BEYOND HEURISTIC CEILING: CO-EVOLVING LLM-GENERATED HEURISTICS AND RL SELECTION FOR COMBINATORIAL OPTIMIZATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Neural combinatorial optimization faces a trade-off between solution quality and learning efficiency. End-to-end construction policies can attain high solution quality but incur substantial training cost and scale poorly due to their growing action space, whereas reinforcement learning (RL)-based heuristic selection learns efficiently over a compact action space but its performance ceiling is limited by the heuristic pool. To break this ceiling while retaining the learning efficiency of heuristic selection, we propose **CoEvoRL**, a collaborative framework that co-evolves LLM-generated heuristics and an RL-based selector. We formulate heuristic selection as a restricted Markov decision process and decompose its suboptimality into an action-abstraction gap induced by insufficient heuristic coverage, and a finite-sample RL training error whose bound increases with pool size. Motivated by this decomposition, CoEvoRL alternates between selector training and heuristic evolution in a closed loop, where the selector identifies coverage gaps in the current heuristic pool to guide the LLM toward complementary heuristics, while the evolved pool reshapes the selection space and drives further policy improvement. Two key designs make this loop practical: a variable-pool heuristic selector that accommodates changing heuristic pools without altering the network architecture, and a selective admission criterion that evaluates candidates by their marginal coverage gains against pool expansion costs to keep the pool compact. CoEvoRL reduces the optimality gap of heuristic selection by 4.6%–39.1%, while matching or outperforming substantially more costly construction policies, using only a lightweight selector and a small fraction of their training budget.

1 INTRODUCTION

Neural combinatorial optimization (NCO) has been widely studied in transportation, logistics, and manufacturing (Bengio et al., 2021; Cappart et al., 2023; Zhang et al., 2023; Chung et al., 2025), where many problems admit sequential decision-making formulations, and reinforcement learning (RL) has become a predominant approach to learning high-quality decision policies (Mazyavkina et al., 2021). From the perspective of action representation, NCO approaches can be broadly categorized into two paradigms: *learning construction policy* and *learning heuristic selection*.

Learning construction policy approaches (e.g., Kool et al., 2019; Kwon et al., 2020; Zhang et al., 2020) select a primitive action at each decision step, such as the next city to visit or the next operation to dispatch. Operating over the full feasible action set, they can represent flexible policies and attain strong solution quality. However, their action space grows with problem size, making learning increasingly costly and sample-inefficient (Joshi et al., 2022), which is especially restrictive in practical settings where training instances are limited or extensive training is prohibitive. In contrast, heuristic selection approaches choose from a small pool of predefined heuristics at each decision step to determine the primitive action to execute (Choong et al., 2018; Zhao et al., 2023; Kallestad et al., 2023). This compact action space substantially improves learning efficiency, but its achievable performance is fundamentally bounded by the heuristic pool: the selector cannot choose an action that the pool never proposes, and improving the selector alone cannot remove this limitation.

Our code and data are available at https://anonymous.4open.science/r/coevorl-923A.

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.

Recent advances in large language models (LLMs) offer a promising way to overcome this limitation by automatically generating and refining executable heuristics with limited human effort (Romera-Paredes et al., 2024; Liu et al., 2024; Ye et al., 2024). However, existing works commonly optimize heuristics for their standalone performance, which does not reflect their value under adaptive selection. For example, a strong standalone heuristic may contribute little if its behavior overlaps with actions already available to the selector, whereas a weaker one can be valuable if it precisely supplies high-quality actions where the current pool fails. This distinction shifts the target of heuristic evolution from standalone solution quality to the collective coverage of the pool. Moreover, updating the heuristic pool reshapes the action space and requires the selector to adapt accordingly. This bidirectional dependence motivates a closed-loop framework, in which selector feedback guides heuristic evolution, while the evolved pool, in turn, drives further selector adaptation.

To quantify how the heuristic pool shapes solution quality and learning efficiency, we first formulate heuristic selection as a restricted Markov decision process (MDP). This formulation decomposes the suboptimality of a learned selector into two terms: 1) an *action-abstraction gap*, incurred when the pool fails to cover high-value primitive actions during solution construction, and 2) a *finite-sample RL training error*, which grows with the size of the selection space under a fixed training budget. The former quantifies the performance ceiling imposed by the heuristic pool, whereas the latter captures the increasing training difficulty and gap over a larger selection space. This decomposition reveals a fundamental trade-off: expanding the pool improves action coverage and thus reduces the abstraction gap, but at the cost of greater learning complexity for the selector.

Building on this decomposition, we propose **CoEvoRL**, a collaborative framework that co-evolves LLM-generated heuristics and an RL-based selector to break the performance ceiling of heuristic selection while retaining a compact pool for efficient learning. Specifically, CoEvoRL alternates between RL-based selector training and LLM-based heuristic evolution in a closed loop. During training, the selector is rolled out to identify states in which the current heuristic pool fails to propose high-quality actions. This coverage diagnosis is then provided to the LLM, directing heuristic evolution toward specific weaknesses of the heuristic pool rather than standalone performance. A candidate heuristic is admitted only if its marginal coverage gain outweighs the incremental statistical learning cost induced by pool expansion. Notably, a key design to support an evolving heuristic pool is that CoEvoRL parameterizes the selector using a variable-pool scorer, which allows heuristics to be added or removed without modifying the policy network. Once the pool is updated, the selector adapts to the resulting action space, exploits the newly introduced heuristics, and exposes the remaining coverage gaps for the next round. This bidirectional co-evolution enables the LLM and RL components to progressively improve action coverage while preserving a compact and learnable selection space. Our main contributions are summarized as follows:

- We formulate RL-based heuristic selection as a restricted MDP, and decompose its suboptimality into an action-abstraction gap and a finite-sample RL training error, formalizing the trade-off between heuristic coverage and learning complexity.
- We propose CoEvoRL, a closed-loop framework that co-evolves LLM-generated heuristics and an RL-based selector, where selector-derived coverage diagnosis guides LLM-based heuristic evolution, while the evolving heuristic pool drives further selector adaptation.
- We develop a variable-pool heuristic selector that accommodates an evolving heuristic pool without modifying the policy network, together with a coverage-aware admission criterion that balances marginal coverage gain against the incremental statistical learning cost of pool expansion.
- CoEvoRL reduces the optimality gap of heuristic selection by 4.6%–39.1%, matching or outperforming far more costly construction policies, and offers a promising path toward combining the learning efficiency of heuristic selection with solution quality approaching construction policies.

2 Related Work

Neural Combinatorial Optimization. NCO learns parameterized policies, typically represented as neural networks, for solving CO problems (Bengio et al., 2021). A major line of work learns construction policies that build a solution by selecting primitive actions in an autoregressive manner (Mazyavkina et al., 2021). Vinyals et al. (2015) introduced a sequence-based construction architecture, and Bello et al. (2016); Nazari et al. (2018) then optimized the decision policy through RL.

Subsequent studies have improved solution quality through stronger graph representations (Kool et al., 2019; Zhang et al., 2020), inference augmentation (Kwon et al., 2020; Kim et al., 2022), and structure-aware optimization (Yi et al., 2026; Xiao et al., 2026). While these approaches can attain strong solution quality, their action space is tied directly to the primitive decision space (e.g., the number of cities or operations), and their sample complexity and training cost grow with problem size, motivating a more compact action abstraction for better learning efficiency.

Learning-Based Heuristic Selection. An alternative line of work uses RL to select over a space of low-level heuristics at each decision step, rather than choosing primitive actions directly (Drake et al., 2020; Burke et al., 2013; Li et al., 2024). For example, Choong et al. (2018) use Q-learning to automatically configure heuristic selection to replace hand-tuned adaptive rules, and Kallestad et al. (2023) integrate deep RL into large neighborhood search to select operators. Since the action space in these methods is confined to a fixed heuristic pool which is independent of the problem size, the resulting policies are learned efficiently and scale well across problem sizes. However, their achievable solution quality is fundamentally bounded by the coverage of the underlying heuristic pool, i.e., the action-abstraction gap formalized in our restricted MDP analysis, which CoEvoRL is designed to improve by evolving the pool itself rather than treating it as fixed.

LLM-Based Automatic Heuristic Design. Recent works have used LLMs for automatic heuristic design through LLM-based contextual operators within an evolutionary search loop (Novikov et al., 2025; Liu et al., 2026; Chen et al., 2026). Representative methods such as FunSearch (Romera-Paredes et al., 2024), EoH (Liu et al., 2024), and ReEvo (Ye et al., 2024) iteratively prompt LLMs to generate and refine heuristic code guided by evaluation signals and reflective feedback. Although these approaches may maintain populations of candidate heuristics, candidates are generally evaluated by their standalone performance, which captures little about their value under adaptive selection. The closest work to ours is HeurAgenix (Yang et al., 2025), which first evolves a heuristic pool offline and trains a selector to choose among the resulting heuristics. Nevertheless, its evolution and selection stages remain strictly separated, and its selector is fine-tuned to imitate LLM preference rather than trained to optimize the objective directly, so selection outcomes never diagnose or feed back into heuristic evolution. In contrast, CoEvoRL closes this loop, where RL-derived coverage diagnosis directly guides LLM-based heuristic evolution, while the evolved pool reshapes the selection MDP and drives further policy improvement.

