MOTIF: Multi-strategy Optimization via Turn-based Interactive Framework

Nguyen Viet Tuan Kiet[1], **Tung Dao**[1], **Cong Dao Tran**[2], **Huynh Thi Thanh Binh**[1*]

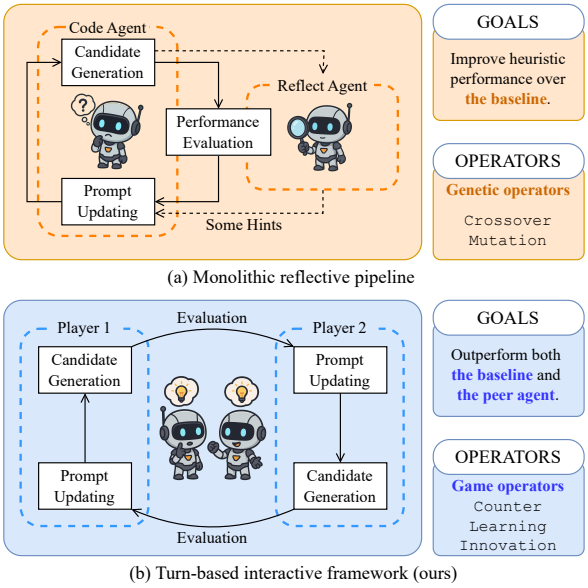
[1]Hanoi University of Science and Technology, Vietnam
[2]FPT Software AI Center, Vietnam
{kiet.nvt220032, tung.dv242050M}@sis.hust.edu.vn, daotc2@fpt.com, binhht@soict.hust.edu.vn

Abstract

Designing effective algorithmic components remains a fundamental obstacle in tackling NP-hard combinatorial optimization problems (COPs), where solvers often rely on carefully hand-crafted strategies. Despite recent advances in using large language models (LLMs) to synthesize high-quality components, most approaches restrict the search to a single element—commonly a heuristic scoring function—thus missing broader opportunities for innovation. In this paper, we introduce a broader formulation of solver design as a multi-strategy optimization problem, which seeks to jointly improve a set of interdependent components under a unified objective. To address this, we propose **M**ulti-strategy **O**ptimization via **T**urn-based **I**nteractive **F**ramework (**MOTIF**)—a novel framework based on Monte Carlo Tree Search that facilitates turn-based optimization between two LLM agents. At each turn, an agent improves one component by leveraging the history of both its own and its opponent's prior updates, promoting both competitive pressure and emergent cooperation. This structured interaction broadens the search landscape and encourages the discovery of diverse, high-performing solutions. Experiments across multiple COP domains show that MOTIF consistently outperforms state-of-the-art methods, highlighting the promise of turn-based, multi-agent prompting for fully automated solver design.

1 Introduction

From vehicle routing and scheduling to circuit design and logistics, Combinatorial Optimization Problem (COP) underpin many of society's most complex decision systems (Phan Duc et al. 2025; Rajendran 1993; Tan, Mohd-Mokhtar, and Arshad 2021). Yet, despite decades of progress, building effective solvers for COPs remains a costly and highly manual endeavor—requiring domain-specific heuristics, iterative tuning, and substantial expert effort. Heuristics and meta-heuristics are currently the most common methods used to tackle these problems (Desale et al. 2015). However, these techniques often require substantial expert input, which increases development costs and limits their flexibility across various problems.



(a) Monolithic reflective pipeline

(b) Turn-based interactive framework (ours)

Figure 1: (a) *Monolithic reflective pipeline*: generation and reflection exchange one-way hints around a single evaluator with minimal behavioral awareness. (b) *Turn-based interactive framework*: two agents take turns generating and updating under shared evaluations, yielding explicit peer feedback, richer diversity, and adaptive explore–exploit balance.

Significant progress has been made in the field of Automatic Heuristic Design (AHD) (Burke et al. 2018). A prominent example is Genetic Programming Hyper Heuristic (GPHH) (Langdon and Poli 2013), a widely recognized algorithm for AHD. However, many existing approaches still rely heavily on predefined heuristic spaces or rules designed by human experts (Pillay and Qu 2018). Recently, Language Hyper-Heuristics (LHH) has emerged as a promising solution to overcome this limitation. Rather than manually designing heuristic functions or relying on fixed search spaces, researchers have begun leveraging the reasoning capabilities of Large Language Model (LLM) to automatically generate more innovative heuristics. For instance, Liu et al. (Liu et al. 2024; Romera-Paredes et al. 2024) integrated evolutionary algorithms with LLM to evolve code with minimal

*Corresponding author

human intervention. Ye et al. introduced Reflective Evolution (ReEvo) (Ye et al. 2024), a framework that combines evolutionary search with self-reflection (Shinn et al. 2023) to further improve the effectiveness of LLM. Subsequent studies, such as HSEvo (Dat, Doan, and Binh 2025), incorporated diversity measurements and harmony search (Shi, Han, and Si 2012) to encourage more creative solutions. In addition to evolutionary methods, other search algorithms have also been explored in LHH. For example, MCTS-AHD (Zheng et al. 2025) employed Monte Carlo Tree Search (MCTS) with progressive widening (Browne et al. 2012; Coulom 2007b) to better explore the heuristic space. These methods, which require minimal expert knowledge, have achieved competitive performance compared to state-of-the-art solvers and neural network–based approaches.

However, previous approaches have generally focused on searching for a single heuristic function within a general framework, which limits the creativity and exploration potential of LLM. To address this limitation, we propose a multi-strategy optimization approach that aims to optimize a set of strategies simultaneously rather than focusing on just one. This broader search space enables LLM to generate more diverse and innovative solutions. Moreover, because multi-strategy optimization requires a significantly more complex search mechanism than existing methods, we introduce a turn-based interactive framework in which two players sequentially compete to outperform each other in discovering the best set of strategies, as presented in Figure 1. In conclusion, our contributions can be summarized as follows:

- We introduce the novel problem of *multi-strategy optimization*, where the goal is to jointly optimize a system of algorithmic components under a shared performance objective. This setting generalizes prior work that only targets single-component improvements.
- We propose a two-round competitive framework that decomposes the problem into (i) *component-wise optimization* using Competitive Monte Carlo Tree Search (CMCTS) under a dynamic baseline, and (ii) *system-aware refinement* in which agents compete in a turn-based manner with fixed baselines and full system visibility.
- We design a library of competitive operators—including *counter*, *learning*, and *innovation*—that structure LLM prompting based on the opponent's behavior, recent history, and baseline context. This enables emergent self-play dynamics and yields superior implementations through adversarial pressure and contextual learning.
- We conduct extensive experiments across three algorithmic frameworks, five COP domains, and diverse strategy sets. Results consistently demonstrate that MOTIF outperforms prior frameworks—both in single-strategy and multi-strategy optimization—validating the power of turn-based self-play.

