Large Language Model for Efficient Algorithm Design of Combinatorial Optimization Solver

Yuyan Zhou
University of Science and Technology of China
Hefei Anhui, China
yuyanzhou@mail.ustc.edu.cn

Jie Wang*
University of Science and Technology of China
Hefei Anhui, China
jiewangx@ustc.edu.cn

***Abstract*—The algorithm design in exact combinatorial optimization (CO) solver plays a fundamental role in operations research. However, due to the extensive requirements on domain knowledge and the large search space for algorithm design, the refinement on these algorithms remains highly challenging for both manual and learning-based paradigms. To tackle this problem, we propose a novel machine learning framework—large language model for exact combinatorial optimization solver (LLM4Solver)—to efficiently design high-quality algorithms of the CO solvers. The core idea is that, instead of searching in the high-dimensional and discrete symbolic space from scratch, we can utilize the prior knowledge learned from large language models to directly search in the space of programming languages. Specifically, we first use an LLM as the generator for high-quality algorithms. Then, to efficiently explore the discrete and non-gradient algorithm space, we employ a derivative-free evolutionary framework as the algorithm optimizer. Experiments on extensive benchmarks show that the algorithms designed by LLM4Solver significantly outperform all the state-of-the-art (SOTA) human-designed and learning-based policies (on GPU) in terms of the solution quality, the solving efficiency, and the cross-benchmark generalization ability. The appealing features of LLM4Solver include 1) the high training efficiency to outperform SOTA methods within ten iterations, and 2) the high cross-benchmark generalization ability on heterogeneous benchmark (i.e. MIPLIB 2017). LLM4Solver shows the encouraging potential to efficiently design algorithms for the next generation of CO solvers.**

***Index Terms*—LLMs, Algorithm Design, Combinatorial Optimization.**

I. Introduction

Combinatorial optimization (CO), which aims to find an optimal object from a finite solution set, is one of the most fundamental models in operations research (OR) [1], [2]. It is widely used to formulate a series of important real-world tasks, e.g., scheduling, transportation, and management [3]–[6]. In these applications, the solving efficiency and the solution quality are usually related to enormous economic value [1], [7]. Thus, the algorithm design in exact CO solvers plays a fundamental role in the field of OR. Popular exact CO solvers like SCIP [8] and Gurobi [9] employ a rich set of hard-coded heuristic algorithms, whose efficacy directly affects the performance of the CO solvers.

Recently, there has been an explosive surge in the use of machine learning (ML) techniques to enhance exact CO solvers. These learning-based approaches can be roughly divided into two classes. One class incorporates deep neural networks (DNNs) to approximate different components in CO solvers, e.g., the branching [10], the cut selection [11], and the primal heuristics [12], [13]. Note that these DNN models fail to explain what patterns they have learned that accelerate the CO solvers [14], and thus they fail to help researchers further optimize the human-designed heuristics in solvers. To tackle this problem, the other class [14], [15] employs symbolic discovery approaches to learn more interpretable heuristics. Currently, these approaches are proven effective mainly on the branching component [1], in which symbolic regression (SR) is employed to learn interpretable scoring functions that outperform DNN policies on purely CPU-based devices.

Previous symbolic discovery approaches [14], [15] , though effective for branching, have two general limitations that severely hinder their potential applications to CO solvers. (1) Due to the high-dimensional and discrete search space for symbolic discovery and its nature of searching from scratch, existing SR-based approaches suffer from high computational costs. (2) The vastly different intrinsic nature of CO problems across various scenarios causes most approaches to fail in generalizing to various benchmarks. However, we hope that learning-based approaches will design more generic heuristics like human-designed ones to enhance the built-in performance of the solvers.

The powerful capabilities of the large language models (LLMs) in text comprehension and logic generation have attracted widespread attention [16], [17], offering new approaches for algorithm design. FunSearch [18] combines LLM with the island-based evolution for mathematical discovery. EoH [19] and Reevo [20] leverage LLM to revise heuristics for online bin packing, traveling salesman problems, and flow shop scheduling problems. AutoSAT [21] creates a multi-agent-based framework to improve the heuristics of SAT problems. These works have demonstrated impressive results in scenarios like mathematical discovery and designing heuristics for classical CO and SAT problems like online bin packing (OBP) and traveling salesman problems (TSP). However, while OBP and TSP are important CO problems, when modeled and solved using general MILP formulations in CO solvers, it is critical to investigate heuristics to find feasible solutions in more general MILP problems. Currently, there

* Corresponding author.

are no LLM-based heuristic optimization methods specifically designed for general MILP problems and research on such methods holds more generic scientific significance.

In this work, we propose an automatic algorithm design framework—large language model for exact combinatorial optimization solver (LLM4Solver)—to *efficiently* design high-quality diving heuristics of the CO solvers. Specifically, we first use the LLM as an algorithm generator, leveraging it on three operators: initialization, crossover, and mutation, to generate new executable algorithm code with the LLM's prior knowledge. Then, we treat the derivative-free evolutionary framework as an optimizer, utilizing it to iteratively optimize in the non-gradient algorithm space. Finally, we extend this framework through *multi-objective evolution* to leverage the heterogeneous characteristics of different CO problems to design more generic algorithms. Extensive experiments show that LLM4Solver-designed *interpretable* diving heuristics *significantly* outperform all the state-of-the-art (SOTA) human-designed and learning-based policies (on GPU) in terms of solution quality, solving efficiency, and cross-benchmark generalization ability.