3 Restricted MDP & Suboptimality Decomposition

3.1 Preliminaries

We consider the sequential construction of a feasible solution x minimizing a cost $c(x)$. For a given problem instance, we model this process as a finite-horizon *primitive MDP* $\mathcal{M} = (\mathcal{S}, \mathcal{A}, P, r, T)$, where $\mathcal{S}$ and $\mathcal{A}$ are the state and primitive action spaces, P and r are the transition kernel and reward function, and T is the decision horizon determined by the problem and construction scheme. The action set $\mathcal{A}_t(s) \subseteq \mathcal{A}$ contains all feasible primitive actions in state s at step t, such as selecting an unvisited city or dispatching an eligible operation, which can be very large as problem size grows.

A primitive construction policy π selects actions according to $\pi_t(\cdot \mid s) \in \Delta(\mathcal{A}_t(s))$, where $\Delta(\cdot)$ is the probability simplex over the specified action set. The expected cumulative return $J_{\mathcal{M}}(\pi) = \mathbb{E}_{P,\pi}[\sum_{t=1}^{T} r_t(s_t, a_t)]$ is typically designed to equal $-\mathbb{E}_{P,\pi}[c(x)]$, and thus maximizing expected return is equivalent to minimizing expected solution cost. Throughout, we assume per-step rewards are bounded and, without loss of generality, affinely rescaled so that $r_t(s, a) \in [0, 1]$. We write $J^*_{\mathcal{M}} := \max_\pi J_{\mathcal{M}}(\pi)$ for the optimal return in the primitive MDP.

For the subsequent analysis, $V^\pi_t(s)$ and $Q^\pi_t(s, a)$ denote the state- and action-value functions, respectively. Their optimal counterparts $V^*_t(s)$ and $Q^*_t(s, a)$ satisfy the Bellman optimality equations.

3.2 Restricted MDP Formulation

Heuristic selection instead restricts the available action space to a small pool of heuristics, independent of the problem size. Consider a fixed, nonempty pool $\mathcal{H} = \{h_1, \ldots, h_K\}$ of deterministic

one-step heuristics, indexed by $[K] = \{1, \ldots, K\}$, where each h_k proposes a feasible primitive action $h_{k,t}(s) \in \mathcal{A}_t(s)$ in state s at step t. The heuristic pool $\mathcal{H}$ induces the available action set

$$\mathcal{A}_t^{\mathcal{H}}(s) = \{h_{k,t}(s) : k \in [K]\} \subseteq \mathcal{A}_t(s), \tag{1}$$

which provides a state-dependent restriction of the primitive action space with $|\mathcal{A}_t^{\mathcal{H}}(s)| \leq K$.

A heuristic selector μ specifies a distribution $\mu_t(\cdot \mid s) \in \Delta([K])$ over the heuristic pool. At each step t, it samples $k_t \sim \mu_t(\cdot \mid s_t)$, and the primitive action $a_t = h_{k_t,t}(s_t)$ is executed. This interaction induces a *restricted MDP* over the heuristic pool.

Definition 1. Given a primitive MDP $\mathcal{M}$ and a heuristic pool $\mathcal{H}$, the induced restricted MDP is

$$\mathcal{M}_{\mathcal{H}} := (\mathcal{S}, [K], P^{\mathcal{H}}, r^{\mathcal{H}}, T), \tag{2}$$

$$\text{with } P_t^{\mathcal{H}}(s' \mid s, k) = P_t(s' \mid s, h_{k,t}(s)), \quad r_t^{\mathcal{H}}(s, k) = r_t(s, h_{k,t}(s)). \tag{3}$$

To relate the restricted MDP to primitive construction, any selector μ induces a primitive policy

$$\pi_t^{\mu,\mathcal{H}}(a \mid s) = \sum_{k=1}^{K} \mu_t(k \mid s)\, \mathbb{1}\{h_{k,t}(s) = a\}. \tag{4}$$

Since selector μ and its induced primitive policy $\pi^{\mu,\mathcal{H}}$ generate identical distributions over states, primitive actions, and rewards, we have $J_{\mathcal{M}_{\mathcal{H}}}(\mu) = J_{\mathcal{M}}(\pi^{\mu,\mathcal{H}})$. Conversely, every primitive policy supported on $\mathcal{A}_t^{\mathcal{H}}(s)$ can be realized by some selector over $\mathcal{M}_{\mathcal{H}}$ (Appendix A.1). Hence, heuristic selection optimizes the primitive objective subject only to actions $\mathcal{A}_t^{\mathcal{H}}(s)$ supported by the pool.

This equivalence enables us to define $J^*_{\mathcal{M}_{\mathcal{H}}} := \max_\mu J_{\mathcal{M}_{\mathcal{H}}}(\mu)$, the best return achievable through heuristic selection within the restricted MDP $\mathcal{M}_{\mathcal{H}}$. It serves as an intermediate benchmark between the primitive optimum $J^*_{\mathcal{M}}$ and the return of a learned selector, separating the coverage limitations of the heuristic pool from the training error of the selector.

3.3 Suboptimality Decomposition

We now decompose the suboptimality of a learned selector into two terms: 1) an *action-abstraction gap* reflecting the coverage of the heuristic pool, and 2) a *finite-sample RL training error* induced by learning under a limited training budget.

Action-Abstraction Gap. Every selector over $\mathcal{M}_{\mathcal{H}}$ induces a primitive policy over $\mathcal{M}$ via (4), but the converse fails once $\mathcal{H}$ leaves high-value actions uncovered, in which case the optimal primitive policy falls outside the representable class. We measure this limitation by an *action-abstraction gap*

$$\delta_{\mathcal{M}}(\mathcal{H}) := J^*_{\mathcal{M}} - J^*_{\mathcal{M}_{\mathcal{H}}} \geq 0. \tag{5}$$

This gap captures the performance ceiling imposed by pool $\mathcal{H}$, which even a perfect selector cannot remove if $\mathcal{H}$ never proposes an optimal primitive action in some visited state. To localize where and to what extent a pool $\mathcal{H}$ falls short, we define a per-step, per-state coverage residual

$$\ell_t^{\mathcal{H}}(s) := V_t^*(s) - \max_{a \in \mathcal{A}_t^{\mathcal{H}}(s)} Q_t^*(s, a) \geq 0. \tag{6}$$

It measures the one-step value lost by restricting the available primitive actions to the pool's proposals, which vanishes if and only if $\mathcal{A}_t^{\mathcal{H}}(s)$ covers an optimal primitive action. To connect local residuals to the overall gap, we introduce a coverage-greedy policy $\bar{\pi}_t^{\mathcal{H}}(s) \in \arg\max_{a \in \mathcal{A}_t^{\mathcal{H}}(s)} Q_t^*(s, a)$, which greedily selects, over $\mathcal{A}_t^{\mathcal{H}}(s)$, the action with the highest $Q_t^*(s, a)$ under $\mathcal{M}$. This policy, which is realizable by the pool, provides an upper bound on the action-abstraction gap.

Proposition 1 (Coverage bound)**.** *For any nonempty heuristic pool $\mathcal{H}$,*

$$0 \leq \delta_{\mathcal{M}}(\mathcal{H}) \leq \mathbb{E}_{P, \bar{\pi}^{\mathcal{H}}}\left[\sum_{t=1}^{T} \ell_t^{\mathcal{H}}(s_t)\right]. \tag{7}$$

The proof is provided in Appendix A.2.

Proposition 1 suggests an important direction for heuristic evolution: even a heuristic with modest standalone performance can still be valuable if it supplies complementary high-value actions in the states where the existing pool falls short. Meanwhile, the residual also provides a theoretical target for coverage diagnosis, with estimated action values and sampled selector trajectories serving as practical surrogates even if V_t^* and Q_t^* are not known exactly (as detailed in Section 4.2).

Finite-Sample RL Training Error. In addition to the pool coverage, a learned selector $\hat{\mu}$ under a finite training budget cannot achieve the optimal return $J^*_{\mathcal{M}_\mathcal{H}}$ exactly. We leverage the following proposition to capture this training error in terms of the heuristic pool size and the training budget.

Proposition 2 (Training error)**.** *Consider a fixed heuristic pool $\mathcal{H}$ inducing a tabular MDP with S states, K actions, per-step rewards normalized to $[0, 1]$, and finite horizon T. For any $\eta \in (0, 1)$ and sufficiently large episode budget N_{ep}, there exists a tabular learning algorithm that uses at most N_{ep} episodes and returns a selector $\hat{\mu}$ satisfying, with probability at least $1 - \eta$,*

$$J^*_{\mathcal{M}_\mathcal{H}} - J_{\mathcal{M}_\mathcal{H}}(\hat{\mu}) \leq c\sqrt{\frac{SKT^3}{N_{\text{ep}}}}, \tag{8}$$

where the coefficient c absorbs the confidence dependence and logarithmic factors.

This proposition follows from Ménard et al. (2021), and a complementary worst-case lower bound is given by Domingues et al. (2021) (Appendix A.3). Notably, given S, T, and N_{ep}, the bound's dependence on K provides a rationale to keep the pool compact under a fixed training budget.

Overall, combining the action-abstraction gap (5) with the finite-sample RL training error (8) under the preceding assumptions yields the following suboptimality decomposition

$$J^*_\mathcal{M} - J_{\mathcal{M}_\mathcal{H}}(\hat{\mu}) = \underbrace{J^*_\mathcal{M} - J^*_{\mathcal{M}_\mathcal{H}}}_{\text{heuristic ceiling}} + \underbrace{J^*_{\mathcal{M}_\mathcal{H}} - J_{\mathcal{M}_\mathcal{H}}(\hat{\mu})}_{\text{training error}} \leq \delta_\mathcal{M}(\mathcal{H}) + c\sqrt{\frac{SKT^3}{N_{\text{ep}}}}. \tag{9}$$

This decomposition reveals a fundamental trade-off: expanding the pool can possibly improve action coverage and reduce the abstraction gap, but it enlarges K and hence the training error under a fixed training budget. Therefore, our aim is to raise the heuristic ceiling through better coverage, while controlling the training error under limited training budgets to preserve the learning efficiency of heuristic selection. This is exactly what CoEvoRL is designed to fulfill.