2 Related Works

2.1 Automatic Heuristics Design

Previous work on LHH (Liu et al. 2024; Ye et al. 2024; Dat, Doan, and Binh 2025; Zheng et al. 2025; Tran et al. 2025) has explored the use of LLMs to design heuristic functions via evolutionary algorithms or tree-based search. While these approaches improve over traditional metaheuristics and Neural Combinatorial Optimization (NCO), they exhibit two core limitations. First, they aim to optimize a single heuristic in isolation, overlooking the potential of weaker components that, when combined, can yield stronger overall performance. This narrow focus restricts the generative flexibility of LLMs. Second, although prior frameworks incorporate self-reflection to assess interactions between heuristics, such mechanisms are limited in scope and depth. Typically based on static comparisons or surface-level summaries, they struggle to capture hidden incompatibilities or potential synergies. As a result, the LLM receives insufficient feedback for meaningful revisions, hampering its ability to conduct deeper exploratory optimization and refine its outputs throughout the search process.

2.2 Learning Through Self-play

Recent work has shown that game-based self-play can enhance LLM reasoning through structured interaction, such as critique, revision, or deception. For instance, SPAG (Cheng et al. 2025) frames reasoning as an attacker–defender game to improve deception detection; CDG (Wang et al. 2025) trains a prover–critic pair to expose flawed reasoning; and (Fu et al. 2023a) uses self-play and AI feedback in negotiation tasks to refine decision-making. While effective for internal rationality and dialogue skills, these methods do not address the co-evolution of multiple algorithmic components for structured tasks. In contrast, our approach applies self-play to multi-strategy optimization, where LLM agents iteratively critique, improve, and compete over distinct heuristics—introducing a new paradigm of inter-strategy competition under system-level feedback. To our knowledge, this is the first use of self-play for optimizing combinatorial solvers in this setting.

3 Multi-strategy Optimization

Prior works (Liu et al. 2024; Ye et al. 2024; Dat, Doan, and Binh 2025; Zheng et al. 2025) has largely focused on tuning a single heuristic assuming it alone drives solver quality. In contrast, we adopt a multi-strategy view that treats each routine as an independent optimization target, enabling coordinated improvements across the solver pipeline. This modular perspective fosters richer interaction among routines and often yields greater overall solver synergy than isolated rule refinement.

Definition 3.1 (Domain, Instance, and Solution)**.** A COP domain d defines a class of discrete problems, such as the Travelling Salesman Problem (TSP) or Capacitated Vehicle Routing Problem (CVRP). Without loss of generality, we assume the objective is to minimize a cost function; maximization problems can be equivalently transformed by negating the objective. Under this convention, each domain specifies:

- a space of instances $\mathcal{X}_d$, where each instance $\mathbf{x} \in \mathcal{X}_d$ encodes problem-specific data (e.g., graph structure, distances matrix, constraints),

- a global solution space $\mathcal{Y}_d$, representing all possible solutions across all instances, and
- an objective function $f_d : \mathcal{X}_d \times \mathcal{Y}_d \to \mathbb{R}$ that quantifies the quality of a solution relative to a given instance.

For a specific instance $\mathbf{x}$, the feasible solutions form a subset $\mathcal{Y}_d(\mathbf{x}) \subset \mathcal{Y}_d$. The goal is to find a solution $\mathbf{y}^* \in \mathcal{Y}_d(\mathbf{x})$ that minimizes $f_d(\mathbf{x}, \mathbf{y})$.

Definition 3.2 (Solver and Strategy)**.** A solver s is an algorithmic framework that generates a solution $\mathbf{y} \in \mathcal{Y}_d(\mathbf{x})$ for a given instance $\mathbf{x} \in \mathcal{X}_d$, guided by a collection of internal routines. Each such routine is referred to as a strategy π_k, which may include heuristic scoring rules, construction policies, neighborhood moves, penalty update mechanisms, or other algorithmic components. The solver operates as:

$$\mathbf{y} = s\big(\mathbf{x} \mid (\pi_1, \pi_2, \ldots, \pi_K)\big). \tag{1}$$

Definition 3.3 (Strategy Space)**.** For each strategy type π, let $\mathcal{S}_\pi$ denote the space of all valid implementations, including both functional variants and parametrizations. All strategies in $\mathcal{S}_\pi$ share a common function signature and optimization objective, ensuring interoperability within the solver. Variations among strategies arise from differences in internal logic, parameter choices, or heuristic design, while maintaining consistent input-output behavior.

Given a solver s equipped with a sequence of K strategies $\mathbf{\Pi} = (\pi_1, \pi_2, \ldots, \pi_K)$, its performance on an instance $\mathbf{x} \in \mathcal{X}_d$ is evaluated by the objective value

$$F_d\big(\mathbf{x} \mid \mathbf{\Pi}\big) = f_d\Big(\mathbf{x}, s\big(\mathbf{x} \mid \mathbf{\Pi}\big)\Big). \tag{2}$$

Since the solver's behavior is determined by its underlying strategies, optimizing solver quality amounts to optimizing the design of these strategies.

Definition 3.4 (Multi-strategy Optimization)**.** The multi-strategy optimization problem aims to simultaneously optimize a sequence of K strategies $\mathbf{\Pi} = (\pi_1, \pi_2, \ldots, \pi_K)$, implemented within solver s, where each π_k belongs to its corresponding strategy space $\mathcal{S}_{\pi_k}$, to minimize the expected solver objective across the domain, as follows:

$$\mathbf{\Pi}^* = \arg\min_{\mathbf{\Pi}} \ \mathbb{E}_{\mathbf{x}\sim\mathcal{X}_d}\Big[F_d\big(\mathbf{x} \mid \mathbf{\Pi}\big)\Big], \tag{3}$$

subject to a total computational budget T (e.g., total solver runs, code evaluation time).

Ours formulation enables a more holistic view of solver design: instead of optimizing components in isolation, we jointly evolve multiple, interacting strategies under a unified optimization objective. By modeling each strategy as an adaptive and parameterizable unit, our approach supports richer design spaces and unlocks synergies that static, handcrafted routines cannot capture.