We summarize the highly appealing features of LLM4Solver as follows. (1) High performance in terms of the solution quality (Table I) and the solving efficiency (Table II)[1]. LLM4Solver outperforms *all* the baselines, including both the human-designed heuristics in SCIP [8], the SOTA learning-based policy on GPU [12] and previous LLM-based methods [18], [19]. (2) Efficient searching. LLM4Solver outperforms the SOTA learning-based approaches within *only* four iterations and converges to optimum within ten iterations, respectively (Figure 2). (3) Strong cross-benchmark generalization ability. LLM4Solver can design generic heuristic algorithms with high *cross-benchmark* generalization ability on different benchmarks (Table III), including the highly challenging heterogeneous MIPLIB 2017 (Table IV). (4) Good interpretability. The programs with comments designed by LLM4Solver (Figure 3) clearly illustrate the execution logic of the algorithms, offering better interpretability compared to neural network parameters [12] and purely symbolic expressions [14]. LLM4Solver shows the potential to efficiently design high-quality and generic algorithms for the next generation of solvers, thereby enhancing their built-in capabilities.

II. Preliminaries

A. Exact Combinatorial Optimization Solvers and Diving Heuristic

In real-world scenarios, a series of CO problems can be modeled as Mixed Integer Linear Programmings (MILPs), taking the form of:

$$\arg\min_{\mathbf{x}}\{\mathbf{c}^\top\mathbf{x}|\mathbf{A}\mathbf{x}\leq\mathbf{b},\mathbf{l}\leq\mathbf{x}\leq\mathbf{u},x_j\in\mathbb{Z}\ \forall j\in\mathcal{I}\},$$

where $\mathbf{c}$ denotes the objective coefficient vector, $\mathbf{A}$ the constraint matrix, $\mathbf{b}$ the constraint right hand side vector, $\mathbf{l},\mathbf{u}$ respectively the lower and upper bounds and $\mathcal{I}$ denotes the index of integer variables. In exact solvers like SCIP [8], MILPs are solved with the branch-and-bound (B&B) algorithm. B&B recursively builds a search tree and expands the tree by selecting a variable x_i to partition the problem into two subproblems. Specifically, one adds constraint $x_i\leq\lfloor x_i^*\rfloor$ and the other adds $x_i\geq\lceil x_i^*\rceil$, where the x_i^* is the fractional value in the solution of the linear programming (LP) relaxation problem. Furthermore, B&B uses objective bounds to prune the tree and direct the exploration.

Primal heuristics help solvers obtain stronger primal bounds and improve solving efficiency. Among them, diving heuristics [1] is one of the most common primal heuristics and has a significant impact on the performance of solvers. They perform a depth-first search by iteratively rounding a variable and solving the modified LP relaxation problems until a feasible solution is found or infeasibility is proven. The scoring function s, formed as shown in Figure 3 and used to select the variables and rounding direction, is the most crucial part of diving heuristics.

B. Evolutionary Algorithms

Given a minimization problem $\arg\min_{v\in\mathcal{V}}h(v)$, evolutionary algorithms (EA) [22] take v as an individual and use *parent selection*, *crossover*, *mutation*, *fitness measure* and *survivor selection* operators to get better individuals. After iteration, EA outputs a population of feasible solutions to the problem.

Multi-objective evolutionary algorithms (MOEAs) [23] can implent on multi-objective minimization problem $\arg\min_{v\in\mathcal{V}}(h_1(v),h_2(v),...,h_m(v))$. For two solutions v,v' in a multi-objective minimization problem, we define that

- v **weakly dominates** v' (denoted as $v\preceq v'$) **iff.** $\forall 1\leq i\leq m,h_i(v)\leq h_i(v')$.
- v **dominates** v' **iff.** $v\preceq v'$ and $\exists 1\leq i\leq m,h_i(v)<h_i(v')$.

A feasible solution that any other solution cannot dominate is called the *Pareto optimal solution*. The set of all Pareto optimal solutions is called the *Pareto Front*. MOEAs leverage the evolution framework and output the Pareto Front of the multi-objective optimization problem.

C. Performance Measurement

It is common to measure the *primal-dual gap* as the solving performance, taking the form as:

$$\gamma_{pd}(\tilde{z},\tilde{z}^*)=\begin{cases}\frac{|\tilde{z}-\tilde{z}^*|}{\max(|\tilde{z}|,|\tilde{z}^*|)}, & if\ 0<\tilde{z}\tilde{z}^*<\infty,\\ 1, & else,\end{cases}$$

where $\tilde{z}$ is the primal bound given by the incumbent feasible solution $\tilde{x}$ and $\tilde{z}^*$ is the dual bound given by the optimal solution of LP relaxation problem.

Primal-dual integral Considering that the primal-dual gap is subject to the final solution and the time limit setting, a more intuitive way is measuring the variation of the primal-dual gap during the solving process, i.e. calculating the *primal-dual integral* along the time steps:

$$PD(T)=\int_{t=0}^{T}\gamma_{pd}(\tilde{z}_t,\tilde{z}_t^*)dt.$$

[1]The code is available at https://github.com/YuyanZhou/LLM4Solver.

