MO-EoH: Multi-Objectivization Evolution of Heuristics with Large Language Models

Anonymous authors
Paper under double-blind review

Abstract

In recent years, Automated Heuristic Design (AHD) based on Large Language Models (LLMs) has achieved significant success. However, existing single-objective optimization methods use only the optimal performance on the target problem as the sole guidance signal during evolution, making it difficult to accurately assess the performance of heuristics and making the search prone to getting trapped in local optima. To address this issue, we introduce helper objectives into the original heuristic design objective, thereby transforming the Single-Objective Optimization Problem (SOP) into a Multi-Objective Optimization Problem (MOP), while the ultimate goal remains to discover better solutions to the original SOP. This process is referred to as multi-objectivization. Building on this idea, we propose Multi-Objectivization Evolution of Heuristics (MO-EoH), the first framework to introduce multi-objectivization into LLM-based AHD. Furthermore, we design a Lexicographic-Pareto mechanism that prioritizes the original objective of heuristic design, thereby improving search efficiency. Experiments across eight classical planning benchmark domains demonstrate that MO-EoH consistently outperforms existing state-of-the-art LLM-based methods, achieving an average 24.5% improvement in original objective.

1 Introduction

Heuristic methods have long served as important tools for solving a wide range of optimization and decision-making tasks (Pearl, 1984), and have been widely applied in various fields, including transportation (Dahmani et al., 2024; Pereira et al., 2021), production scheduling (Mihoubi et al., 2021; Palacio et al., 2022), and robotics (Ghallab et al., 2004). Traditional heuristic functions are typically designed by human experts. This process relies heavily on extensive expert knowledge and practical experience, and involves labor-intensive iterative trial-and-error and manual design, making it costly. To reduce the cost of manually customizing heuristic methods for different tasks, researchers have proposed Automated Heuristic Design (AHD), which aims to automatically configure, select, and combine heuristic methods suitable for different problems.

In recent years, the combination of Large Language Models (LLMs) (Huang et al., 2023; Zelikman et al., 2024) with evolutionary computation (EC) for automated heuristic design has achieved significant progress (Dat et al., 2025; Ye et al., 2024; Liu et al., 2024; Zhang et al., 2024; Zheng et al., 2025; Ma et al., 2024). This paradigm first leverages the in-context learning and reasoning capabilities of LLMs to generate heuristic functions, which are then evaluated through a predefined evaluation framework (Yao et al., 2025). The heuristics with superior objective performance are subsequently retained and fed back to the LLM for further iterative evolution, thereby forming an evolutionary loop driven by objective performance feedback(Liu et al., 2026). Using this paradigm for heuristic design significantly reduces the need for manual design while still achieving competitive performance.

As shown in Table 1, existing single-objective optimization methods evaluate heuristic functions solely based on their optimal performance on the target problem. This can mainly lead to the following two issues: First, existing methods cannot accurately assess the performance limitations of heuristic functions, which may lead the evolutionary search of LLMs in misguided directions, thereby wasting substantial computational resources (Zhang et al., 2024). Second, although some individuals may exhibit significant advantages in specific performance dimensions beyond the orig-

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.

Table 1: Comparison of Different Optimization Paradigms for LLM-based AHD.

Opt. Paradigm	Initial Problem	Optimization Formulation	Final Purpose
Single-objective (**EoH**)	Single-objective problem	Optimize the single objective	Obtain the optimum of the single objective
Multi-objective (**MEoH**)	Multi-objective problem	Jointly optimize multiple objectives	Obtain Pareto-optimal solutions
Multi-objectivization (**MO-EoH**)	Single-objective problem	Reformulate as a multi-objective problem for optimization	Obtain a high-quality solution for the original objective

inal objective, they are likely to be prematurely eliminated. However, such distinctive advantages can help the entire population escape local optima (Yao et al., 2025).

A common approach to addressing the above issues is multi-objectivization (Ma et al., 2023; Drugan & Thierens, 2013; Sushil et al., 2018; Hel, 2011; Tanaka et al., 2024), which reformulates the original single-objective optimization problem into a multi-objective optimization problem and solves the transformed multi-objective problem to obtain high-quality solutions for the original problem. Therefore, it is natural to introduce this idea into LLM-based automated heuristic design. In this study, we propose the first LLM-based multi-objectivized heuristic design framework, termed Multi-Objectivization Evolution of Heuristics (MO-EoH). Specifically, MO-EoH extends the objective of heuristic design from the original single objective, e.g., the number of solved tasks, to a multi-objective formulation that additionally considers several helper objectives (Hel, 2011; Tanaka et al., 2024), e.g., search time and expanded nodes. Finally, MO-EoH selects the heuristic function that achieves the best performance on the original objective as the final output. This multi-objectivization framework broadens the evaluation dimensions of heuristic search and retains heuristics with different potential advantages, thereby providing more opportunities to obtain high-quality heuristics.

Our main contributions are summarized as follows: 1) We propose MO-EoH, the first framework to introduce multi-objectivization into LLM-based automatic heuristic design, thereby enabling more efficient design of high-performance heuristics for the original single optimization objective. 2) We design a Lexicographic-Pareto mechanism that prioritizes the original objective of heuristic design, thereby improving search efficiency. 3) Experiments across eight classical planning benchmark domains were conducted in comparison with existing LLM-based, traditional, and neural methods. Our proposed MO-EoH is able to discover heuristics that outperform those of existing state-of-the-art LLM-based methods, achieving a 24.5% improvement in the original objective.

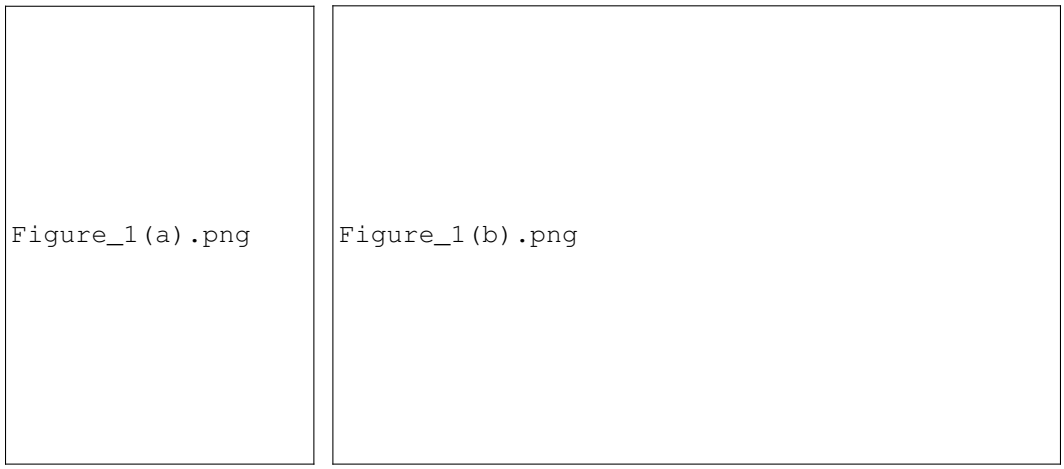
2 Background

2.1 Automated Heuristic Design

Automated heuristic algorithm design, often referred to as hyper-heuristics, aims to automatically perform algorithm configuration, selection and combination for specific problems (Pillay & Qu, 2021), thereby reducing the cost of manual design while improving solution efficiency and generalization ability. Among these approaches, genetic programming (Sun et al., 2020; Alfaro-Fernández et al., 2020) generates new heuristic methods by searching over algorithmic structures, providing an interpretable approach to automated algorithm design. Meanwhile, machine learning techniques(Bengio et al., 2021; He et al., 2021; Li et al., 2023) have also been widely applied to automated algorithm design by learning the relationships between problem characteristics and algorithm performance to assist in algorithm selection, configuration, and generation. However, existing methods still rely heavily on manually designed algorithmic components and domain knowledge, which not only increases the cost of algorithm design but also limits the degree of automation.