4 MOTIF

4.1 Architectural Overview

MOTIF adopts a two-round optimization process that reflects a natural progression from local specialization to global integration. The goal is to gradually increase the cognitive burden placed on the agents, allowing them to first perform focused improvements under simplified context before advancing to more complex, system-aware reasoning.

Component-wise Competition. In the first phase, the solver is decomposed into a set of individual strategies $\{\pi_1, \pi_2, \ldots, \pi_K\}$, each optimized independently. A separate competitive tree is constructed for each strategy, where two agents alternately propose revisions to a single component at a time. During this phase, agents operate with partial context: they have access only to the implementation and performance of the target strategy, its dynamic baseline, and the opponent's proposal.

System-aware Refinement. Once all components have been optimized in isolation, the second phase begins. Here, agents revisit each strategy sequentially, but now with access to the full system configuration. At each turn, a player proposes a modification to one strategy while observing the entire combination of current implementations. A fixed global baseline is used during the optimization of each component, ensuring fairness and stability across turns. This phase encourages the emergence of synergistic adaptations, where the effectiveness of one strategy depends on how well it integrates with others.

Turn-based Dual-agent Game. Throughout both phases, the optimization proceeds as a turn-based game. At each turn, a player selects an operator to generate a new implementation, aiming not only to beat the baseline but also to outperform the opponent. This structure fosters both adversarial behavior—by incentivizing relative gains—and cooperative dynamics—by reinforcing global cost reduction. The continuous interplay between agents creates a rich and adaptive optimization trajectory that evolves from competitive local improvements to system-level coherence.

4.2 Component-wise Competition

Outer Controller and Strategy Selection. In the first phase, MOTIF maintains a separate tree $\mathcal{T}_k$ for each strategy π_k. At each outer iteration, a controller selects one tree to optimize using a UCB-based rule:

$$k = \arg\max_{1\le k\le K} \left(\frac{R_j}{N_j} + C_{\text{out}} \sqrt{\frac{\ln \sum_i N_i}{N_j}} \right), \tag{4}$$

where R_j and N_j are the total reward and visit count of tree $\mathcal{T}_j$, and C_{out} controls exploration.

The selected strategy is then optimized via a two-player competition. Although only one component is updated per turn, its impact is assessed globally via the system cost.

Node Representation. Each node represents a game state where $\mathcal{P}_1$ and $\mathcal{P}_2$ each hold a distinct implementation of π_k, along with their respective code, and improvement metrics.

Nodes are linked through operator-based transformations, and updated using turn-aware reward signals. Since only one player acts per turn, the structure supports asymmetric credit assignment and promotes adaptive behavior. Each $\mathcal{T}_k$ evolves via a Competitive MCTS, as described next.

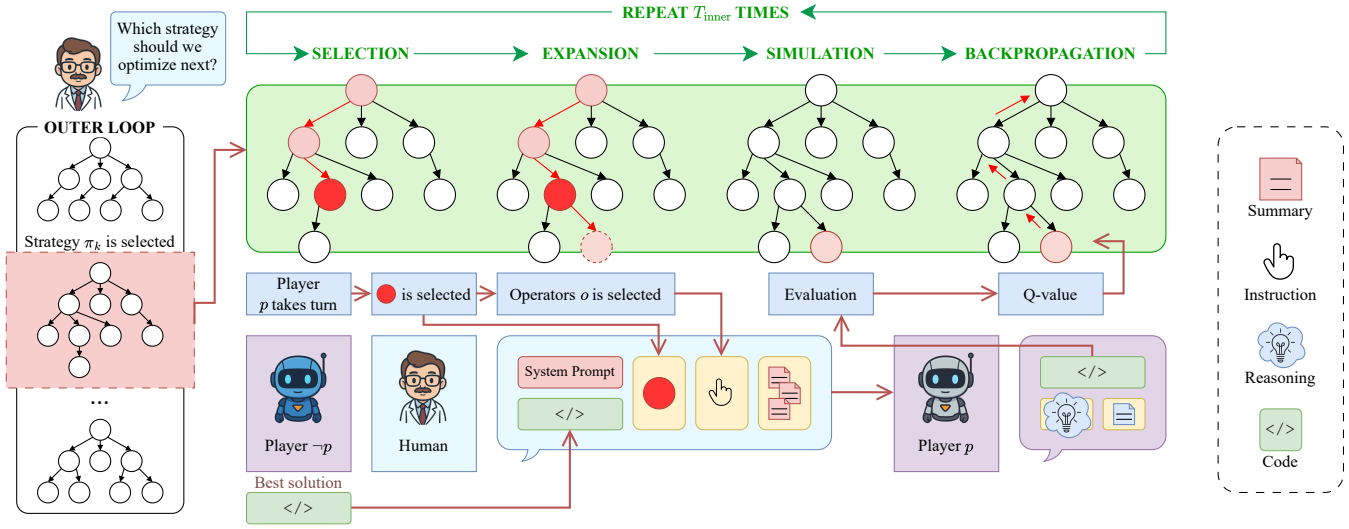


Figure 2: Overview of the component-wise competition framework. **Left**: The outer controller selects a strategy tree $\mathcal{T}_k$ to optimize in each iteration. **Right**: The selected tree is improved via a two-player CMCTS, where agents alternate turns using one operator. Each move prompts the LLM with contextualized information about the current and opponent implementations, as well as prior history. Generated code is evaluated and backpropagated through the tree based on a Q-value that accounts for both absolute and relative improvements. The best solution is retained for potential system-level baseline updates.

Competitive Monte Carlo Tree Search. In CMCTS, each node represents a two-player duel over a specific strategy π_k, where both players maintain separate implementations and cost estimates. The tree expands through one of three competitive operators that guide the language model $\mathcal{L}$ in proposing new code:

- *Counter* targets weaknesses in the opponent's code, prompting the LLM to design adversarial improvements that exploit inefficiencies or limitations.
- *Learning* encourages synthesis by integrating strengths from the opponent's implementation into the agent's own solution.
- *Innovation* promotes exploration by instructing the LLM to ignore prior solutions and propose novel, potentially unconventional approaches.

Together, these operators span a rich spectrum of behaviors—from adversarial exploitation to constructive integration to exploratory invention. During search, the UCB formula is applied at the operator level to ensure systematic coverage of diverse transformation types.