Primal gap As the diving heuristics only aim to improve the primal performance, there is a necessity to introduce the *relative primal gap* to assess the effectiveness of diving heuristics, which is given by:

$$\gamma_p(\tilde{z}) = \frac{|\tilde{z} - z^\dagger|}{|z^\dagger|},$$

where $z^\dagger$ is the objective value of the optimal solution resolved. If $|z^\dagger| = 0$, we would use the *primal gap*:

$$\gamma'_p(\tilde{z}) = |\tilde{z} - z^\dagger|$$

III. Methods

Due to the large search space and lack of prior knowledge, previous manual and learning-based paradigms are inefficient for designing heuristics in CO solvers. Thus, we propose a novel framework —large language model for exact combinatorial optimization solver (LLM4Solver)—to efficiently design high-quality and generic diving heuristics. The core idea of LLM4Solver is that, instead of defining complex symbols and searching in the space of symbolic trees [14], we can *utilize the prior knowledge learned from large language models to conduct efficient searching directly at the program space*. We further extend this framework through multi-objective evolution to leverage the heterogeneous characteristics of different CO problems and design more generic heuristics.

Generally, as shown in Figure 1, LLM4Solver begins evolution with population initialization, iteratively optimizing the algorithms through the following steps: parent selection, crossover, mutation, fitness evaluation, and survivor selection. It leverages the prior knowledge of LLM to generate new algorithm candidates during initialization, crossover, and mutation (the bottom part of Figure 1). During fitness evaluation, single-objective evolution uses one fitness function, f_1, while multi-objective evolution considers multiple fitness functions, $f_1, f_2, \ldots, f_n$, to assess performance across various CO problems and design a more generic algorithm.

A. Individual Representation and Fitness Evaluation

As a population-based optimization strategy, each individual in the evolutionary process is represented by a diving score function s and the description of its logic. The fitness of each individual is assessed based on its solving performance in specific instances. During the evolutionary process, a population of N individuals is maintained to facilitate optimization.

Diving Score Function The diving score function s is the key decision-making component of the diving heuristic. It determines the next diving variable and the rounding direction, which directly impacts the solving performance. For a diving score function s with Python format, we employ a variable's 13 features (as shown in Figure 3) as input and output *score* (float, the score of the variable) and *roundup* (bool, **True** if round the variable up). These 13 features represent the union of all features used by the human-designed diving heuristics in SCIP. They are cheap to obtain, interpretable, and effectively describe the state of variables.

As previous methods based on neural networks [12] and symbolic discovery [14] do not consider the description of algorithm logic, we treat both the diving score function and its logical description as an individual like [19]. As shown in the bottom part of Figure 1, these logical descriptions provide a high-level idea for the corresponding algorithms, helping both the LLM and humans understand the algorithms. Through steps like crossover and mutation, LLM can combine and mutate these ideas to guide the generation of new algorithms.

Fitness Evaluation As primal heuristics aim to find better feasible solutions, we use the quality of solutions found by the diving heuristics as their fitness. Specifically, we embed s in SCIP, turn off the other heuristics, dive into the root node, and the fitness is

$$f(s) = mean(\gamma_p(\tilde{z}_1^s), \gamma_p(\tilde{z}_2^s), ..., \gamma_p(\tilde{z}_{N_{ins}}^s)), \quad (1)$$

where N_{ins} is the number of instances used for fitness evaluation, $\tilde{z}_k^s$ is the incumbent feasible solution found by s in the k-th instance, $\gamma_p(\tilde{z}_k^s)$ is the relative primal gap of s in the k-th instance. For a diving function s, the smaller the value of $f(s)$, the better s is.

B. Generating New Individuals with LLM

The text comprehension and logic generation abilities exhibited by LLMs closely align with the algorithm design. Therefore, we leverage the capabilities of LLM with prompt engineering to quickly generate new individuals in initialization, crossover and mutation steps.

Prompt Engineering As general LLM lacks sufficient information in specific domains, we need to provide more details of diving heuristics and specific instructions through prompt engineering. For the convenience of LLM's comprehension, we categorize the prompts into two groups: 1) the *background prompts* to provide the details about diving heuristics and 2) the *task-specific prompts* to instruct how to generate a new individual in initialization, crossover or mutation steps. Specifically, the background prompts consist of the introduction of MILP and diving heuristics and give the pseudo-code of diving for task-specific prompts. The task-specific prompts comprise the input features description, in-out format description, and task-specific instruction.

Initialization, Crossover and Mutation In the initialization step, we need to leverage the capabilities of LLM to generate an individual from scratch. We directly use initialization-specific prompts and the LLM to generate a new individual. After running N times, we can get the first population $P = \{s_1, s_2, ..., s_N\}$. In the crossover step, we need to recombine the advantages of the parent individuals to obtain a better offspring individual. Specifically, we provide l parent algorithms and recombine them with crossover-specific prompts to generate r offspring individuals. Then, we employ the LLM with mutation-specific prompts to mutate the offspring individuals generated by crossover, thereby exploring new individuals in the vicinity of the current one. We run the crossover and mutation for N times in each generation and get rN new individuals after that.

