±0.05	22.41 ±0.05	1410.2 ± 7.8	1214.6 ±11.5	1475.3 ± 6.6
w/o composition	22.28 ±0.16	22.11 ±0.11	22.34 ±0.17	1418.4 ±11.3	1226.7 ±13.9	1482.3 ±10.7
w/o conflict	22.19 ±0.09	21.73 ±0.31	22.34 ±0.06	1411.9 ± 8.7	1216.7 ±13.2	1477.0 ± 7.3
w/o transfer repair	22.37 ±0.09	22.08 ±0.05	22.47 ±0.10	1411.4 ± 4.9	1215.2 ± 3.9	1476.8 ± 5.2
w/o diversification	22.32 ±0.11	21.89 ±0.31	22.46 ±0.16	1424.0 ±24.7	1224.0 ±18.5	1490.7 ±26.7
MECo	**21.98** ±0.50	**21.60** ±0.70	**22.11** ±0.43	**1408.7** ± 1.1	**1211.3** ± 3.5	**1474.4** ± 1.1

4.5 Generation-model sensitivity

We examine how the LLM affects MECo by comparing GPT-4o-mini, Gemini-3.1-Flash-Lite, and GPT-5.4-nano under the same allowance of 2,000 heuristic proposals. As shown in Table 5, Gemini-3.1-Flash-Lite achieves the lowest mean costs on both ID and OOD tasks in both families. GPT-5.4-nano has higher mean costs than GPT-4o-mini on VRP and slightly lower costs on FJSP. Thus, the relative performance of the latter two models varies across problem families.

Table 5: MECo performance with different generation models.

	VRP distance ↓			FJSP makespan ↓		
Model	All (#16)	ID (#4)	OOD (#12)	All (#16)	ID (#4)	OOD (#12)
gpt-4o-mini	21.98 ±0.50	21.60 ±0.70	22.11 ±0.43	1408.7 ± 1.1	1211.3 ± 3.5	1474.4 ± 1.1
gpt-5.4-nano	22.29 ±0.46	21.72 ±0.43	22.48 ±0.48	1405.8 ± 1.7	1205.1 ± 3.4	1472.7 ± 1.9
gemini-3.1-flash-lite	**21.72** ±0.36	**21.36** ±0.28	**21.84** ±0.38	**1367.6** ±22.3	**1163.2** ±13.6	**1435.8** ±25.1

Further analysis. We study the adaptation on OB and BLTW. MECo shares 500 additional proposals across both targets, while the task-specific baselines use 500 per target. Although MECo is not the best on either target individually, its costs remain close to the best observed values and its average rank is the lowest, using half the total proposal allowance. Appendix E.6 reports these results. We further study the joint design across TSP, VRP, and FJSP under a shared budget of 1,000 proposals. MECo yields TSP and VRP costs that are lower than those of EMO-STA and MoH at both test sizes. It also improves over EMO-STA on FJSP, where MoH performs best. This extends joint design beyond variants of one problem family, as reported in Appendix E.8. Appendix E.9 reports resource use.

5 Discussion and Conclusion

This work introduces MECo, an LLM-driven multi-task evolutionary framework for zero-shot cross-problem generalization. Transfer gaps guide interactions among task populations, enabling heuristics to be adapted and recombined. Source feedback supports both evolution and complementary set selection, without target evaluations during design. Experiments on VRP and FJSP demonstrate improved ID and OOD performance under a shared design budget. Moreover, integrating MECo with EoH, ReEvo, and HSEvo consistently improves their performance. Together, these results support coordinated multi-task search as an effective approach to designing heuristic sets that generalize beyond their source problems.

AI USE STATEMENT

In this work, we used generative AI tools for assisting with translation. We have not used generative AI tools for generating synthetic data sets, assisting in the writing of proofs, proposing or refining hypotheses, designing or providing feedback on research methodology or experiments, implementing methods, cleaning and reformatting datasets, supporting qualitative and thematic data analysis, interpreting results, helping develop theoretical models or conceptual frameworks, formulating mathematical claims, or providing critical ingredients for proving mathematical claims, as these uses are not applicable to this work. Additionally, we used generative AI tools for editing a research paper to improve readability. We have reviewed all AI-assisted work. We checked LLM-generated research ideas for potential plagiarism through a manual literature survey. We take responsibility for the final content of this work, including text, claims, or artifacts produced with the aid of generative AI.

REFERENCES

Alessandro Achille, Michael Lam, Rahul Tewari, Avinash Ravichandran, Subhransu Maji, Charless C Fowlkes, Stefano Soatto, and Pietro Perona. Task2vec: Task embedding for meta-learning. In *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 6430–6439, 2019.

Kavitesh Kumar Bali, Yew-Soon Ong, Abhishek Gupta, and Puay Siew Tan. Multifactorial evolutionary algorithm with online transfer parameter estimation: Mfea-ii. *IEEE Transactions on Evolutionary Computation*, 24(1):69–83, 2019.

Lucas Berterottière, Stéphane Dauzère-Pérès, and Claude Yugma. Flexible job-shop scheduling with transportation resources. *European Journal of Operational Research*, 312(3):890–909, 2024.

Geoff Clarke and John W Wright. Scheduling of vehicles from a central depot to a number of delivery points. In *The roots of logistics*, pp. 229–244. Springer, 2012.

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.

Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and TAMT Meyarivan. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE transactions on evolutionary computation*, 6(2): 182–197, 2002.

Halil Alperen Gozeten, Xuechen Zhang, Emrullah Ildiz, Ege Onur Taga, Tara Javidi, and Samet Oymak. Evolutionary multi-task optimization for llm-guided program discovery. *arXiv preprint arXiv:2605.22613*, 2026.

Abhishek Gupta, Yew-Soon Ong, and Liang Feng. Multifactorial evolution: Toward evolutionary multitasking. *IEEE Transactions on Evolutionary Computation*, 20(3):343–357, 2015.

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.

Kun Lei, Peng Guo, Wenchao Zhao, Yi Wang, Linmao Qian, Xiangyin Meng, and Liansheng Tang. A multi-action deep reinforcement learning framework for flexible job-shop scheduling problem. *Expert Systems with Applications*, 205:117796, 2022.

Wenhui Lin, Qianwang Deng, Wenwu Han, Guiliang Gong, and Kexin Li. An effective algorithm for flexible assembly job-shop scheduling with tight job constraints. *International Transactions in Operational Research*, 29(1):496–525, 2022.

Fei Liu, Xi Lin, Zhenkun Wang, Qingfu Zhang, Tong Xialiang, and Mingxuan Yuan. Multi-task learning for routing problem with cross-problem zero-shot generalization. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pp. 1898–1908, 2024a.

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 41st International Conference on Machine Learning*, pp. 32201–32223, 2024b.

Fei Liu, Yilu Liu, Qingfu Zhang, Xialiang Tong, and Mingxuan Yuan. EoH-S: Evolution of heuristic set using LLMs for automated heuristic design. *Proceedings of the AAAI Conference on Artificial Intelligence*, 40(43):37090–37098, 2026a.

Fei Liu, Yiming Yao, Ping Guo, Zhiyuan Yang, Xi Lin, Zhe Zhao, Xialiang Tong, Kun Mao, Zhichao Lu, Zhenkun Wang, et al. A systematic survey on large language models for algorithm design. *ACM Computing Surveys*, 58(8):1–32, 2026b.

Fei Liu, Rui Zhang, Shunyu Yao, Qinglong Hu, Zhichao Lu, Qingfu Zhang, et al. Hierarchical representations for cross-task automated heuristic design using llms. In *Forty-third International Conference on Machine Learning*, 2026c.

Robbert Reijnen, Igor G Smit, Hongxiang Zhang, Yaoxin Wu, Zaharah Bukhsh, and Yingqian Zhang. Job shop scheduling benchmark: Environments and instances for learning and non-learning methods. *Annals of Mathematics and Artificial Intelligence*, pp. 1–31, 2026.

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.

Liji Shen, Stéphane Dauzère-Pérès, and Janis S Neufeld. Solving the flexible job shop scheduling problem with sequence-dependent setup times. *European journal of operational research*, 265 (2):503–516, 2018.

Yiding Shi, Jianan Zhou, Wen Song, Jieyi Bi, Yaoxin Wu, Zhiguang Cao, and Jie Zhang. Generalizable heuristic generation through llms with meta-optimization. In *International Conference on Learning Representations*, volume 2026, pp. 119314–119349, 2026.

Paolo Toth and Daniele Vigo. *Vehicle routing: problems, methods, and applications*. SIAM, 2014.

Niki Van Stein and Thomas Bäck. Llamea: A large language model evolutionary algorithm for automatically generating metaheuristics. *IEEE Transactions on Evolutionary Computation*, 29 (2):331–345, 2024.

Zhuoliang Xie, Fei Liu, Genghui Li, Zhilin Mao, Yu Zhang, Zhenkun Wang, and Qingfu Zhang. Multipopulation optimization with llm-driven knowledge discovery for large-scale hfvrp. *IEEE Transactions on Computational Social Systems*, 12(6):5449–5459, 2025a.

Zhuoliang Xie, Fei Liu, Zhenkun Wang, and Qingfu Zhang. Llm-driven neighborhood search for efficient heuristic design. In *2025 IEEE Congress on Evolutionary Computation (CEC)*, pp. 1–8. IEEE, 2025b.

Zhuoliang Xie, Fei Liu, Zhenkun Wang, and Qingfu Zhang. Enhancing cvrp solver through llm-driven automatic heuristic design. *arXiv preprint arXiv:2602.23092*, 2026.

Lin Xu, Holger Hoos, and Kevin Leyton-Brown. Hydra: Automatically configuring algorithms for portfolio-based selection. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 24, pp. 210–216, 2010.

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. In *Proceedings of the 38th International Conference on Neural Information Processing Systems*, pp. 43571–43608, 2024.

