DRAGON: LLM-Driven Decomposition and Reconstruction Agents for Large-Scale Combinatorial Optimization

Shengkai Chen
Institute for Infocomm Research
A*STAR, Singapore
Chen_Shengkai@a-star.edu.sg

Zhiguang Cao
Singapore Management University
Singapore
zgcao@smu.edu.sg

Jianan Zhou
Nanyang Technological University
Singapore
jianan004@e.ntu.edu.sg

Yaoxin Wu
Eindhoven University of Technology
Eindhoven, Netherlands
y.wu2@tue.nl

Senthilnath Jayavelu
National University of Singapore &
Institute for Infocomm Research
A*STAR, Singapore
J_Senthilnath@a-star.edu.sg

Zhuoyi Lin*
Institute for Infocomm Research
A*STAR, Singapore
Lin_Zhuoyi@a-star.edu.sg

Xiaoli Li
Singapore University of Technology
and Design, Singapore
xiaoli_li@sutd.edu.sg

Shili Xiang
Institute for Infocomm Research
A*STAR, Singapore
Xiang_Shili@a-star.edu.sg

ABSTRACT

Large Language Models (LLMs) have recently shown promise in addressing combinatorial optimization problems (COPs) through prompt-based strategies. However, their scalability and generalization remain limited, and their effectiveness diminishes as problem size increases, particularly in routing problems involving more than 30 nodes. We propose **DRAGON**, which stands for **D**ecomposition and **R**econstruction **A**gents **G**uided **O**ptimizatio**N**, a novel framework that combines the strengths of metaheuristic design and LLM reasoning. DRAGON autonomously identifies regions in initial solution with high optimization potential and strategically decompose large-scale COPs into manageable subproblems. Each subproblem is then reformulated as a concise, localized optimization task and solved through targeted LLM prompting guided by accumulated experiences. Finally, the locally optimized solutions are systematically reintegrated into the global context to yield a significantly improved outcome. By continuously interacting with the optimization environment and experience memory, the agents iteratively learn from feedback, effectively coupling symbolic reasoning with heuristic search. Empirical results show that, DRAGON consistently produces feasible solutions on TSPLIB, CVRPLIB, and Weibull-5k bin packing benchmarks, and achieves near-optimal results (0.16% gap) on knapsack problems with over 3M variables. This work shows the potential of feedback-driven language agents as a new paradigm for generalizable and interpretable large-scale optimization.

KEYWORDS

Agentic AI; Combinatorial Optimization; Metaheuristics

*Corresponding author

Proc. of the 25th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2026), C. Amato, L. Dennis, V. Mascardi, J. Thangarajah (eds.), May 25 – 29, 2026, Paphos, Cyprus. © 2026 International Foundation for Autonomous Agents and Multiagent Systems (www.ifaamas.org). https://doi.org/10.65109/SBAY6258

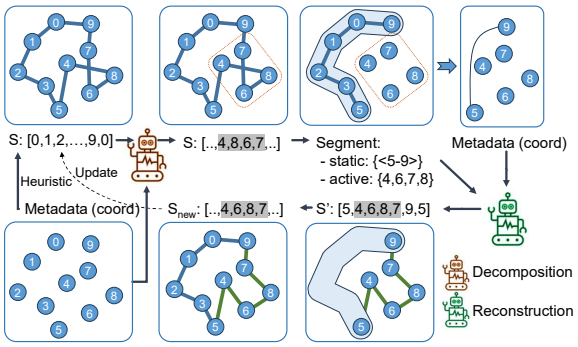

Figure 1: Illustration of DRAGON on a TSP instance. The decomposer identifies a suboptimal segment {4,6,7,8} (grayed) from a global solution. The segment is locally refined and reintegrated by the reconstructor to improve the global tour.

ACM Reference Format:
Shengkai Chen, Zhiguang Cao, Jianan Zhou, Yaoxin Wu, Senthilnath Jayavelu, Zhuoyi Lin, Xiaoli Li, and Shili Xiang. 2026. DRAGON: LLM-Driven Decomposition and Reconstruction Agents for Large-Scale Combinatorial Optimization. In *Proc. of the 25th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2026), Paphos, Cyprus, May 25 – 29, 2026*, IFAAMAS, 9 pages. https://doi.org/10.65109/SBAY6258

1 INTRODUCTION

Combinatorial optimization problems (COPs), such as the Traveling Salesman Problem (TSP), Vehicle Routing Problem (VRP), and Knapsack Problem (KP), are notoriously challenging due to the NP-hard nature [4]. COPs are typically addressed using exact algorithms or heuristic methods, which often require substantial manual design and parameter tuning [9, 14, 22, 25, 32], thus limiting scalability and adaptability to varying problem instances and domains.

Large Language Models (LLMs) have led to a series of breakthroughs for various challenging problems, including language

understanding, text generation, code synthesis, and reasoning tasks [1, 8, 10]. Recent evidence reveals that LLMs have emerged as powerful tools capable of addressing a variety of COPs, leveraging their reasoning capabilities for direct solution generation [37, 41] or heuristic design [23, 33, 42, 45]. Their inherent strengths in abstraction, generalization, and semantic understanding allow them to generate promising solutions even without explicit algorithmic instructions. However, the current capabilities of LLMs for directly generating solutions remain largely confined to relatively small-scale instances, such as TSP with fewer than 30 nodes [17, 24, 41]. As problem size and complexity increase, LLM-based solutions often deteriorate due to restricted context length, reduced logical coherence, and difficulty in representing combinatorial structures [17, 43]. These limitations significantly impede the practical deployment of LLM-driven methods in real large-scale applications, such as logistics, transportation, and supply chain management, where problems commonly involve hundreds to thousands of nodes [7].

In this study, we aim to answer two research questions: (1) *Can an LLM agent, without solving the full COP, identify and isolate regions of the solution that are likely suboptimal or have improvement potential?* (2) *Can an LLM agent effectively solve small-scale COPs locally while following additional, customized non-trivial constraints?* These questions highlight the core challenges of our framework: utilizing LLMs to guide decomposition by detecting potential improvement areas, and ensuring reconstructed solutions remain feasible.