The two propositions further indicate how to optimize toward this aim. Proposition 1 suggests that selector trajectories can provide states for coverage diagnosis that guide the LLM toward specific weaknesses of the current pool, and Proposition 2 shows that every newly added heuristic induces a statistical cost through increased training error. Therefore, CoEvoRL combines this bidirectional dependence in a closed loop: it alternates coverage-guided heuristic evolution with RL-based selector training, while selectively admitting candidate heuristics by evaluating their marginal coverage gains against the additional learning cost of pool expansion, which we describe in the next section.

4 CoEvoRL

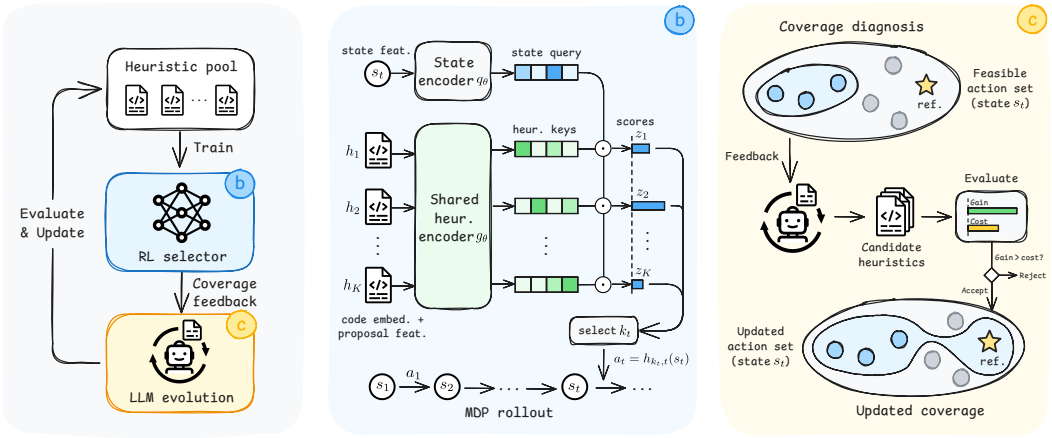
CoEvoRL addresses both terms of (9) by co-evolving LLM-generated heuristics and an RL-based selector in a closed loop (Fig. 1(a)). The selector adapts to the evolving pool through a variable-pool scorer (Section 4.1), and its coverage diagnosis feeds back into heuristic evolution, with candidates evaluated by their marginal coverage gains relative to pool-size cost (Section 4.2).

4.1 Variable-Pool Selection

Variable-pool selector. In contrast to existing learned selectors that read out a fixed heuristic pool through a K-way output head (Choong et al., 2018; Kallestad et al., 2023), CoEvoRL requires the selector to adaptively operate on a variable pool that is repeatedly updated. Therefore, we parameterize the selector $\mu_\theta(k \mid s)$ using a content-based, pointer-style attention mechanism that treats each heuristic as one token, so the parameter θ never depends on specific heuristics or the pool size.

As shown in Fig. 1(b), at each decision step, a state encoder maps s_t to a query vector $q_\theta(s_t) \in \mathbb{R}^d$. Each heuristic h_k is represented by a token $x_k = [e(h_k);\ f(h_{k,t}(s_t))]$, which concatenates a frozen code-model embedding $e(h_k)$ of its source code, identifying its contextual information, and a feature vector $f(h_{k,t}(s_t))$ of its proposed action, which exposes its state-dependent behavior. The heuristic tokens are projected into K keys through a shared heuristic encoder g_θ, yielding the selection policy through pointer-style attention scores

$$z_k(s_t) = \frac{q_\theta(s_t)^\top g_\theta(x_k)}{\sqrt{d}}, \quad \mu_t(k \mid s_t) = \text{softmax}_k(z_k(s_t)), \tag{10}$$



(a) CoEvoRL framework (b) Variable-pool selector (c) Coverage-guided evolution

Figure 1: Overview of CoEvoRL. (a) CoEvoRL alternates between RL-based selector training and LLM-driven heuristic evolution in a closed loop. (b) The selector leverages a variable-pool scorer to select heuristics from an evolving pool. (c) The heuristic evolution is guided by coverage diagnosis, and evaluated by marginal coverage gains against pool-size cost.

where d is the dimension of the query and keys. Since the heuristic encoder g_θ is shared across tokens, updating the heuristic pool only adds or drops a token, while the selector architecture and parameters remain unchanged, and the policy is also invariant to the ordering of the pool. Moreover, a new heuristic that is semantically and behaviorally close to an existing one will be embedded into a similar token and therefore receive a similar selection probability, which stabilizes training over an evolving pool and enables the selector to adapt smoothly rather than relearning from scratch.

Off-pool critic. Alongside the selector $\mu_\theta(k \mid s)$, we train an off-pool critic $Q_\phi(s, a)$ that estimates the values of all feasible primitive actions $a \in \mathcal{A}_t(s)$, including those not currently covered by the heuristic pool. This critic is used to approximate the coverage residual (6) by comparing the highest estimated action-value over the full primitive action set with that over the current pool's proposals. To obtain learning signals beyond the pool coverage, we sample a small fraction of training steps uniformly from the feasible primitive actions, and the critic updates bootstrap over the full primitive action set. The resulting critic provides a surrogate for the coverage residual, guiding heuristic evolution toward missing high-value actions.

4.2 Coverage-Guided Evolution

Coverage diagnosis. Building on the action-value estimates, we diagnose where the current pool lacks high-value proposals and use this diagnosis to guide heuristic evolution. After each RL training block, we freeze the pool and critic, and collect states by rolling out the selector to approximate the expectation of Proposition 1. Specifically, we record two types of diagnostic signals at each visited state s_t. The first is the approximated *coverage residual*

$$\hat{\ell}_t^{\mathcal{H}}(s_t) = \max_{a \in \mathcal{A}_t(s_t)} Q_\phi(s_t, a) - \max_{k \in [K]} Q_\phi(s_t, h_{k,t}(s_t)). \tag{11}$$

A large residual means that the critic has identified a promising primitive action beyond the current pool's proposals, suggesting a potential opportunity for complementary heuristic design. The second is a *reference miss*, a binary indicator identifying whether the current pool covers the reference optimal action taken at s_t (if available), which is precomputed offline on the diagnostic instances. The two signals serve complementary roles: the former is quantitative but approximate, as it estimates a concrete magnitude of the residual yet inherits approximation bias from the critic $Q_\phi(s, a)$, while the latter is exact but only qualitative, since it precisely indicates whether there exists a coverage gap but cannot quantify its specific magnitude. Together, they characterize where coverage gaps arise in the current heuristic pool and how severe they are. From the collected trajectories, we select a representative subset of visited states based on their residual severity, instance coverage, and diversity of feature contrasts, which forms a compact feedback to prompt heuristic evolution.

Heuristic evolution. An LLM is prompted with the aforementioned diagnosis, together with summaries of current heuristic programs and their usage statistics. The LLM is prompted to design a novel heuristic, typically represented as a function following a predefined interface, to address the diagnosed coverage gaps and complement the current pool. Each generated candidate undergoes compilation and execution checks, and only valid ones are kept for further evaluation.

Evaluation and admission. Different from existing LLM-based heuristic evolution methods that rank candidates by their standalone solution quality, we evaluate each candidate heuristic by comparing its marginal contribution to the pool coverage against the statistical learning cost of enlarging the selection space. Specifically, we score a candidate heuristic h by

$$F(h) = \frac{\sum_{(t,\, s_t)\in\mathcal{D}} \hat{\ell}_t^{\mathcal{H}}(s_t) - \sum_{(t,\, s_t)\in\mathcal{D}} \hat{\ell}_t^{\mathcal{H}\cup\{h\}}(s_t)}{\sum_{(t,\, s_t)\in\mathcal{D}} \hat{\ell}_t^{\mathcal{H}}(s_t)} - \beta \frac{\sqrt{K+1} - \sqrt{K}}{\sqrt{K_{\min}+1} - \sqrt{K_{\min}}}, \tag{12}$$

where $\mathcal{D}$ is an evaluation state set (obtained similarly to the diagnostic states), and K denotes the size of the current heuristic pool $\mathcal{H}$. The first term measures the improvement in coverage residual recovered by adding the new heuristic h, rewarding complementary high-value proposals rather than standalone solution quality. The second term captures the marginal increment in the $\sqrt{K}$ training-error bound from Proposition 2, normalized at a prescribed minimum pool size $K_{\min}$. This criterion translates the suboptimality decomposition (9) into a practical admission rule: only candidate heuristics with positive scores, i.e., whose coverage gains outweigh the additional cost of pool expansion, are accepted to the heuristic pool for further co-evolution.

Pruning. Moreover, to make the pool reversible and leave room for promising heuristics in subsequent evolution, we re-examine the heuristic pool after each candidate admission and remove redundant heuristics. Specifically, a heuristic is removed when its proposals frequently overlap with those of other pool members and it carries no standalone credit in solution quality, which is measured through a leave-one-out check. A minimum of $K_{\min}$ heuristics is always retained to preserve the pool's basic capability and diversity. The updated pool is then frozen for the next RL training block, allowing the selector to adapt to new proposals for further improvement.