Amir R Zamir, Alexander Sax, William Shen, Leonidas J Guibas, Jitendra Malik, and Silvio Savarese. Taskonomy: Disentangling task transfer learning. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 3712–3722, 2018.

Rui Zhang and Zhichao Lu. Rethinking code similarity for automated algorithm design with LLMs. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=HIUqeO9OOr.

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*, 2025a.

Yu Zhang, Kefeng Zheng, Fei Liu, Qingfu Zhang, and Zhenkun Wang. Autoturb: Using large language models for automatic algebraic turbulence model discovery. *Physics of Fluids*, 37(1), 2025b.

Zhi Zheng, Zhuoliang Xie, Zhenkun Wang, and Bryan Hooi. Monte carlo tree search for comprehensive exploration in llm-based automatic heuristic design. In *Forty-second International Conference on Machine Learning*, 2025.

APPENDIX CONTENTS

A Related work 14
- A.1 Task-specific automated heuristic design 14
- A.2 Cross-task automated heuristic design 14
- A.3 Knowledge transfer across optimization tasks 14

B More Method Details 15
- B.1 Search Procedure 15
- B.2 Source feedback and transfer 15
- B.3 Heuristic Generation 16
- B.4 Population survival 16
- B.5 Final heuristic-set selection 17
- B.6 Construction and candidate validity 17

C Task definitions and constraint semantics 18
- C.1 VRP 18
- C.2 FJSP 18

D Benchmark variants and experimental parameters 20
- D.1 Complete variant specification 20
- D.2 Problem parameters and data construction 20
- D.3 Search and baseline parameters 22
- D.4 Classical constructive reference rules 22

E More Experiments 24
- E.1 Complete task results 24
- E.2 Performance profiles across all variants 26
- E.3 Performance by composition depth 26
- E.4 Task pairing and generation operators 27
- E.5 Transfer across local search engines 27
- E.6 VRP target adaptation 28
- E.7 Generalization to larger VRP instances 29
- E.8 Search across heterogeneous tasks 29
- E.9 Resource use 30

F Generation prompts and interfaces 32
- F.1 Task descriptions and input templates 32
- F.2 Descriptions for active attributes 33
- F.3 Generation instructions 33

G Representative evolved heuristics 35
- G.1 VRP example 35
- G.2 FJSP example 36

H Licenses 37

A Related work

A.1 Task-specific automated heuristic design

LLM-driven AHD commonly uses evolutionary search to generate and refine executable heuristics from problem-specific feedback (Romera-Paredes et al., 2024; Liu et al., 2024b; Van Stein & Bäck, 2024). Natural-language descriptions and reflection provide additional guidance for program evolution (Liu et al., 2024b; Ye et al., 2024). Other search strategies include diversity-aware evolution (Dat et al., 2025), Monte Carlo tree search (Zheng et al., 2025), and neighborhood search (Xie et al., 2025b). These approaches differ primarily in how they explore the program space and reuse evaluation feedback.

A complementary direction addresses variation among instances of the same problem. Heuristic-set evolution (Liu et al., 2026a) and instance specialization through feature-based partitioning or performance-based clustering (Zhang et al., 2025a; Hu et al., 2026) seek complementary strengths across instance groups. Multi-objective search additionally considers trade-offs between solution quality and computational cost (Yao et al., 2025). Applications extend to parameter adaptation and ruin-operator design within vehicle-routing solvers (Xie et al., 2025a; 2026), as well as algebraic turbulence-model discovery (Zhang et al., 2025b).

A.2 Cross-task automated heuristic design

Cross-task AHD investigates reusable knowledge beyond an individual design task. Existing approaches transfer task-agnostic algorithmic templates through hierarchical representations (Liu et al., 2026c), learn heuristic optimizers through meta-optimization (Shi et al., 2026), or share program archives before task-specific adaptation (Gozeten et al., 2026). The reusable knowledge consequently ranges from abstract search principles to executable programs, with downstream feedback guiding their specialization.

A.3 Knowledge transfer across optimization tasks

Knowledge transfer across optimization tasks has also been studied through learned solvers and shared search populations. In neural combinatorial optimization, multi-task VRP learning trains a shared solver across problem definitions and studies zero-shot transfer to unseen constraint combinations (Liu et al., 2024a). Evolutionary multitasking exploits shared representations and cross-task reproduction, with adaptive transfer mechanisms regulating exchange as populations develop (Gupta et al., 2015; Bali et al., 2019). The usefulness of such exchange motivates studying task relationships. Related work in representation learning characterizes these relationships through Fisher-information embeddings (Achille et al., 2019) and directed transfer graphs estimated from predictive performance (Zamir et al., 2018), distinguishing representational similarity from empirical transfer utility.

B More Method Details

This section expands the task interactions, population updates, and heuristic-set selection introduced in Section 3. We retain the notation T_C for a design task, $\mathcal{P}_C$ for its population, and D_{T_j} for a source dataset. All search decisions use source evaluations, while target specifications describe the problems that the generated heuristics are intended to solve.

B.1 Search Procedure

The explored tasks form a directed acyclic graph. Its roots are the single-attribute tasks, and an interaction between T_A and T_B produces $T_{A\cup B}$. Eligibility requires both $A \subsetneq A\cup B$ and $B \subsetneq A\cup B$, so each parent contributes an attribute absent from the other. Every edge therefore increases the attribute set. Several parent pairs can reach the same task, allowing its population to incorporate heuristics developed through different combinations of prior knowledge.

Algorithm 1 summarizes the search under a proposal budget Q. Each source receives an initial allowance q_s, and each expansion receives a local allowance q, with population capacity N. A completed interaction retires its parent pair while leaving both parent tasks available for other interactions. Repeatedly reached tasks retain a single merged population. Consequently, task exploration and local evolution share the same budget, and the graph grows according to source feedback.

Algorithm 1 MECo multi-task evolution

Input Source datasets $\{D_{T_j}\}_{j=1}^{m}$, task specifications, proposal budget Q
Input Allowances q_s, q, population capacity N, output size K
Evolve source populations on their respective datasets with allowance q_s each
Initialize explored attribute sets $\mathcal{V}$ with the roots and completed pairs $\mathcal{U} = \varnothing$
while proposal budget remains **do**
  Form eligible pairs of attribute sets in $\mathcal{V}$ excluding $\mathcal{U}$
  if no eligible pair remains **then break**
  Prefer unexplored unions, then select a pair (A, B) with smallest $G(A, B)$
  Set $C = A \cup B$ and choose the native champion of each parent population
  Initialize a local population $\mathcal{P} = \varnothing$ and successful-candidate set $\mathcal{A}_C = \varnothing$
  while local allowance q and total budget remain **do**
    Choose parent programs from the champions initially, then from $\mathcal{P}$
    Generate candidates for T_C with crossover or guided mutation
    Evaluate on source datasets and obtain the valid candidate set $\mathcal{Q}_C$
    $\mathcal{A}_C \leftarrow \mathcal{A}_C \cup \mathcal{Q}_C$
    $\mathcal{P} \leftarrow \mathrm{Survive}_C(\mathcal{P} \cup \mathcal{Q}_C, N)$
    if initialization produces no valid candidate **then break**
  if $\mathcal{A}_C \neq \varnothing$ **then**
    Select a retained population $\mathcal{S}_C$ by J_C and behavioral priority
    if $C \in \mathcal{V}$ **then** $\mathcal{P}_C \leftarrow \mathrm{Survive}_C(\mathcal{P}_C \cup \mathcal{S}_C, N)$
    else set $\mathcal{P}_C \leftarrow \mathcal{S}_C$ and add C to $\mathcal{V}$
    Add (A, B) to $\mathcal{U}$
Collect representatives from all task populations
Filter repeated behaviors to obtain the eligible pool $\mathcal{E}$
$\mathcal{H} \leftarrow \{\arg\min_{h\in\mathcal{E}} m^{-1} \sum_j f_j(h)\}$
while $|\mathcal{H}| < K$ **do**
  Add the candidate with greatest marginal reduction in $J_{\mathrm{comp}}(\mathcal{H})$
Output Heuristic set $\mathcal{H}$, frozen before target evaluation

The local survival operator preserves source-quality trade-offs and distinct heuristic logic. The retained population then supplies evidence for subsequent task interactions. Final selection draws from all explored tasks. Therefore a heuristic can remain useful even when it was developed for a different task specification.

B.2 Source Feedback and Transfer

The normalized score $f_j(h)$ compares a heuristic with a simple reference rule on source T_j, as defined in Equation 4. Each scale b_j is initially computed on D_{T_j} and kept fixed. This normalization

gives each constituent source a comparable contribution to $J_C(h)$. Evaluating a combined-task heuristic produces a complete source vector, while its local quality depends only on the dimensions associated with C.

For transfer from T_A to T_B, both populations are assessed on exactly the same destination sources. Expanding the definition in the main text gives

$$s(A \to B) = \frac{1}{|\mathcal{P}_A|\,|B|} \sum_{h \in \mathcal{P}_A} \sum_{j \mid c_j \in B} \frac{1}{b_j |D_{T_j}|} \sum_{x \in D_{T_j}} \ell(h, x). \tag{10}$$

The native score $s(B \to B)$ is obtained by replacing $\mathcal{P}_A$ with $\mathcal{P}_B$ in this expression. Both scores average over all members of their respective populations and all instances in the same destination source datasets. Comparing $s(A \to B)$ with $s(B \to B)$ gives the directional component $g(A \to B)$. Task pairing uses the symmetric transfer gap $G(A, B)$, which averages $g(A \to B)$ and $g(B \to A)$.

The positive part in Equation 5 measures relative deterioration upon transfer. A direction with no deterioration contributes zero to the symmetric gap. Prioritizing unexplored unions promotes task coverage, while minimizing the gap among equally prioritized pairs favors populations with stronger mutual transfer. Once a population changes, subsequent pair decisions use its updated membership and scores.