Prompt-based LLM methods often fail on large-scale instances due to the absence of domain knowledge and the context length limitation, while code-generation approaches require extensive task-specific training to develop robust heuristics, leading to substantial computational costs and poor cross-domain transferability. Meanwhile, classical metaheuristics such as divide-and-conquer and large neighborhood search (LNS) [29] exhibit strong scalability in large-scale COPs by iteratively decomposing and refining solutions, though they rely heavily on handcrafted heuristics and expert knowledge. To address the gap, we propose **DRAGON**, a framework that transforms complex large COP solving into decomposition and reconstruction tasks guided by LLM agents. DRAGON first strategically identifies regions with high potential of improvement, decomposing large-scale COPs into context-manageable subproblems, and locally optimizes each subregion through reconstruction.

To concretely illustrate this idea, Figure 1 demonstrates an example of how the proposed Dragon improves TSP solution locally and globally. Given an initial global solution, the decomposition agent $\mathcal{D}$ identifies nodes $\{4, 6, 7, 8\}$ (highlighted in gray) as a subproblem with high potential for further improvement, while the remaining nodes remain static and impose boundary constraints. During reconstruction, the suboptimal local segment (4-8-6-7) is locally optimized with LLM agents, leading to a better sub-solution (4-6-7-8). Consequently, the improved local segment is integrated into the global solution, yielding a reduced tour length solution.

Overall, our contributions are threefold: (1) To the best of our knowledge, this is the first work to demonstrate that LLM agents can be effectively leveraged to directly generate high-quality solutions for large-scale COPs, opening new possibilities for LLM-driven optimization. (2) We propose DRAGON, a divide-and-conquer framework that dynamically decomposes large-scale COPs and subsequently refines compact subproblems by state passing between agents. (3) We empirically validate DRAGON on large-scale COPs benchmarks, demonstrating substantial improvements in solution quality and scalability compared to state-of-the-art LLM-based baselines. Our findings validate the feasibility of leveraging LLM agents for large-scale COPs that are prevalent in real-world applications.

2 RELATED WORK

LLMs for Combinatorial Optimization. Recent studies have increasingly examined the capability of LLMs to solve COPs by leveraging prompt engineering techniques. Among these efforts, a prominent direction involves using LLMs directly as solution generators, where carefully designed prompts and enriched input information guide the models to produce viable solutions. Early research by Wang et al. [37] provided evidence that LLMs could solve certain graph-based COPs, although this capability was strongly tied to the quality of the provided prompts. Subsequent work has sought to refine this process by incorporating contextual information. Examples include integrating graph structural and topological data [38], existing heuristic solutions [17], explicit relationships between solutions and objectives [41], and visual representations of problems or solutions [12, 16]. Another research pathway involves leveraging LLMs as components within structured algorithms or frameworks. For instance, Liu et al. [24] demonstrated the utility of LLMs in evolutionary computation by prompting them to perform essential algorithmic operations such as selection, crossover, and mutation. Similarly, Elhenawy et al. [13] deployed multiple LLMs as collaborative agents, each responsible for distinct roles such as initialization, critique, and evaluation within a predefined optimization pipeline. Despite these promising advancements, current capabilities for directly generating solutions via LLMs remain largely restricted to relatively small-scale problems, typically limited to instances such as TSP with fewer than 30 nodes. To circumvent this limitation, Iklassov et al. [17] introduced a decomposition approach wherein an LLM assesses problem complexity autonomously. If deemed difficult, the model recursively subdivides the problem into simpler, manageable subproblems. Nonetheless, its scalability remains limited.

Beyond direct solution generation, other studies have explored leveraging LLMs for heuristic design [23, 33, 42] and mathematical modeling [2, 20, 40]. The former aims to use LLMs to discover heuristics expressed in code that have potential to outperform those crafted by human experts, while the latter focuses on translating natural language problem descriptions into mathematical formulations compatible with traditional OR solvers. More recently, research has investigated training large language models for end-to-end combinatorial optimization, rather than relying solely on prompting paradigms [18, 19]. These directions fall outside the scope of this work, and we refer interested readers to Da Ros et al. [11] for a comprehensive survey.

Large-Scale Combinatorial Optimization. Another research direction related to this work aiming to design metaheuristics for solving large-scale COPs, either with machine learning [21, 35] or through traditional methods [3, 34]. These approaches generally follow decomposition or divide-and-conquer principles. While they have shown promising results, their effectiveness often relies

on problem-specific strategies (e.g., domain-specific decomposition rules), which limits their generality. In this work, we propose DRAGON, a general decomposition-reconstruction framework leveraging the semantic capabilities of LLMs. Unlike previous methods, DRAGON identifies suboptimal regions without explicit domain-specific rules and iteratively reconstructs globally consistent solutions. Our framework effectively addresses LLM context-length constraints, combining learned decomposition advantages with the generalization provided by LLM agents.

3 METHODOLOGY

DRAGON enables LLM agents to effectively solve large-scale COPs via state passing communication. As illustrated in Figure 2, DRAGON iteratively alternates between two core tasks: Decomposition and Reconstruction. Starting from fast heuristic initialization, DRAGON progressively refines it by identifying regions with high improvement potential and solving localized subproblems under explicit constraint awareness. In each iteration, the current solution is decomposed into manageable subproblems, locally optimized through reconstruction agents, and then reassembled into a globally improvement. This hierarchical process allows DRAGON to scale from small instances to complex large-scale COPs. Algorithm 1 summarized our workflow, where implicit communication between agents occurs via solution state passing.