As illustrated in Figure 2, the competitive search follows a standard MCTS procedure with the following four steps:

1. Selection. The search begins at the root and follows the child nodes created by the current player. If all operators have been explored at a node, the child with the highest average Q-value is selected. Otherwise, an unexplored operator is chosen based on a UCB rule:

$$\text{UCB}(o,\pi) = \frac{Q(o,\pi)}{N(o,\pi)} + C_{\text{in}}\sqrt{\frac{\ln N(\pi)}{N(o,\pi)}}, \tag{5}$$

where $Q(o,\pi)$ is the cumulative reward, $N(o,\pi)$ the visit count for operator o, and $N(\pi)$ the total visits to node π. The constant C_{in} balances exploration and exploitation.

2. Expansion. A new node is created by applying the selected operator to the current player's implementation. Let $p \in \{\mathcal{P}_1, \mathcal{P}_2\}$ be the active player and $\pi^{(p)}$ their current code. The language model $\mathcal{L}$ generates a new implementation via:

$$\pi' \leftarrow \mathcal{L}\left(\text{Prompt}\left(o; \left\{\pi^{(p)}, \pi^{(\neg p)}\right\}; \left\{\mathcal{H}^{(p)}, \mathcal{H}^{(\neg p)}\right\}; \mathcal{B}\right)\right), \tag{6}$$

where $o \in \{\text{counter, learning, innovation}\}$ is the selected operator type; $\pi^{(p)}$, $\pi^{(\neg p)}$ are the current implementations of the active player and their opponent, respectively; $\mathcal{H}^{(p)}$, $\mathcal{H}^{(\neg p)}$ are the recent move histories of each player, including summaries of past improvements and operator usage; $\mathcal{B}$ denotes the current baseline, consisting of the reference implementation and its associated system cost.

3. Evaluation. The modified implementation π' is inserted into the system, replacing the corresponding strategy. The resulting cost is compared to the fixed baseline to compute the improvement $I^{(p)}$ (%) for the current player p. The opponent's value $I^{(\neg p)}$ (%) reflects their unchanged performance at the same node, inherited from the parent.

4. Backpropagation. The resulting reward is propagated upward through the tree. For player p, the Q-value combines both absolute improvement and competitive gain:

$$Q^{(p)} = \lambda \cdot \sigma(I^{(p)}) + (1-\lambda) \cdot \sigma(I^{(p)} - I^{(\neg p)}), \tag{7}$$

where $\sigma(x) = \frac{1}{1+e^{-kx}}$ and $\lambda \in [0,1]$ controls the balance. This reward updates statistics for both the visited nodes and

the operator used, supporting informed selection in future iterations.

Dynamic Baseline. To drive meaningful progress during optimization, we adopt a dynamic baseline mechanism. At any outer iteration, the baseline refers to the best-known implementation of a strategy—initially handcrafted or warm-started—and is updated whenever an agent produces a strictly better implementation. Each agent competes not against a static reference, but against the most recent high-performing solution from the opposite player.

4.3 System-aware Refinement

In the second phase, each strategy π_k is re-optimized with full system visibility. Unlike the first phase, players now operate over the entire configuration $\mathbf{\Pi} = (\pi_1, \ldots, \pi_K)$, using a fixed baseline for each strategy.

The key distinction lies in the prompt construction: rather than focusing on local context and opponent-specific features, the prompt now includes the full system configuration, global baseline cost, and historical search traces. This enables the language model $\mathcal{L}$ to reason about system-level dependencies, hyperparameter synergies, and global optimization behavior:

$$\pi_k' \leftarrow \mathcal{L}\left(\text{Prompt}\left(\mathbf{\Pi}; \left\{\pi_k^{(p)}, \pi_k^{(\neg p)}\right\}; \left\{\mathcal{H}^{(p)}, \mathcal{H}^{(\neg p)}\right\}\right)\right). \tag{8}$$

An update is accepted only if it improves upon both the baseline and the opponent's best result, promoting coordination across components and discovering configurations not reachable through isolated optimization. See Appendix C.4 for more details.

5 Experiments

Settings. We employ the `gpt-4o-mini-2024-07-18` model for both agents throughout our experiments. This model, although not highly capable in coding tasks, was deliberately chosen for its affordability, fast inference, and support for structured outputs, which are essential for reliable parsing and downstream processing. Given its limited reasoning ability, we design a structured response format that enforces clarity in its generation process.

Specifically, we adopt a lightweight variant of chain-of-thought prompting (Wei et al. 2022). Each response must contain a `reasoning` field, in which the model is required to explain its thought process in no more than five concise sentences. The `code` field contains the generated Python implementation, while the `summary` field briefly describes the key changes introduced in the current turn. This tri-field schema helps us better monitor reasoning quality, code correctness, and strategic intent during each optimization step.

Training and Evaluation Setup. All optimization is conducted on a lightweight training set, while final performance is measured on a held-out test set. Full details of the setup, including dataset construction and evaluation protocol, are provided in Appendix D.1.[1]

[1]Our code is available at: github.com/HaiAu2501/MOTIF

Methods	TSP50	TSP100	TSP200	TSP500
EoH	0.000 ± 0.000	0.012 ± 0.009	0.639 ± 0.097	5.796 ± 0.146
ReEvo	0.000 ± 0.000	0.335 ± 0.449	2.081 ± 1.927	7.918 ± 3.051
HSEvo	0.108 ± 0.138	3.095 ± 2.363	11.254 ± 6.424	24.593 ± 7.314
MCTS-AHD	0.000 ± 0.000	0.024 ± 0.007	0.652 ± 0.111	5.611 ± 0.216
MOTIF	$\mathbf{0.000 \pm 0.000}$	$\mathbf{0.007 \pm 0.006}$	$\mathbf{0.610 \pm 0.100}$	$\mathbf{5.577 \pm 0.255}$

Table 1: Average optimal gap (‰) on TSP with GLS: comparison across methods (3 runs).

5.1 Single-strategy Search

Guided Local Search. Among classical metaheuristics for combinatorial optimization, one of the most effective is Guided Local Search (GLS) (Voudouris and Tsang 1999), which enhances local search by penalizing frequent features of poor local optima. It has demonstrated strong performance on combinatorial problems such as the TSP. In practice, GLS is often deployed with handcrafted heuristics that guide the search toward promising regions of the solution space.

To preserve the efficiency of GLS, prior works typically optimize only a single heuristic component—usually the precomputed penalty scoring function—while keeping the surrounding search logic fixed. We adopt the same design philosophy to ensure fair comparison and fast runtime during evaluation.

As shown in Table 1, even under this restricted single-strategy setting, our method significantly outperforms several recent LLM-based methods.