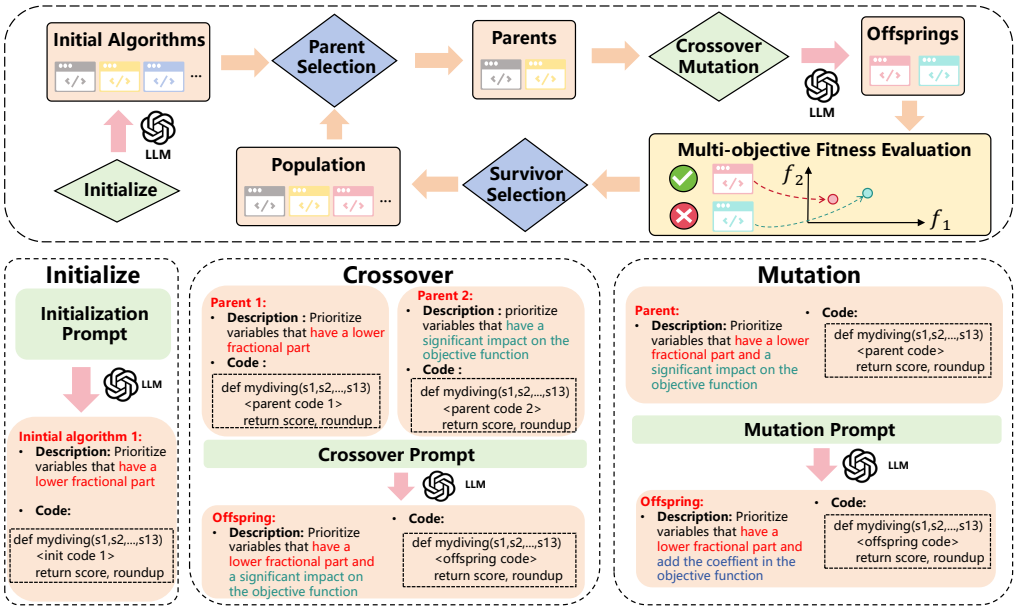

Fig. 1: Illustration of the automatic algorithm design framework LLM4Solver. The **top** flowchart outlines the evolutionary process of the algorithms. LLM4Solver leverages the prior knowledge of LLM to generate new algorithm candidates in the initialization, crossover, and mutation steps. In the fitness evaluation step, LLM4Solver utilizes the solving performance of the candidates on one or *multiple* CO problems for single- or multi-objective evolution. The three parts at the **bottom** give examples of initialization, crossover, and mutation with LLM.

Parent Selection Before crossover, we need to select l parent algorithms from the population. As parent selection needs to balance randomness and optimality, we adopt the *fitness proportional selection* [22] to determine the probability of each individual s_n being selected, i.e.

$$g(s_n) = \frac{\frac{1}{f(s_n)+eps}}{\sum_{k=1}^{N} \frac{1}{f(s_k)+eps}}, \quad n = 1, 2, ..., N, \tag{2}$$

where eps is a small number to avoid division by 0. It can be observed that the probability of better individuals being selected as parents is higher, which effectively balances randomness and optimality, allowing for exploration of a larger algorithm space.

C. Elitism Survivor Selection

The Survivor Selection Policy plays a vital role in determining which individuals are kept and which are removed from the next generation. It is essential to ensure that the best individuals are preserved within the population. Specifically, we employ *elitism survivor selection* [22] which always select the N individuals with best fitness from the total $(r+1)N$ ones after crossover and mutation. Elitism survivor selection can excellently ensure the optimality of individuals, which is helpful for efficiently finding high-quality diving heuristics.

D. Employing Multi-objective Evolution for Cross-benchmark Generalization

To design a unified algorithm with cross-benchmark generalization ability for the CO solver, we need to leverage the information of different benchmarks simultaneously. However, previous gradient-based work has only one optimization objective [10], which is the algorithm's performance on a single benchmark, making it difficult to simultaneously utilize information from multiple benchmarks. To tackle this problem, we treat the performance of the algorithm on different benchmarks as separate objectives and utilize multi-objective evolution to harness information from different benchmarks. Specifically, the differences between single-objective and multi-objective are in the *fitness evaluation*, *parent selection*, and *survivor selection* steps.

Fitness Evaluation of Multi-objective Evolution We treat the fitness of the diving heuristics on different benchmarks as different objectives. Specifically, the m-th objective is

$$f_m(s) = mean(\gamma_p(\tilde{z}^s_{m,1}), \gamma_p(\tilde{z}^s_{m,2}), ..., \gamma_p(\tilde{z}^s_{m,N_{ins\_m}})) \tag{3}$$

where $\tilde{z}^s_{m,k}$ is the incumbent feasible solution found by s in the k-th instance of the m-th benchmark.

Parent and Survivor Selection of Multi-objective Evolution We use the *binary tournament selection* [23] for parent selection. Specifically, we randomly choose 2 individuals from the population and select the better one as a parent. We use the *elitism survivor selection* for multi-objective evolution to select the best next generation. Since multi-objective optimization cannot simply compare two individuals using a single fitness function, we employ *Non-dominated Sorting* and *Crowding Distance Sorting* (see appendix for more details) to perform the comparisons.