Algorithm 1 DRAGON for Large-Scale COPs

Require: Metadata M, initial solution S_0, maximum time $T_{\max}$, rejection threshold N
Ensure: Improved solution S

$S \leftarrow S_0$ // Current solution
$t \leftarrow 0$ // Time counter
$r \leftarrow 0$ // Rejection counter
while $t < T_{\max}$ **and** $r < N$ **do**
  $(\mathbf{a}, \mathbf{s}) \leftarrow \mathcal{D}(M, S)$ // **Decomposition**
  $(M', S', C) \leftarrow \texttt{Compress}(M, \mathbf{a}, \mathbf{s})$
  $S'_{\text{new}} \leftarrow \mathcal{R}(M', S', C)$ // **Reconstruction**
  if $\texttt{Accept}(S', S'_{\text{new}})$ **then**
    $S' \leftarrow S'_{\text{new}}$
    $S \leftarrow \texttt{Integration}(S', S)$
    $r \leftarrow 0$ // Reset rejection counter
  else
    $r \leftarrow r + 1$
  end if
  $t \leftarrow \texttt{ElapsedTime}()$ // Reach running time limit.
end while
return S

3.1 Framework Inputs and Solution Updates

DRAGON interacts with problem environment that contains two fundamental inputs: metadata M and the current solution S. Metadata M, represented as structured JSON, encodes essential problem-specific parameters, and the solution S is represented as an ordered sequence of elements, such as city indices for TSP or item identifiers for Knapsack, and is formatted in either JSON or XML, as illustrated in Figure 2(a).

Formally, given metadata M, the global solution at iteration i is obtained as follows:

$$S_i = \begin{cases} \texttt{Fast\_Heuristic}(M), & i = 0 \\ \texttt{Integration}(S'_{i-1}, S_{i-1}), & i > 0 \\ S'_{i-1} = \mathcal{R}\left(\mathcal{D}\left(M, S_{i-1}\right)\right), & i > 0 \end{cases} \tag{1}$$

Here, S'_{i-1} denotes the locally refined sub-solution from the previous iteration, identified by the decomposer $\mathcal{D}$ and enhanced by the reconstructor $\mathcal{R}$ (both detailed in subsequent sections). `Integration` involves simply concatenation and sorting operations, while specific implementation of `Fast_Heuristic` are provided in the experimental section. Furthermore, a new solution is conditionally accepted using a probabilistic criterion (assuming minimization):

$$\texttt{Accept}(S_1, S_2) = \min\left\{1, \exp\left(-\frac{f(S_2) - f(S_1)}{T}\right)\right\} \tag{2}$$

where f denotes the objective function and T is a temperature parameter controlling the acceptance probability. As shown in Algorithm 1, we substitute the current local solution S'_i and its $\mathcal{R}$ processed version $S'_{\text{new},i}$ into Eq. 2 as S_1 and S_2, respectively, and accepted solutions are then integrated back into the global solution to ensure consistency and feasibility. It is worth noting that even when the new local solution is worse i.e., $f(S'_{\text{new},i}) \geq f(S'_i)$, there remains a small chance of acceptance and integration, allowing DRAGON to occasionally accept inferior solutions, helping it escape local optima, promote exploration of the solution space, and maintain a balancing between exploitation and exploration.

Balancing exploitation (greedy improvement) and exploration (diversification) is critical to the performance of heuristic search methods. In many cases, domain experts must carefully design problem-specific strategies to achieve this balance, and the convergence rate of the solution gap can vary significantly depending on the designer's expertise. However, for large-scale COPs where analyzing the problem structure is extremely challenging and generalization across domains is difficult, we leverage LLM-based agents, which are well-suited for both exploitation and exploration, because they can exploit stored experiences from previous trails, while their generative and adaptive capabilities enable diverse solution proposals, supporting broader exploration of the solution space.

3.2 Decomposition for Large-Scale COPs

A central challenge in applying LLMs to large-scale COP decomposition lies in their limited capability to reason over complex global constraints. Although existing studies demonstrate the potential of LLMs in solving COPs [39, 41], their capability to accurately detect and isolate sub-problems suitable for local refinement within large-scale COPs remains largely unexplored. To this end, we propose a decomposition agent ($\mathcal{D}$) which leverages LLMs to strategically partition COPs into smaller, manageable sub-problems for further refinement, rather than directly solving the entire COP. Specifically, the decomposer $\mathcal{D}$ functions as a high-level planner, analyzing metadata M and the current solution S_i. It partitions the solution into *active segments* ($\mathbf{a}_i$), which have substantial improvement potential, and *static segments* ($\mathbf{s}_i$), which remain unchanged during the current iteration. Specifically, decomposition is formulated as:

$$(\mathbf{a}_i, \mathbf{s}_i) = \mathcal{D}(M, S_i) \tag{3}$$

DRAGON demonstrates that well-designed prompts (illustrated in Figure 2(b) and Supplementary Materials (Prompt designs)) enable LLMs to effectively identify segments with significant potential for improvement while introducing additional constraints (C_i) to maintain feasibility and global consistency:

$$(M'_i, S'_i, C_i) = \texttt{Compress}(M, \mathbf{a}_i, \mathbf{s}_i) \tag{4}$$

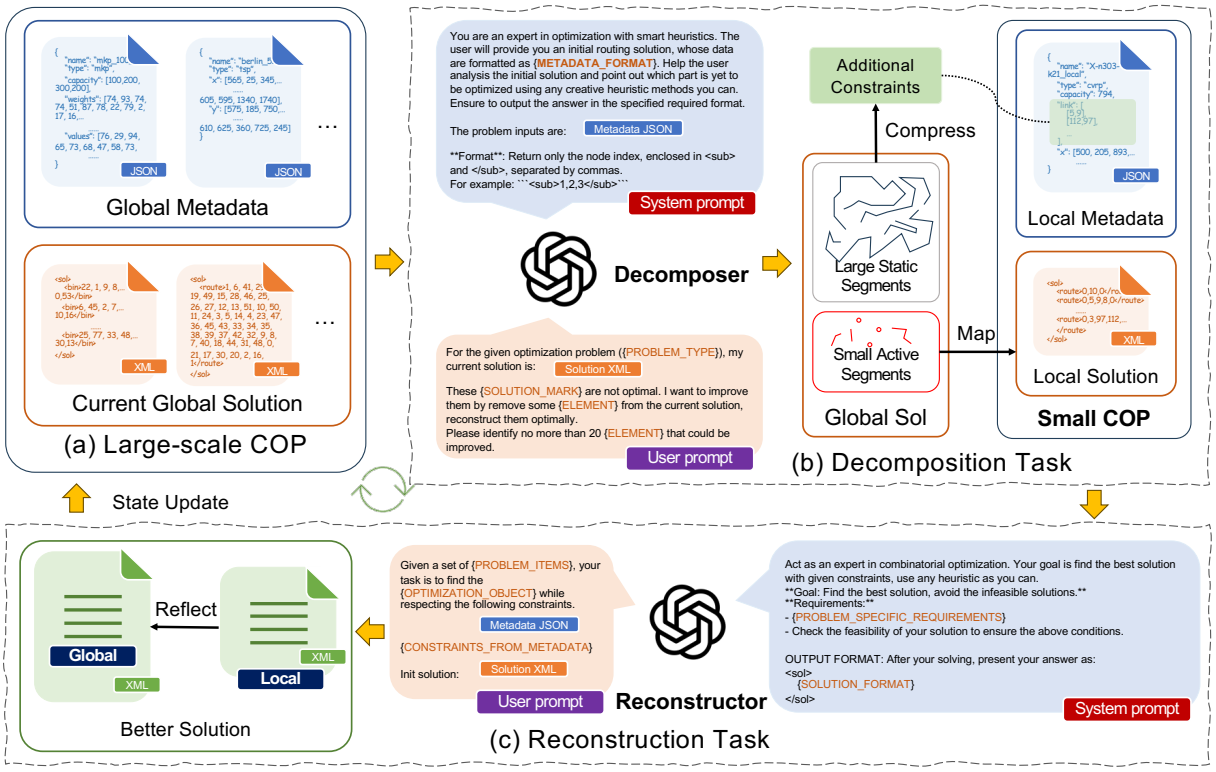

Figure 2: Overview of the state passing among agents in DRAGON framework. With (a) given COP data input and current global solution, the DRAGON pipeline process alternates between two key stages: (b) A decomposition step uses an LLM to split a large-scale COP into manageable active and static segments. These are compressed into a smaller subproblem; (c) A reconstruction step solves the reduced problem with additional constraints to yield a refined local solution.

where $M_i' \subseteq M$ denotes the subset of original metadata relevant only to the active segments. For example, in the TSP scenario illustrated in Figure 1, the decomposer identifies an active segment $\mathbf{a}_i = \{4, 6, 7, 8\}$ and a static segment $\mathbf{s}_i = [9, 0, 1, 2, 3, 5]$. The `Compress` step significantly reduces the size of the local data $|M_i'| << |M|$, making it manageable to downstream agents. However, it will introduce extra specific constraints, which are a set of compressed representations of static segment $\mathbf{s}_i$.

3.3 Reconstruction with Constraint Integration

Recent studies [17, 24, 41] highlight the effectiveness of LLMs in solving small-scale COPs, typically with fewer than 30 decision variables. Building on this capability, our reconstruction agent ($\mathcal{R}$) is designed to solve compressed COP instances iteratively, derived from the decomposition stage.

Specifically, reconstructor $\mathcal{R}$ utilizes localized metadata M_i' augmented with explicit constraints C_i to ensure global feasibility and consistency. Constraints C_i are explicitly integrated into natural language within the LLM prompt, with the formats varying by COP type. Formally, the reconstruction process is defined as:

$$S'_{\text{new},i} = \mathcal{R}(M_i', S_i', C_i) \tag{5}$$

where M_i' encapsulates localized metadata, and C_i encodes global consistency requirements derived from static segments $\mathbf{s}_i$.

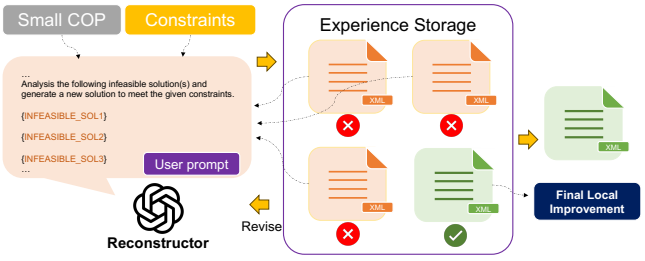

Figure 3: Constraint satisfaction in reconstruction is ensured via experience accumulation.

Note that the format and interpretation of constraints depend on the specific COP type. In routing problems such as TSP and CVRP, constraints typically enforce particular edge sequences to ensure route continuity, while in packing problems like MKP or BPP, constraints explicitly mandate item inclusion or exclusion. For instance, a constraint prompt for TSP may state: *"the following edges must be visited consecutively, with no additional points permitted between them."* It is essential to maintain consistency between constraints and compressed metadata. For example, required edges (a, b) in CVRP represent condensed segments from the original route, with

associated demands accurately reflecting the cumulative demands of intermediate nodes.

As illustrated in Figure 3, reconstruction agent utilizes iterative experience storage collected along previous trails to self-revise candidate solutions until all specified constraints are satisfied. The agent's experience storage records all previous attempts, regardless of whether they result in feasible solutions or not, along with annotations on the type and cause of any constraint violations. This memory allows the agent to learn from past infeasible solutions, avoid repeating similar errors, and identify patterns within feasible solutions to guide future improvements. Through this process of iterative refinement, localized solutions are gradually adjusted to remain coherent and globally feasible. Extensive experiments detailed in the experimental section and Supplementary Materials demonstrate that, when guided by well-structured prompts and clearly defined constraints, LLM agents can consistently generate high-quality solutions to constrained subproblems.

4 EXPERIMENTS

4.1 Setup

4.1.1 Problem Descriptions. To evaluate the effectiveness and generalizability of the proposed DRAGON framework, we conduct experiments on four representative COPs across routing, packing and assignment domains:

• Traveling Salesman Problem (**TSP**): A classical combinatorial routing problem that seeks the shortest possible tour visiting each city exactly once and returning to the starting point.