B.3 Heuristic Generation

An expansion begins with the best native heuristic from each selected task. The target description specifies T_C, and the parent descriptions and source scores provide context for transferring their decision rules. Subsequent proposals use representatives of the local population. Grouping programs by their decisions on common construction states prevents behaviorally repeated candidates from dominating parent selection.

The composition and conflict-resolution operators, c_1 and c_2, draw distinct parents with a preference for lower J_C. For a representative set $\mathcal{R}_C$, quality-based sampling uses

$$p_C(h) = \frac{w_C(h)}{\sum_{g \in \mathcal{R}_C} w_C(g)}, \qquad w_C(h) = \max_{g \in \mathcal{R}_C} J_C(g) - J_C(h) + \eta, \tag{11}$$

where $\eta > 0$ gives every representative a positive weight. We sample the first parent according to p_C, remove it from the candidate set, and recompute the probabilities over the remaining candidates to sample the second parent. Composition combines compatible mechanisms, while conflict resolution reconciles competing rules through conditional or hierarchical decisions.

Exploratory crossover c_3 selects the pair with the greatest difference in constituent-source scores. For distinct candidates $h, g \in \mathcal{R}_C$, we define their distance and select the parents as

$$\begin{aligned} d_C(h, g) &= \frac{1}{|C|} \sum_{j \mid c_j \in C} |f_j(h) - f_j(g)|, \\ (h_1, h_2) &\in \arg \max_{\substack{h, g \in \mathcal{R}_C \\ h \neq g}} d_C(h, g). \end{aligned} \tag{12}$$

The selected parents provide contrasting performance profiles, and the generation prompt directs exploration toward an under-covered source dimension. Guided mutation m_1 uses a strong local parent and its source feedback to revise a decision mechanism while retaining its backbone. These operators act within the task chosen by the outer search and share its source-based evaluation criterion.

B.4 Population Survival

Local survival first compares candidates on their constituent-source vectors. A heuristic h dominates g for T_C when

$$f_j(h) \leq f_j(g) \quad \forall j \text{ with } c_j \in C, \qquad f_j(h) < f_j(g) \quad \exists j \text{ with } c_j \in C. \tag{13}$$

Nondominated fronts define the initial ordering. Within a front, crowding distance favors candidates in less densely populated regions of the source-score space (Deb et al., 2002). This preserves alternative trade-offs among constituent sources during local evolution.

We characterize a heuristic's behavior by its action choices on a fixed set of construction states $\mathcal{S}$ collected from reference trajectories on source problems. Each state contains the instance information, partial solution, and available actions. In VRP, an action selects the next node to visit. In FJSP, it selects an operation and the machine on which to process it. Every heuristic receives the same states, so its behavior is represented by the action vector $(a(h,s))_{s\in\mathcal{S}}$, where $a(h,s)$ is the action chosen by h at state s. The disagreement rate is

$$d_{\mathrm{beh}}(h,g) = \frac{1}{|\mathcal{S}|}\sum_{s\in\mathcal{S}} \mathbf{1}\{a(h,s) \neq a(g,s)\}. \tag{14}$$

We scan candidates in the quality order and assign each to the first existing representative within distance δ. A candidate with no such representative starts a new group. Representatives are placed before repeated behaviors, preserving the quality order within each category. Repeated behaviors fill remaining positions when needed. Thus, programs that choose the same actions across the reference states can have distance zero even when their code differs. This comparison follows behavior-aware program comparison (Zhang & Lu, 2026).

Population retention has two stages: (1) During local evolution, the current members and new candidates are ordered by nondominated fronts and crowding distance, followed by behavioral priority, and at most N survive. (2) After expansion, if T_C is new, these heuristics establish its population. If another parent pair has already produced the same target specification C, the existing and newly retained populations are pooled and reduced to at most N members using nondominated sorting, crowding distance, and behavioral priority again.

B.5 Final Heuristic Set Selection

We pool candidate heuristics from all explored task populations, including the sources, and sort them by mean normalized source cost. Behavioral grouping then retains the lowest-cost representative of each group to form $\mathcal{E}$. Multiple candidates from the same task can remain eligible.

We initialize $\mathcal{H}$ with the candidate having the lowest mean normalized source cost. For each combination $C \in \mathcal{C}_{\geq 2}$, let $q_C(\mathcal{H}) = \min_{g\in\mathcal{H}} J_C(g)$ be the current set's best mean score on its constituent sources. Adding a candidate $h \in \mathcal{E} \setminus \mathcal{H}$ reduces the set objective by

$$\Delta(h \mid \mathcal{H}) = \frac{1}{|\mathcal{C}_{\geq 2}|}\sum_{C\in\mathcal{C}_{\geq 2}} \max\{0, q_C(\mathcal{H}) - J_C(h)\}. \tag{15}$$

Only improvements over the current best score contribute to this gain. We add the candidate with the largest Δ, update the combination scores, and repeat until $|\mathcal{H}| = K$. Ties favor lower mean normalized source cost. The objective covers combinations with $|C| \geq 2$, using source scores alone. The selected set is frozen before target evaluation.

B.6 Constructive decision interface

Within each problem family, programs use a shared constructive decision interface. The current instance and partial solution determine the admissible actions through task-specific feasibility masks. A heuristic ranks or selects among these actions, and construction updates the partial solution accordingly.

C TASK DEFINITIONS AND CONSTRAINT SEMANTICS

We address two families of tasks in this paper. Each contains a base problem and four optional attributes. Active attributes determine the feasibility rules and cost conventions. These models draw on published problems, while their joint instances and augmentation parameters define the task families used in this study.

C.1 VRP

The base capacitated vehicle routing problem assigns every customer to exactly one route starting at a depot. Vehicles have capacity Q, and the objective is total traveled distance. We use the four attributes considered in multi-task VRP solving by Liu et al. (2024a), with the following load and route conventions.

Open routes O. A vehicle may end its route at its final customer. The objective excludes the return leg to the depot. This convention models deliveries for which the vehicle need not return to its original dispatch point. It also changes the travel required when checking the remaining route budget.

Time windows TW. Service at customer i must start within $[a_i, b_i]$. Early arrivals wait until a_i, and departure follows service completion. If t is the departure time from the preceding node and τ_i is the travel time to i, service starts at $\max\{t + \tau_i, a_i\}$. Closed routes also respect the depot return deadline. Time windows represent customer availability and scheduled delivery commitments.

Route-length limits L. Each route's total traveled distance is bounded by a prescribed limit. The return leg counts only for closed routes. This bound models a limited travel allowance, such as a distance-based operating range, and remains distinct from waiting and service times under TW.

Mixed deliveries and pickups B. B introduces pickups alongside deliveries and permits their visits to interleave. A route whose first customer is a pickup departs empty and serves pickups only. A route beginning with a delivery departs with load Q. Let d_i be positive for deliveries and negative for pickups. After visiting customer v_k, the load obeys

$$q_k = q_{k-1} - d_{v_k}, \qquad 0 \le q_k \le Q. \tag{16}$$

The total delivered quantity and total collected quantity on each route are each bounded by Q. These conditions remain active when B is combined with O, TW, or L. Customer selection preserves all applicable conditions, and the resulting routes are checked for coverage and feasibility. The attribute represents a vehicle sharing capacity between outbound deliveries and inbound collections. The pickup-first convention separates empty-departure collection routes from routes carrying delivery loads.

Composition and feasibility. For a task P_C, all attributes in C apply to the same route. For example, O+TW+B combines open-route accounting, service-window checks, and the mixed-load trajectory. At each construction step, a customer is available only if it satisfies every active condition. Capacity and customer coverage remain part of the base problem.

C.2 FLEXIBLE JOB SHOP SCHEDULING

Each job consists of ordered operations. Operation o must be assigned to one eligible machine m and processed without interruption for p_{om} time units. Operations on the same machine cannot overlap. The objective is the makespan, the largest operation completion time.

Release dates R. Each job has a release date known before scheduling. Its operations must respect this availability time. The constraint represents the arrival of material or a production order. We specialize the job-arrival setting in Reijnen et al. (2026) to dates available at the start of construction.

Assembly precedence A. A sparse acyclic set of cross-job edges augments the within-job operation chains. An operation can begin only after all its predecessors finish. Such edges model components that must be produced before a downstream assembly or dependent processing step. We draw on the coupled-job setting of Lin et al. (2022) and generate additional predecessor graphs for our instances.

Sequence-dependent setups S. The setup duration depends on the machine and the families of two consecutive operations on that machine. This models changeovers such as tool replacement or cleaning. We follow the sequence-dependent setup semantics studied by Shen et al. (2018). The machine is unavailable for processing during its setup, and its first operation uses an initial setup duration.

Transportation delays T. Successive operations of a job incur a deterministic delay when assigned to different machines. This represents the time required to move a part between workstations. We adapt the transportation setting of Berterottière et al. (2024) to fixed machine-to-machine lags. The present model imposes no competition for a finite fleet of transport vehicles.

Composition and constructive actions. Let u be the preceding operation of o within its job, m_u its assigned machine, v the last operation assigned to m, and $\mathcal{A}(o)$ the active cross-job predecessors. Denote completion times by C, the job release date by r_j, transportation time by t, and setup time by s. Construction appends o to machine m at

$$S_o = \max\left\{r_j,\ C_u + t_{m_u m},\ \max_{a\in\mathcal{A}(o)} C_a,\ C_v + s_{mvo}\right\}, \qquad C_o = S_o + p_{om}. \tag{17}$$

Absent predecessors contribute zero, and inactive attributes contribute zero release dates, lags, or setup times. An idle machine may perform setup before its next operation becomes available. The first operation on a machine uses its initial setup time. Transport applies only to within-job edges and is zero between identical machines. An action is eligible once its predecessors have been assigned and its machine is eligible. The time bounds then ensure temporal feasibility.