2.2 LLM-based AHD

In recent years, the combination of large language models with evolutionary computation for automated heuristic design has achieved significant progress (Dat et al., 2025; Liu et al., 2024; Zhang et al., 2024; Zheng et al., 2025; Ma et al., 2024). FunSearch (Romera-Paredes et al., 2023) and



(a) Existing Methods (b) MO-EoH

Figure 1: **a) Existing LLM-based AHD methods** solely optimize the original single objective. **b) MO-EoH** introduces multi-objectivization into LLM-based AHD and employs a Lexicographic-Pareto mechanism to jointly consider the original and helper objectives, ultimately guiding the search toward better solutions to the original objective.

EoH (Liu et al., 2024) are pioneering works in this direction. FunSearch adopts an island model and uses a single prompt to guide LLMs in searching for heuristics, while EoH proposes to evolve both thoughts and codes simultaneously to improve the efficiency of heuristic search. Subsequently, ReEvo (Ye et al., 2024) introduces the concept of language hyper-heuristics and further enhances performance by incorporating both short-term and long-term reflection mechanisms. MEoH (Yao et al., 2025) formulates heuristic evolution as a multi-objective Pareto optimization problem. Unlike the aforementioned works, we introduce a multi-objectivization strategy into the LLM-based automated heuristic design framework, with the goal of designing stronger heuristics for the single original objective.

2.3 MULTI-OBJECTIVIZATION

In recent years, multi-objectivization (Ma et al., 2023; Drugan & Thierens, 2013; Sushil et al., 2018; Hel, 2011; Tanaka et al., 2024) has gradually emerged as a promising paradigm for solving SOPs in evolutionary computation. Its core idea is to reformulate the original SOP as an MOP and then leverage evolutionary algorithms to solve the reformulated problem, thereby obtaining optimal solutions to the original problem. Knowles et al. first proposed the multi-objectivization paradigm and demonstrated its impact on search performance (Knowles et al., 2001). Subsequently, Jensen transformed an SOP into a multi-objectivized problem by introducing a helper objective that differs from the original objective function (Jensen, 2004). Lochtefeld and Ciarallo made the first attempt to investigate the effects of different helper-objective sequences (Lochtefeld & Ciarallo, 2015). A two-step multi-objectivization method was proposed for scheduling problems (Syberfeldt & Rogström, 2018). Inspired by these advances, we seek to introduce the idea of multi-objectivization into an LLM-based AHD framework. This approach can reshape the search structure of the original language search space, reduce the risk of the search process becoming trapped in local optima, and further improve the search efficiency and final performance of LLM-based AHD.

3 MO-EOH

3.1 FRAMEWORK

We propose a framework named Multi-Objectivization Evolution of Heuristics (MO-EoH), which integrates multi-objectivization (Ma et al., 2023; Drugan & Thierens, 2013; Sushil et al., 2018) with LLM-based automated heuristic design to design high-quality heuristics for the original single-

objective problem (Holland, 1975; Bäck et al., 2000). As shown in Figure 1, existing single-objective LLM-based AHD methods typically evaluate heuristic individuals solely based on their performance on the target problem, resulting in an evolutionary loop driven by single-objective performance feedback. In contrast, MO-EoH adopts a multi-objectivization strategy to transform the single-objective LLM-based AHD problem (Dat et al., 2025; Ye et al., 2024; Liu et al., 2024) into a multi-objective LLM-based AHD problem (Yao et al., 2025), thereby facilitating a more effective search for high-quality heuristics for the original objective. Importantly, a key advantage of multi-objectivization is its flexibility in defining helper objectives according to the characteristics of the problem. Therefore, both the original objective and the helper objectives can be flexibly selected and adjusted depending on the specific problem being solved.

The MO-EoH method procedure is as follows:

1. **Multi-objectivization:** Before the evolutionary process begins, MO-EoH first transforms the original single-objective AHD problem into a multi-objective formulation. Thus, each heuristic individual h is evaluated according to a multi-objective vector:

$$\mathbf{F}(h) = (f_1(h), f_2(h), \ldots, f_m(h)), \tag{1}$$

   where $f_1(h)$ denotes the original objective, while $f_2(h), \ldots, f_m(h)$ represent a set of helper objectives. The resulting multi-objective formulation provides the basis for the subsequent population initialization and evolutionary search.

2. **Initialization:** MO-EoH starts from an initial population P_0 consisting of n heuristic individuals $\{h_1, \ldots, h_n\}$. The heuristics are generated by repeatedly prompting LLMs using an initialization operator $Init$:

$$h_i = Init(LLM, p_i). \tag{2}$$

   Here, p_i is the prompt for initialization, which includes the task description and an example heuristic. It instructs the LLM to generate both a natural language description and the corresponding heuristic function implementation in code. Due to space limitations, the complete prompt details used in MO-EoH are provided in the Appendix.

3. **Evolutionary Cycle:** After the initialization is completed, MO-EoH iteratively performs the following steps:

   (a) **Evolutionary Search.** Two strategies are alternately used to generate new heuristics based on the current population:

      i. **Local Optimization Search (LOS):**

$$h_o = LOS(LLM, p_{los}, h_p). \tag{3}$$

      Here, p_{los} is the prompt for local optimization search, and h_p is a parent heuristic selected from the population. In each iteration, a single parent heuristic h_p is selected from the population primarily based on its performance on the original objective.

      ii. **Performance-Guided Search (PGS):**

$$h_o = PGS(LLM, p_{pgs}, (h_{p1}, h_{p2})). \tag{4}$$

      Here, p_{pgs} is the prompt for performance-guided search, and h_{p1} and h_{p2} are two parent heuristics selected from the population based on their complementary performance across multiple objectives.

   (b) **Population Management.** After generating K new individuals through evolutionary search, these new individuals replace existing individuals in the population through a population management mechanism, thereby maintaining a stable population size while preserving population diversity.

4. **Termination:** The evolutionary process terminates when a predefined stopping criterion is met, specifically reaching the maximum number of iterations T_{max}. The method outputs the best heuristic discovered across all iterations.

3.2 LEXICOGRAPHIC-PARETO MECHANISM

Traditional multi-objectivization methods lack an effective means of distinguishing the priorities among different objectives. They typically incorporate the original task objective and helper objectives into a unified Pareto optimization framework, which may allow helper objectives to exert excessive influence on the performance of the original task. To address this, we propose a ***Lexicographic-Pareto mechanism (LPM)***, which prioritizes the original objective while further leveraging the performance information provided by the helper objectives. Specifically, the proposed mechanism first treats the original objective, e.g., task-solving capability, as the highest-priority evaluation criterion. The heuristic functions are grouped according to their ability to solve the single original objective, thereby partitioning the population into multiple heuristic sets. Within each heuristic set with the same or similar original-objective solving capability, helper objectives, e.g., search time and the number of expanded nodes, are further considered. Subsequently, the heuristic functions are subjected to non-dominated sorting based on their Pareto dominance relationships with respect to the helper objectives, forming multiple non-dominated layers to further distinguish heuristic functions with the same performance on the single original objective.

Through the above Lexicographic-Pareto mechanism, we obtain a hierarchical organization and non-dominated sorting of heuristic functions that jointly account for the priority of task-solving capability and the differences in helper-objective performance within each level. The resulting ranking provides a unified priority criterion for subsequent parent selection and population management.

3.3 EVOLUTIONARY SEARCH

During the evolutionary cycle, heuristic generation is primarily driven by two strategies: local optimization search and performance-guided search. Each strategy defines two types of operators to guide the search process, namely l_1 and l_2 for local optimization search, and p_1 and p_2 for performance-guided search, enabling efficient exploration of new heuristics.