• Capacitated Vehicle Routing Problem (**CVRP**): Extension of TSP with multiple vehicles and demand constraints. The goal is to minimize total travel distance while satisfying delivery demands.

• Bin Packing Problem (**BPP**): Classical combinatorial optimization problem where items of various sizes must be packed into a minimal number of fixed-capacity bins without exceeding their limits.

• Multiple Knapsack Problem (**MKP**): A generalization of the 0–1 knapsack problem involving multiple independent knapsacks. Unlike the multi-dimensional knapsack problem [9], where each item is constrained by several resource limits, the MKP aims to maximize the total value by optimally assigning items to knapsacks without exceeding their individual capacities.

4.1.2 Datasets. Detailed datasets for the aforementioned problems:

• **TSP** We use 77 benchmark instances from TSPLIB [31] of the EUC_2D type, with sizes ranging from 50 to 20k, covering a broad range of scales and spatial layouts. **Oracle:** optimal values or lower bounds reported in [31].

• **CVRP** We select subsets of 19 instances from CVRPLIB, including both X-type [36] (up to 1k nodes) and larger XML-type instances [30] distribution, which was synthesized at large scale (up to 5k nodes) by [44]. **Oracle:** optimal values or lower bounds reported in [36, 44].

• **BPP** We adopt instances from FunSearch [33] and EoH [23] of all Weibull-5k, which is reflecting real-world scheduling and allocation scenarios. **Oracle:** L2 lower bounds determined by [33].

• **MKP** Following Google OR-Tools guidelines [28], we generate 10 synthetic instances by with knapsack capacities uniformly sampled from $U(100, 500)$, using 10 to 100 knapsacks. Item values and weights drawn uniformly from $U(1, 100)$. All instances will be released in supplementary JSON files. **Oracle:** optimal values or uppers bound searched by solver.

4.1.3 Implementation Details. We evaluate our framework using four representative LLMs accessed via API: two general-purpose models, OpenAI's `gpt-4o` and `gpt-4.1`, and two reasoning-specialized models, OpenAI's `o3` and DeepSeek's `r1`. Due to time and cost constraints, we limit our experiments to these models. Prompting strategies for each stage of DRAGON are illustrated in Figure 2, with complete prompt templates provided in the Appendix (Prompt designs). We adopt simple global solution initialization algorithms to each problem: 1) Random Insertion [5] for TSP, 2) Greedy Nearest Neighbor heuristic [26] for CVRP, and 3) First-Fit Decreasing [6] for both BPP and MKP. For comparative evaluation, we implement OR-Tools [28] solvers as: CVRP is solved via the routing model [15] with *Guided_Local_Search* as the metaheuristic, while BPP uses the CP-SAT [27]. All experiments run on an Ubuntu 20.04 server with an Intel Core i9-10900X (20 cores at 4.6 GHz) without GPU. For detailed prompt designs used in the decomposition and reconstruction processes, please refer to the Supplementary Materials.

4.1.4 Feasibility check. We implement a checker based on the inherent constraints of each COP type. If a solution is found to be infeasible, it is stored in the agent's memory along with comments explaining the reason for infeasibility. This feedback helps the agent avoid repeating the same mistakes, thereby reducing search time and accelerating convergence to feasible solutions. In the results presented later, we denote an infeasible solution as "`infe`" and use "`inf`" to indicate cases where the solving time exceeds the given time limit $T_{\max}$.

4.1.5 Metrics. We evaluate performance using four metrics: (1) Optimality Gap defined as $Gap = \frac{|v - v^*|}{v^*} \times 100\%$, where v and v^* are the objective value and the optimal value, respectively; (2) Running Time (seconds); (3) Input/Output Token Counts; and (4) Number of API Calls. All experiments use consistent settings with a 1-hour time limit $T_{\max} = 3600$ and early stopping after $N = 5$ consecutive non-improving iterations.

4.2 Performance Evaluation

To evaluate performance on large-scale routing problems (TSP and CVRP), Table 1 presents the average optimality gap (%) across different instance size groups, comparing DRAGON with existing LLM-based solvers, including prompt-based methods (OPRO [41], SGE [17]) and code-generation approaches (LMEA [24], ReEvo [42], with ReEvo(a) using Ant Colony Optimization (ACO) and ReEvo(c) employing a constructive heuristic). To ensure a fair comparison, all LLM-based methods use `gpt-4o`, with token usage and inference time recorded. Instances are grouped by node size for conciseness, and detailed results are listed in Supplementary Materials.

OPRO fails to return valid solutions for all groups, as some hard instances in groups exceed the runtime limit, resulting in an infinite gap `inf`. SGE consistently produces infeasible solutions (`infe`), likely because it lacks explicit feasibility checks, since its original design is for small problems (under 30 nodes). LMEA handles slightly larger instances but exhibits high gaps and fails beyond 2k-node due to prompt length limits and generation timeouts.

Table 1: Average optimality gap (%) on routing problems using subsets from TSPLIB (EUC_2D) and CVRPLIB (Set X/XML).