5 Experiments

5.1 Experimental Setup

Problems. We consider four CO problems: job-shop scheduling problem (JSSP), traveling salesman problem (TSP), multidimensional knapsack problem (MKP), and online bin packing problem (OBPP), each at three sizes. These problems span diverse optimization structures covering routing, scheduling, and offline and online packing, which require different sequential decisions. Details of the problem definitions and instance generation are provided in Appendix B.1.

Baselines. We compare CoEvoRL with three categories of baselines. The first comprises heuristics without learned selection: *Best Heur.*, the best-performing of the heuristics we collect from the literature (Table 6), which also form the initial pool, and *EoH* (Liu et al., 2024), which evolves heuristics by their standalone performance. The second comprises heuristic selection that does not co-evolve, including learning a selector based on a fixed heuristic pool (*Fixed Pool*), which reflects the paradigm of most existing works, and *HeurAgenix* (Yang et al., 2025), a two-stage framework that first evolves a pool offline and then trains a selector over the resulting pool. For fair comparison, all methods start from the same initial pool, LLM-based methods use the same backbone and call budget, and all selection-based methods share the same selector architecture and training budget.

The third category comprises representative construction policies (*Construct.*): L2D (Zhang et al., 2020) for JSSP, POMO (Kwon et al., 2020) for TSP, RL-to-Rank (Woo et al., 2022) for MKP, and ORL (Balaji et al., 2019) for OBPP, respectively. Each is trained under two budgets: *Full* follows the original papers and reflects the performance attainable when extensive training is affordable, and *Limit* matches the training budget (i.e., training episodes) of CoEvoRL to isolate the effect of the action space under an identical budget. Notably, CoEvoRL is trained entirely on CPUs owing to its lightweight selector, while construction baselines use GPUs in both budget settings. Implementation details and experimental protocols are given in Appendix B.2–B.4.

Table 1: Reference optimality gap on JSSP, TSP, MKP, and OBPP (lower is better).

Problem	JSSP			TSP			MKP			OBPP		
Size	10×10	15×15	20×20	20	50	100	100	250	500	100	250	500
Construct. (Full)	15.87%	20.79%	23.68%	0.08%	0.35%	0.77%	4.05%	8.52%	9.89%	6.56%	5.02%	4.21%
Construct. (Limit)	22.12%	27.08%	33.52%	18.32%	40.12%	58.58%	4.45%	10.34%	9.97%	7.14%	5.88%	5.26%
Best Heur.	22.13%	26.92%	28.59%	17.62%	22.75%	24.75%	2.71%	1.98%	1.51%	6.72%	5.46%	4.90%
EoH	21.60%	25.95%	28.28%	16.50%	21.98%	21.90%	2.01%	1.66%	1.39%	6.67%	5.44%	4.90%
Fixed Pool	15.77%	22.16%	24.19%	10.66%	16.72%	19.93%	1.51%	0.74%	0.48%	6.51%	4.92%	4.30%
HeurAgenix	15.11%	21.27%	23.81%	7.62%	12.92%	16.66%	1.40%	0.82%	0.57%	6.86%	19.97%	13.25%
CoEvoRL (Ours)	**14.09%**	**19.84%**	**22.72%**	**7.12%**	**11.21%**	**13.94%**	**1.32%**	**0.55%**	**0.38%**	**5.09%**	**3.41%**	**2.62%**

* All gaps are computed relative to the best solution obtained by the reference solvers with a time limit of 3,600 seconds per instance.

Table 2: Ablation study on key co-evolution components.

Problem	JSSP			TSP			MKP			OBPP		
Size	10×10	15×15	20×20	20	50	100	100	250	500	100	250	500
w/o diagnosis	15.36%	20.41%	23.23%	7.39%	14.49%	16.20%	1.48%	0.77%	0.51%	5.22%	4.03%	3.37%
w/o residual	15.04%	20.20%	**22.37%**	8.46%	12.98%	15.48%	1.42%	0.76%	0.50%	6.51%	4.92%	4.65%
w/o penalty	14.95%	20.34%	23.05%	8.13%	11.80%	14.96%	1.51%	0.77%	0.49%	5.66%	4.03%	3.30%
CoEvoRL	**14.09%**	**19.84%**	22.72%	**7.12%**	**11.21%**	**13.94%**	**1.32%**	**0.55%**	**0.38%**	**5.09%**	**3.41%**	**2.62%**

5.2 Main Results

Co-evolution breaks heuristic ceiling and improves overall performance. Table 1 reports the optimality gap on four benchmark problems. Full results and statistical significance are supplemented in Appendix C.1–C.2. CoEvoRL achieves the best solution quality over all heuristic-based methods in all settings. Specifically, EoH enhances given heuristics but remains generally weaker than learned selection, while Fixed Pool improves over single heuristics through adaptive selection, but remains bounded by the coverage of its fixed pool. HeurAgenix evolves heuristics offline without selector feedback, yielding heuristics with similar decision patterns that offer limited gains in coverage and can even degrade final performance. In contrast, CoEvoRL reduces the optimality gap by 4.6%–39.1% relative to the best competing heuristic selector in each setting, supporting the benefit of closed-loop co-evolution, where the evolved heuristic pool raises the performance ceiling through coverage feedback, and the selector realizes it through RL-based training and adaptation.

Competing with construction policies with a compact action space. CoEvoRL consistently outperforms Construct. (Limit) in every setting, demonstrating the effectiveness of heuristic selection with an evolving pool under constrained training budgets. This advantage becomes more pronounced as the problem size grows, with the advantage over Construct. (Limit) widening from 8.03% to 10.80% on JSSP and from 11.20% to 44.64% on TSP. This growing advantage can be attributed to the compact selection space of CoEvoRL, whose size is determined by the heuristic pool, allowing it to achieve better scalability to larger problems. Notably, CoEvoRL even outperforms Construct. (Full) with substantially larger training budgets on JSSP, MKP, and OBPP, suggesting that retaining a compact heuristic selection space need not entail lower solution quality compared with learning construction policies, provided that the pool itself can improve in a reasonable manner.

5.3 Ablation Study on Co-Evolution Components

We isolate the contribution of key co-evolution components with three ablated variants: 1) *w/o diagnosis* removes the coverage diagnosis from the prompts to LLM; 2) *w/o residual* replaces the residual gain in (12) with the fraction of instances on which a candidate outperforms the best heuristic in the current pool, thereby substituting instance-level complementarity for state-level coverage gain; and 3) *w/o penalty* drops the training-error term in (12) and instead sets a maximum pool size limit to prevent unrestricted expansion. The ablation results are shown in Table 2.

Removing any component leads to performance degradation across problems, while each component exhibits a distinct failure mode. Without coverage diagnosis, the degradation is most pronounced on problems with large primitive action spaces, where the actions missed by the pool are hard for the LLM to infer from heuristic programs alone. Replacing the residual gain with instance-level scoring

Table 3: Wall-clock runtime (in minutes) and LLM usage on JSSP 10×10.

	T_{Total}	T_{LLM}	# Calls
Construct. (Full)	106.45	–	–
Construct. (Limit)	38.34	–	–
EoH	3.98	3.01	50.0
Fixed Pool	18.97	–	–
HeurAgenix	26.01	3.67	50.2
CoEvoRL (Ours)	24.20	3.42	49.4

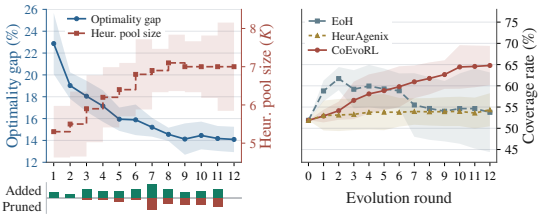


Figure 2: CoEvoRL dynamics: optimality gap and pool size (left) and optimal coverage rate (right) on JSSP 10×10.

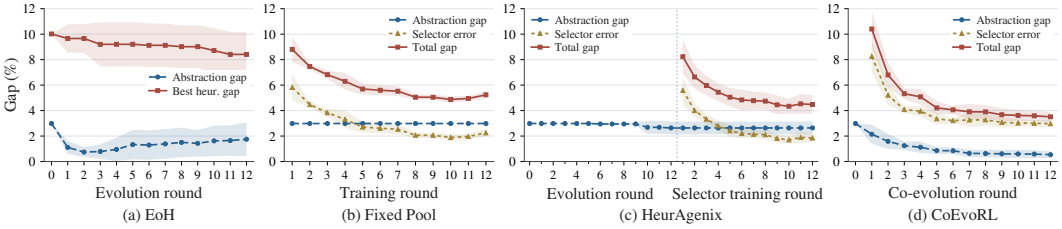


Figure 3: Exact suboptimality decomposition on small-sized TSP-10.

even reduces CoEvoRL to nearly the level of Fixed Pool on OBPP, since a heuristic that proposes high-value actions in only a small subset of states rarely outperforms the best heuristic over an entire instance and thus complementary candidates are frequently rejected. This confirms that state-level residual gain, as motivated by Proposition 1, is essential for admitting complementary heuristics. Finally, removing the penalty admits heuristics with only marginal coverage gains, and the enlarged pool increases the training error of the selector under a fixed budget, consistent with Proposition 2.

5.4 Computational Cost and Co-Evolution Analysis

Table 3 and Figure 2 report the computational cost and co-evolution dynamics on JSSP 10×10. Using a lightweight selector, CoEvoRL runs on CPUs and still takes far less wall-clock time than the GPU-trained construction policies. Its overhead over Fixed Pool comes mainly from LLM queries, with time cost and call counts comparable to EoH and HeurAgenix, suggesting that the gains of CoEvoRL stem from more effective evolution guidance rather than larger LLM budgets. The co-evolution dynamics show an overall decrease in the optimality gap as the heuristic pool first expands and then stabilizes, with admissions and pruning progressively refining its composition while keeping the selection space compact. Meanwhile, CoEvoRL steadily improves its optimal-action coverage, whereas EoH and HeurAgenix show either declining coverage after early gains or persistently low coverage. Together, these trends suggest that co-evolution improves the pool's coverage to propose high-quality actions while enabling the selector to exploit the evolving pool.