Local Optimization Search The target of local optimization search is to optimize a single-parent heuristic function with strong performance on the original objective, while exploring new local variants and maintaining population diversity. During parent selection, for the single-parent operators l_1 and l_2, a rank-weighted random selection method is employed to select one parent based on the original objective. Heuristic functions with better performance on the original objective have a higher probability of being selected. The prompt strategy for l_1 is to design an improved heuristic based on the existing one, refining or extending the original design without completely discarding it. For l_2, the model is instructed to design a new heuristic that differs from the existing one as much as possible in order to explore novel ideas and behaviors.

Performance-Guided Search The objective of performance-guided search is to improve heuristic functions by leveraging heuristics with complementary performance across multiple dimensions, thereby further enhancing the overall performance of the population. Benefiting from the multi-objectivization strategy, heuristic functions can be directly and comprehensively evaluated across multiple dimensions, providing an effective basis for identifying heuristics with complementary advantages. For the two-parent operators p_1 and p_2, the LPM first identifies the set with the highest original-objective performance. Within this set, a Pareto front is constructed based on the helper objectives. Two parent individuals are then selected based on this Pareto front. Specifically, a rank-weighted random selection method is separately applied to different helper objectives to select one parent individual for each objective. For p_1, the prompt strategy is to generate a new heuristic function that differs as much as possible from the selected parent heuristics, so as to explore novel combinations while leveraging their complementary performance advantages. For p_2, the goal is to preserve the common ideas of the selected parent heuristics while introducing new elements to improve performance and leveraging their complementary performance advantages.

3.4 POPULATION MANAGEMENT

As newly generated heuristics are continuously added, the population size gradually increases. To maintain a predefined population size, we employ a population management strategy based on the proposed Lexicographic-Pareto mechanism. When the population has not yet reached its capacity,

qualified newly generated heuristic functions are directly added to the population. Once the population reaches its capacity, the Lexicographic-Pareto mechanism is first used to identify the set of heuristics with the worst performance on the original target problem, followed by non-dominated sorting of this set and the identification of its worst-performing non-dominated layer. Subsequently, the heuristic individual to be removed is selected from this layer based on the crowding mechanism and removed to complete the population update.

4 EXPERIMENTS

4.1 EXPERIMENTAL SETTINGS

Benchmarks. We consider eight classical planning domains, namely Blocksworld, Childsnack, Floortile, Miconic, Rovers, Sokoban, Spanner, and Transport, all of which are taken from the IPC 2023 Learning Track (Taitler et al., 2024). Different resource constraints are imposed during training and testing: the maximum runtime and memory limits are 5 seconds and 1 GB during training, and 30 seconds and 8 GB during testing. In addition, each domain contains 16 training tasks and 90 test tasks. The main parameters and corresponding ranges of the training and testing tasks for each domain are summarized in Table 2. Detailed descriptions are provided in the Appendix. When multiple candidate heuristics achieve the same performance on the original objective (Solved Tasks) across the training tasks, we select the heuristic with the shortest search time for evaluation on the test tasks.

Table 2: Size of training and testing tasks for each domain based on their main parameters.

Domain	Training	Testing
Blocksworld	$n \in [10, 14]$	$n \in [5, 488]$
Childsnack	$c \in [4, 5]$	$c \in [4, 40]$
Floortile	$t \in [4, 12]$	$t \in [9, 952]$
Miconic	$p \in [6, 8]$	$p \in [1, 485]$
Rovers	$r \in [1, 2]$	$r \in [1, 10]$
Sokoban	$b \in [1, 2]$	$b \in [1, 79]$
Spanner	$s \in [6, 8]$	$s \in [1, 487]$
Transport	$v \in [1, 3]$	$v \in [3, 50]$

Objectives. **1) *Solved Tasks*:** We use the number of solved tasks as the original objective to measure the effectiveness of the designed heuristics in achieving the original objective of solving classical planning tasks. **2) *Search Time*:** We use search time as a helper objective to reflect the search efficiency of the designed heuristics. **3) *Expanded Nodes*:** We use the number of expanded nodes as a helper objective to reflect the informativeness of the designed heuristics. Classical planning involves helper objectives that are closely related to the original objective, which is one of the important reasons why we choose classical planning as the evaluation setting to validate the effectiveness of our proposed multi-objectivization method, MO-EoH.

Baselines. ***1) Traditional methods:*** we consider a set of classical domain-independent heuristics, including the blind heuristic h^0, maximum heuristic $h^{\max}$ (Bonet & Geffner, 2001), landmark-count heuristic h^{lmc} (Büchner et al., 2023; Richter et al., 2008), set-additive heuristic h^{sa} (Bonet & Geffner, 2001; Helmert, 2006b), FF heuristic h^{FF} (Hoffmann & Nebel, 2001), and additive heuristic h^{add} (Bonet & Geffner, 2001). These heuristics are widely used manually designed approaches in classical planning and serve as strong baselines for evaluating heuristic search performance. ***2) Neural methods:*** we additionally include $h^{\text{WLF}}_{\text{GPR}}$ (Chen et al., 2024) as a baseline. It is considered a state-of-the-art method for heuristic learning in classical planning and is implemented on top of Fast Downward. ***3) LLM-based methods:*** LHG [1] (Corrêa et al., 2025), FunSearch (Romera-Paredes et al., 2023), EoH (Liu et al., 2024), and ReEvo (Ye et al., 2024), which represent recent advances in large language model-based algorithm discovery.

[1]Since the original paper does not assign a specific name to this method, we refer to it as LHG in the remainder of this paper.

Table 3: Comparison of successfully solved tasks of LLM-based methods using GPT-4o-mini across eight classical planning domains.

	GPT-4o-mini				
Domain	LHG	FunSearch	EoH	ReEvo	MO-EoH
Blocksworld	28.00	27.33	35.33	31.33	**50.33**
Childsnack	13.00	16.00	21.00	20.33	**30.67**
Floortile	1.00	1.00	1.33	**1.67**	**1.67**
Miconic	62.33	29.00	53.00	53.33	**66.33**
Rovers	26.67	16.67	**28.33**	12.00	**28.33**
Sokoban	21.00	20.00	22.67	20.00	**23.33**
Spanner	50.00	34.33	53.33	43.33	**62.00**
Transport	29.33	6.00	21.33	15.67	**31.67**
Total	231.33	150.33	236.33	197.66	**294.33**

Implementation Details. MO-EoH uses a population size of 20. All LLM-based methods generate domain-specific heuristics by prompting large language models. To mitigate the impact of randomness in large language models, all LLM-based methods are independently executed three times, and the final results are reported as the average over these runs. Each run consists of 300 fixed iterations. In terms of LLM usage, we adopt GPT-4o-mini as the default configuration and apply it uniformly across all LLM-based methods to ensure a fair comparison. To further investigate the impact of model capability, in addition to the default GPT-4o-mini, MO-EoH also introduces GPT-3.5-turbo and a stronger model, GPT-5.1. The temperature parameter for all language models is set to 1. We assume unit action costs and adopt satisficing planning, where any plan that reaches the goal is acceptable. Unless explicitly stated otherwise, all tasks are solved using the Greedy Best-First Search (GBFS) (Helmert, 2006a) algorithm based on the Pyperplan (Alkhazraji et al., 2020) planner. All experiments are conducted on a unified hardware platform equipped with an AMD Ryzen 9 7945HX processor (2.50 GHz).

Table 4: Comparison of traditional heuristics, Neural heuristics, and MO-EoH using GPT-5.1 across eight classical planning domains.