Method	TSPLIB (EUC_2D)					
(size, k	0.05-0.5	0.5-1	1-2	2-5	5-20	All
#)	42	6	15	7	7	77
OPRO	inf	inf	inf	inf	inf	inf
SGE	infe	infe	infe	infe	infe	infe
LMEA	820.30	2279.35	3578.21	inf	inf	inf
ReEvo(c)	12.69	*16.43*	*17.01*	inf	inf	inf
ReEvo(a)	**8.17**	16.61	18.23	*21.20*	*18.62*	*13.10*
DRAGON	*9.74*	**11.24**	**15.24**	**19.37**	**15.42**	**12.24**

Method	CVRPLIB (X/XML)						
(size, k	0.1-0.2	0.2-0.5	0.5-1	1-2	2-5	≥5	All
#)	2	3	5	3	3	3	19
OR-Tools	inf	inf	inf	inf	inf	inf	inf
ReEvo(a)	**21.49**	**19.27**	29.44	25.39	22.00	11.24	24.17
DRAGON	25.56	29.35	**26.73**	**15.51**	**15.48**	**6.45**	**20.15**

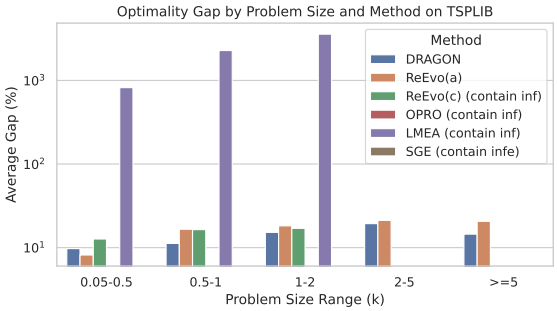

Figure 4: Average Optimality Gap (%) (log scale) across problem size groups for different methods on TSPLIB.

As shown in Figure 4, the results in TSPLIB reveal that SGE is excluded due to infeasibility, while OPRO is omitted as "inf" so that no solution was found due to its slow inference time in all groups. LMEA is able to solve up to medium-sized instances but exhibits relatively large optimality gaps. ReEvo(c) achieves strong results up to the medium-size group, whereas ReEvo(a) excels on smaller instances. Although DRAGON is not the best on every group, it shows clear advantages on larger-scale problems, driven by its task-allocation mechanism, where decomposition and reconstruction work effectively together. Figure 5 presents the methods capable of obtaining feasible solutions for CVRP instances. As CVRP is inherently more complex than TSP, DRAGON outperforms ReEvo(a) primarily on large-scale problems, where its decomposition–reconstruction mechanism demonstrates greater advantages.

These findings highlight a major limitation of pure prompt-based methods: their performance degrades significantly as problem size increases. This is mainly due to long context lengths and high decoding cost—making token-by-token generation is unstable when

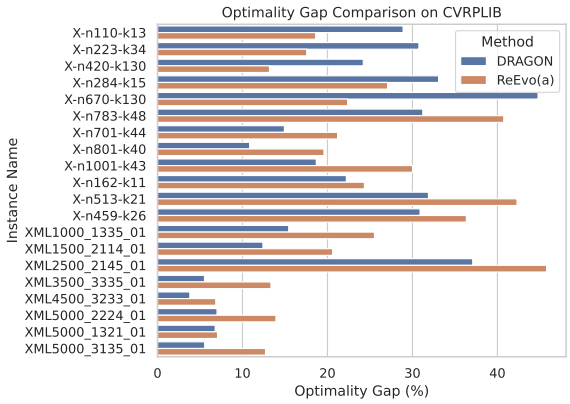

Figure 5: Optimality Gap (%) for methods on CVRPLIB.

Table 2: Domain expertise levels across decomposition–reconstruction strategy combinations. Each cell shows expertise needed: ✗ (none), ✓ (some), ✓✓ (extensive) for decomposition and reconstruction, respectively.

Decomposition	Reconstruction		
	Heuristic	Solver	LLM
Random	(✗, ✓)	(✗, ✓✓)	(✗, ✗)
Heuristic	(✓, ✓)	(✓, ✓✓)	(✓, ✗)
LLM	(✗, ✓)	(✗, ✓✓)	(✗, ✗)

encounter large input loads. Code-generation methods like ReEvo improve scalability. We tested ReEvo's released heuristic code, including both constructive (ReEvo(c)) and ACO-enhanced (ReEvo(a)) variants. On TSPLIB, DRAGON consistently outperforms ReEvo(c) across all size groups and also surpasses ReEvo(a) on all groups beyond 500, and achieving an average gap reduction of 0.86%. On CVRPLIB, where OR-Tools fails to find feasible solutions within time limit, DRAGON leads around 3.98% lower gaps than ReEvo(a) on all cases. Overall, DRAGON demonstrates the best average performance across routing benchmarks, validating its effectiveness and scalability.

4.3 Ablation Study

4.3.1 Strategies for Decomposition and Reconstruction. DRAGON supports configurable strategies for each task, allowing us to evaluate the impact of different implementations. Table 2 presents a 3×3 combination of decomposition and reconstruction strategies, along with the domain expertise required for each pair, offering insight into their implementation difficulty. For decomposition, "Random" selects elements arbitrarily, "Heuristic" applies domain-specific rules, and "LLM" leverages large language models. Similarly for strategies in reconstruction, the only new word "Solver" refers to implement general purpose solver. Figure 6 shows the performance of DRAGON on a TSPLIB subset using these nine combinations.

Among reconstruction strategies, the solver-based approach achieves the best performance by optimally solving small size local

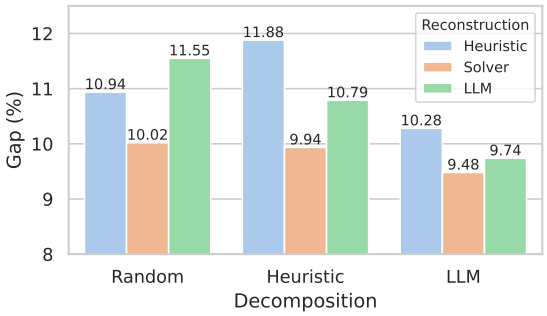

Figure 6: Ablation study of DRAGON across 9 decomposition–reconstruction combinations as listed in Table 2. Bars show the optimality gap (%) compared to the known optimal value on a TSPLIB subset.

COPs. However, this method has key drawbacks: (1) in routing problems for example, enforcing must-visit edges via zero distances with penalty trick may not lead to the constraint satisfied outcomes, and (2) encoding custom constraints across diverse COPs requires substantial domain expertise, as reflected in Table 2. Despite requiring the most domain knowledge, the "Heuristic×Solver" combination does not yield the best results. In contrast, our DRAGON pipeline "LLM×LLM" outperforms it, highlighting the effectiveness of LLM-based decomposition agent in capturing promising substructures. Notably, reconstruction agent offers ease of implementation and integrates naturally with decomposition agent. Hence, DRAGON strikes a strong balance between performance and usability.