IV. Experiments

We evaluate LLM4Solver through extensive experiments and benchmarks. These experiments aim to 1) show

Methods	Setcover	Cauctions	Facilities	Indset
LLM4Solver with SOEA	**3.36 (0.25)**	**1.83 (0.16)**	**0.65 (0.03)**	**0.84 (0.07)**
Best Human-designed	*6.99 (0.38)*	*3.00 (0.21)*	*2.17 (0.09)*	*4.91 (0.45)*
farkas	*6.99 (0.38)*	5.67 (0.29)	2.19 (0.09)	–
pseudocost	18.62 (1.47)	*3.00 (0.21)*	*2.17 (0.09)*	9.82 (0.49)
vectorlength	232.43 (3.47)	61.67 (0.55)	6.78 (0.31)	*4.91 (0.45)*
EoH	3.41 (0.27)	1.98 (0.33)	0.85 (0.07)	1.03 (0.12)
FunSearch	3.45 (0.33)	2.28 (0.26)	1.17 (0.09)	1.11 (0.07)
L2DIVE	3.58	2.60	0.71	1.37

TABLE I: The average relative primal gap with standard error of different diving heuristics. The results compare LLM4Solver with SOEA to human-designed, learning-based and LLM-based baselines to illustrate the superior quality of solutions found by the designed diving heuristics.

that LLM4Solver with single-objective evolution algorithm (SOEA) achieves better solution quality and higher solving and search efficiency; 2) illustrate that LLM4Solver with a multi-objective evolution algorithm (MOEA) designs heuristics with high cross-benchmark generalization ability; 3) show the interpretability of LLM4Solver.

A. Experimental Settings

Baselines There are 6 baselines corresponding to solution quality, including 3 human-designed diving heuristics (e.g. pseudocost diving) in the open-source solver SCIP [1], a learning-based GNN diving method L2DIVE [12] and 2 LLM-based algorithm design methods EoH [19] and FunSearch [18]. For solving efficiency, we compare our method with the default and tuned SCIP and L2DIVE. Specifically, we do not change any parameters for default SCIP, but we tune two important parameters (i.e. *freq* and *freqofs*) for tuned SCIP as a more comparable baseline.

Benchmarks The same as previous work [12], we employ four standard and two real-world benchmarks to compare the solution quality and solving efficiency. The four standard ones include set covering (Setcover), combinatorial auctions (Cauctions), capacitated facility location (Facilities), and maximum independent sets (Indset), and the two real-world ones include server load balancing in distributed computing (LoadBalance) [24] and neural network verification (NNVerify) [13]. We utilize the heterogeneous benchmark MIPLIB2017 containing 20 instances [25] to further demonstrate the cross-benchmark generation ability. These 20 instances ensure that at least one diving heuristic can find a feasible solution, facilitating the comparison of different diving heuristics.

Implementation Details (1) For the solution quality, Diving is solely implemented in the root node of each instance, with branching, cutting planes, and other primal heuristics disabled, emphasizing the quality of feasible solutions found by diving heuristics. For fitness evaluation in the evolution, we generate 50 instances for each benchmark. We validate and test the discovered diving heuristics on 100 instances each. (2) For the solving efficiency, we embed the discovered diving heuristics into the SCIP. We use the LoadBalance dataset [10] with 100 instances for validation and testing respectively. These instances cannot be solved within 3600 seconds, so we set a limit time $T_{limit} = 900$ seconds and measure its primal-dual integral $PD(T_{limit})$ as the solving performance. We use the NNVerify dataset with 50 instances for validation and 523 for testing and measure the solving time T. We use solution quality as the evaluation criterion and then utilize the primal-dual integral or solving time for validation and testing, selecting the diving heuristic that demonstrates the best performance in solving efficiency. (3) For LLM4Solver with multi-objective evolution algorithm (MOEA), we set the primal gap on four benchmarks as four objectives. Then, we chose the diving heuristic in the Pareto front with the highest average improvement ratio compared to the human-designed heuristic on four benchmarks as the output. We directly employ the designed diving heuristic on the MIPLIB instances to show the cross-benchmark generalization ability.

We use the GPT-4o-mini in our experiments. We run all the experiments with 3 random seeds on Intel(R) Xeon(R) CPU E5-2667 v4 @ 3.20GHz and NVIDIA GeForce RTX 2080 Ti.

	LoadBalance		NNVerify	
	Primal-dual Integral	Wins	Solving Time	Wins
Default SCIP	7340.7 (58.1)	0 (0.0)	76.9 (4.08)	44 (9.2)
Tuned SCIP	5445.7 (100.8)	1 (0.5)	65.8 (1.09)	103 (10.1)
LLM4Solver	**4543.0 (53.1)**	**99 (0.5)**	**52.9 (1.49)**	**376 (14.6)**

TABLE II: Compare LLM4Solver with SCIP to illustrate that it can improve the quality of solutions while leveraging better solutions to enhance the solving efficiency. We tune two important parameters (i.e. *freq* and *freqofs*) for tuned SCIP.