Figure 3 further assesses the dynamics of the two terms in suboptimality decomposition (9) on small-sized TSP-10, where $J^*_{\mathcal{M}_{\mathcal{H}}}$ is computed exactly by dynamic programming. Compared with the baselines, CoEvoRL substantially raises the heuristic ceiling through coverage diagnosis while keeping the training error manageable by controlling K, therefore achieving the lowest total gap.

6 Conclusions

We proposed CoEvoRL, a co-evolution framework that collaboratively evolves a pool of heuristics using LLMs and trains a selector via RL for heuristic selection. The restricted MDP formulation separates the action-abstraction gap induced by the heuristic pool from the finite-sample training error of the selector, motivating coverage-guided heuristic evolution, the variable-pool selector, and selective pool updates. Across four CO problems, CoEvoRL consistently outperforms all heuristic selection baselines while achieving performance comparable to far more costly construction policies, providing a promising path toward combining the learning efficiency of heuristic selection with solution quality approaching end-to-end construction. Future work includes improving training stability under evolving heuristic pools and test-time augmentation for further improvement.

AI USE STATEMENT

LLMs serve as key methodological components for heuristic evolution in our framework. We also used LLMs to assist code writing, figure drafting, and text polishing. We have not used generative AI tools for other tasks with required disclosure. The core ideas and the manuscript were conceived, prepared, verified, and finalized by the authors. We take responsibility for the final content of this work, including text, claims, or artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

We have made every effort to ensure the reproducibility of our results. The problems in our experiments follow standard definitions, with instance generation details specified in Appendix B.1. The implementation details of CoEvoRL and baselines, training and testing configurations, and hyperparameter settings are described in Appendix B.2–B.4. Our code is available in the supplementary material and at https://anonymous.4open.science/r/coevorl-923A.

REFERENCES

Bharathan Balaji, Jordan Bell-Masterson, Enes Bilgin, Andreas Damianou, Pablo Moreno Garcia, Arpit Jain, Runfei Luo, Alvaro Maggiar, Balakrishnan Narayanaswamy, and Chun Ye. ORL: Reinforcement learning benchmarks for online stochastic optimization problems. *arXiv preprint arXiv:1911.10641*, 2019.

Irwan Bello, Hieu Pham, Quoc V Le, Mohammad Norouzi, and Samy Bengio. Neural combinatorial optimization with reinforcement learning. *arXiv preprint arXiv:1611.09940*, 2016.

Yoshua Bengio, Andrea Lodi, and Antoine Prouvost. Machine learning for combinatorial optimization: A methodological tour d'horizon. *European Journal of Operational Research*, 290(2): 405–421, 2021.

Edmund K Burke, Michel Gendreau, Matthew Hyde, Graham Kendall, Gabriela Ochoa, Ender Özcan, and Rong Qu. Hyper-heuristics: A survey of the state of the art. *Journal of the Operational Research Society*, 64(12):1695–1724, 2013.

Quentin Cappart, Didier Chételat, Elias B Khalil, Andrea Lodi, Christopher Morris, and Petar Veličković. Combinatorial optimization and reasoning with graph neural networks. *Journal of Machine Learning Research*, 24(130):1–61, 2023.

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

Hongzheng Chen, Yingheng Wang, Yaohui Cai, Hins Hu, Jiajie Li, Shirley Huang, Chenhui Deng, Rongjian Liang, Shufeng Kong, Haoxing Ren, Samitha Samaranayake, Carla P. Gomes, and Zhiru Zhang. HeuriGym: An agentic benchmark for LLM-crafted heuristics in combinatorial optimization. In *International Conference on Learning Representations*, 2026.

Shin Siang Choong, Li-Pei Wong, and Chee Peng Lim. Automatic design of hyper-heuristic based on reinforcement learning. *Information Sciences*, 436:89–107, 2018.

Paul C Chu and John E Beasley. A genetic algorithm for the multidimensional knapsack problem. *Journal of Heuristics*, 4(1):63–86, 1998.

Kwok Tung Chung, Carman KM Lee, and Yung Po Tsang. Neural combinatorial optimization with reinforcement learning in industrial engineering: A survey. *Artificial Intelligence Review*, 58(5): 130, 2025.

Omar Darwiche Domingues, Pierre Ménard, Emilie Kaufmann, and Michal Valko. Episodic reinforcement learning in finite MDPs: Minimax lower bounds revisited. In *International Conference on Algorithmic Learning Theory*, pp. 578–598, 2021.

John H Drake, Ahmed Kheiri, Ender Özcan, and Edmund K Burke. Recent advances in selection hyper-heuristics. *European Journal of Operational Research*, 285(2):405–428, 2020.

Chaitanya K Joshi, Quentin Cappart, Louis-Martin Rousseau, and Thomas Laurent. Learning the travelling salesperson problem requires rethinking generalization. *Constraints*, 27(1):70–98, 2022.

Jakob Kallestad, Ramin Hasibi, Ahmad Hemmati, and Kenneth Sörensen. A general deep reinforcement learning hyperheuristic framework for solving combinatorial optimization problems. *European Journal of Operational Research*, 309(1):446–468, 2023.

Minsu Kim, Junyoung Park, and Jinkyoo Park. Sym-NCO: Leveraging symmetricity for neural combinatorial optimization. *Advances in Neural Information Processing Systems*, 35:1936–1949, 2022.

Wouter Kool, Herke Van Hoof, and Max Welling. Attention, learn to solve routing problems! In *International Conference on Learning Representations*, 2019.

Yeong-Dae Kwon, Jinho Choo, Byoungjip Kim, Iljoo Yoon, Youngjune Gwon, and Seungjai Min. POMO: Policy optimization with multiple optima for reinforcement learning. *Advances in Neural Information Processing Systems*, 33:21188–21198, 2020.

Cuixia Li, Xiang Wei, Jing Wang, Shuozhe Wang, and Shuyan Zhang. A review of reinforcement learning based hyper-heuristics. *PeerJ Computer Science*, 10:e2141, 2024.

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 *International Conference on Machine Learning*, 2024.

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

Nina Mazyavkina, Sergey Sviridov, Sergei Ivanov, and Evgeny Burnaev. Reinforcement learning for combinatorial optimization: A survey. *Computers & Operations Research*, 134:105400, 2021.

Pierre Ménard, Omar Darwiche Domingues, Anders Jonsson, Emilie Kaufmann, Edouard Leurent, and Michal Valko. Fast active learning for pure exploration in reinforcement learning. In *International Conference on Machine Learning*, pp. 7599–7608. PMLR, 2021.

Mohammadreza Nazari, Afshin Oroojlooy, Lawrence Snyder, and Martin Takác. Reinforcement learning for solving the vehicle routing problem. *Advances in Neural Information Processing Systems*, 31, 2018.

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

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

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

Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. Pointer networks. *Advances in Neural Information Processing Systems*, 28, 2015.

Yue Wang, Hung Le, Akhilesh Gotmare, Nghi Bui, Junnan Li, and Steven Hoi. CodeT5+: Open code large language models for code understanding and generation. In *The Conference on Empirical Methods in Natural Language Processing*, pp. 1069–1088, 2023.

Honguk Woo, Hyunsung Lee, and Sangwoo Cho. An efficient combinatorial optimization model using learning-to-rank distillation. In *AAAI Conference on Artificial Intelligence*, volume 36, pp. 8666–8674, 2022.

Xiangjie Xiao, Cong Zhang, Wen Song, and Zhiguang Cao. RESCHED: Rethinking flexible job shop scheduling from a transformer-based architecture with simplified states. In *International Conference on Learning Representations*, 2026.

Xianliang Yang, Ling Zhang, Haolong Qian, Lei Song, and Jiang Bian. HeurAgenix: Leveraging LLMs for solving complex combinatorial optimization challenges. *arXiv preprint arXiv:2506.15196*, 2025.

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

Hang Yi, Ziwei Huang, Yining Ma, and Zhiguang Cao. RADAR: Learning to route with asymmetry-aware distance representations. In *International Conference on Learning Representations*, 2026.

Cong Zhang, Wen Song, Zhiguang Cao, Jie Zhang, Puay Siew Tan, and Xu Chi. Learning to dispatch for job shop scheduling via deep reinforcement learning. *Advances in Neural Information Processing Systems*, 33:1621–1632, 2020.

Cong Zhang, Yaoxin Wu, Yining Ma, Wen Song, Zhang Le, Zhiguang Cao, and Jie Zhang. A review on learning to solve combinatorial optimisation problems in manufacturing. *IET Collaborative Intelligent Manufacturing*, 5(1):e12072, 2023.

Fuqing Zhao, Yuebao Liu, Ningning Zhu, Tianpeng Xu, and Jonrinaldi. A selection hyper-heuristic algorithm with Q-learning mechanism. *Applied Soft Computing*, 147:110815, 2023.