4.3.2 Impact of LLM backbones. As shown in Figure 7, we evaluate the impact of different LLM backbones within the DRAGON framework, including general-purpose models (OpenAI `gpt-4o`, `gpt-4.1`) and reasoning models (OpenAI `o3`, DeepSeek `r1`). Due to cost considerations, we do not evaluate all TSPLIB instances. As `r1` exceeds the model's input token limit of 65,536 on instances larger than 10k nodes, it fails to produce responses in the largest case (listed in Appendix (Experiment results)).

While `o3` achieves the lowest average optimality gap, it incurs extremely high output token counts. Given that OpenAI's API typically prices output tokens at 4 times the rate of input tokens, this leads to significantly higher costs and longer inference time. `r1` exhibits a relatively low number of API calls, suggesting that each call takes longer—likely due to more intensive reasoning per iteration. Considering the trade-off between objective quality, inference speed, and token efficiency, `gpt-4.1` achieves the best overall balance and emerges as the current most practical choice.

4.4 Generalization Case Study

To assess the robustness of DRAGON, we evaluate its performance across more domains, it highlights the method's ability to generalization for the subsequent results.

4.4.1 Bin Packing Problem. In addition to routing problems, we evaluate DRAGON on the packing domain. As shown in Table 3, DRAGON consistently outperforms ReEvo(a), FunSearch, EoH, and

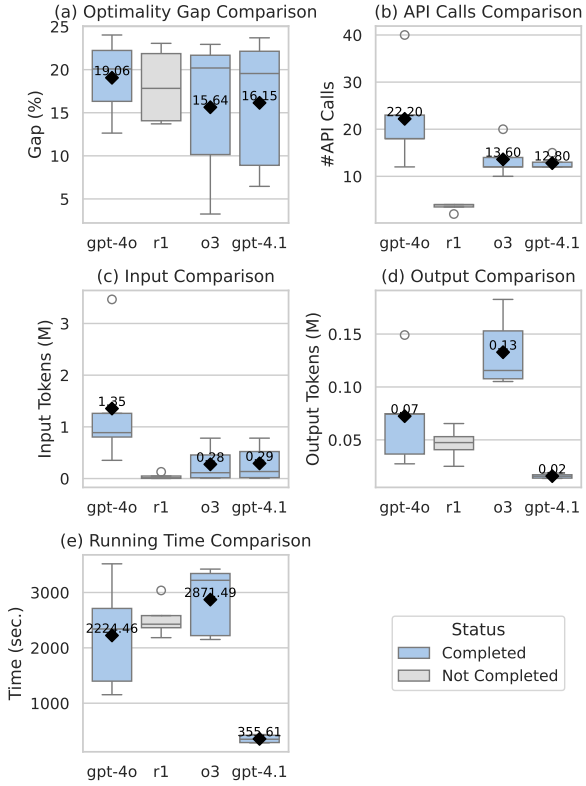

Figure 7: Comparison of LLM models by: (a) optimality gap, (b) number of API calls, (c) input tokens, (d) output tokens, (e) running time. We evaluate general-purpose models (`gpt-4o`, `gpt-4.1`) and reasoning-oriented models (`o3`, `r1`). Gray bars indicate incomplete results due to token limit violations.

Table 3: Average optimality gap (%) w.r.t. L2 lower bound and inference time (sec.) on the Weibull-5K dataset.

Method	ReEvo(a)	FunSearch	EoH	EoH expert	Ours
Gap (%) ↓	3.46	0.69	0.66	0.55	**0.33**
Time (sec.) ↓	56.677	2.292	-	-	487.873

EoH expert on the Weibull-5K dataset. We omit OR-Tools solvers (both MP and CP-SAT backends) as they failed to produce feasible solutions before timeouts.

Compared to routing, packing problems are generally easier for LLMs to reason about due to their more intuitive and straightforward constraints. For example, merging static segments (**s**) into packed items requires less complex reasoning, allowing DRAGON to scale more effectively to larger instances. Results for EoH and EoH expert are reported directly from [23], where running time was not provided. While DRAGON achieves the lowest optimality gap, it is slower than code-generation based methods. However, code-generation approaches come with high API costs and substantial time and resources to evolve or fine-tune high-quality heuristic

functions. These overheads are not captured in current comparison. Each paradigm has its strengths. For large batches of similar instances, code-generation methods are effective as the learned heuristic can be reused across the dataset. In contrast, prompt-based approaches like DRAGON are better suited for dynamic environments where instance structures vary, offering flexibility without the need for retraining or code evolution.

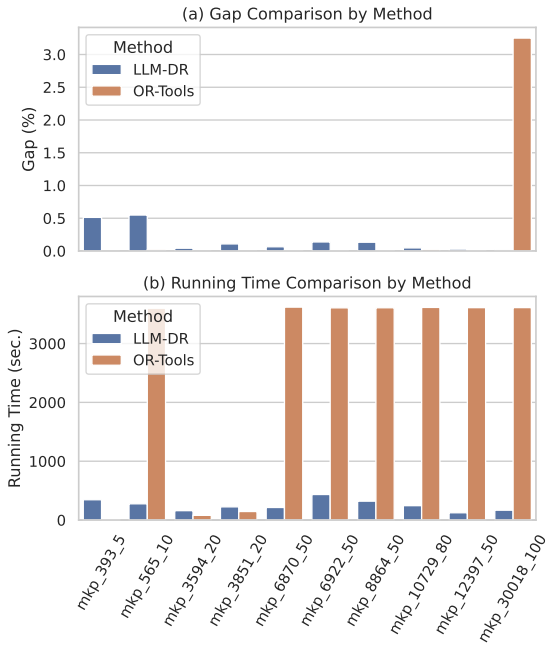

Figure 8: Comparison on synthetic Multiple Knapsack Problem instances between DRAGON and OR-Tools (CP-SAT).