B. Results

Solution Quality We compare LLM4Solver to other baselines in Table I. Results show that LLM4Solver with single-objective evolution algorithm (SOEA) finds better feasible solutions than *all* human-designed and learning-based SOTA diving heuristics on four different problem classes. The results show that combining the prior knowledge of LLMs and evolutionary search is effective for designing new algorithms of CO solvers.

Solving Efficiency We compare LLM4Solver with default and tuned SCIP in Table II. Results show that LLM4Solver improves the solution quality while leveraging better solutions to enhance the solving efficiency. LLM4Solver improves the primal-dual integral by 38% (15%) on LoadBalance and

Methods	Setcover	Cauctions	Facilities	Indset
LLM4Solver + MOEA	4.01 (0.32)	2.49 (0.19)	1.30 (0.08)	1.28 (0.11)
LLM4Solver + Setcover	**3.36 (0.25)**	2.77 (0.21)	3.33 (0.18)	9.75 (0.49)
LLM4Solver + Cauctions	5.30 (0.36)	**1.83 (0.16)**	3.28 (0.20)	14.87 (0.48)
LLM4Solver + Facilities	6.86 (0.54)	11.37 (1.21)	**0.65 (0.03)**	5.93 (0.35)
LLM4Solver + Indset	70.96 (1.50)	11.52 (0.52)	3.18 (0.16)	**0.84 (0.07)**
LLM4Solver + Mean	4.57 (0.42)	2.54 (0.23)	1.37 (0.12)	1.38 (0.15)
EoH + Mean	4.65 (0.43)	2.65 (0.26)	1.48 (0.17)	1.38 (0.16)
FunSearch + Mean	4.78 (0.38)	2.71 (0.23)	1.44 (0.15)	1.41 (0.21)
Best Human-designed	6.99 (0.38)	3.00 (0.21)	2.17 (0.09)	4.91 (0.45)

TABLE III: The average relative primal gap of LLM4Solver with MOEA, LLM4Solver with SOEA, and human-designed diving heuristics. The results show that using multi-objective evolution can leverage the characteristics of different CO problems to achieve cross-benchmark generalization ability.

Heuristics	Primal Gap	Wins
Coefficient	1510	0/18
Distributional	4826	1/15
Farkas	1180	3/11
Fractional	1268	1/14
Linesearch	1589	1/16
Pseudocost	1242	4/15
Vectorlength	4803	2/17
LLM4Solver	**844**	**8/18**

TABLE IV: Compare LLM4Solver with human-designed diving heuristics to illustrate high generalization ability on the heterogeneous benchmark MIPLIB of 20 instances.

reduces the solving time by 31% (20%) on NNVerify over the default (tuned) settings of SCIP comparing that L2DIVE improves 35% (7%) on LoadBalance and 29% (20%) on NNVerify.

Generalization Ability of LLM4Solver with MOEA Table III compares algorithms across multiple CO problems, where "+ Setcover" and "+ Mean" denote training on a single benchmark and a joint average objective, respectively. The results indicate that: 1) Although the diving heuristics designed by LLM4Solver with SOEA perform well on their corresponding benchmarks, they struggle to generalize effectively to other benchmarks; 2) LLM4Solver with MOEA consistently outperforms expert-designed heuristics and demonstrates robust cross-benchmark generalization; 3) LLM4Solver with MOEA outperforms the approach of simply using the average performance across the 4 benchmarks as a single objective. Furthermore, evaluations on 20 MIPLIB instances (Table IV) show that MOEA-designed heuristics find better solutions even on heterogeneous, unseen instances. This underscores MOEA's ability to synthesize diverse problem characteristics into a unified, high-performance algorithm.

Efficient Searching We show the convergence process of LLM4Solver with SOEA on Setcover dataset in Figure 2. It illustrates that LLM4Solver can design an algorithm better than the best human-designed ones in the *first* generation and better than the SOTA learning-based method L2DIVE in the *fourth* generation. The convergence time with 10 iterations is 3503.5±73.4s for Setcover, and 1835.4±56.6s, 4568.2±116.3s, 1213.6±48.9s for Cauctions, Facilities, and Indset respectively. In comparison, EoH and FunSearch require 4104.3±69.3s seconds and 7335.1±81.9s respectively

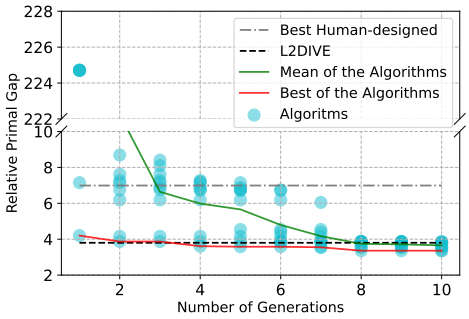

Fig. 2: The convergence curve of LLM4Solver with SOEA on the Setcover problem, where each point represents an algorithm during the evolution.

Methods	Setcover	Cauctions
LLM (No Evolution)	3.84 (0.31)	2.84 (0.22)
LLM4Solver (No Crossover)	3.67 (0.28)	2.77 (0.21)
LLM4Solver (No Mutation)	3.55 (0.27)	2.34 (0.19)
LLM4Solver	3.36 (0.25)	1.83 (0.16)

TABLE V: A comparison of different parts in evolution. The average relative primal gap with standard error.

to converge for Setcover. The result shows that LLM4Solver is *efficient* for designing high-quality diving heuristics.