D BENCHMARK VARIANTS AND EXPERIMENTAL PARAMETERS

D.1 COMPLETE VARIANT SPECIFICATION

Table 6 enumerates the two task families. Capacity, customer coverage, and a common depot belong to the VRP base problem. Job precedence, machine eligibility, and non-overlapping processing belong to the FJSP base problem. The four attributes in each family are optional additions. Each family therefore contains $1 + 4 + 6 + 4 + 1 = 16$ variants. The four single-attribute variants supply training instances. All 16 variants have separate test instances. In terms of task attributes, four task types are in distribution and twelve are held out, comprising the base task and eleven multi-attribute combinations.

Table 6: Benchmark variants and active constraints. ID and OOD denote source and held-out task types.

VRP							FJSP						
Variant	O	B	TW	L	Depth	Split	Variant	R	A	S	T	Depth	Split
Base					0	OOD	Base					0	OOD
B		✓			1	ID	A		✓			1	ID
L				✓	1	ID	R	✓				1	ID
TW			✓		1	ID	S			✓		1	ID
O	✓				1	ID	T				✓	1	ID
BL		✓		✓	2	OOD	AS		✓	✓		2	OOD
BTW		✓	✓		2	OOD	AT		✓		✓	2	OOD
LTW			✓	✓	2	OOD	RA	✓	✓			2	OOD
OB	✓	✓			2	OOD	RS	✓		✓		2	OOD
OL	✓			✓	2	OOD	RT	✓			✓	2	OOD
OTW	✓		✓		2	OOD	ST			✓	✓	2	OOD
BLTW		✓	✓	✓	3	OOD	AST		✓	✓	✓	3	OOD
OBL	✓	✓		✓	3	OOD	RAS	✓	✓	✓		3	OOD
OBTW	✓	✓	✓		3	OOD	RAT	✓	✓		✓	3	OOD
OLTW	✓		✓	✓	3	OOD	RST	✓		✓	✓	3	OOD
OBLTW	✓	✓	✓	✓	4	OOD	RAST	✓	✓	✓	✓	4	OOD

D.2 PROBLEM PARAMETERS AND DATA CONSTRUCTION

All methods use the same source datasets and held-out test instances within each problem family. Training covers the single-attribute tasks, while testing covers both source and unseen task types at a larger instance size. Table 7 summarizes the data splits.

VRP instances. Training coordinates and demand magnitudes are sampled independently. Optional attributes add pickups, travel limits, and time windows according to Table 8. For time windows, the center depends on the customer's distance from the depot, and the sampled interval is clipped to the depot horizon. Instances that fail the single-customer return-feasibility check are regenerated. We use fixed evaluation instances from the public VRP benchmark repository.[1]

FJSP instances. We use the public templates provided by Reijnen et al. (2026), preserving their processing times and eligible-machine sets. Training uses the e-data and r-data versions of Hurink la01 to la05, and validation uses the corresponding s-data and v-data versions. Validation data are excluded from search feedback. Test templates are disjoint from both splits. Each template receives independent realizations of the optional constraints, with each realization shared across all attribute subsets. Thus, variants differ only in which constraints are active. The base variant repeats the unchanged scheduling template across its realizations.

[1] https://github.com/CIAM-Group/URS

Table 7: Dataset sizes and base parameters.

Setting	VRP	FJSP
Training tasks × instances	4×10	4×10
Training size	50 customers and a depot	50 operations, 10 jobs, 5 machines
Test tasks × instances	$16 \times 1{,}000$	$16 \times 1{,}000$
Test size	100 customers and a depot	100 operations, 10 or 20 jobs
Test machines	–	5, 10, 11, 12, or 13
Coordinates / processing data	Uniform coordinates in $[0, 1]^2$	Public processing times and eligibility sets
Demand magnitudes	Uniform in $\{1, \ldots, 9\}$	–
Vehicle capacity	$Q = 50$	–
Training templates	–	10 Hurink e-data and r-data templates
Validation templates	–	10 Hurink s-data and v-data templates
Test templates	–	92 Hurink, 7 Barnes, 1 Brandimarte
Realizations per test template	–	10

FJSP timing attributes are scaled by the template workload and processing times. It is defined as

$$H = \max\left\{1, \operatorname{round}\left(\frac{\sum_o \min_{m \in \mathcal{M}_o} p_{om}}{M}\right)\right\}, \qquad \bar{p} = \frac{\sum_o \sum_{m \in \mathcal{M}_o} p_{om}}{\sum_o |\mathcal{M}_o|}, \tag{18}$$

where M is the number of machines, $\mathcal{M}_o$ contains the eligible machines for operation o, and p_{om} is its processing time on machine m. Table 8 gives the resulting constraint settings, with n denoting the number of operations. Integer-valued durations are sampled uniformly up to the rounded bound, with a minimum upper bound of one.

Table 8: Constraint generation parameters.

Family	Attribute	Setting
VRP	O	Routes end at the last customer without returning to the depot
	B	Pickups comprise 20% of customers, chosen uniformly
	L	Route-distance bound of 3
	TW	Speed 1, service duration $s = 0.2$, depot horizon $[0, 3]$
		Window center $\mathcal{U}(d_i, 3 - d_i - s)$ and half-width $\mathcal{U}(s/2, 1)$, where d_i is depot distance
FJSP	R	Release dates from 0 to $0.80H$, with the first job released at zero
	A	From the third job onward, probability 0.15 of one extra predecessor chosen from earlier jobs
	S	Setup bound $0.60\bar{p}$, zero within a family, and $F = \min\{5, \max\{3, \operatorname{round}(\sqrt{n}/2)\}\}$ balanced operation families
	T	Symmetric transportation delays bounded by $1.20\bar{p}$, with zero delay on the same machine

Shared feasibility and task information. All methods use the same feasibility masks and receive the same task information, including active-constraint indicators and VRP time-window features. For testing a variant without TW, service durations are zero and every node receives the same nonbinding time window. With n nodes, distance matrix D, and speed v, its upper bound is $H_{\text{wide}} = \max\{1, \max_{i,j} D_{ij}/v\}(5n + 21)$. Variants with TW retain their actual windows.

D.3 SEARCH AND BASELINE PARAMETERS

Common protocol. Search uses only source-task feedback, and the final heuristic sets are fixed before target evaluation. The common budget counts newly proposed solver programs, including invalid and duplicate proposals. Table 9 lists the common settings and principal method parameters.

Table 9: Common search settings and method parameters.

Common setting		Value
Generation model		GPT-4o-mini
New solver proposals per search		At most 2,000
Independent searches / final set size		3 / 4
Method	**Population**	**Principal parameters**
EoH	20	Initial allowance 40, two crossover parents
ReEvo	20	Mutation rate 0.5, short- and long-term reflection
MCTS-AHD	10	Initial population 4, $\alpha = 0.5$, $\lambda_0 = 0.1$
EoH-S	10	First-stage fraction 0.2, instance-level set selection
MEoH	20	Four source objectives, dominance-dissimilarity survival
HSEvo	10	Initial allowance 30, mutation rate 0.5, harmony memory 5
EMO-STA	40	Four islands, archive 16, shared allowance 1,000 and source adaptation 4×250
MoH	Heuristics 10 Optimizers 10	At most 10 outer iterations
MECo	10 retained per task	Source allowance 4×100, expansion allowance 16×100 Behavioral comparison on 32 states per source with tolerance 0.02

Final heuristic sets. EoH and ReEvo rank their final populations by mean normalized source cost. MCTS-AHD applies the same ranking to feasible programs in its search tree, and HSEvo uses its valid-program archive and final population, including numerical variants. Each selects the best distinct programs up to the prescribed set size.

EoH-S greedily adds the program that most reduces the sum of per-instance best-member losses, starting with the program of lowest mean loss. MEoH pools its final population and elitist archive, ranks candidates in descending order of its native dominance-dissimilarity survival score, and selects four programs. A predetermined random ordering resolves score ties. This four-member extraction adapts the native scoring principle to our common evaluation protocol.

EMO-STA selects each source's final adapted program, and MoH selects each source's best heuristic with its associated optimizer. All selections use training feedback alone.

D.4 CLASSICAL CONSTRUCTIVE REFERENCE RULES

We compare against deterministic construction rules using the same feasibility masks and state transitions as the generated heuristics. Remaining ties are resolved deterministically. Table 10 reports their costs on the common test sets.

VRP. Nearest-feasible chooses the admissible customer nearest to the current node. Savings-priority begins each route with the farthest admissible customer and then selects the customer maximizing $d(i, 0) + d(0, j) - d(i, j)$, where i is the current node, j the candidate, and 0 the depot. This adapts the savings expression of Clarke & Wright (2012) to sequential construction. The expression serves as a selection priority for both closed and open routes. A route ends when no customer is admissible.

FJSP. Earliest completion time (ECT) selects the feasible operation and machine assignment with the earliest completion under all active constraints. Most-work-remaining with ECT (MWKR-ECT) first favors the job with the greatest remaining workload,

$$W_j = \sum_{\substack{o \in j \\ o \text{ unscheduled}}} \min_{m \in \mathcal{M}_o} p_{om}, \tag{19}$$

and then uses earliest completion to choose its machine assignment. This extends compound dispatching to the present constraints (Lei et al., 2022). Each deterministic rule is evaluated once, and group means weight variants equally.

Table 10: Classical heuristic costs. Lower is better.