	Traditional Heuristics						Neural	GPT-5.1
Domain	h^0	$h^{\max}$	h^{lmc}	h^{sa}	h^{FF}	h^{add}	$h^{\text{WLF}}_{\text{GPR}}$	MO-EoH
Blocksworld	5	9	11	13	14	26	34	**53.33**
Childsnack	5	5	5	10	10	9	36	**39.67**
Floortile	1	5	4	4	6	**10**	2	2.00
Miconic	27	21	30	45	47	54	**83**	76.33
Rovers	10	10	18	24	24	21	31	**33.67**
Sokoban	20	23	21	27	26	26	29	**31.67**
Spanner	30	30	30	30	30	30	58	**58.33**
Transport	6	8	8	17	17	30	41	**49.00**
Total	104	111	127	170	174	206	314	**344.00**

4.2 Results Analysis

Comparison with LLM-based methods. Table 3 summarizes the performance of MO-EoH and LLM-based methods, all implemented using GPT-4o-mini, across eight classical planning domains. Among all LLM-based methods, including LHG (Corrêa et al., 2025), FunSearch (Romera-Paredes et al., 2023), EoH (Liu et al., 2024), and ReEvo (Ye et al., 2024), MO-EoH achieves the best or tied-best performance across all eight domains, significantly outperforming existing LLM-based methods. Specifically, MO-EoH successfully solves 294.33 out of the 720 test tasks, solving 58 more tasks than the second-best method, EoH, which solves 236.33 tasks, corresponding to an im-



Figure 2: Comparison of different methods in terms of the two helper objectives.

provement of approximately 24.5%. In the Blocksworld, Childsnack, and Spanner domains, MO-EoH solves 15, 9.67, and 8.67 more tasks, respectively, than the strongest baseline in each domain, with EoH being the strongest baseline in all three cases. Moreover, in the Miconic domain, MO-EoH solves 4 more tasks than the best-performing baseline, LHG. These results demonstrate that introducing a multi-objective formulation into single-objective LLM-based AHD can significantly improve search efficiency.

Comparison with Traditional and Neural Methods. Table 4 compares the performance of MO-EoH using GPT-5.1 with that of traditional handcrafted heuristics and neural methods. Overall, MO-EoH using GPT-5.1 generates heuristics that substantially outperform traditional handcrafted heuristics, solving 138 more tasks than the strongest handcrafted heuristic, h_{add}. Although the neural method $h^{\mathrm{GPR}}_{\mathrm{WLF}}$ (Chen et al., 2024) also demonstrates strong performance, MO-EoH based on GPT-5.1 still solves 30 more tasks. These results further demonstrate that LLM-based multi-objective evolutionary search can generate high-quality heuristics with strong problem-solving performance.

4.3 DISCUSSION

Helper Objectives. Figure 2 compares MO-EoH based on GPT-4o-mini with four other methods from the perspective of helper objectives, specifically search time and the number of expanded nodes. The comparison methods include LHG and EoH based on GPT-4o-mini, MO-EoH based on GPT-5.1, and the traditional method h^{add}. In the figure, the red regions indicate cases where the comparison method performs better, while the blue regions indicate cases where MO-EoH based on GPT-4o-mini performs better. When either method fails to find a valid solution, the corresponding result is placed on the boundary of the plot. Overall, compared with the three baseline methods other than MO-EoH based on GPT-5.1, MO-EoH based on GPT-4o-mini demonstrates significantly better overall per-

Table 5: Comparison of MO-EoH using different large language models.

Domain	3.5-turbo	4o-mini	GPT-5.1
Blocksworld	30.33	50.33	**53.33**
Childsnack	25.67	30.67	**39.67**
Floortile	**2.00**	1.67	**2.00**
Miconic	68.67	66.33	**76.33**
Rovers	24.00	28.33	**33.67**
Sokoban	22.67	23.33	**31.67**
Spanner	61.67	**62.00**	58.33
Transport	34.67	31.67	**49.00**
Total	269.67	294.33	**344.00**

formance. Furthermore, as expected, MO-EoH based on GPT-5.1 consistently outperforms MO-EoH based on GPT-4o-mini across all evaluated metrics. This result suggests that leveraging a more capable underlying LLM can further improve the search performance of MO-EoH.

Different LLMs. As shown in Table 5, the comparison of different language models reveals a clear trend: in general, stronger models tend to generate higher-quality heuristics and achieve better overall performance. Specifically, across most domains, the progression from GPT-3.5-turbo to GPT-4o-mini and then to GPT-5.1 consistently leads to improved results, indicating that enhanced reasoning and generation capabilities translate directly into more effective heuristic design. However, an exception is observed in the Spanner domain, where GPT-5.1 performs slightly worse than both GPT-3.5-turbo and GPT-4o-mini. We speculate that this may be because the training instances in this domain are relatively simple, causing the more expressive model to overfit to superficial patterns, which in turn reduces its generalization ability.

Impact of operators. We evaluate the contribution of each operator by measuring the performance of MO-EoH after removing a specific operator (Table 6). The results show that all operators contribute positively to the quality of the generated heuristics. The four operators l_1, l_2, p_1, and p_2 all play important roles—removing any one of them leads to a noticeable degradation in the performance of the resulting heuristics.

Table 6: The ablation results of MO-EoH on the Blocksworld and Childsnack domains.

Method	Blocksworld	Childsnack
MO-EoH	**50.33**	**30.67**
w/o l_1	40.00	28.67
w/o l_2	45.33	28.00
w/o p_1	46.00	26.33
w/o p_2	43.00	29.67
Size 10	43.63	26.00
Size 30	41.33	23.00
w/o LPM	39.33	20.33

Population size. We investigate the impact of population size on the performance of the heuristics designed by MO-EoH by varying the population size (Table 6). The results show that a smaller population size limits the number of candidate solutions available for the LLM to reference when generating improved heuristics, thereby narrowing the search space and reducing the population's exploration capability. In contrast, an excessively large population size may cause useful information to become overly dispersed among candidates, making it difficult for the LLM to effectively focus on and leverage high-quality heuristics. Therefore, the population size of 20 adopted in this study achieves a good balance between exploration and exploitation.

Impact of Lexicographic-Pareto mechanism. We analyze the impact of the proposed Lexicographic-Pareto mechanism on the performance of MO-EoH by removing it and adopting the traditional Pareto-front-based selection combined with a crowding mechanism for population selection, as shown in Table 6. The results show that removing the Lexicographic-Pareto mechanism leads to a significant degradation in the overall performance of the heuristics. This indicates that the traditional Pareto front and crowding mechanism cannot effectively distinguish the priorities between the original objective and the helper objectives, potentially allowing the helper objectives to exert excessive influence on the performance of the original task.

5 Conclusion

We propose a framework named Multi-Objectivization Evolution of Heuristics, which integrates multi-objectivization with LLM-based AHD to improve heuristic design for the original single-objective problem. By introducing helper objectives, MO-EoH transforms single-objective AHD into multi-objective AHD, providing richer search information and improving optimization of the original objective. We also acknowledge certain limitations of MO-EoH, such as the selection of helper objectives. Different problems may require different helper objectives; therefore, how to design and select effective helper objectives according to the characteristics of a given problem remains an important direction for future research. Moreover, although the current study primarily focuses on classical planning problems, MO-EoH has the potential to be extended to a broader range of optimization problems, which we leave for future work.

6 LLM USAGE

During the preparation of this manuscript, we used large language models as auxiliary tools for language editing, proofreading, and improving the clarity and readability of the manuscript. LLMs were also used, where applicable, to assist with minor coding and debugging tasks. The LLMs did not contribute to the formulation of the core research ideas, experimental design, scientific claims, or conclusions. All LLM-assisted content was carefully reviewed and verified by the authors. The authors take full responsibility for the accuracy, originality, and integrity of the submitted manuscript.

REFERENCES

Helper-objective optimization strategies for the job-shop scheduling problem. *Applied Soft Computing*, 11(6):4161–4174, 2011. doi: 10.1016/j.asoc.2011.03.007.

Pedro Alfaro-Fernández, Rubén Ruiz, Federico Pagnozzi, and Thomas Stützle. Automatic algorithm design for hybrid flowshop scheduling problems. *European Journal of Operational Research*, 282 (3):835–845, 2020. doi: 10.1016/j.ejor.2019.10.004.

Yusra Alkhazraji, Matthias Frorath, Markus Grützner, Malte Helmert, Thomas Liebetraut, Robert Mattmüller, Manuela Ortlieb, Jendrik Seipp, Tobias Springenberg, Philip Stahl, and Jan Wülfing. Pyperplan, March 2020. URL https://doi.org/10.5281/zenodo.3701399. Version v1.3.