A Proofs and Remarks on the Suboptimality Analysis

A.1 Equivalence of Selector and Primitive Policy

Let $\Pi_{\mathcal{H}}$ be the class of primitive policies supported on $\mathcal{A}_t^{\mathcal{H}}(s)$ at every decision state, i.e., satisfying

$$\pi_t(a \mid s) = 0, \quad \forall a \notin \mathcal{A}_t^{\mathcal{H}}(s). \tag{13}$$

From selector to primitive policy. For any selector μ, define $\pi^{\mu,\mathcal{H}}$ by (4). Nonnegativity and

$$\sum_{a \in \mathcal{A}_t(s)} \pi_t^{\mu,\mathcal{H}}(a \mid s) = \sum_{k=1}^{K} \mu_t(k \mid s) = 1 \tag{14}$$

show that $\pi^{\mu,\mathcal{H}}$ is a valid member of $\Pi_{\mathcal{H}}$. Therefore, conditional on a given state, the executed primitive action has precisely this distribution under either description; conditional on that action, the next-state distribution and reward agree by (3). Induction from the common initial distribution therefore gives equality of the joint laws of states, primitive actions, and rewards. In particular,

$$J_{\mathcal{M}_{\mathcal{H}}}(\mu) = J_{\mathcal{M}}(\pi^{\mu,\mathcal{H}}). \tag{15}$$

From supported primitive policy to selector. Fix $\pi \in \Pi_{\mathcal{H}}$. For each covered action, let

$$I_t(s,a) = \{k \in [K] : h_{k,t}(s) = a\}, \qquad n_t(s,a) = |I_t(s,a)|. \tag{16}$$

Here, $I_t(s,a)$ collects the indices of all heuristics that propose primitive action a at state s step t, and $n_t(s,a)$ counts these heuristics. Define

$$\mu_t(k \mid s) = \frac{\pi_t(h_{k,t}(s) \mid s)}{n_t(s, h_{k,t}(s))}. \tag{17}$$

The denominator is positive given the definition of $\Pi_{\mathcal{H}}$. For every covered action,

$$\sum_{k \in I_t(s,a)} \mu_t(k \mid s) = \pi_t(a \mid s). \tag{18}$$

For uncovered actions, both sides of (4) are 0. Moreover, grouping indices by their proposals gives $\sum_k \mu_t(k \mid s) = \sum_{a \in \mathcal{A}_t^{\mathcal{H}}(s)} \pi_t(a \mid s) = 1$. Thus the induced primitive policy is exactly π.

Therefore, the two constructions establish that optimizing heuristic selection is exactly equivalent to optimizing the primitive objective over primitive policies supported by the heuristic pool, and

$$J^*_{\mathcal{M}_{\mathcal{H}}} = \max_{\pi \in \Pi_{\mathcal{H}}} J_{\mathcal{M}}(\pi) \le J^*_{\mathcal{M}}. \tag{19}$$

A.2 Proof of Proposition 1

The optimal state- and action-values satisfy the Bellman equations

$$V_t^*(s) = \max_{a \in \mathcal{A}_t(s)} Q_t^*(s,a), \quad V_{T+1}^*(s) \equiv 0, \tag{20}$$

$$Q_t^*(s,a) = r_t(s,a) + \mathbb{E}_{s' \sim P_t(\cdot \mid s,a)} V_{t+1}^*(s'). \tag{21}$$

For fixed initial state s_1, combining $J^*_{\mathcal{M}} = V_1^*(s_1)$ with the tower property yields

$$\begin{aligned} &\mathbb{E}_{P,\pi}\left[\sum_{t=1}^{T} \big(V_t^*(s_t) - Q_t^*(s_t,a_t)\big)\right] \\ &= \mathbb{E}_{P,\pi}\left[\sum_{t=1}^{T} \big(V_t^*(s_t) - r_t(s_t,a_t) - V_{t+1}^*(s_{t+1})\big)\right] \\ &= V_1^*(s_1) - J_{\mathcal{M}}(\pi) = J^*_{\mathcal{M}} - J_{\mathcal{M}}(\pi). \end{aligned} \tag{22}$$

Proof of Proposition 1. The deterministic policy $\bar{\pi}^{\mathcal{H}}$ selects a maximizer of Q_t^* among the covered actions, with any fixed tie-breaking rule. The deterministic policy $\bar{\pi}^{\mathcal{H}}$ belongs to $\Pi_{\mathcal{H}}$, and

$$V_t^*(s) - Q_t^*(s, \bar{\pi}_t^{\mathcal{H}}(s)) = \ell_t^{\mathcal{H}}(s). \tag{23}$$

Substituting $\bar{\pi}^{\mathcal{H}}$ into (22) gives the exact identity

$$J_{\mathcal{M}}^* - J_{\mathcal{M}}(\bar{\pi}^{\mathcal{H}}) = \mathbb{E}_{P,\bar{\pi}^{\mathcal{H}}}\left[\sum_{t=1}^{T} \ell_t^{\mathcal{H}}(s_t)\right]. \tag{24}$$

Equation (19) implies $J_{\mathcal{M}}(\bar{\pi}^{\mathcal{H}}) \le J_{\mathcal{M}_{\mathcal{H}}}^* \le J_{\mathcal{M}}^*$, which proves the claimed bound. □

A.3 Remarks on Proposition 2

Proposition 2 considers a heuristic pool $\mathcal{H}$ that induces a tabular MDP with S states, K actions, finite horizon T, and deterministic per-step rewards normalized to $[0, 1]$. The upper bound of the training error follows from Theorem 2 of Ménard et al. (2021). We give an explicit sufficient budget condition to retain the accuracy restriction in this theorem. For an episode budget $N_{\mathrm{ep}} \ge 2$ and a universal constant $C \ge 1$ large enough to absorb the numerical constants in that theorem, we define

$$\begin{aligned} b &= 1 + \log(6SKT/\eta), \\ L &= b\left[1 + \log\big((S+b)SKT^3(N_{\mathrm{ep}}+1)\big)\right]^2, \\ \varepsilon &= \sqrt{\frac{CSKT^3L}{N_{\mathrm{ep}}-1}}. \end{aligned} \tag{25}$$

Here, b and L collect the logarithmic factors, and ε is the target accuracy for the given budget, where a sufficient condition for the sufficiently large budget is $\varepsilon \le S^{-2}$, i.e., $N_{\mathrm{ep}} \ge CS^5KT^3L + 1$. Under this condition, applying the cited theorem at the above accuracy ε and confidence parameter $\eta/2$ provides an ε-optimal recommendation with probability at least $1 - \eta/2$. Its stopping time τ also satisfies, with probability at least $1 - \eta/2$,

$$\tau \le 1 + \frac{CSKT^3 b}{\varepsilon^2}\left[1 + \log\left(\frac{(S+b)SKT^3}{\varepsilon}\right)\right]^2. \tag{26}$$

Since $\varepsilon \ge 1/(N_{\mathrm{ep}}+1)$, the logarithmic factor in (26) is bounded by the corresponding factor in L. The right-hand side is therefore at most $1 + CSKT^3L/\varepsilon^2 = N_{\mathrm{ep}}$.

Run this algorithm with a hard limit of N_{ep} episodes, returning its recommendation if it stops by that limit and an arbitrary feasible selector otherwise. The resulting procedure always respects the budget. A union bound over the recommendation and stopping-time guarantees shows that, with probability at least $1 - \eta$, its output satisfies

$$J_{\mathcal{M}_{\mathcal{H}}}^* - J_{\mathcal{M}_{\mathcal{H}}}(\hat{\mu}) \le \varepsilon \le \sqrt{2CL}\sqrt{\frac{SKT^3}{N_{\mathrm{ep}}}} = \widetilde{O}\left(\sqrt{\frac{SKT^3}{N_{\mathrm{ep}}}}\right). \tag{27}$$

Thus, the factor c in (8) abbreviates confidence and logarithmic dependence, made explicit here by $\sqrt{2CL}$. The budget condition is sufficient, rather than necessary.

On the other hand, the lower bound of Domingues et al. (2021) provides complementary worst-case context: under stated conditions, for every algorithm there exists a family of MDPs in the same $[0, 1]$-reward class on which $\Omega\big(SKT^3 \log(1/\eta)/\varepsilon^2\big)$ episodes are necessary to return an ε-optimal policy with probability $1 - \eta$. Hence, the polynomial dependence of (8) on S, K, and T is worst-case optimal up to logarithmic factors in that regime. Taken together, the matching upper and lower bounds indicate that the $\sqrt{K}$ dependence cannot be removed in the worst case, which motivates our design of selective pool updates by criterion (12) to control the size of the selection space.

B Experimental Settings

B.1 Problems and Data Generation

We evaluate CoEvoRL on four CO problems: JSSP, TSP, MKP, and OBPP. Table 4 summarizes the problem sizes, objectives, and primitive decisions. The same problem definitions and instances are used to evaluate all methods within each setting.

Table 4: Overview of the experimental problems.

Problem	Sizes*	Objective	Primitive decision
JSSP	10×10, 15×15, 20×20	Minimize makespan (↓)	Dispatch an operation
TSP	20, 50, 100	Minimize tour length (↓)	Visit an unvisited city
MKP	100, 250, 500	Maximize total profit (↑)	Select a feasible item
OBPP	100, 250, 500	Minimize bin count (↓)	Place the arriving item

* For JSSP, size denotes the number of jobs $\times$ the number of machines; for TSP, it denotes the number of cities; for MKP and OBPP, it denotes the number of items.

Job-Shop Scheduling Problem (JSSP). JSSP contains J jobs and M machines. Each job consists of M operations that follow a prescribed order, with each operation processed non-preemptively on a specified machine. Each machine can process at most one operation at a time. The objective is to minimize the makespan, i.e., the completion time of the last operation.