Problem	Rule	All (#16)	ID (#4)	OOD (#12)
VRP	Nearest-feasible	**23.53**	23.04	**23.69**
VRP	Savings-priority	24.51	**22.48**	25.19
FJSP	ECT	**1486.8**	**1301.4**	**1548.6**
FJSP	MWKR(min)-ECT	1751.7	1436.3	1856.9

VRP	Nearest-feasible	Savings-priority	FJSP	ECT	MWKR ECT
Base	20.85	**19.80**	Base	**1146.7**	1152.3
B	21.98	**20.33**	A	**1324.9**	1528.9
L	20.91	**19.79**	R	**1299.8**	1500.5
TW	35.61	**33.90**	S	**1291.8**	1366.4
O	**13.68**	15.88	T	**1289.2**	1349.2
BL	21.43	**19.44**	AS	**1462.2**	1725.0
BTW	38.92	**37.06**	AT	**1604.7**	1922.7
LTW	35.62	**33.85**	RA	**1464.6**	1875.3
OB	**15.62**	17.82	RS	**1429.0**	1689.4
OL	**13.68**	15.80	RT	**1469.4**	1687.8
OTW	**20.14**	25.10	ST	**1441.4**	1540.6
BLTW	38.89	**36.92**	AST	**1744.5**	2104.8
OBL	**15.40**	17.15	RAS	**1582.7**	2047.9
OBTW	**21.89**	27.25	RAT	**1768.3**	2256.9
OLTW	**20.13**	25.03	RST	**1589.6**	1859.2
OBLTW	**21.76**	27.00	RAST	**1880.7**	2420.8

E MORE EXPERIMENTS

E.1 COMPLETE TASK RESULTS

Table 11 and 12 give the results for every test variant. The four single-constraint variants are ID tasks. The base problem and the eleven multi-constraint combinations are OOD tasks. Best4 selects the lowest whole-dataset mean among four frozen heuristics, as defined in Equation 9. AHD results report means and sample standard deviations over three independent searches, while deterministic rules are evaluated once.

For each method, the main-table rank averages its per-variant midranks over all sixteen variants and all eleven methods. Ties use the displayed precision. The relative gap averages $100(\bar{f}_{m,v}/\min_k \bar{f}_{k,v} - 1)$ across variants, where $\bar{f}_{m,v}$ is method m's mean cost on variant v. Thus, the reference is the best observed method mean, not a known optimum. Group means and standard deviations are computed by first averaging the relevant variants within each search. In the detailed tables, bold and underlined entries indicate the best and second-best displayed means, and pale green highlights the best entries.

Table 11: VRP results on all variants. Lower is better.

Method	Base	B	L	TW
Nearest-feasible	20.85	21.98	20.91	35.61
Savings-priority	**19.80**	**20.33**	**19.79**	33.90
EoH	20.85 ±0.00	22.00 ±0.04	21.13 ±0.24	<u>31.77</u> ±0.07
ReEvo	20.89 ±0.15	22.05 ±0.10	20.95 ±0.22	32.84 ±1.43
MCTS-AHD	20.86 ±0.00	21.99 ±0.03	20.95 ±0.06	**31.76** ±0.10
EoH-S	20.69 ±0.07	22.10 ±0.14	20.82 ±0.11	32.09 ±0.54
MEoH	20.86 ±0.01	21.98 ±0.01	20.94 ±0.02	33.53 ±0.72
HSEvo	20.85 ±0.00	21.91 ±0.23	20.91 ±0.00	32.00 ±0.28
EMO-STA	20.37 ±0.84	21.08 ±1.55	20.50 ±0.70	35.05 ±0.97
MoH	<u>20.25</u> ±0.76	<u>21.01</u> ±1.50	20.39 ±0.63	33.19 ±0.61
MECo	20.37 ±0.84	21.08 ±1.55	<u>20.32</u> ±0.61	31.80 ±0.15

Method	O	BL	BTW	LTW
Nearest-feasible	13.68	21.43	38.92	35.62
Savings-priority	15.88	**19.44**	37.06	33.85
EoH	13.68 ±0.00	22.27 ±0.74	34.80 ±0.28	34.00 ±2.76
ReEvo	13.77 ±0.13	21.82 ±0.48	35.78 ±1.00	33.00 ±1.16
MCTS-AHD	13.68 ±0.00	21.50 ±0.12	34.87 ±0.43	<u>31.77</u> ±0.12
EoH-S	13.71 ±0.05	21.56 ±0.19	35.08 ±0.32	32.10 ±0.53
MEoH	13.68 ±0.01	21.46 ±0.02	36.20 ±0.75	34.27 ±0.54
HSEvo	13.66 ±0.03	21.33 ±0.33	<u>34.78</u> ±0.90	31.97 ±0.16
EMO-STA	<u>13.21</u> ±0.81	20.72 ±1.22	38.29 ±1.10	35.08 ±0.94
MoH	**12.94** ±0.70	<u>20.55</u> ±1.10	35.74 ±0.55	33.16 ±0.61
MECo	<u>13.21</u> ±0.81	20.58 ±1.09	**34.54** ±0.15	**31.74** ±0.10

Method	OB	OL	OTW	BLTW
Nearest-feasible	15.62	13.68	20.14	38.89
Savings-priority	17.82	15.80	25.10	36.92
EoH	15.62 ±0.00	14.65 ±1.14	**19.65** ±0.37	37.77 ±3.01
ReEvo	15.78 ±0.11	13.83 ±0.14	21.24 ±1.30	35.92 ±0.83
MCTS-AHD	15.61 ±0.01	13.71 ±0.05	20.00 ±0.03	34.85 ±0.60
EoH-S	15.84 ±0.41	13.66 ±0.03	20.25 ±0.42	35.46 ±0.99
MEoH	15.63 ±0.01	13.70 ±0.01	<u>19.83</u> ±0.13	36.95 ±0.59
HSEvo	15.57 ±0.04	13.67 ±0.02	20.02 ±0.28	<u>34.68</u> ±0.83
EMO-STA	<u>14.64</u> ±1.69	13.20 ±0.83	20.00 ±0.25	38.26 ±1.08
MoH	**14.35** ±1.50	<u>12.91</u> ±0.73	20.13 ±0.02	35.60 ±0.61
MECo	<u>14.64</u> ±1.69	**12.62** ±0.76	19.94 ±0.14	**34.41** ±0.16

Table 11 continued

Method	OBL	OBTW	OLTW	OBLTW
Nearest-feasible	15.40	21.89	20.13	21.76
Savings-priority	17.15	27.25	25.03	27.00
EoH	16.50 ±0.95	**21.42** ±0.54	21.04 ±1.86	23.36 ±1.90
ReEvo	15.78 ±0.38	23.10 ±1.01	21.27 ±1.17	23.13 ±0.90
MCTS-AHD	15.49 ±0.16	21.84 ±0.34	19.88 ±0.06	21.92 ±0.70
EoH-S	15.61 ±0.28	22.40 ±0.26	20.02 ±0.57	22.62 ±0.50
MEoH	15.44 ±0.06	21.45 ±0.10	19.78 ±0.15	21.37 ±0.11
HSEvo	15.35 ±0.08	21.66 ±0.91	**19.74** ±0.18	**21.35** ±0.75
EMO-STA	14.49 ±1.58	21.73 ±0.28	19.96 ±0.31	21.61 ±0.25
MoH	14.18 ±1.39	21.89 ±0.00	20.13 ±0.00	21.76 ±0.00
MECo	**13.62** ±0.83	21.59 ±0.12	19.83 ±0.15	21.45 ±0.10

Table 12: FJSP results on all variants. Lower is better.

Method	Base	A	R	S
ECT	1146.7	1324.9	1299.8	1291.8
MWKR-ECT	1152.3	1528.9	1500.5	1366.4
EoH	1061.8 ±28.3	1300.4 ±34.4	1216.5 ±37.0	1243.0 ±57.6
ReEvo	1044.4 ±12.1	1264.1 ±17.5	1245.1 ±38.0	1200.1 ±23.7
MCTS-AHD	1049.4 ±43.3	1269.6 ±10.2	1247.5 ±87.7	1214.2 ±30.5
EoH-S	1049.7 ±40.7	1258.9 ± 6.4	1243.9 ±76.2	1245.8 ±37.1
MEoH	1049.2 ±32.1	1258.2 ± 8.5	1239.5 ±27.1	1231.9 ±51.9
HSEvo	1036.2 ± 9.6	1268.0 ± 8.9	1218.3 ±11.0	1215.0 ±11.2
EMO-STA	1091.7 ±56.0	1279.8 ±37.5	1259.3 ±34.1	1232.5 ±51.5
MoH	1049.4 ± 2.0	1264.6 ±12.0	1233.0 ±11.5	1199.6 ±17.5
MECo	**1035.2** ± 5.6	**1251.8** ± 0.7	**1211.6** ± 0.8	**1181.3** ± 2.6

Method	T	AS	AT	RA
ECT	1289.2	1462.2	1604.7	1464.6
MWKR-ECT	1349.2	1725.0	1922.7	1875.3
EoH	1228.3 ±23.1	1464.0 ±58.9	1649.2 ±27.8	1428.7 ±30.1
ReEvo	1207.5 ±23.2	1415.2 ±26.8	1589.9 ±52.7	1446.2 ±48.2
MCTS-AHD	1227.3 ±77.7	1418.7 ±32.9	1617.5 ±50.5	1416.8 ±37.7
EoH-S	1218.0 ±64.1	1439.8 ± 4.7	1616.1 ±54.1	1405.2 ±23.6
MEoH	1219.2 ±27.1	1421.5 ±28.1	1617.0 ±40.6	1428.9 ±55.8
HSEvo	1199.4 ±14.2	1434.3 ± 6.1	1599.4 ±43.9	1431.0 ±36.0
EMO-STA	1251.3 ±47.2	1418.8 ±34.2	1580.3 ±20.4	1429.2 ±30.4
MoH	**1196.9** ±17.9	1393.8 ±10.2	1568.9 ± 9.4	1412.5 ±11.9
MECo	1200.8 ±11.7	**1388.8** ± 1.6	**1561.9** ± 1.0	**1402.5** ± 0.5