Thomas Bäck, David B. Fogel, and Zbigniew Michalewicz (eds.). *Evolutionary Computation 1: Basic Algorithms and Operators*. Institute of Physics Publishing, 2000.

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. doi: 10.1016/j.ejor.2020.07.063.

Blai Bonet and Héctor Geffner. Planning as heuristic search. *Artificial Intelligence*, 129(1):5–33, 2001. URL https://doi.org/10.1016/S0004-3702(01)00108-4.

Clemens Büchner, Salomé Eriksson, Thomas Keller, and Malte Helmert. Landmark progression in heuristic search. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, pp. 70–79, 2023.

Dillon Z. Chen, Felipe W. Trevizan, and Sylvie Thiébaux. Return to tradition: Learning reliable heuristics with classical machine learning. In *Proceedings of the Thirty-Fourth International Conference on Automated Planning and Scheduling*, pp. 68–76. AAAI Press, 2024.

Augusto B. Corrêa, André G. Pereira, and Jendrik Seipp. Classical planning with llm-generated heuristics: Challenging the state of the art with python code. NeurIPS 2025, 2025. URL https://openreview.net/forum?id=UCV21BsuqA. NeurIPS 2025 Poster Track.

Nadia Dahmani, Ines Sbai, Takwa Tlili, and Saoussen Krichen. On solving the 2l-cvrp using an adaptive chemical reaction algorithm: postal transportation real-case. *International Journal of System Assurance Engineering and Management*, 2024. doi: 10.1007/s13198-024-02452-2.

Pham Vu Tuan Dat, Long Doan, and Huynh Thi Thanh Binh. Hsevo: Elevating automatic heuristic design with diversity-driven harmony search and genetic algorithm using llms. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pp. 26931–26938, 2025. URL https://arxiv.org/abs/2409.XXXX.

Madalina M. Drugan and Dirk Thierens. Multiobjectivization with nsga-ii on the noiseless bbob testbed. In *Proceedings of the 15th Annual Conference Companion on Genetic and Evolutionary Computation*, pp. 851–858. ACM, 2013. doi: 10.1145/2464576.2482700.

Malik Ghallab, Dana Nau, and Paolo Traverso. *Automated planning. Theory & practice*. 05 2004. ISBN 978-1-55860-856-6.

Xin He, Kaiyong Zhao, and Xiaowen Chu. Automl: A survey of the state-of-the-art. *Knowledge-Based Systems*, 212:106622, 2021. doi: 10.1016/j.knosys.2020.106622.

Malte Helmert. The fast downward planning system. *Journal of Artificial Intelligence Research*, 26: 191–246, 2006a.

Malte Helmert. The fast downward planning system. *Journal of Artificial Intelligence Research*, 26: 191–246, 2006b. URL https://doi.org/10.1613/jair.1705.

J. Hoffmann and Bernhard Nebel. The ff planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research*, 14:253–302, 05 2001.

John H. Holland. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, 1975.

Jiaxin Huang, Shixiang Gu, Le Hou, Yuexin Wu, Xuezhi Wang, Hongkun Yu, and Jiawei Han. Large language models can self-improve. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 1051–1068. Association for Computational Linguistics, 2023. doi: 10.18653/v1/2023.emnlp-main.67.

Mikkel T. Jensen. Helper-objectives: Using multi-objective evolutionary algorithms for single-objective optimisation. *Journal of Mathematical Modelling and Algorithms*, 3(4):323–347, 2004. doi: 10.1023/B:JMMA.0000049422.81900.26.

Joshua D. Knowles, Richard A. Watson, and David W. Corne. Reducing local optima in single-objective problems by multi-objectivization. In *Evolutionary Multi-Criterion Optimization (EMO 2001)*, volume 1993 of *Lecture Notes in Computer Science*, pp. 269–283. Springer, 2001. doi: 10.1007/3-540-44719-9_19.

Nan Li, Lianbo Ma, Guo Yu, Bing Xue, Mengjie Zhang, and Yaochu Jin. Survey on evolutionary deep learning: Principles, algorithms, applications, and open issues. *ACM Computing Surveys*, 56(2):1–34, 2023. doi: 10.1145/3603704.

Fei Liu, Xialiang Tong, Mingxuan Yuan, Xi Lin, Fu Luo, Zhenkun Wang, Zhichao Lu, and Qingfu Zhang. Evolution of heuristics: Towards efficient automatic algorithm design using large language model. In *Proceedings of the 41st International Conference on Machine Learning (ICML)*, volume 235 of *Proceedings of Machine Learning Research*, pp. 32201–32223. PMLR, 2024. URL https://proceedings.mlr.press/v235/liu24bs.html.

Fei Liu, Yilu Liu, Qingfu Zhang, Xialiang Tong, and Mingxuan Yuan. Eoh-s: Evolution of heuristic set using llms for automated heuristic design. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pp. 37090–37098, 2026. URL https://ojs.aaai.org/index.php/AAAI/article/view/41038.

Darrell F. Lochtefeld and Frank W. Ciarallo. Multi-objectivization via decomposition: An analysis of helper-objectives and complete decomposition. *European Journal of Operational Research*, 243(2):395–404, 2015. doi: 10.1016/j.ejor.2014.11.041.

Xiaoliang Ma, Zhitao Huang, Xiaodong Li, Yutao Qi, Lei Wang, and Zexuan Zhu. Multiobjectivization of single-objective optimization in evolutionary computation: A survey. *IEEE Transactions on Cybernetics*, 53(6):3702–3715, 2023. doi: 10.1109/TCYB.2021.3120788.

Yecheng Jason Ma, William Liang, Guanzhi Wang, De-An Huang, Osbert Bastani, Dinesh Jayaraman, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Eureka: Human-level reward design via coding large language models. ICLR 2024, 2024. URL https://openreview.net/forum?id=IEduRUO55F.

Bachir Mihoubi, Brahim Bouzouia, and Mehdi Gaham. Reactive scheduling approach for solving a realistic flexible job shop scheduling problem. *International Journal of Production Research*, 59 (19):5790–5808, 2021. doi: 10.1080/00207543.2020.1790686.

Jessica Coto Palacio, Yailen Martinez Jimenez, Leander Schietgat, Bart Van Doninck, and Ann Nowe. A q-learning algorithm for flexible job shop scheduling in a real-world manufacturing scenario. *Procedia CIRP*, 106:227–232, 2022. doi: 10.1016/j.procir.2022.02.183.

Judea Pearl. *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. Addison-Wesley Series in Artificial Intelligence. Addison-Wesley Publishing Company, 1984. ISBN 978-0201055948. URL https://books.google.com/books?id=1HpQAAAAMAAJ.

Rafael H. M. Pereira, Marcus Saraiva, Daniel Herszenhut, Carlos Kaue Vieira Braga, and Matthew Wigginton Conway. r5r: Rapid realistic routing on multimodal transport networks with R^5 in r. *Findings*, 2021. doi: 10.32866/001c.21262.

Nelishia Pillay and Rong Qu (eds.). *Automated Design of Machine Learning and Search Algorithms*. Natural Computing Series. Springer, Cham, 2021. ISBN 978-3-030-72069-8. doi: 10.1007/978-3-030-72069-8.

Silvia Richter, Malte Helmert, and Matthias Westphal. Landmarks revisited. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, pp. 975–982, 2008.

Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Matej Balog, M. Pawan Kumar, Emilien Dupont, Francisco J. R. Ruiz, Jordan Ellenberg, Pengming Wang, Omar Fawzi, Pushmeet Kohli, and Alhussein Fawzi. Mathematical discoveries from program search with large language models. *Nature*, 2023.