Interpretability We report the diving heuristic designed by LLM4Solver with MOEA in Figure 3. There are two key features that enable LLM4Solver to achieve both interpretability and high performance. (1) Leveraging LLMs' text and code generation capabilities, LLM4Solver directly generates code and provides comments for the code. This offers greater interpretability compared to black-box neural networks [12] and purely numerical symbolic methods [14]. (2) LLM4Solver can refine parameters or specific computational methods to achieve higher performance for particular problems. For example, after multiple rounds of evolution, it ultimately selected a penalty value of "0.2" for the "penalize limited rounding options" strategy. Similarly, for "prioritize low pseudo costs," it uses the formula "min(1/(1+(pscostdown+pscostup)), 1)" instead of directly applying "-(pscostdown+pscostup)". These algorithms not only improve the solving performance but also help experts obtain **insights** into solving patterns.

C. Ablation Study

We conduct ablation studies to compare the contribution of different parts in the evolutionary process. For "LLM (No Evolution)", we disable all evolutionary processes and solely use the LLM to generate 100 algorithm candidates. Additionally, we compare LLM4Solver where either crossover or mutation was excluded individually. The results in Table V indicate that crossover is more crucial for the outcomes. Without crossover, the solution quality decreases by 10%-28%, while not using any evolutionary process results in a 14%-55% reduction in solution quality.

V. Conclusions

In this paper, we propose a novel LLM-based automatic algorithm design framework for combinatorial optimization

```
def myheurdiving(mayrounddown, mayroundup, candsfrac, candsol,
nlocksdown, nlocksup, obj, objnorm, pscostdown, pscostup, rootsolval,
nNonz, isBinary):
    # Initialization
    score = 0.0
    roundup = False
    # Penalize limited rounding options to encourage exploration
    score -= 0.2 if mayrounddown or mayroundup else 0.0
    # Prioritize variables with high fractional values and low pseudo costs
    score += candsfrac
    score += min(1 / (1 + (pscostdown + pscostup)), 1)
    # Consider the impact of objective function value and Euclidean norm
    score += (obj / max(1, objnorm)) * (1 - candsfrac) / (nNonz + 1)
    # Incorporate historical solution values
    score += rootsolval * 0.3
    # Penalize excessive sparsity
    if nNonz < 5:
        score -= 0.2
    # Adjust based on the number of locks for rounding down/up of a
special type
    score -= min(0.2 * (nlocksdown + nlocksup), 0.4)
    # Differentiate based on the binary nature of the variable
    score *= 1.5 if isBinary else 1.0
    # Determine rounding direction based on the score
    if candsfrac > 0.5:
        roundup = True
    return score, roundup
```

Fig. 3: The code generated by LLM4Solver.

solvers to efficiently design high-quality and generic diving heuristics. To leverage the heterogeneous characteristics of different CO problems, we extend this framework through *multi-objective evolution*. Extensive experiments show that LLM4Solver significantly outperforms all the SOTA human-designed and learning-based methods in terms of solution quality, solving efficiency, and cross-benchmark generalization ability. Furthermore, the appealing features of LLM4Solver include high performance, efficient searching, and interpretability of the designed algorithms. The results show an encouraging step towards efficient automatic algorithm design on modern exact CO solvers via large language models. Applying LLM4Solver to more components in modern CO solvers like branching [10], [14], presolve [7], and cut generation [11], [26] are exciting avenues for further work. Finally, LLM4Solver shows the potential to efficiently design high-quality and generic algorithms for the next generation of solvers.

VI. Acknowledgements

We used LLMs solely for language editing during manuscript preparation. The authors conceived all intellectual content and take full responsibility for the integrity of this publication.

References

[1] T. Achterberg, "Constraint integer programming," Ph.D. dissertation, Berlin Institute of Technology, 2007. [Online]. Available: http://opus.kobv.de/tuberlin/volltexte/2007/1611/

[2] Y. Bengio, A. Lodi, and A. Prouvost, "Machine learning for combinatorial optimization: A methodological tour d'horizon," *Eur. J. Oper. Res.*, vol. 290, no. 2, pp. 405–421, 2021.

[3] S. Liu, J. M. Pinto, and L. G. Papageorgiou, "A tsp-based milp model for medium-term planning of single-stage continuous multiproduct plants," *Industrial & Engineering Chemistry Research*, vol. 47, no. 20, pp. 7733–7743, 2008.

[4] Z.-L. Chen, "Integrated production and outbound distribution scheduling: review and extensions," *Operations research*, vol. 58, no. 1, pp. 130–148, 2010.

[5] K. Ma, L. Xiao, J. Zhang, and T. Li, "Accelerating an fpga-based sat solver by software and hardware co-design," *Chinese Journal of Electronics*, vol. 28, no. 5, pp. 953–961, 2019.

[6] V. T. Paschos, *Applications of combinatorial optimization*. John Wiley & Sons, 2014, vol. 3.