Method	RS	RT	ST	AST
ECT	1429.0	1469.4	1441.4	1744.5
MWKR-ECT	1689.4	1687.8	1540.6	2104.8
EoH	1365.3 ±43.8	1423.6 ±28.9	1418.4 ±39.5	1810.1 ±44.1
ReEvo	1376.0 ±43.9	1427.4 ±27.5	1358.5 ±32.3	1740.1 ±57.0
MCTS-AHD	1385.4 ±71.0	1484.5 ±63.7	1387.3 ±67.9	1755.2 ±49.0
EoH-S	1397.5 ±65.6	1464.5 ±22.1	1399.4 ±56.7	1773.5 ±34.1
MEoH	1378.2 ±43.3	1441.0 ±40.8	1384.5 ±45.3	1751.5 ±51.6
HSEvo	1358.0 ±12.9	1415.6 ±31.6	1365.9 ±10.9	1755.8 ±29.4
EMO-STA	1372.7 ±48.1	1441.9 ±18.7	1388.2 ±45.6	1722.1 ±23.9
MoH	1345.4 ±14.6	1417.7 ±12.7	1349.0 ±23.7	1699.2 ±12.4
MECo	**1326.9** ± 1.0	**1393.9** ± 1.6	**1334.6** ± 1.2	**1692.2** ± 5.7

Table 12 continued

Method	RAS	RAT	RST	RAST
ECT	1582.7	1768.3	1589.6	1880.7
MWKR-ECT	2047.9	2256.9	1859.2	2420.8
EoH	1561.4 ±40.8	1799.7 ±22.7	1570.6 ±27.0	1927.8 ±23.2
ReEvo	1574.4 ±53.3	1782.3 ±58.8	1554.0 ±43.2	1910.3 ±69.1
MCTS-AHD	1541.1 ±35.8	1786.0 ±69.0	1612.6 ±47.8	1896.3 ±53.6
EoH-S	1556.9 ±25.9	1786.0 ±71.5	1600.4 ±08.6	1908.2 ±49.3
MEoH	1555.5 ±58.9	1801.2 ±76.2	1562.2 ±54.8	1909.0 ±80.6
HSEvo	1561.8 ±32.1	1781.4 ±83.6	1545.9 ±22.6	1907.2 ±68.6
EMO-STA	1543.2 ±36.0	1743.6 ±21.2	1549.6 ±41.5	1856.6 ±24.6
MoH	1523.2 ±11.3	1729.2 ±14.6	1521.4 ±19.3	1838.3 ±15.1
MECo	**1511.9** ± 2.5	**1717.9** ± 1.4	**1500.4** ± 2.0	**1827.1** ± 3.0

E.2 Performance profiles across all variants

Figure 6 presents the per-variant relative gaps defined above. The radius uses $\log(1 + g)$, where g is the percentage gap, and the tick labels retain the original percentages. This scale makes small differences visible. A smaller radius indicates a lower cost relative to the best observed method. The shaded regions identify methods rather than uncertainty, and their areas should not be interpreted as an aggregate score.

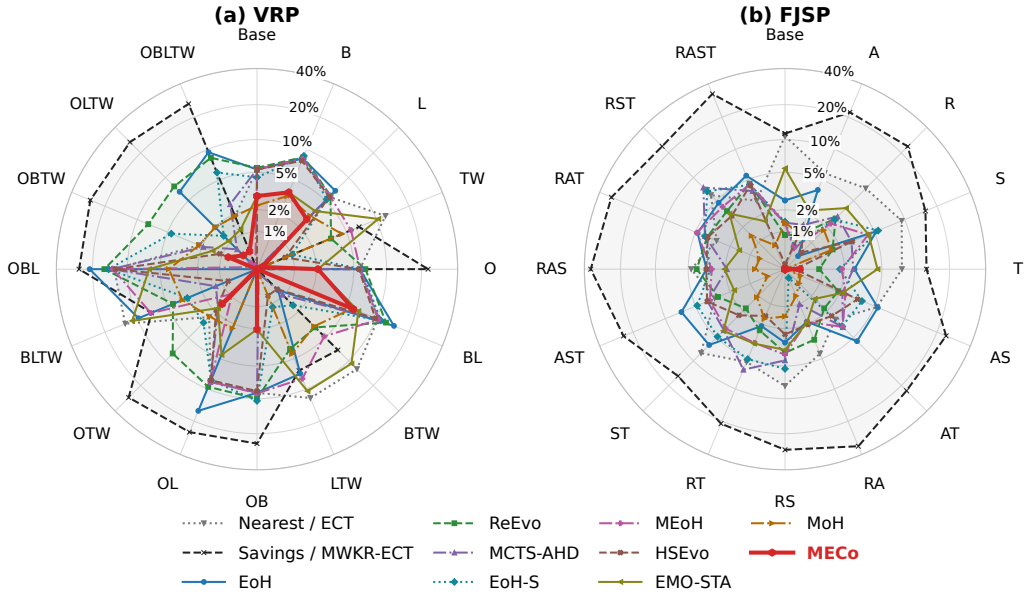

Figure 6: Relative gaps across all variants. Lower is better. MECo is red.

E.3 Performance by composition depth

Composition depth is the number of active optional constraints. Figure 7 and Table 13 summarize the multi-constraint variants at each depth. Costs are first averaged within each search and depth, then summarized across the three searches. The figure shows selected methods for readability, while the table includes the complete comparison. MECo leads at depths two and three in VRP and at all reported depths in FJSP, while HSEvo has the lowest mean on the four-constraint VRP variant.

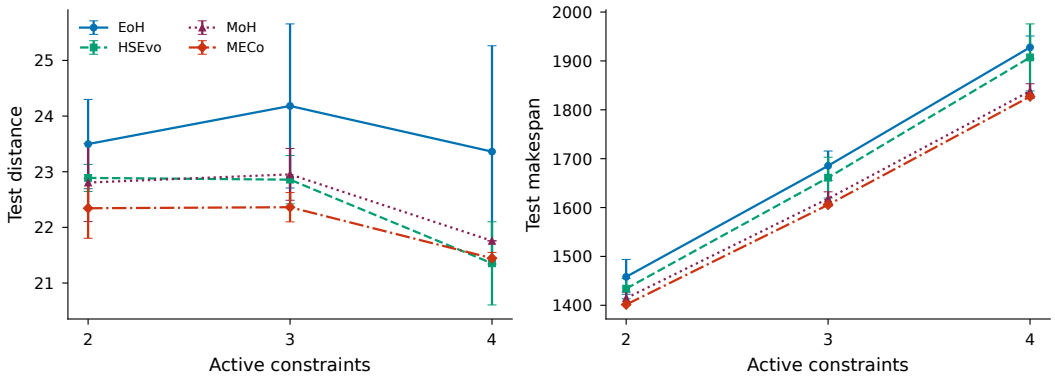

Figure 7: Costs by composition depth, with sample standard deviations.

Table 13: Costs by composition depth. Lower is better.

	VRP			FJSP		
Method	Depth 2	Depth 3	Depth 4	Depth 2	Depth 3	Depth 4
Nearest-feasible (VRP)						
ECT (FJSP)	24.23	24.08	21.76	1478.5	1671.3	1880.7
Savings-priority (VRP)						
MWKR-ECT (FJSP)	24.85	26.59	27.00	1740.1	2067.2	2420.8
EoH	23.50 ±0.80	24.18 ±1.47	23.36 ±1.90	1458.2 ±35.7	1685.5 ±30.2	1927.8 ±23.2
ReEvo	23.57 ±0.55	24.02 ±0.74	23.13 ±0.90	1435.5 ±29.4	1662.7 ±47.3	1910.3 ±69.1
MCTS-AHD	22.91 ±0.12	23.01 ±0.29	21.92 ±0.70	1451.7 ±53.6	1673.7 ±51.3	1896.3 ±53.6
EoH-S	23.08 ±0.15	23.37 ±0.17	22.62 ±0.50	1453.8 ±48.4	1679.2 ±53.2	1908.2 ±49.3
MEoH	23.52 ±0.04	23.40 ±0.22	21.37 ±0.11	1445.2 ±34.4	1667.6 ±54.4	1909.0 ±80.6
HSEvo	22.89 ±0.24	22.86 ±0.43	**21.35** ±0.75	1434.0 ±20.7	1661.2 ±41.8	1907.2 ±68.6
EMO-STA	23.66 ±0.54	23.61 ±0.41	21.61 ±0.25	1438.5 ±32.4	1639.6 ±30.2	1856.6 ±24.6
MoH	22.81 ±0.70	22.95 ±0.47	21.76 ±0.00	1414.6 ±12.5	1618.2 ±14.0	1838.3 ±15.1
MECo	**22.34** ±0.54	**22.36** ±0.26	21.45 ±0.10	**1401.4** ±0.7	**1605.6** ±2.9	**1827.1** ±3.0

E.4 Task pairing and generation operators

The component study in Table 4 replaces transfer-guided pairing with uniform sampling over admissible unused pairs, or removes one generation operator throughout search. Random pairing also removes the priority for unexplored unions. Removing transfer-oriented mutation does not disable execution-error correction. These experiments use the same proposal allowance.

E.5 Transferring the framework across local search engines

The framework study retains each engine's native program generation and survival rules, while MECo supplies the task populations, task pairing, and final heuristic-set selection. No additional execution-error repair is introduced. The main comparison in Table 3 reports three searches for each engine and family. If transferred parents cannot support native initialization, the hybrids use the engine's initialization procedure. For HSEvo, distinct programs with equal fitness may serve as parents when its usual selection rule cannot proceed.

For the additional FJSP reliability study, Table 14 lists ten independent searches for EoH, ReEvo, and their MECo counterparts. Each value averages a search's costs over the ID or OOD variants. The ten searches include the initial three and seven additional searches. We separately compare EoH with EoH-MECo and ReEvo with ReEvo-MECo on each split, giving four tests.

Each test is a two-sided exact independent permutation test. Its statistic is the absolute difference between the two groups' mean costs, and its null distribution enumerates all $\binom{20}{10}$ assignments of the twenty searches to groups of ten. Holm adjustment is then applied jointly to the four resulting p-values. Searches, rather than individual instances, are the independent observations. The adjusted values are reported on the MECo rows. All four are below 0.05, supporting the lower ID and OOD costs observed in Figure 5.