Yanan Sun, Bing Xue, Mengjie Zhang, Gary G. Yen, and Jiancheng Lv. Automatically designing cnn architectures using the genetic algorithm for image classification. *IEEE Transactions on Cybernetics*, 50(9):3840–3854, 2020. doi: 10.1109/TCYB.2020.2983860.

Kumar Sushil et al. A penalty-based multi-objectivization approach for single objective optimization. *Information Sciences*, 467:543–562, 2018. doi: 10.1016/j.ins.2018.02.034.

Anna Syberfeldt and Joel Rogström. A two-step multi-objectivization method for improved evolutionary optimization of industrial problems. *Applied Soft Computing*, 64:331–340, 2018. doi: 10.1016/j.asoc.2017.12.027.

Ayal Taitler, Ron Alford, Joan Espasa, Gregor Behnke, Daniel Fišer, Michael Gimelfarb, Florian Pommerening, Scott Sanner, Enrico Scala, Dominik Schreiber, Javier Segovia-Aguas, and Jendrik Seipp. The 2023 international planning competition. *AI Magazine*, 45(2):280–296, 2024. URL https://doi.org/10.1002/aaai.12169.

Shoichiro Tanaka, Arnaud Liefooghe, Keiki Takadama, and Hiroyuki Sato. Designing helper objectives in multi-objectivization. In *2024 IEEE Congress on Evolutionary Computation (CEC)*, 2024. doi: 10.1109/CEC60901.2024.10612125.

Shunyu Yao, Fei Liu, Xi Lin, Zhichao Lu, Zhenkun Wang, and Qingfu Zhang. Multi-objective evolution of heuristic using large language model. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pp. 27144–27152, 2025. URL https://arxiv.org/abs/2409.16867.

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. NeurIPS 2024, 2024. URL https://github.com/ai4co/reevo.

Eric Zelikman, Eliana Lorch, Lester Mackey, and Adam Tauman Kalai. Self-taught optimizer (stop): Recursively self-improving code generation. In *Conference on Language Modeling*, 2024.

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

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

A Detailed Benchmark Description

- **Blocksworld Domain**: A classical planning domain in which a robotic arm rearranges blocks on a table from an initial to a target configuration. A block can be manipulated only when it is clear, and the arm can hold at most one block at a time. The main actions are ***pick-up***, ***put-down***, ***stack***, and ***unstack***, which allow the arm to move and rearrange blocks.
- **Childsnack Domain**: Involves preparing and delivering sandwiches to children at different locations. Sandwiches are prepared in the kitchen from available bread and content portions, placed on trays, and delivered to the children. Children with gluten allergies must receive gluten-free sandwiches. The main actions include ***make-sandwich***, ***put-on-tray***, ***move-tray***, and ***serve-sandwich***, which together enable sandwich preparation, transportation, and delivery.
- **Floortile Domain**: Involves robots painting a grid of floor tiles according to a target color pattern. Robots can move between adjacent unpainted tiles, change the color of their spray gun, and paint tiles above or below their current position. The main actions are ***move***, ***change-color***, ***paint-up***, and ***paint-down***. Once painted, a tile can no longer be occupied by a robot.
- **Miconic Domain**: Involves an elevator transporting passengers from their origin floors to their destination floors. Passengers can board only when the elevator is at their origin and can leave at their destination. The main actions are ***board***, ***up***, ***down***, and ***depart***, which coordinate elevator movement and passenger transportation.
- **Rovers Domain**: Involves autonomous rovers exploring a planetary surface represented by connected waypoints. Rovers can navigate, collect soil and rock samples, and acquire images of designated objectives. Cameras must be calibrated before imaging, and collected data must eventually be communicated to a lander. Key actions include ***navigate***, ***sample-soil***, ***sample-rock***, ***calibrate***, ***take-image***, and ***communicate***.
- **Sokoban Domain**: Involves navigating an agent through a warehouse to push boxes onto designated goal locations. The agent can move horizontally or vertically and push boxes when the destination cell is free, but boxes cannot be pulled. The main actions are ***move*** and ***push***. Since pushing can be irreversible, inappropriate actions may create deadlocks that prevent the goals from being reached.
- **Spanner Domain**: Involves an agent moving through a linear corridor from a shed to a gate while collecting spanners and tightening nuts. The agent can move only forward, and each spanner can be used only once before it breaks. The key actions are ***walk***, ***pick-up-spanner***, and ***tighten***, requiring careful allocation of consumable spanners.
- **Transport Domain**: Involves vehicles delivering packages across a directed road network. Each vehicle has limited carrying capacity, which is consumed when packages are picked up and released when they are dropped off. The main actions are ***drive***, ***pick-up***, and ***drop***, requiring the planner to jointly determine vehicle routes and package assignments while respecting capacity constraints.

B MO-EoH procedure

Algorithm 1: MO-EoH Method Procedure

Require: LLM, population size n, maximum iterations T_{max}
Ensure: Best heuristic h^*

1: Multi-objectivize the original single-objective AHD problem into a multi-objective formulation
2: Define evaluation vector $F(h) = (f_1(h), f_2(h), \ldots, f_m(h))$, where f_1 is the original objective and $f_2, \ldots, f_m$ are helper objectives
3: Initialize population $P_0 = \{h_1, \ldots, h_n\}$
4: **for** $i = 1$ to n **do**
5: $\quad h_i \leftarrow Init(LLM, p_i)$
6: **end for**
7: **for** $t = 1$ to T_{max} **do**
8: $\quad$ Initialize offspring set $Q_t \leftarrow \emptyset$

for $k = 1$ to K **do**
  Select evolutionary strategy from {LOS, PGS}
  if LOS is selected **then**
    Select one parent heuristic h_p from P_t based on the original objective
    using rank-weighted random selection
    $h_o \leftarrow LOS(LLM, p_{los}, h_p)$
  else
    Construct the Pareto front using the Lexicographic-Pareto mechanism
    Select two parent heuristics h_{p1}, h_{p2} from the Pareto front based on
    helper objectives using rank-weighted random selection
    $h_o \leftarrow PGS(LLM, p_{pgs}, (h_{p1}, h_{p2}))$
  end if
  $Q_t \leftarrow Q_t \cup \{h_o\}$
end for
$P_{t+1} \leftarrow PopulationManagement(P_t, Q_t)$
end for
return best heuristic h^* in population

C PROMPTS

Figure_3.png

Figure 3: Prompt template for MO-EoH in the Blocksworld domain.