In its primitive MDP, the state records the partial schedule, the next unscheduled operation of each job, job and machine availability times, and remaining processing requirements. An action selects an unfinished job and dispatches its next operation. This operation is appended to the schedule of its specified machine, and starts when both the job and machine are available; the transition updates the schedule, availability times, and remaining workloads. The reward is $r_t = L(s_t) - L(s_{t+1})$, where $L(s)$ is the maximum of the current makespan, each job's availability time plus its remaining workload, and each machine's availability time plus its remaining workload. An episode ends after all operations have been scheduled. Since the terminal bound equals the completed makespan $C_{\max}$, the cumulative reward is $L(s_1) - C_{\max}$, aligning return maximization with makespan minimization.

For data generation, we follow the random-instance distribution of Taillard (1993), also adopted in Zhang et al. (2020). We consider $(J, M) \in \{(10, 10), (15, 15), (20, 20)\}$. Each job's machine sequence is sampled independently as a random permutation of the M machines. Operation processing times are sampled independently and uniformly from the integers $\{1, \ldots, 99\}$.

Traveling Salesman Problem (TSP). We consider symmetric Euclidean TSP, where each instance consists of N cities, with edge costs given by the Euclidean distances between their coordinates. The objective is to find a minimum-length closed tour that visits every city exactly once.

In its primitive MDP, the state contains the city coordinates, the current city, the visited-city set, and the accumulated tour length. Construction starts at city 0, and each action selects an unvisited city. The transition moves to the selected city, marks it as visited, and extends the partial tour. The reward is the negative length of the traversed edge. After the last city, the environment closes the tour by returning to city 0, with the return-edge cost included in the final reward. An episode therefore contains $N - 1$ decisions, and its cumulative reward equals the negative length of the complete tour.

For data generation, we follow the Euclidean TSP setting of Kool et al. (2019). We consider $N \in \{20, 50, 100\}$ and sample city coordinates independently and uniformly from the unit square $[0, 1]^2$. Edge costs are computed as Euclidean distances between the sampled coordinates.

Multidimensional Knapsack Problem (MKP). We consider the multidimensional 0–1 knapsack problem with N items and D resource constraints. Each item i has profit p_i and resource consumption w_{ir} in dimension r. The objective is

$$\max_{x \in \{0,1\}^N} \sum_{i=1}^{N} p_i x_i, \quad \text{s.t.} \sum_{i=1}^{N} w_{ir} x_i \le C_r, \quad r = 1, \ldots, D,$$

where C_r denotes the capacity of resource r.

In its primitive MDP, the state contains the item profits and resource consumptions, the selected items, and the remaining capacities. An action chooses an unselected item that meets all remaining resource capacities. The transition adds the item to the solution and subtracts its resource consumptions from the corresponding capacities, yielding a reward equal to its profit. The episode terminates when no additional item is feasible. The episode length depends on the selected items, and the cumulative reward equals the total profit of the constructed solution.

For data generation, we construct synthetic instances inspired by the Chu–Beasley benchmark family (Chu & Beasley, 1998), using correlated profits and a fixed capacity tightness. We consider $N \in \{100, 250, 500\}$ items and $D = 5$ resource dimensions. In our generator, resource consumptions are sampled independently and uniformly from $\{0, \dots, 1000\}$. Profits are computed as

$$p_i = \max\left\{1,\ \left\lfloor \frac{1}{D}\sum_{r=1}^{D} w_{ir} + 500u_i \right\rfloor\right\} \Big/ 1000, \quad u_i \sim \mathcal{U}[0,1),$$

where the u_i are independent. With a tightness parameter $\rho = 0.5$, capacities are set to

$$C_r = \max\left\{1,\ \left\lfloor \rho \sum_{i=1}^{N} w_{ir} \right\rfloor\right\}.$$

Online Bin Packing Problem (OBPP). OBPP consists of an online sequence of N items that must be packed into bins of capacity C. Each item must be assigned upon arrival, without access to future item sizes. The objective is to minimize the total number of bins used.

In its primitive MDP, the (observable) state contains the arrival index, the size of the current item, and the remaining capacities of the opened bins. An action places the current item into an existing bin with sufficient remaining capacity or opens a new bin. The transition updates the selected bin's remaining capacity and advances to the next arrival. The reward is -1 when a new bin is opened and 0 otherwise. The episode terminates after all N items have been placed, so the cumulative reward equals the negative number of bins used.

For data generation, we adopt the sampling scheme based on the Weibull-based bin-packing benchmarks (Castiñeiras et al., 2012). We consider $N \in \{100, 250, 500\}$ and bin capacity $C = 100$. In our experiments, item sizes are sampled independently from a Weibull distribution with shape parameter 3 and scale parameter 45, then rounded to the nearest integer and clipped to $[1, C]$. Equivalently, drawing z_i from a unit-scale Weibull distribution with shape parameter 3, we set

$$s_i = \min\left\{C,\ \max\left\{1,\ \mathrm{round}(45 z_i)\right\}\right\},$$

and items arrive in their generated order.

Reference optimal solutions. The solutions are computed offline and reused across methods. For all problems, we use the OR-Tools solver with a time limit of 3,600 seconds per instance. Note that, for OBPP, the optimal reference is the solution to the corresponding static (offline) BPP, where the entire item sequence is known in advance, whereas the evaluated methods make decisions without observing future items. The final solutions are used to evaluate the relative gap in our experiments.

B.2 CoEvoRL Implementation

Representations and networks. Each feasible primitive action is represented as a 10-dimensional feature vector. The state representation combines problem-specific global features with the element-wise mean, minimum, and maximum of the feasible-action features. Each heuristic is a deterministic Python function that proposes its highest-scoring feasible action. Its source code is encoded by the frozen CodeT5+ model (Wang et al., 2023), and the 256-dimensional code embedding is projected to 16 dimensions using a fixed Gaussian projection and ℓ_2-normalized, then concatenated with the proposed action's features to form a 26-dimensional token.

The selector maps states and heuristic tokens to 128-dimensional queries and keys using multilayer perceptrons (MLPs) and computes selection probabilities through scaled dot-product attention. The off-pool critic (Section 4.1) estimates a scalar action-value from the state and primitive action features through an MLP. A separate state-value network provides the baseline value for selector training. Both value networks have two hidden layers of width 128.

Training and heuristic evolution. The selector is trained by policy gradient with learned state-value baseline. The off-pool critic uses temporal-difference targets, bootstrapping over all feasible primitive actions, where training directly samples a feasible primitive action with probability 0.08. After each training block, two disjoint subsets of 12 training instances are rolled out with the current

Table 5: Hyperparameters of CoEvoRL.

Parameter	Value	Parameter	Value
RL selector / critic training		*LLM heuristic evolution*	
Hidden width / code dimension	128 / 16	Instances per diagnostic subset	12
Selector / value learning rate	3×10^{-4}	Candidates per evolution round	4
Off-pool critic learning rate	10^{-3}	Pool size (initial / min / max)	5 / 3 / 12
TD steps for off-pool critic	5	Maximum admissions per round	2
Entropy coefficient	0.01	Candidate agreement threshold	0.9
Primitive-action exploration rate	0.08	β in admission criterion	0.25

Table 6: Initial heuristic pools.

Problem	Initial heuristics
JSSP	Shortest processing time, most work remaining, earliest available job, longest processing time, and most operations remaining
TSP	Nearest node, farthest node, nearest to depot, minimum detour, and isolation-based selection
MKP	Highest profit, profit density, bottleneck efficiency, balanced resource consumption, and scarcity-aware selection
OBPP	Best fit, first fit, worst fit, tight-fit preference, and gap-aware placement

selector. One supplies diagnosis to the LLM, and the other evaluates candidate residual gains, where all 12 examples enter the prompt, and candidates are scored on at most 150 replay states.

Each evolution round requests four candidates, which undergo syntax and execution checks with at most one repair attempt each. Candidate admission follows (12). The pool starts with five rules, admits at most two candidates per round, and is kept between 3 and 12 members in the pool. Highly overlapping candidates are rejected, and redundant rules with non-positive leave-one-out credit are pruned while protecting three elite rules. Following pool changes, stored proposals are relabeled and the selector briefly adapts by transferring probability mass to new rules that propose the same primitive actions. The main hyperparameters are summarized in Table 5.

B.3 Baseline Implementations

All heuristic-based methods start from the same five rules for each problem, listed in Table 6. *Best Heur.* reports the initial heuristic with the best solution quality. *Fixed Pool* trains the same networks as CoEvoRL based on the same initial pool. *EoH* (Liu et al., 2024) ranks candidates by their standalone solution quality and retains a population of at most 12 heuristics (the same as CoEvoRL). *HeurAgenix* (Yang et al., 2025) first evolves a pool offline and then freezes it for selector training. The selector remains untrained during evolution, and all neural training occurs in the second stage.

For JSSP, we implement L2D (Zhang et al., 2020) following its graph architecture and PPO configuration. For TSP, we follow POMO (Kwon et al., 2020), using single-start greedy decoding to keep the inference computation comparable across methods. For MKP, we implement the RL teacher of RL-to-Rank (Woo et al., 2022), following its network architecture and deterministic greedy baseline. For OBPP, we adapt the masked PPO policy of ORL (Balaji et al., 2019). Unless otherwise specified, all implementations follow their original papers or official code.

B.4 Experimental Protocols

CoEvoRL runs for 12 rounds, each comprising 300 updates with 4 episodes per update, totaling 14,400 training episodes. Evolution follows the first 11 rounds, and the final round trains without further pool changes. *Fixed Pool*, *EoH*, and *HeurAgenix* use the same training or evolution schedule and budgets for fair comparison. All LLM-based methods use GLM-4-Flash with temperature 1.0. *Construct. (Limit)* uses the same budget of 14,400 training episodes as CoEvoRL to compare solution quality under a matched training budget, where *Construct. (Full)* follows the original papers to assess their performance with the complete training budget.