Table 14: FJSP costs over ten searches and Holm-adjusted p-values.

Method	Run 1	Run 2	Run 3	Run 4	Run 5	p
FJSP ID (#4)						
EoH	1203.90	1273.08	1264.14	1226.42	1216.71	–
EoH-MECo	1226.10	1206.31	1200.84	1212.34	1214.94	0.01146
ReEvo	1229.89	1216.86	1240.74	1213.55	1232.26	–
ReEvo-MECo	1189.72	1225.70	1206.62	1208.58	1230.63	0.01146
FJSP OOD (#12)						
EoH	1503.36	1563.39	1553.41	1499.68	1512.43	–
EoH-MECo	1508.95	1508.01	1500.05	1509.39	1493.24	0.00351
ReEvo	1534.43	1477.15	1543.06	1477.61	1521.41	–
ReEvo-MECo	1463.86	1522.43	1469.90	1469.16	1523.56	0.01146

Method	Run 6	Run 7	Run 8	Run 9	Run 10	p
FJSP ID (#4)						
EoH	1344.73	1242.49	1231.08	1210.22	1284.11	–
EoH-MECo	1212.56	1215.72	1216.31	1214.25	1213.56	0.01146
ReEvo	1284.63	1218.15	1208.12	1248.48	1323.22	–
ReEvo-MECo	1158.15	1204.30	1197.26	1215.17	1212.56	0.01146
FJSP OOD (#12)						
EoH	1722.40	1541.62	1533.27	1503.76	1627.10	–
EoH-MECo	1475.81	1497.05	1497.31	1493.51	1485.92	0.00351
ReEvo	1576.14	1492.95	1490.40	1552.26	1692.36	–
ReEvo-MECo	1441.29	1467.63	1460.41	1477.38	1474.04	0.01146

Both panels repeat the same p-values, computed from all ten searches.

E.6 VRP TARGET ADAPTATION

We further allow the frozen heuristic sets to adapt using training data from two target tasks, OB and BLTW. Each task provides ten size-50 training instances, and evaluation uses the unchanged size-100 test set. Local baselines search independently on each target, whereas EMO-STA, MoH, and MECo share their additional proposal allowance across the two targets. Table 15 reports the final costs and the total allowance for both targets.

All final programs are selected using target training feedback, and Best4 is computed only after selection. The entries report means and sample standard deviations over three searches. Average rank is the mean of the two target ranks among all listed methods, with ties assigned midranks at the displayed precision. EMO-STA uses its native scoring mechanism. Although MECo does not achieve the lowest cost on either target, its costs remain close to the best observed values and it achieves the best average rank. Moreover, MECo uses half as many additional proposals as the task-specific baselines, demonstrating more efficient use of the adaptation proposal budget.

Table 15: VRP target adaptation. Costs are mean ± sample SD over three searches.

Method	Total proposals	OB ↓	BLTW ↓	Avg. rank ↓
EoH	500 + 500	15.7304 ±0.1173	34.1699 ±0.4925	5.00
EoH-S	500 + 500	15.5708 ±0.1741	34.1896 ±0.4038	4.50
ReEvo	500 + 500	15.7903 ±0.0376	**33.8441** ±0.0562	4.00
MEoH	500 + 500	15.7702 ±0.0267	34.1351 ±0.5875	5.00
MCTS-AHD	500 + 500	15.8940 ±0.4708	34.5688 ±0.0305	8.00
HSEvo	500 + 500	15.9109 ±0.1988	34.1125 ±0.4006	6.00
EMO-STA	500	15.6168 ±0.0000	38.8891 ±0.0000	6.50
MoH	500	**14.3453** ±1.5030	34.2801 ±0.3966	4.00
MECo	500	14.6400 ±1.6918	33.8955 ±0.1653	**2.00**

E.7 Generalization to larger VRP instances

We evaluate the frozen MECo sets on 1,000-customer VRP instances and compare them with the public 100-customer MTNCO checkpoint (Liu et al., 2024a). Both methods use the same 100 test instances per variant. MTNCO uses 1,000 starting points, either without augmentation or with eight augmentations. MECo retains whole-dataset Best4 evaluation and reports variation across its three searches. This is a scale-generalization comparison, not an equal-training-budget comparison. ID and OOD follow MECo's source-task split and do not imply that all OOD variants were unseen during MTNCO training. As shown in Table 16, MECo has lower aggregate costs in both groups.

Table 16: Frozen evaluation on 1,000-customer VRP instances. Lower is better.

Method	All (#16)	ID (#4)	OOD (#12)
MTN	148.67	115.16	159.85
MTN 8aug	137.05	109.33	146.29
MECo	**106.60** ± 2.54	**88.75** ± 2.40	**112.55** ± 2.62

E.8 Search across heterogeneous tasks

We also study joint heuristic design across TSP, base VRP, and FJSP. Each method shares an allowance of 1,000 new programs across the three tasks and selects one executable expert per task using training feedback. Each task uses 64 training instances of size 50 and 1,000 test instances at each of sizes 50 and 100, measured in nodes, customers, or operations. The same expert is evaluated at both test sizes. Table 17 reports means and sample standard deviations over three searches, without averaging incompatible objective units.

MECo achieves lower TSP and VRP costs than EMO-STA and MoH at both test sizes. It also improves over EMO-STA on FJSP, where MoH performs best. These results extend the evaluation of MECo from constraint variants within a family to joint design across different constructive problems. The experiment concerns the participating tasks, rather than zero-shot transfer to an unseen problem type.

Table 17: Joint design across three tasks. Mean ± sample SD.

Task	Size	EMO-STA	MoH	MECo
3 tasks				
TSP	50	6.99 ±0.00	6.46 ±0.19	**6.34** ±0.07
	100	9.69 ±0.00	9.04 ±0.32	**8.83** ±0.11
VRP	50	13.53 ±0.00	13.41 ±0.63	**12.97** ±0.51
	100	23.35 ±0.00	23.03 ±1.16	**22.31** ±1.02
FJSP	50	132.87 ±7.03	**115.02** ±0.30	123.00 ±9.15
	100	236.38 ±12.29	**210.38** ±0.92	222.57 ±11.72

E.9 Resource use

Table 18 reports training time, test time, and token use for the main experiments. Training includes program generation, source evaluation, and final selection. Testing covers the frozen evaluation.

Table 18: Resource use per search, with sample standard deviations.

Method	Train (h)	Test (h)	Input (M)	Output (M)	Tokens (M)
VRP					
EoH	0.77 ±0.26	1.26 ±0.13	2.16 ±0.09	0.84 ±0.03	3.00 ±0.11
ReEvo	1.04 ±0.08	0.87 ±0.17	4.97 ±0.17	1.26 ±0.07	6.23 ±0.23
MCTS-AHD	5.64 ±0.07	0.94 ±0.16	8.17 ±0.55	1.18 ±0.02	9.35 ±0.56
EoH-S	0.99 ±0.32	1.25 ±0.18	2.54 ±0.06	1.03 ±0.04	3.57 ±0.11
MEoH	0.92 ±0.29	1.25 ±0.09	2.21 ±0.18	0.88 ±0.05	3.09 ±0.22
HSEvo	5.17 ±0.67	1.06 ±0.08	5.72 ±0.36	1.39 ±0.13	7.11 ±0.49
EMO-STA	3.90 ±0.29	1.48 ±0.10	12.05 ±1.22	0.92 ±0.09	12.97 ±1.32
MoH	1.39 ±0.48	1.55 ±0.43	1.47 ±0.89	1.11 ±0.66	2.58 ±1.55
MECo	2.26 ±0.12	0.98 ±0.20	3.41 ±0.06	1.27 ±0.03	4.68 ±0.09
FJSP					
EoH	0.97 ±0.15	0.09 ±0.01	2.57 ±0.08	1.07 ±0.04	3.64 ±0.12
ReEvo	1.01 ±0.21	0.06 ±0.01	7.13 ±0.91	1.86 ±0.31	8.99 ±1.22
MCTS-AHD	7.32 ±2.57	0.04 ±0.00	11.66 ±1.92	1.63 ±0.08	13.28 ±1.99
EoH-S	0.66 ±0.02	0.06 ±0.04	2.82 ±0.10	1.16 ±0.05	3.98 ±0.15
MEoH	0.82 ±0.17	0.07 ±0.02	2.69 ±0.30	1.09 ±0.14	3.77 ±0.43
HSEvo	2.46 ±0.63	0.06 ±0.02	7.14 ±0.10	1.64 ±0.07	8.78 ±0.18
EMO-STA	6.25 ±0.35	0.09 ±0.01	23.52 ±1.94	1.69 ±0.14	25.21 ±2.08
MoH	2.82 ±0.18	0.26 ±0.21	3.51 ±0.09	2.01 ±0.17	5.51 ±0.20
MECo	1.79 ±0.04	0.04 ±0.01	4.35 ±0.02	1.33 ±0.01	5.68 ±0.03

F GENERATION PROMPTS AND INTERFACES

A generation prompt combines a task description, a function interface, the descriptions of active constraints, and an operator instruction. Parent programs and their source scores provide the search context. We present these components in that order.

F.1 TASK DESCRIPTIONS AND INPUT TEMPLATES

The VRP contract below comes from the active-constraint-aware search interface. Its accompanying template specifies the five function arguments.