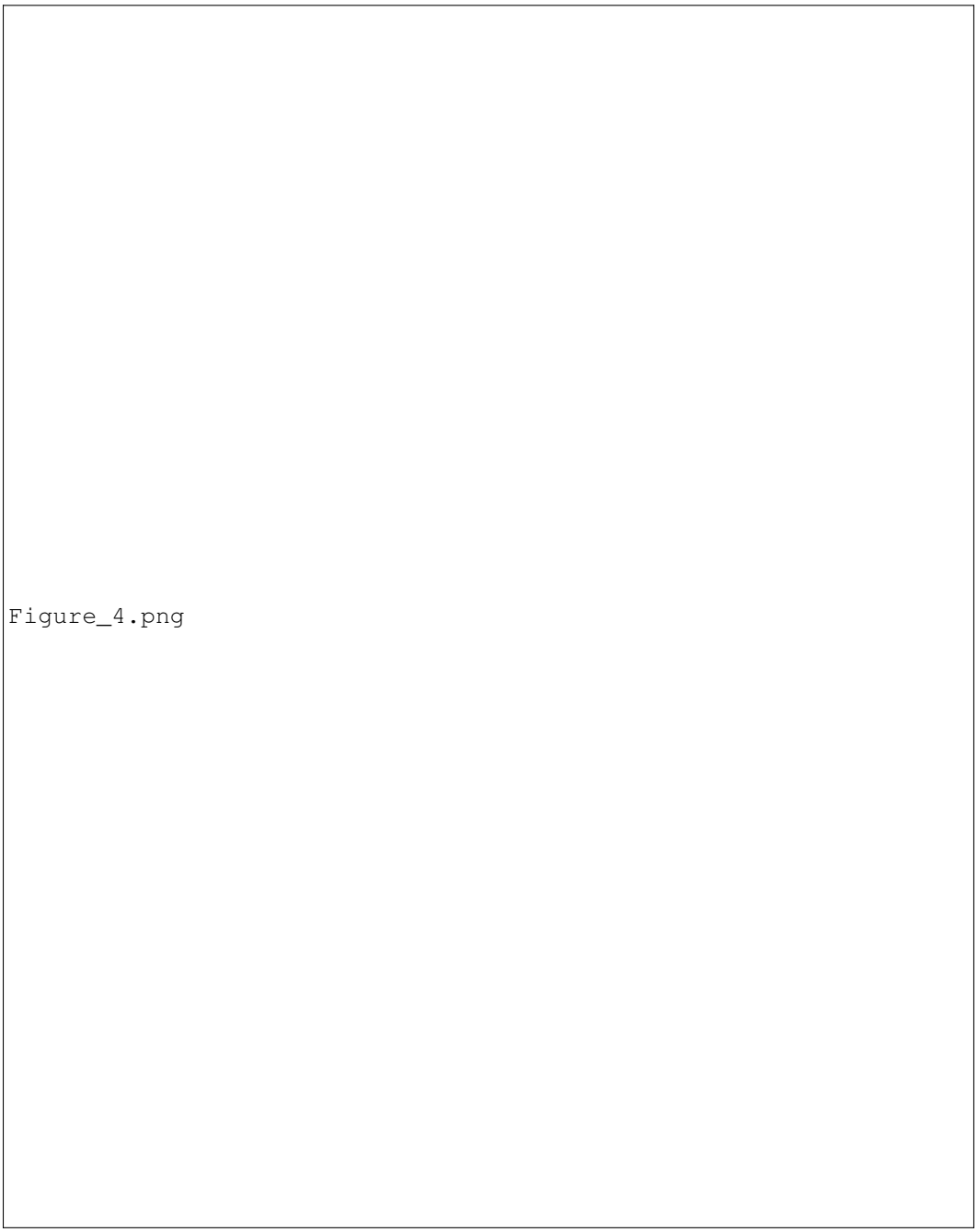


Figure 4: Operator instructions for MO-EoH.

D EXAMPLES OF LLM-GENERATED HEURISTICS

Algorithm 2: A High-Quality LLM-Generated Heuristic for the Blocksworld Domain

```
from heuristics.heuristic_base import Heuristic


def _parse_fact(fact):
    """
    Convert a fact string like '(on a b)' into ('on', ['a', 'b']).
    """
```

```
    parts = fact[1:-1].split()
    return parts[0], parts[1:]


class BlocksworldHeuristic(Heuristic):
    def __init__(self, task):
        # Store goal facts
        self.goals = set(task.goals)
        self.static_facts = task.static

        # Goal relations
        self.goal_on = {}
        self.goal_above = {}
        self.goal_on_table = set()
        self.goal_clear = set()
        self.goal_holding = None
        self.goal_arm_empty = False

        for fact in self.goals:
            pred, args = _parse_fact(fact)
            if pred == "on":
                b, under = args
                self.goal_on[b] = under
                self.goal_above[under] = b
            elif pred == "on-table":
                b = args[0]
                self.goal_on[b] = None
                self.goal_on_table.add(b)
            elif pred == "clear":
                self.goal_clear.add(args[0])
            elif pred == "holding":
                self.goal_holding = args[0]
            elif pred == "arm-empty":
                self.goal_arm_empty = True

        # All blocks that appear in goal conditions
        # (location or clearness or holding)
        all_goal_blocks = (
            set(self.goal_on.keys())
            | set(self.goal_on_table)
            | set(self.goal_above.keys())
            | set(self.goal_clear)
        )

        if self.goal_holding is not None:
            all_goal_blocks.add(self.goal_holding)

        self.goal_blocks = all_goal_blocks

        # Precompute goal towers (bottom to top) for faster evaluation
        self.goal_towers = []
        visited = set()

        for b in list(self.goal_on.keys()) + list(self.goal_on_table):
            # Find base of the tower containing b
            base = b
            while base in self.goal_on and self.goal_on[base] is not None:
                base = self.goal_on[base]

            if base in visited:
                continue

            visited.add(base)

            # Build tower from base upward using goal_above
            tower = [base]
```

```
            cur = base

            while cur in self.goal_above:
                nxt = self.goal_above[cur]
                tower.append(nxt)
                cur = nxt

            self.goal_towers.append(tower)

    def __call__(self, node):
        state = set(node.state)

        # Current structural relations
        current_on = {}
        current_under = {}
        current_on_table = set()
        current_clear = set()
        current_holding = None
        arm_empty = False

        for fact in state:
            pred, args = _parse_fact(fact)

            if pred == "on":
                b, under = args
                current_on[b] = under
                current_under[under] = b
            elif pred == "on-table":
                current_on_table.add(args[0])
            elif pred == "clear":
                current_clear.add(args[0])
            elif pred == "holding":
                current_holding = args[0]
            elif pred == "arm-empty":
                arm_empty = True

        total_cost = 0

        # Helper: compute height above a block in current state
        def height_above(block):
            h = 0
            cur = current_under.get(block)

            while cur is not None:
                h += 1
                cur = current_under.get(cur)

            return h

        # 1. Misplaced blocks: wrong support/table relative to goal,
        #    with stronger penalty if deeply buried.
        for b in self.goal_blocks:
            if b in self.goal_on:
                desired_support = self.goal_on[b]

                if desired_support is None:
                    # Should be on table
                    if b not in current_on_table:
                        if b not in current_clear:
                            h = height_above(b)
                            total_cost += 2 * h
                            total_cost += 2

                        total_cost += 2

                else:
```

```
                # Should be on another block
                if current_on.get(b) != desired_support:
                    if b not in current_clear:
                        h = height_above(b)
                        total_cost += 2 * h
                        total_cost += 2

                    total_cost += 2

            elif b in self.goal_on_table:
                if b not in current_on_table:
                    if b not in current_clear:
                        h = height_above(b)
                        total_cost += 2 * h
                        total_cost += 2

                    total_cost += 2

        # 2. Goal clear requirements
        for b in self.goal_clear:
            if b not in current_clear:
                total_cost += 2

        # 3. Hand status consistency
        if self.goal_arm_empty:
            if not arm_empty:
                held = current_holding
                penalty = 1

                if held is not None:
                    if held in self.goal_on:
                        desired_support = self.goal_on[held]

                        if desired_support is None and held in current_on_table:
                            penalty += 2
                        elif (
                            desired_support is not None
                            and current_on.get(held) == desired_support
                        ):
                            penalty += 2

                total_cost += penalty

        elif self.goal_holding is not None:
            g = self.goal_holding

            if current_holding != g:
                if current_holding is not None and current_holding != g:
                    total_cost += 1
                    arm_empty = True

                if g in current_on:
                    if g not in current_clear:
                        h = height_above(g)
                        total_cost += 2 * h
                        total_cost += 2

                    total_cost += 1

                elif g in current_on_table or g in self.goal_blocks:
                    if g not in current_clear:
                        h = height_above(g)
                        total_cost += 2 * h
                        total_cost += 2

                    total_cost += 1
```

```
        # 4. Obstruction penalty
        for support, correct_child in self.goal_above.items():
            cur_top = current_under.get(support)

            if cur_top is None:
                continue

            if cur_top != correct_child:
                depth = 0
                b = cur_top

                while b is not None:
                    depth += 1
                    b = current_under.get(b)

                total_cost += 2 * depth

        # 5. Tower alignment
        for tower in self.goal_towers:
            base = tower[0]
            correct_prefix = 0

            if base in self.goal_on_table:
                if base in current_on_table:
                    correct_prefix = 1
            else:
                if (
                    base in self.goal_on
                    and current_on.get(base) == self.goal_on[base]
                ):
                    correct_prefix = 1

            for i in range(1, len(tower)):
                if correct_prefix != i:
                    break

                lower = tower[i - 1]
                upper = tower[i]

                if current_on.get(upper) == lower:
                    correct_prefix += 1
                else:
                    break

            misaligned = len(tower) - correct_prefix

            if misaligned > 0:
                total_cost += misaligned

            internal_misorder = 0
            goal_set = set(tower)

            for b in tower:
                sup = current_on.get(b)

                if (
                    sup in goal_set
                    and (
                        sup not in self.goal_on
                        or self.goal_on[b] != sup
                    )
                ):
                    internal_misorder += 1

            if internal_misorder > 0:
```

```
                total_cost += internal_misorder

        return total_cost


class DomainHeuristic(Heuristic):
    def __init__(self, task):
        self._inner = BlocksworldHeuristic(task)
        self.goals = self._inner.goals
        self.static_facts = self._inner.static_facts

    def __call__(self, node):
        return self._inner(node)
```