5.2 Multi-strategy Search

Ant Colony Optimization. Simulating the pheromone-based foraging behavior of ants, Ant Colony Optimization (ACO) (Dorigo, Birattari, and Stutzle 2007; Dorigo and Stützle 2018) has established itself as a versatile and competitive framework for solving NP-hard combinatorial problems. Classical ACO implementations consist of multiple expert-designed components, each responsible for a specific sub-behavior of the colony. In our view, these components—often treated as fixed formulas—can themselves be subject to optimization. Specifically, we target the following sub-strategies: (i) the initialization scheme for heuristic and pheromone scores, (ii) the construction policy that combines heuristic and pheromone information into a transition probability distribution, and (iii) the pheromone update rule that determines how feedback from previous solutions reinforces or weakens certain paths.

Figure 3 compares the performance of our proposed MOTIF framework with several competitive baselines across five distinct optimization problems, each evaluated at five problem scales. The results demonstrate that jointly optimizing multiple strategies in ACO yields substantial performance gains over both human-designed and LLM-generated single-strategy baselines.

Deconstruction-then-Repair. We introduce a simple yet effective constructive framework as the starting point for strategy exploration. The framework operates in three sequential stages: (i) a greedy initialization guided by an edge

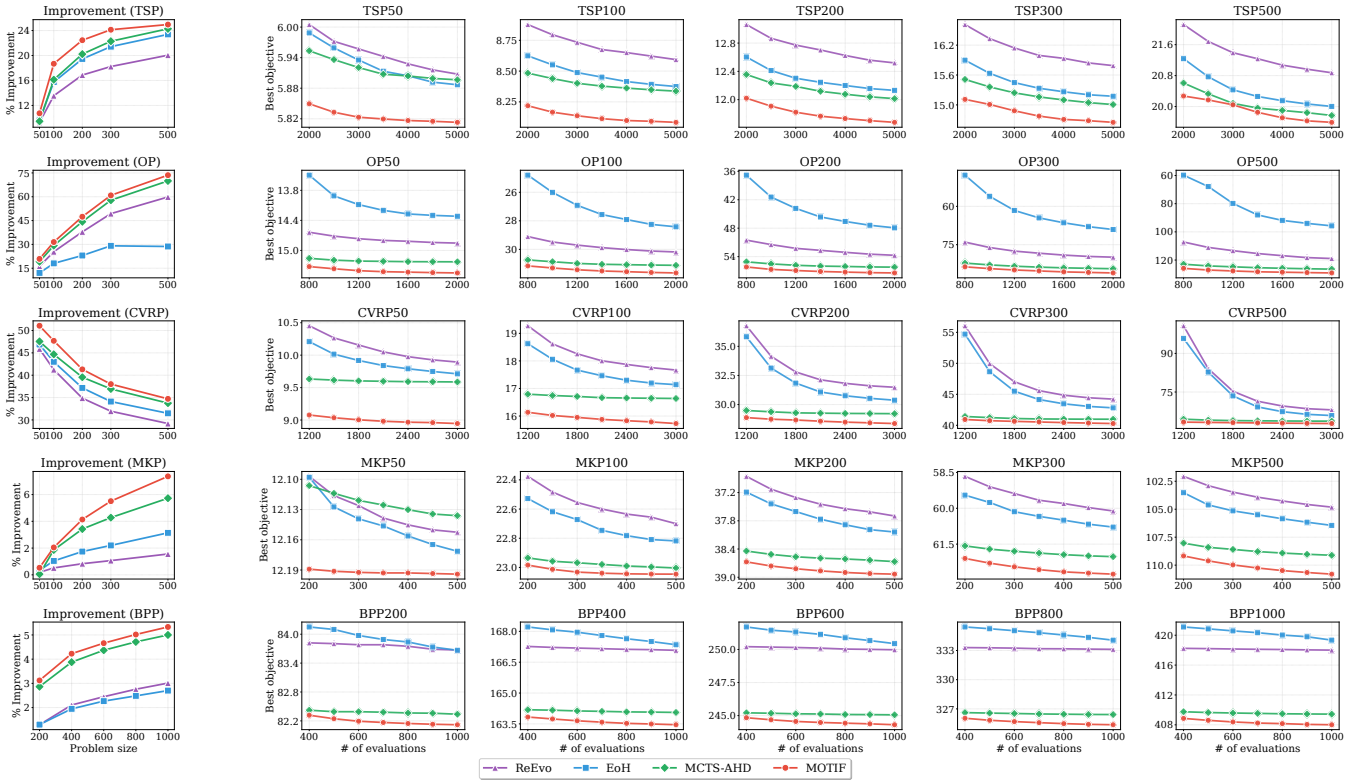


Figure 3: Comparison of AHD frameworks applied to the ACO algorithm. EoH, ReEvo, and MCTS-AHD optimize only a single strategy component (the heuristic function), while MOTIF concurrently optimizes two or three strategy components. Left: Relative performance compared to the human-designed baseline. Right: Evaluation curves showing the best objective value over time (measured by the number of evaluations), averaged over three independent runs.

scoring function π_1; (ii) a partial destruction phase that removes low-quality elements based on a badness scoring strategy π_2; and (iii) a repair phase that incrementally reconstructs the solution using a placement selection criterion π_3. This process reflects a natural design philosophy: producing reasonably good solutions quickly without engaging in costly search loops.

Table 2 reports the improvement margins achieved by optimizing one, two, or all three strategies in the framework. Each component contributes differently depending on the problem domain—for example, π_3 is critical in TSP, while π_2 plays a larger role in CVRP and BPP. Overall, jointly optimizing two or more strategies consistently outperforms the single-strategy setting, demonstrating the synergistic benefits of multi-strategy optimization.

These results also suggest that, beyond merely tuning isolated heuristics, LLMs hold promise in co-designing full algorithmic pipelines. This supports the broader vision of transforming a simple, human-sketched pipeline into a strong, domain-adapted algorithm through iterative LLM-guided search.

5.3 Convergence and Diversity Analysis

Figure 4 plots the best optimality gap achieved at each outer iteration across five independent training runs. While occasional regressions are observed—due to the exploratory nature of operator moves—the overall trend is clearly downward, indicating steady convergence.