VRP task description

```
Choose one next node for the active route. `instance['active_constraints']` lists
the lowercase constraint primitives active in the current variant; compose behavior
 from primitive membership instead of matching a complete variant name. Only fields
 belonging to active constraints are documented in the target task description.
Return a feasible unvisited customer when continuing the route; check `
feasibility_check(node)` before returning it. Return `destination_node` only to end
 the current route.
```

```python
from collections.abc import Callable

import numpy as np


def select_next_node(
    instance: dict,
    current_node: int,
    destination_node: int,
    unvisited_nodes: np.ndarray,
    feasibility_check: Callable[[int], bool],
) -> int | None:
    """Select one feasible next node for the active route.

    Args:
        instance: State and candidate metrics described in the task. The
            active_constraints list identifies primitive constraint membership.
        current_node: int, last node on the active route.
        destination_node: int, depot/destination. Returning it ends the route.
        unvisited_nodes: np.ndarray[int], shape (m,).
        feasibility_check: callable that checks whether a node can be appended.

    Returns:
        A feasible node from unvisited_nodes, or destination_node to end the route.
    """
    pass
```

The FJSP example describes the release-date source task. The interface takes the current instance, the partial schedule, and the admissible actions.

FJSP task description

```
Design a constructive operation-machine dispatch heuristic for a target with active
 primitives `'r'`. `instance['active_constraints']` lists the active lowercase
primitives. Compose behavior from membership in this list; do not hard-code
complete variant names. Each action is `(operation_id, machine_id)`. `actions`
already contains only ready and eligible choices; return one member and rank
choices for makespan quality. Do not reimplement readiness. Always available: `
operations`, `job_operations`, and the partial schedule in `state`, including `
scheduled`, `machine_sequences`, and `job_previous`. Active `r` provides `
release_dates`.
```

```python
import math
import numpy as np

def select_action(
    instance: dict,
    state: dict,
    actions: list[tuple[int, int]],
) -> tuple[int, int]:
    """Choose one ready operation-machine action.

    Args:
        instance: Current FJSP instance with active constraint IDs.
        state: Partial schedule and predecessor information.
        actions: Ready and eligible (operation_id, machine_id) pairs.

    Returns:
        One pair contained in actions.
    """
    pass
```

F.2 Descriptions for active attributes

Only the descriptions of active constraints are appended to the task prompt. Each family has four such descriptions.

VRP constraints

```
O: open routes

B: interleavable linehaul and backhaul service; a route starting with a pickup
starts empty and contains only pickups
Linehaul/backhaul membership is available in the instance state.

TW: Time-window candidate metrics provide normalized `arrival_ratio`, `
waiting_ratio`, `slack_ratio`, and `finish_ratio`; route state provides `
current_time_ratio` and `time_window_slack_ratio`.

L: Route-distance state provides `route_distance`, `route_limit`, and `
remaining_route_budget_ratio`.
```

FJSP constraints

```
R: Active `r` provides `release_dates`.

A: Active `a` provides `assembly_predecessors`.

S: Active `s` provides `num_setup_families`, `setup_families`, and `setup_times`.

T: Active `t` provides `transport_times`.
```

F.3 Generation instructions

Each proposal uses one of the following instructions together with its selected parents and search context.

Mechanism composition c_1

```
Identify one concrete decision mechanism in each parent (for example a ranking
signal, look-ahead, or route-ending rule). Create a child that composes only
compatible mechanisms. Do not paste two complete score functions together; explain
the composed mechanism briefly in the algorithm description.
```

Conflict resolution c_2

```
Identify a concrete conflict between the decision biases of Parent A and Parent B.
Create a compact conditional or hierarchical rule that resolves the conflict from
the documented runtime state. The result must be one coherent heuristic, not an
average of two arbitrary scores.
```

Exploratory crossover c_3

```
The parents occupy different regions of the measured root-performance Pareto set.
Design a coherent candidate that improves the under-covered target dimension named
in the search context using a genuinely different decision principle or look-ahead
structure. Do not diversify by renaming variables or perturbing constants.
```

The mutation instruction below uses one parent and its source evaluation.

Single-parent mutation m_1

```
Keep the parent algorithm as the backbone. Use the task description and source
evaluation to identify a decision mechanism to improve. Revise that mechanism while
 preserving the parent's useful structure and the documented function interface.
```

G REPRESENTATIVE EVOLVED HEURISTICS

We present two detailed heuristics from the frozen sets produced by the MECo searches, selected for their program structure rather than test performance. The listings preserve their executable code and omit only the main function's interface documentation.

G.1 VRP EXAMPLE

The VRP example is the third selected member from the first search. It combines feasibility filtering with a score based on travel distance and temporal features. After selecting a customer, an additional route-budget check can terminate the route instead.

```python
from collections.abc import Callable
import numpy as np

def select_next_node(instance: dict, current_node: int, destination_node: int, unvisited_nodes: np.ndarray, feasibility_check: Callable[[int], bool]) -> int | None:
    active_constraints = instance.get('active_constraints', [])
    candidate_metrics = instance.get('candidate_metrics', {})
    distance_matrix = instance.get('dismatrix', {})
    remaining_capacity = instance.get('remaining_capacity', float('inf'))
    route_distance = instance.get('route_distance', 0)
    route_limit = instance.get('route_limit', float('inf'))
    remaining_route_budget_ratio = instance.get('remaining_route_budget_ratio', 1.0)
    feasible_nodes = []
    for node in unvisited_nodes:
        node = int(node)
        if not feasibility_check(node):
            continue
        metrics = candidate_metrics.get(node)
        if metrics is None:
            continue
        if 'capacity' in active_constraints:
            demand = metrics['demand']
            remaining_after = metrics['remaining_after']
            if demand > remaining_capacity:
                continue
            if 'l' in active_constraints and demand > 0 and (remaining_after <= 0):
                continue
        if 'tw' in active_constraints and (not metrics.get('feasible', True)):
            continue
        feasible_nodes.append(node)
    if not feasible_nodes:
        return int(destination_node)
    scores = {}
    for node in feasible_nodes:
        dist_to_node = distance_matrix[current_node][node]
        metrics = candidate_metrics[node]
        wait_ratio = metrics.get('waiting_ratio', 0.0)
        finish_ratio = metrics.get('finish_ratio', 0.0)
        arrival_ratio = metrics.get('arrival_ratio', 0.0)
        slack_ratio = metrics.get('slack_ratio', 0.0)
        score = dist_to_node + wait_ratio + finish_ratio + arrival_ratio * 0.5
        if 'tw' in active_constraints and (not metrics.get('feasible', True)):
            score += 1000
        scores[node] = score
    selected_node = min(scores, key=scores.get)
    dist_next = distance_matrix[current_node][selected_node]
    if route_distance + dist_next > route_limit * remaining_route_budget_ratio:
        return int(destination_node)
    return int(selected_node)
```

G.2 FJSP EXAMPLE

The FJSP example is the fourth selected member from the third search. It estimates start times from release dates, predecessor completion, transportation delays, and machine availability. Its ranking adds an extra setup penalty to estimated completion time when setups are active, favoring assignments with smaller setup costs. The environment continues to enforce action feasibility.

```python
import math
import numpy as np

def select_action(instance: dict, state: dict, actions: list[tuple[int, int]]) -> tuple[int, int]:
    if not actions:
        raise ValueError('No ready action')
    operations = {int(operation['id']): operation for operation in instance['operations']}
    active = set(instance.get('active_constraints', ()))

    def duration(operation_id: int, machine: int) -> int:
        """Fetch the processing duration for a given operation on a specified machine."""
        for alternative in operations[operation_id]['alternatives']:
            if int(alternative['machine']) == int(machine):
                return int(alternative['duration'])
        return 10 ** 12

    def compute_start_time(operation_id: int, machine: int) -> int:
        """Compute the start time for an operation on a machine considering release and transport times."""
        start_time = int(instance['release_dates'][int(operations[operation_id]['job'])]) if 'r' in active else 0
        previous = state['job_previous'].get(operation_id)
        if previous is not None:
            previous_record = state['scheduled'][previous]
            start_time = max(start_time, int(previous_record['end']))
            if 't' in active:
                start_time = max(start_time, int(previous_record['end']) + int(instance['transport_times'][int(previous_record['machine'])][machine]))
        sequence = state['machine_sequences'][machine]
        last_operation = sequence[-1] if sequence else None
        machine_ready_time = 0 if last_operation is None else int(state['scheduled'][last_operation]['end'])
        return max(start_time, machine_ready_time)

    def compute_setup_time(operation_id: int, machine: int) -> int:
        """Calculate setup time based on the last operation's family."""
        if 's' in active:
            families = instance['setup_families']
            last_operation = state['machine_sequences'][machine][-1] if state['machine_sequences'][machine] else None
            previous_family = int(instance['num_setup_families']) if last_operation is None else int(families[last_operation])
            return int(instance['setup_times'][machine][previous_family][int(families[operation_id])])
        return 0

    def compute_completion(action: tuple[int, int]) -> int:
        """Calculate the total completion time for an action."""
        (operation_id, machine) = map(int, action)
        start_time = compute_start_time(operation_id, machine)
        setup_time = compute_setup_time(operation_id, machine)
        start_time += setup_time
        processing_duration = duration(operation_id, machine)
        return start_time + processing_duration
    best_action = min(actions, key=lambda action: compute_completion(action) + (compute_setup_time(action[0], action[1]) if 's' in active else 0))
    return tuple(map(int, best_action))
```

H LICENSES

The licenses and URLs of baselines are listed in Table 19.

Table 19: Licenses and URLs of the baseline implementations used in this work.

Method	License	URL
EoH	BSD 2-Clause	https://github.com/Optima-CityU/LLM4AD
ReEvo	BSD 2-Clause	https://github.com/Optima-CityU/LLM4AD
MCTS-AHD	BSD 2-Clause	https://github.com/Optima-CityU/LLM4AD
EoH-S	MIT	https://github.com/FeiLiu36/EoH-S
MEoH	BSD 2-Clause	https://github.com/Optima-CityU/LLM4AD
HSEvo	MIT	https://github.com/datphamvn/HSEvo
MoH	MIT	https://github.com/yiding-s/MoH
EMO-STA	Apache 2.0	https://github.com/alperengozeten/emo