Algorithm 3: A High-Quality LLM-Generated Heuristic for the Childsnack Domain

```
from heuristics.heuristic_base import Heuristic


class ChildsnackHeuristic(Heuristic):
    def __init__(self, task):
        # Cache goals and static information
        self.goals = set(task.goals)
        self.static_facts = set(task.static)

        # Static child data
        self.static_child_place = {}       # child -> place
        self.static_allergic = set()
        self.static_non_allergic = set()

        for fact in self.static_facts:
            parts = fact[1:-1].split()
            if not parts:
                continue
            pred, *args = parts
            if pred == "waiting" and len(args) == 2:
                c, p = args
                self.static_child_place[c] = p
            elif pred == "allergic_gluten" and len(args) == 1:
                self.static_allergic.add(args[0])
            elif pred == "not_allergic_gluten" and len(args) == 1:
                self.static_non_allergic.add(args[0])

        # Goal children
        self.goal_children = set()
        for g in self.goals:
            parts = g[1:-1].split()
            if len(parts) == 2 and parts[0] == "served":
                self.goal_children.add(parts[1])

    def __call__(self, node):
        state = set(node.state)

        def parse(f):
            return f[1:-1].split()

        # ---------- Single pass over state ----------
        served_children = set()
        dyn_child_place = dict(self.static_child_place)
        dyn_allergic = set(self.static_allergic)
        dyn_non_allergic = set(self.static_non_allergic)

        tray_pos = {}                  # tray -> place
        ontray = {}                    # sandwich -> tray
        kitchen_sandwich = set()
```

```
        gf_sandwich = set()

        bread = set()
        content = set()
        gf_bread_ids = set()
        gf_content_ids = set()
        creatable_sandwiches = set()

        for fact in state:
            parts = parse(fact)
            if not parts:
                continue
            pred, *args = parts

            # Children-related
            if pred == "served" and len(args) == 1:
                served_children.add(args[0])
            elif pred == "waiting" and len(args) == 2:
                c, p = args
                dyn_child_place[c] = p
            elif pred == "allergic_gluten" and len(args) == 1:
                dyn_allergic.add(args[0])
            elif pred == "not_allergic_gluten" and len(args) == 1:
                dyn_non_allergic.add(args[0])

            # Trays and sandwiches
            elif pred == "at" and len(args) == 2:
                t, p = args
                tray_pos[t] = p
            elif pred == "at_kitchen_sandwich" and len(args) == 1:
                kitchen_sandwich.add(args[0])
            elif pred == "no_gluten_sandwich" and len(args) == 1:
                gf_sandwich.add(args[0])
            elif pred == "ontray" and len(args) == 2:
                s, t = args
                ontray[s] = t

            # Ingredients
            elif pred == "at_kitchen_bread" and len(args) == 1:
                bread.add(args[0])
            elif pred == "at_kitchen_content" and len(args) == 1:
                content.add(args[0])
            elif pred == "no_gluten_bread" and len(args) == 1:
                gf_bread_ids.add(args[0])
            elif pred == "no_gluten_content" and len(args) == 1:
                gf_content_ids.add(args[0])

            # Sandwich slots
            elif pred == "notexist" and len(args) == 1:
                creatable_sandwiches.add(args[0])

        # ---------- Remaining children ----------
        remaining_children = [c for c in self.goal_children if c not in
        ↪  served_children]
        if not remaining_children:
            return 0

        rem_allergic = sum(1 for c in remaining_children if c in dyn_allergic)
        rem_total = len(remaining_children)
        rem_non_allergic = rem_total - rem_allergic

        # ---------- Count existing sandwiches ----------
        gf_ontray = 0
        n_ontray = 0
        for s in ontray:
            if s in gf_sandwich:
```

```
            gf_ontray += 1
        else:
            n_ontray += 1

    gf_kitchen = {s for s in kitchen_sandwich if s in gf_sandwich}
    n_kitchen = kitchen_sandwich - gf_kitchen

    gf_ready = gf_ontray + len(gf_kitchen)
    n_ready = n_ontray + len(n_kitchen)

    # ---------- Ingredient capacity ----------
    gf_bread = bread & gf_bread_ids
    gf_content = content & gf_content_ids

    max_pairs_any = min(len(bread), len(content), len(creatable_sandwiches))
    max_pairs_gf = min(len(gf_bread), len(gf_content),
    ↪  len(creatable_sandwiches))

    # ---------- Lower bound on serves ----------
    serve_lb = rem_total

    # ---------- Lower bound on required sandwiches (make actions) ----------
    # First satisfy GF demand
    miss_gf = max(0, rem_allergic - gf_ready)
    # Then overall demand
    total_ready = gf_ready + n_ready
    miss_total = max(0, rem_total - total_ready)
    miss_norm = max(0, miss_total - miss_gf)

    make_lb = 0
    if miss_gf > 0:
        make_lb += miss_gf
    if miss_norm > 0:
        make_lb += miss_norm

    # Capacity shortfalls add penalty (keeps heuristic informative in
    ↪  dead-like regions)
    if miss_gf > max_pairs_gf:
        make_lb += (miss_gf - max_pairs_gf)
    if miss_total > max_pairs_any:
        make_lb += (miss_total - max_pairs_any)

    # ---------- Lower bound on put_on_tray ----------
    trays_exist = len(tray_pos) > 0
    already_ontray = gf_ontray + n_ontray

    # Each child needs one sandwich; any that is not already on a tray must
    ↪  be put there
    need_from_non_ontray_or_new = max(0, rem_total - already_ontray)
    put_lb = need_from_non_ontray_or_new if trays_exist else 0

    # ---------- Lower bound on tray moves ----------
    move_lb = 0
    if trays_exist:
        # Distinct places with remaining children
        place_demand = {}
        for c in remaining_children:
            p = dyn_child_place.get(c, "kitchen")
            place_demand[p] = place_demand.get(p, 0) + 1

        distinct_child_places = set(place_demand.keys())
        tray_places = set(tray_pos.values())

        # At least one move for each place currently without a tray
        uncovered = distinct_child_places - tray_places
        if uncovered:
```

```
                move_lb += len(uncovered)

            # If no tray at kitchen but we need to load sandwiches from kitchen
            ↪  or newly made,
            # we need at least one round trip to/from kitchen.
            if "kitchen" not in tray_places:
                potential_supply = len(kitchen_sandwich) + max_pairs_any
                if need_from_non_ontray_or_new > 0 and potential_supply > 0:
                    move_lb = max(move_lb, 1)

        total_cost = serve_lb + make_lb + put_lb + move_lb
        if total_cost < 0:
            total_cost = 0
        return int(total_cost)


class DomainHeuristic(Heuristic):
    def __init__(self, task):
        self.goals = set(task.goals)
        self.static_facts = task.static
        self.inner = ChildsnackHeuristic(task)

    def __call__(self, node):
        state = set(node.state)
        total_cost = self.inner(node)
        return total_cost
```