[7] Y. Kuang, X. Li, J. Wang, F. Zhu, M. Lu, Z. Wang, J. Zeng, H. Li, Y. Zhang, and F. Wu, "Accelerate presolve in large-scale linear programming via reinforcement learning," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2025.

[8] A. Gleixner, M. Bastubbe, L. Eifler, T. Gally, G. Gamrath, R. L. Gottwald, G. Hendel, C. Hojny, T. Koch, M. E. Lübbecke *et al.*, "The scip optimization suite 6.0. technical report," *Optimization Online*, 2018.

[9] Gurobi Optimization, LLC, "Gurobi Optimizer Reference Manual," 2023. [Online]. Available: https://www.gurobi.com

[10] M. Gasse, D. Chételat, N. Ferroni, L. Charlin, and A. Lodi, "Exact combinatorial optimization with graph convolutional neural networks," in *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, H. M. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. B. Fox, and R. Garnett, Eds., 2019, pp. 15 554–15 566.

[11] J. Wang, Z. Wang, X. Li, Y. Kuang, Z. Shi, F. Zhu, M. Yuan, J. Zeng, Y. Zhang, and F. Wu, "Learning to cut via hierarchical sequence/set model for efficient mixed-integer programming," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2024.

[12] M. Paulus and A. Krause, "Learning to dive in branch and bound," *Advances in Neural Information Processing Systems*, vol. 36, 2024.

[13] V. Nair, S. Bartunov, F. Gimeno, I. Von Glehn, P. Lichocki, I. Lobov, B. O'Donoghue, N. Sonnerat, C. Tjandraatmadja, P. Wang *et al.*, "Solving mixed integer programs using neural networks," *arXiv preprint arXiv:2012.13349*, 2020.

[14] Y. Kuang, J. Wang, H. Liu, F. Zhu, X. Li, J. Zeng, J. Hao, B. Li, and F. Wu, "Rethinking branching on exact combinatorial optimization solver: The first deep symbolic discovery framework," in *The Twelfth International Conference on Learning Representations*, 2024.

[15] Y. Kuang, J. Wang, Y. Zhou, X. Li, F. Zhu, J. Hao, and F. Wu, "Towards general algorithm discovery for combinatorial optimization: Learning symbolic branching policy from bipartite graph," in *International Conference on Machine Learning*. PMLR, 2024, pp. 25 623–25 641.

[16] H. Naveed, A. U. Khan, S. Qiu, M. Saqib, S. Anwar, M. Usman, N. Akhtar, N. Barnes, and A. Mian, "A comprehensive overview of large language models," *arXiv preprint arXiv:2307.06435*, 2023.

[17] C. Yang, X. Wang, Y. Lu, H. Liu, Q. V. Le, D. Zhou, and X. Chen, "Large language models as optimizers," *arXiv preprint arXiv:2309.03409*, 2023.

[18] B. Romera-Paredes, M. Barekatain, A. Novikov, M. Balog, M. P. Kumar, E. Dupont, F. J. Ruiz, J. S. Ellenberg, P. Wang, O. Fawzi *et al.*, "Mathematical discoveries from program search with large language models," *Nature*, vol. 625, no. 7995, pp. 468–475, 2024.

[19] F. Liu, T. Xialiang, M. Yuan, X. Lin, F. Luo, Z. Wang, Z. Lu, and Q. Zhang, "Evolution of heuristics: Towards efficient automatic algorithm design using large language model," in *Forty-first International Conference on Machine Learning*, 2024.

[20] H. Ye, J. Wang, Z. Cao, and G. Song, "Reevo: Large language models as hyper-heuristics with reflective evolution," *arXiv preprint arXiv:2402.01145*, 2024.

[21] Y. Sun, X. Zhang, S. Huang, S. Cai, B.-Z. Zhang, and K. Wei, "Autosat: Automatically optimize sat solvers via large language models," *arXiv preprint arXiv:2402.10705*, 2024.

[22] Z.-H. Zhou, Y. Yu, and C. Qian, *Evolutionary learning: Advances in theories and algorithms*. Springer, 2019.

[23] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: Nsga-ii," *IEEE transactions on evolutionary computation*, vol. 6, no. 2, pp. 182–197, 2002.

[24] M. Gasse, S. Bowly, Q. Cappart, J. Charfreitag, L. Charlin, D. Chételat, A. Chmiela, J. Dumouchelle, A. Gleixner, A. M. Kazachkov *et al.*, "The machine learning for combinatorial optimization competition (ml4co): Results and insights," in *NeurIPS 2021 competitions and demonstrations track*. PMLR, 2022, pp. 220–231.

[25] A. Gleixner, G. Hendel, G. Gamrath, T. Achterberg, M. Bastubbe, T. Berthold, P. Christophel, K. Jarck, T. Koch, J. Linderoth *et al.*, "Miplib 2017: data-driven compilation of the 6th mixed-integer programming library," *Mathematical Programming Computation*, vol. 13, no. 3, pp. 443–490, 2021.

[26] Z. Huang, K. Wang, F. Liu, H.-L. Zhen, W. Zhang, M. Yuan, J. Hao, Y. Yu, and J. Wang, "Learning to select cuts for efficient mixed-integer programming," *Pattern Recognition*, vol. 123, p. 108353, 2022.