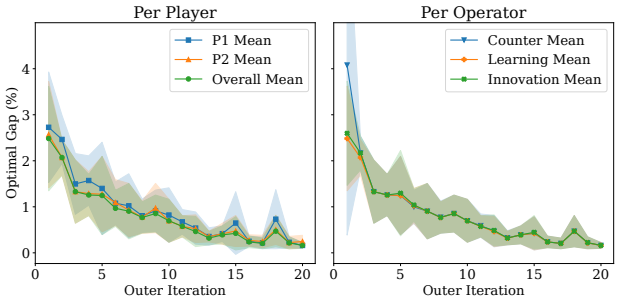


Figure 4: Convergence behavior of the MOTIF framework during training, averaged over five independent runs. **Left:** Best optimality gap achieved at each outer iteration, shown separately for Player 1 (P1), Player 2 (P2), and the overall best. **Right:** Performance breakdown by operator type.

Interestingly, both players exhibit closely aligned performance curves throughout training, suggesting a dynamic equilibrium in which competitive pressure drives mutual improvement. Furthermore, as shown on the right, all three

Strategies	TSP			CVRP			BPP		
	50	100	200	50	100	200	100	200	300
π_1	0.39 ± 0.17	0.81 ± 0.27	0.72 ± 0.45	1.47 ± 0.16	2.96 ± 0.25	2.30 ± 0.39	3.17 ± 0.00	4.82 ± 0.00	4.77 ± 0.00
π_2	3.13 ± 0.05	6.24 ± 0.21	8.18 ± 0.26	4.27 ± 0.24	7.04 ± 0.73	6.85 ± 0.75	23.19 ± 0.05	24.42 ± 0.02	25.00 ± 0.00
π_3	$\mathbf{3.88 \pm 0.04}$	7.91 ± 0.01	11.35 ± 0.03	6.34 ± 0.08	5.07 ± 0.02	4.28 ± 0.03	12.70 ± 7.51	12.84 ± 7.15	12.15 ± 7.49
(π_1, π_2)	2.58 ± 0.50	5.04 ± 0.78	5.84 ± 1.26	6.06 ± 1.64	7.94 ± 2.71	8.85 ± 2.64	23.15 ± 0.15	24.40 ± 0.06	24.98 ± 0.02
(π_2, π_3)	3.83 ± 0.16	$\mathbf{7.97 \pm 0.14}$	11.59 ± 0.07	10.64 ± 0.74	12.31 ± 0.57	11.71 ± 1.03	23.05 ± 0.11	24.32 ± 0.15	24.89 ± 0.15
(π_1, π_2, π_3)	$\mathbf{3.88 \pm 0.04}$	7.96 ± 0.03	$\mathbf{11.65 \pm 0.05}$	$\mathbf{10.98 \pm 1.64}$	$\mathbf{12.84 \pm 3.17}$	$\mathbf{13.06 \pm 3.67}$	$\mathbf{23.94 \pm 0.55}$	$\mathbf{25.02 \pm 0.41}$	$\mathbf{25.41 \pm 0.26}$

Table 2: Performance comparison between single-strategy and multi-strategy optimization across three combinatorial problems: TSP, CVRP, and BPP, each evaluated at various instance sizes. Results indicate percentage improvement over the human-designed baseline, averaged over 3 runs. Strategies π_1, π_2, π_3 denote initialization, deconstruction, and repair respectively.

operators—*counter*, *learning*, and *innovation*—demonstrate similar convergence profiles, reflecting the robustness and adaptability of our operator design across varying strategic contexts.

Operator	Success rate (↑)	Novelty score (↑)	Silhouette score (↑)
Counter	93 ± 2 %	0.0136 ± 0.0067	0.5121 ± 0.0208
Learning	92 ± 2 %	0.0118 ± 0.0054	$\mathbf{0.5325 \pm 0.0300}$
Innovation	97 ± 1 %	$\mathbf{0.0175 \pm 0.0110}$	0.4793 ± 0.0240

Table 3: Diversity and success analysis for each operator. Success rate measures the proportion of generated implementations that improve upon the current baseline. Novelty score captures the average semantic distance to other operators' outputs within the same strategy. Silhouette score quantifies intra-operator cohesion and inter-operator separation in the embedding space. All metrics are averaged over five independent runs.

Another question in operator design is whether LLM-based strategies can break free from conventional coding patterns and exhibit genuinely novel behavior. To examine this, we conduct a semantic diversity analysis of the generated implementations. Specifically, we compute two metrics—*novelty* and *silhouette score*—based on code embeddings to assess how distinct and well-separated each operator's outputs are. Formal definitions and computation details of these metrics are provided in Appendix D.4.

Table 3 compares the three operators across success rate, novelty, and silhouette score. *Innovation* exhibits the highest novelty, reflecting its broad exploration of new code regions. However, it has the lowest silhouette score, suggesting its outputs are scattered and lack internal consistency. *Counter* achieves moderate novelty and silhouette, indicating a balanced behavior—exploring new directions while maintaining some cohesion. *Learning* ranks lowest in novelty but highest in silhouette, showing that it tends to exploit familiar patterns with consistent and stable outputs.

These trends align with the intended design: *innovation* promotes diversity, *learning* refines known ideas, and *counter* responds adaptively to the opponent.

5.4 Ablation Study

Table 4 reports the impact of removing various components on the final optimal gap (lower is better) for two solvers: ACO and GLS. Among three components, removing the dynamic baseline causes the most severe performance drop, especially in the test phase—indicating that without continual baseline updates, the system lacks incentive to improve and tends to stagnate.

Disabling reasoning also leads to a notable degradation, highlighting its importance in enabling the model to reflect on prior failures and generate meaningful revisions.

Methods	ACO		GLS	
	Train	Test	Train	Test
w/o Outer Controller	0.74 ± 0.39	5.61 ± 0.86	0.81 ± 0.14	2.88 ± 0.11
w/o Dynamic Baseline	1.55 ± 0.26	9.50 ± 4.24	0.62 ± 0.20	2.94 ± 0.18
w/o Final Round	1.26 ± 0.52	5.72 ± 0.78	0.39 ± 0.13	2.82 ± 0.06
w/o Reasoning	1.63 ± 0.94	7.88 ± 3.76	0.80 ± 0.23	3.05 ± 0.04
w/o Player Context	0.82 ± 0.63	6.06 ± 1.27	0.53 ± 0.07	2.97 ± 0.04
w/o Opponent Context	0.97 ± 0.50	5.58 ± 1.31	0.48 ± 0.00	2.98 ± 0.13
w/o Counter	1.42 ± 0.07	6.71 ± 0.52	0.64 ± 0.19	3.20 ± 0.12
w/o Learning	0.91 ± 0.32	9.59 ± 5.74	0.62 ± 0.02	3.00 ± 0.12
w/o Innovation	1.73 ± 0.31	6.65 ± 0.80	0.42 ± 0.09	2.86 ± 0.00
MOTIF (original)	0.80 ± 0.28	$\mathbf{5.21 \pm 0.35}$	0.34 ± 0.19	$\mathbf{2.73 \pm 0.05}$

Table 4: Ablations on components, prompting, and operators. Numbers denote optimality gap (%, lower is better), averaged over 3 runs. Results are reported under a small evaluation budget, using TSP50 for training and TSP100 for testing, to highlight performance differences more clearly.