CoEvoRL and heuristic-selection baselines run on CPU (AMD Ryzen Threadripper 3990X) while construction baselines run on GPUs (NVIDIA GeForce RTX 3080). Wall-clock time includes initialization, training, internal evaluation, and LLM interaction. All results, except *Construct. (Full)* and *Best Heur.*, are obtained from 10 independent runs with different random seeds.

C Additional Experiments

C.1 Full Performance Results

Table 7 reports the full results, including the objective values and optimality gaps for all methods on JSSP, TSP, MKP, and OBPP, where results from multiple seeds are summarized as mean ± standard deviation. The arrows indicate whether the objective is minimization (↓) or maximization (↑).

Table 7: Full performance comparison including objective values and standard deviation.

Problem	JSSP					
Size	$J = 10,\ M = 10$		$J = 15,\ M = 15$		$J = 20,\ M = 20$	
	Obj. ↓	Gap	Obj. ↓	Gap	Obj. ↓	Gap
Optimal	815.4	–	1195.7	–	1581.6	–
Construct. (Full)	945.1	15.87%	1444.6	20.79%	1956.8	23.68%
Construct. (Limit)	996.0±8.8	22.12%±1.10%	1519.5±40.9	27.08%±3.44%	2111.9±66.3	33.52%±5.71%
Best Heur.	995.2	22.13%	1517.8	26.92%	2033.9	28.59%
EoH	990.9±11.4	21.60%±1.41%	1506.2±19.1	25.95%±1.60%	2029.0±8.5	28.28%±0.54%
Fixed Pool	944.1±5.3	15.77%±0.65%	1461.1±8.0	22.16%±0.66%	1964.5±4.0	24.19%±0.25%
HeurAgenix	938.7±9.8	15.11%±1.18%	1450.3±10.7	21.27%±0.89%	1958.4±30.8	23.81%±1.95%
CoEvoRL (Ours)	**930.4±7.9**	**14.09%±0.97%**	**1433.1±10.0**	**19.84%±0.83%**	**1941.1±16.6**	**22.72%±1.05%**

Problem	TSP					
Size	$N = 20$		$N = 50$		$N = 100$	
	Obj. ↓	Gap	Obj. ↓	Gap	Obj. ↓	Gap
Optimal	3.781	–	5.706	–	7.761	–
Construct. (Full)	3.784	0.08%	5.726	0.35%	7.820	0.77%
Construct. (Limit)	4.470±0.020	18.32%±0.54%	7.993±0.093	40.12%±1.63%	12.306±0.331	58.58%±4.27%
Best Heur.	4.445	17.62%	7.003	22.75%	9.679	24.75%
EoH	4.403±0.178	16.50%±4.69%	6.957±0.136	21.98%±2.41%	9.459±0.293	21.90%±3.79%
Fixed Pool	4.181±0.015	10.66%±0.40%	6.660±0.028	16.72%±0.50%	9.307±0.023	19.93%±0.30%
HeurAgenix	4.067±0.035	7.62%±0.91%	6.443±0.070	12.92%±1.21%	9.054±0.159	16.66%±2.05%
CoEvoRL (Ours)	**4.047±0.030**	**7.12%±0.80%**	**6.345±0.067**	**11.21%±1.18%**	**8.842±0.119**	**13.94%±1.53%**

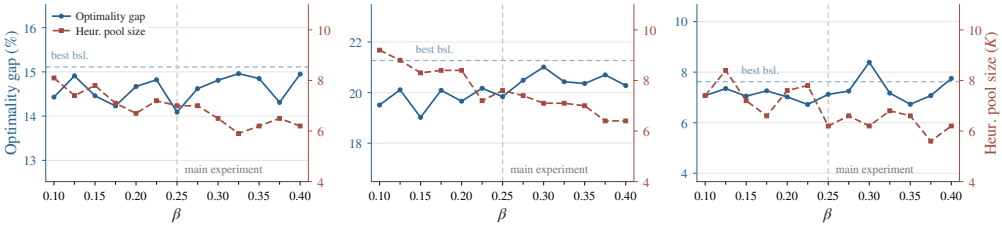
Problem	MKP					
Size	$N = 100$		$N = 250$		$N = 500$	
	Obj. ↑	Gap	Obj. ↑	Gap	Obj. ↑	Gap
Optimal	43.55	–	109.97	–	220.16	–
Construct. (Full)	41.78	4.05%	100.61	8.52%	198.39	9.89%
Construct. (Limit)	41.61±0.91	4.45%±2.08%	98.61±7.98	10.34%±7.26%	198.21±11.53	9.97%±5.24%
Best Heur.	42.37	2.71%	107.80	1.98%	216.82	1.51%
EoH	42.67±0.19	2.01%±0.43%	108.15±0.41	1.66%±0.37%	217.09±0.61	1.39%±0.28%
Fixed Pool	42.89±0.03	1.51%±0.06%	109.16±0.04	0.74%±0.04%	219.10±0.07	0.48%±0.03%
HeurAgenix	42.94±0.05	1.40%±0.11%	109.07±0.11	0.82%±0.10%	218.91±0.39	0.57%±0.18%
CoEvoRL (Ours)	**42.97±0.08**	**1.32%±0.18%**	**109.36±0.08**	**0.55%±0.07%**	**219.32±0.07**	**0.38%±0.03%**

Problem	OBPP					
Size	$N = 100$		$N = 250$		$N = 500$	
	Obj. ↓	Gap	Obj. ↓	Gap	Obj. ↓	Gap
Optimal	40.83	–	101.06	–	201.31	–
Construct. (Full)	43.51	6.56%	106.13	5.02%	209.78	4.21%
Construct. (Limit)	43.75±0.05	7.14%±0.11%	107.00±0.07	5.88%±0.07%	211.90±0.14	5.26%±0.07%
Best Heur.	43.58	6.72%	106.58	5.46%	211.18	4.90%
EoH	43.56±0.03	6.67%±0.08%	106.55±0.08	5.44%±0.08%	211.18±0.00	4.90%±0.00%
Fixed Pool	43.49±0.06	6.51%±0.16%	106.03±0.12	4.92%±0.11%	209.98±0.34	4.30%±0.17%
HeurAgenix	43.63±1.31	6.86%±3.26%	121.18±26.89	19.97%±26.69%	227.96±50.15	13.25%±24.94%
CoEvoRL (Ours)	**42.91±0.31**	**5.09%±0.77%**	**104.51±0.60**	**3.41%±0.59%**	**206.59±1.98**	**2.62%±0.98%**

Table 8: Wilcoxon signed-rank test p-values comparing CoEvoRL with the best baseline.

Problem	Size	Best Bsl.	CoEvoRL	p
JSSP	10×10	15.11%	14.09%	2.05×10^{-6}
	15×15	21.27%	19.84%	1.18×10^{-12}
	20×20	23.81%	22.72%	6.59×10^{-9}
TSP	20	7.62%	7.12%	3.60×10^{-2}
	50	12.92%	11.21%	3.58×10^{-8}
	100	16.66%	13.94%	5.95×10^{-18}

Problem	Size	Best Bsl.	CoEvoRL	p
MKP	100	1.40%	1.32%	4.71×10^{-2}
	250	0.74%	0.55%	3.48×10^{-15}
	500	0.48%	0.38%	2.75×10^{-12}
OBPP	100	6.51%	5.09%	3.08×10^{-23}
	250	4.92%	3.41%	1.58×10^{-30}
	500	4.30%	2.62%	1.58×10^{-30}



(a) JSSP 10×10 (b) JSSP 15×15 (c) TSP-20

Figure 4: Sensitivity analysis on the pool-expansion penalty coefficient β.

C.2 Statistical Significance Analysis

Table 8 reports mean optimality gaps and paired Wilcoxon signed-rank test p-values computed from per-instance gaps. For each problem setting, we compare CoEvoRL with the best heuristic-based baseline. CoEvoRL achieves better mean gaps in all 12 settings, with all comparisons statistically significant at the 0.05 level. Specifically, 10 of the 12 comparisons yield $p < 10^{-5}$, and the evidence strengthens with instance size. The two weakest, TSP-20 ($p = 0.0360$) and MKP-100 ($p = 0.0471$), are the smallest instances of their problems, where a fixed pool already covers most high-value actions and leaves little room for coverage-guided evolution.

C.3 Sensitivity Analysis on β

We examine the sensitivity to β, which controls the pool-expansion penalty in (12), by varying it from 0.10 to 0.40. Figure 4 compares JSSP 10×10 and 15×15 to assess sensitivity across problem sizes, and compares TSP-20 to assess sensitivity across problem types. Increasing β generally reduces the heuristic pool size, whereas the optimality gap varies non-monotonically. Smaller penalties admit larger pools without consistently improving solution quality, while stronger penalties further restrict pool expansion but can increase the gap, particularly on JSSP 15×15. These trends are consistent with the trade-off between action coverage and selector learning complexity underlying the admission criterion. Meanwhile, over a broad range of β, CoEvoRL maintains competitive solution quality despite changes in pool size. This suggests that our coverage diagnosis can effectively guide the heuristic evolution under both weaker and stronger penalties on pool expansion, supporting the adaptability of our framework under different pool-size budgets.

Note that we selected the default $\beta = 0.25$ in our main experiments solely based on the results of JSSP 10×10, and fixed it across all problems and sizes. Although this β value may not be optimal for every setting, CoEvoRL still outperforms all heuristic-based baselines, as reported in our results.