6 Conclusion

We introduced a two-phase competitive optimization framework that leverages turn-based interactions between LLM agents to improve multi-strategy solvers. Through dynamic baselines, self-play prompting, and system-aware refinement, our method consistently outperforms prior approaches across diverse combinatorial problems. The results highlight the importance of both adversarial pressure and structured cooperation in driving algorithmic innovation.

Acknowledgements

This research was funded by Hanoi University of Science and Technology under project code T2024-PC-038.

References

André, H.; and Kevin, T. 2020. *Neural Large Neighborhood Search for the Capacitated Vehicle Routing Problem*. IOS Press.

Browne, C. B.; Powley, E.; Whitehouse, D.; Lucas, S. M.; Cowling, P. I.; Rohlfshagen, P.; Tavener, S.; Perez, D.; Samothrakis, S.; and Colton, S. 2012. A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in games*, 4(1): 1–43.

Burke, E.; Gendreau, M.; Hyde, M.; Kendall, G.; Ochoa, G.; Özcan, E.; and Qu, R. 2013. Hyper-heuristics: A survey of the state of the art. *Journal of the Operational Research Society*, 64: 1695–1724.

Burke, E.; Hyde, M.; Kendall, G.; Ochoa, G.; and Özcan, E. 2010a. *A classification of hyper-heuristic approaches*, 449–468. ISBN 9783319910857.

Burke, E. K.; Hyde, M.; Kendall, G.; and Woodward, J. 2010b. A genetic programming hyper-heuristic approach for evolving 2-D strip packing heuristics. *IEEE Transactions on Evolutionary Computation*, 14(6): 942–958.

Burke, E. K.; Hyde, M. R.; Kendall, G.; Ochoa, G.; Ozcan, E.; and Woodward, J. R. 2009. Exploring hyper-heuristic methodologies with genetic programming. In *Computational intelligence: Collaboration, fusion and emergence*, 177–201. Springer.

Burke, E. K.; Hyde, M. R.; Kendall, G.; Ochoa, G.; Özcan, E.; and Woodward, J. R. 2018. A classification of hyper-heuristic approaches: revisited. In *Handbook of metaheuristics*, 453–477. Springer.

Cheng, P.; Hu, T.; Xu, H.; Zhang, Z.; Yuan, Z.; Dai, Y.; Han, L.; Du, N.; and Li, X. 2025. Self-playing Adversarial Language Game Enhances LLM Reasoning. arXiv:2404.10642.

Coulom, R. 2007a. Computing Elo Ratings of Move Patterns in the Game of Go. *ICGA Journal*, 30.

Coulom, R. 2007b. Computing "elo ratings" of move patterns in the game of go. *ICGA journal*, 30(4): 198–208.

Dam, T.; Chalvatzaki, G.; Peters, J.; and Pajarinen, J. 2022. Monte-carlo robot path planning. *IEEE Robotics and Automation Letters*, 7(4): 11213–11220.

Dat, P. V. T.; Doan, L.; and Binh, H. T. T. 2025. Hsevo: Elevating automatic heuristic design with diversity-driven harmony search and genetic algorithm using llms. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, 26931–26938.

Desale, S.; Rasool, A.; Andhale, S.; and Rane, P. 2015. Heuristic and meta-heuristic algorithms and their relevance to the real world: a survey. *Int. J. Comput. Eng. Res. Trends*, 351(5): 2349–7084.

Dorigo, M.; Birattari, M.; and Stutzle, T. 2007. Ant colony optimization. *IEEE computational intelligence magazine*, 1(4): 28–39.

Dorigo, M.; and Gambardella, L. 1997. Ant colony system: a cooperative learning approach to the traveling salesman problem. *IEEE Transactions on Evolutionary Computation*, 1(1): 53–66.

Dorigo, M.; Maniezzo, V.; and Colorni, A. 1996. Ant System: Optimization by a colony of cooperating agents. IEEE Trans Syst Man Cybernetics - Part B. *IEEE transactions on systems, man, and cybernetics. Part B, Cybernetics : a publication of the IEEE Systems, Man, and Cybernetics Society*, 26: 29–41.

Dorigo, M.; and Stützle, T. 2018. Ant colony optimization: overview and recent advances. *Handbook of metaheuristics*, 311–351.

Fu, Y.; Peng, H.; Khot, T.; and Lapata, M. 2023a. Improving language model negotiation with self-play and in-context learning from ai feedback. *arXiv preprint arXiv:2305.10142*.

Fu, Z.-H.; Sun, S.; Ren, J.; Yu, T.; Zhang, H.; Liu, Y.; Huang, L.; Yan, X.; and Lu, P. 2023b. A Hierarchical Destroy and Repair Approach for Solving Very Large-Scale Travelling Salesman Problem. arXiv:2308.04639.

Hudson, B.; Li, Q.; Malencia, M.; and Prorok, A. 2022. Graph Neural Network Guided Local Search for the Traveling Salesperson Problem. arXiv:2110.05291.

Kong, A.; Zhao, S.; Chen, H.; Li, Q.; Qin, Y.; Sun, R.; Zhou, X.; Wang, E.; and Dong, X. 2024. Better Zero-Shot Reasoning with Role-Play Prompting. arXiv:2308.07702.

Koza, J. R. 1994. Genetic programming as a means for programming computers by natural selection. *Statistics and computing*, 4(2): 87–112.

Langdon, W. B.; and Poli, R. 2013. *Foundations of genetic programming*. Springer Science & Business Media.

Li, K.; Liu, F.; Wang, Z.; and Zhang, Q. 2025. Destroy and Repair Using Hyper Graphs for Routing. arXiv:2502.16170.

Liu, F.; Tong, X.; Yuan, M.; Lin, X.; Luo, F.; Wang, Z.; Lu, Z.; and Zhang, Q. 2024. Evolution of heuristics: towards efficient automatic algorithm design using large language model. In *Proceedings of the 41st International Conference on Machine Learning*, 32201–32223.

Liu, N. F.; Lin, K.; Hewitt, J.; Paranjape, A.; Bevilacqua, M.; Petroni, F.; and Liang, P. 2023. Lost in the Middle: How Language Models Use Long Contexts. arXiv:2307.03172.

Ng, A. Y.; Harada, D.; and Russell, S. 1999. Policy invariance under reward transformations: Theory and application to reward shaping. In *Icml*, volume 99, 278–287. Citeseer.

Phan Duc, H.; Bui Trong, D.; Nguyen Thi, T.; and Huynh Thi Thanh, B. 2025. Pareto Front Grid Guided Multiobjective Optimization In Dynamic Pickup And Delivery Problem Considering Two-Sided Fairness. In *Proceedings of the Genetic and Evolutionary Computation Conference*, 277–285.

Pillay, N.; and Qu, R. 2018. *Hyper-heuristics: theory and applications*. Springer.

Rajendran, C. 1993. Heuristic algorithm for scheduling in a flowshop to minimize total flowtime. *International Journal of Production Economics*, 29(1): 65–73.

Romera-Paredes, B.; Barekatain, M.; Novikov, A.; Balog, M.; Kumar, M. P.; Dupont, E.; Ruiz, F. J.; Ellenberg, J. S.; Wang, P.; Fawzi, O.; et al. 2024. Mathematical discoveries from program search with large language models. *Nature*, 625(7995): 468–475.

Shaw, P. 1997. A new local search algorithm providing high quality solutions to vehicle routing problems. *APES Group, Dept of Computer Science, University of Strathclyde, Glasgow, Scotland, UK*, 46.

Shaw, P. 1998. Using constraint programming and local search methods to solve vehicle routing problems. In *International conference on principles and practice of constraint programming*, 417–431. Springer.

Shi, W.-W.; Han, W.; and Si, W.-C. 2012. A hybrid genetic algorithm based on harmony search and its improving. In *Informatics and Management Science I*, 101–109. Springer.

Shinn, N.; Cassano, F.; Gopinath, A.; Narasimhan, K.; and Yao, S. 2023. Reflexion: Language agents with verbal reinforcement learning. *Advances in Neural Information Processing Systems*, 36: 8634–8652.

Silver, D.; Huang, A.; Maddison, C. J.; Guez, A.; Sifre, L.; Van Den Driessche, G.; Schrittwieser, J.; Antonoglou, I.; Panneershelvam, V.; Lanctot, M.; et al. 2016. Mastering the game of Go with deep neural networks and tree search. *nature*, 529(7587): 484–489.

Sui, J.; Ding, S.; Xia, B.; Liu, R.; and Bu, D. 2024. NeuralGLS: learning to guide local search with graph convolutional network for the traveling salesman problem. *Neural Computing and Applications*, 36(17): 9687–9706.

Tan, C. S.; Mohd-Mokhtar, R.; and Arshad, M. R. 2021. A comprehensive review of coverage path planning in robotics using classical and heuristic algorithms. *IEEE Access*, 9: 119310–119342.

Tran, C. D.; Nguyen-Tri, Q.; Binh, H. T. T.; and Thanh-Tung, H. 2025. Large language models powered neural solvers for generalized vehicle routing problems. In *Towards Agentic AI for Science: Hypothesis Generation, Comprehension, Quantification, and Validation*.

Voudouris, C.; and Tsang, E. 1999. Guided local search and its application to the traveling salesman problem. *European Journal of Operational Research*, 113(2): 469–499.

Wang, P.; Li, J.; Tang, Z.; Gui, H.; et al. 2025. Improving Rationality in the Reasoning Process of Language Models through Self-playing Game. In *Forty-second International Conference on Machine Learning*.

Wang, Y.; Le, H.; Gotmare, A. D.; Bui, N. D. Q.; Li, J.; and Hoi, S. C. H. 2023. CodeT5+: Open Code Large Language Models for Code Understanding and Generation. arXiv:2305.07922.

Wei, J.; Wang, X.; Schuurmans, D.; Bosma, M.; Xia, F.; Chi, E.; Le, Q. V.; Zhou, D.; et al. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35: 24824–24837.

Wu, Y.; Song, W.; Cao, Z.; and Zhang, J. 2021. Learning Large Neighborhood Search Policy for Integer Programming. arXiv:2111.03466.

Xin, H.; Ren, Z.; Song, J.; Shao, Z.; Zhao, W.; Wang, H.; Liu, B.; Zhang, L.; Lu, X.; Du, Q.; et al. 2025. DeepSeek-Prover-V1. 5: Harnessing Proof Assistant Feedback for Reinforcement Learning and Monte-Carlo Tree Search. In *The Thirteenth International Conference on Learning Representations*.

Yao, S.; Yu, D.; Zhao, J.; Shafran, I.; Griffiths, T. L.; Cao, Y.; and Narasimhan, K. 2023. Tree of Thoughts: Deliberate Problem Solving with Large Language Models. arXiv:2305.10601.

Ye, H.; Wang, J.; Cao, Z.; Berto, F.; Hua, C.; Kim, H.; Park, J.; and Song, G. 2024. Reevo: Large language models as hyper-heuristics with reflective evolution. *Advances in neural information processing systems*, 37: 43571–43608.

Ye, H.; Wang, J.; Cao, Z.; Liang, H.; and Li, Y. 2023. DeepACO: Neural-enhanced Ant Systems for Combinatorial Optimization. arXiv:2309.14032.

Zhang, F.; Mei, Y.; Nguyen, S.; and Zhang, M. 2020. Evolving scheduling heuristics via genetic programming with feature selection in dynamic flexible job-shop scheduling. *ieee transactions on cybernetics*, 51(4): 1797–1811.

Zhang, F.; Mei, Y.; Nguyen, S.; and Zhang, M. 2023. Survey on genetic programming and machine learning techniques for heuristic design in job shop scheduling. *IEEE Transactions on Evolutionary Computation*, 28(1): 147–167.

Zhang, F.; Mei, Y.; Nguyen, S.; Zhang, M.; and Tan, K. C. 2021. Surrogate-assisted evolutionary multitask genetic programming for dynamic flexible job shop scheduling. *IEEE Transactions on Evolutionary Computation*, 25(4): 651–665.

Zheng, Z.; Xie, Z.; Wang, Z.; and Hooi, B. 2025. Monte Carlo Tree Search for Comprehensive Exploration in LLM-Based Automatic Heuristic Design. In *Forty-second International Conference on Machine Learning*.