4.4.2 Multiple Knapsack Problem. We compare DRAGON with the CP-SAT on a series of synthetic MKP instances. Each instance is named using the format $\texttt{mkp\_}n_{item}\texttt{\_}n_{knapsack}$, where the decision space grows as $n_{item} \times n_{knapsack}$, hence the maximum of decison size here is around 3M.

As shown in Figure 8, CP-SAT typically achieves lower optimality gaps and can even find optimal solutions when the instance size is manageable. However, its performance starts to degrade on larger instances, evident from the largest case, where the gap spikes to around 3% and it hits the time limit. DRAGON performs favorably across all instances. While its gap is generally slightly higher than CP-SAT on small cases, it remains consistently below 0.5%, even on large-scale instances. In terms of trade-offs between performance and efficiency, DRAGON offers strong scalability and competitive quality, delivering high-quality solutions within reasonable reasoning time, making it highly effective for large-scale MKPs where traditional solvers may struggle.

4.5 Limitations and Future Directions

While DRAGON demonstrates strong potential in solving large-scale COPs, several limitations remain to be addressed. The quality of the reconstructed global solution is sensitive to both the decomposition strategy and the prompt formulation. Inappropriate segmentation or poorly structured prompts may lead to suboptimal or degenerated solutions, emphasizing the need for more robust and adaptive decomposition mechanisms. Moreover, the iterative nature of DRAGON introduces substantial computational overhead due to the large context required during each divide-and-conquer cycle, resulting in extended runtime. Although the current pipeline may not be efficient for small-scale problems, it proves particularly advantageous for super large-scale scenarios where traditional solvers and other learning-based methods struggle to deliver feasible or high-quality solutions across all testing tasks.

DRAGON also demonstrates that purely language-based agents can effectively address large-scale COPs, outperforming other prompt-based or code-generation-based approaches. Additionally, incorporating external optimization tools such as OR-Tools within the reconstruction stage enhances local refinement. However, the use of such tools often demands additional modeling expertise, especially for non-standard constrained sub-problem settings.

In the on-going work, we are integrating more external tools to automatically model customized constraints not only within the reconstruction agent but also into the decomposition process, with the goal of further reducing runtime and API costs while maintaining or improving solution quality. We also envision introducing a central coordination agent to better manage the interaction among decomposition and reconstruction agents, thereby improving the overall consistency and efficiency of the optimization process.

5 CONCLUSION

This paper presents DRAGON, a novel framework that leverages LLM agents to solve large-scale COPs via an iterative decomposition reconstruction process inspired by divide-and-conquer. Experiments across diverse COP domains (e.g., routing and packing) show that DRAGON consistently achieves strong performance, especially on extreme large instances where traditional solvers or basic prompt-based methods often fail due to scalability or timeouts. By decomposing complex problems into context manageable subproblems, DRAGON enables LLMs to reason effectively over intricate structures, identifying promising regions, and reconstructing feasible global solutions. This training-free, modular, and solver-agnostic approach supports flexible constraint handling and adapts easily to diverse COPs without specialized algorithm design. Although not optimized for speed, DRAGON avoids expensive code search and offers a strong balance between solution quality, adaptability, and cost, making it particularly well-suited for dynamic and large-scale settings. Future work will focus on enhancing decomposition strategies, improving prompt robustness, and integrating hybrid solvers to explore promising regions more efficiently. Overall, DRAGON establishes a solid foundation toward general-purpose, interpretable, and scalable LLM-based optimization.

ACKNOWLEDGMENTS

This research/project is supported by the National Research Foundation, Singapore under its AI Singapore Programme (AISG Award No: AISG3-RPGV-2025-017, AISG3-RP-2025-036-USNSF). For Supplementary Materials: https://arxiv.org/abs/2601.06502

REFERENCES

[1] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. 2024. Gpt-4 technical report. *arXiv:2303.08774* (2024). arXiv:2303.08774 [cs.CL]

[2] Ali AhmadiTeshnizi, Wenzhi Gao, and Madeleine Udell. 2024. OptiMUS: Scalable Optimization Modeling with (MI)LP Solvers and Large Language Models. In *International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 235)*, Ruslan Salakhutdinov, Zico Kolter, Katherine Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp (Eds.). PMLR, 577–596.

[3] Florian Arnold, Michel Gendreau, and Kenneth Sörensen. 2019. Efficiently solving very large-scale routing problems. *Computers & operations research* 107 (2019), 32–42.

[4] Giorgio Ausiello, Pierluigi Crescenzi, Giorgio Gambosi, Viggo Kann, Alberto Marchetti-Spaccamela, and Marco Protasi. 2012. *Complexity and approximation: Combinatorial optimization problems and their approximability properties*. Springer Science & Business Media.

[5] Yossi Azar. 1994. Lower bounds for insertion methods for TSP. *Combinatorics, Probability and Computing* 3, 3 (1994), 285–292.

[6] Brenda S Baker. 1985. A new proof for the first-fit decreasing bin-packing algorithm. *Journal of Algorithms* 6, 1 (1985), 49–70.

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

[8] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems* 33 (2020), 1877–1901.

[9] Valentina Cacchiani, Manuel Iori, Alberto Locatelli, and Silvano Martello. 2022. Knapsack Problems — An Overview of Recent Advances. Part II: Multiple, Multidimensional, and Quadratic Knapsack Problems. *Computers & Operations Research* 143 (2022), 105693.

[10] Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, et al. 2023. Palm: Scaling language modeling with pathways. *Journal of Machine Learning Research* 24, 240 (2023), 1–113.

[11] Francesca Da Ros, Michael Soprano, Luca Di Gaspero, and Kevin Roitero. 2025. Large Language Models for Combinatorial Optimization: A Systematic Review. *arXiv:2507.03637* (2025).

[12] Mohammed Elhenawy, Ahmed Abdelhay, Taqwa I Alhadidi, Huthaifa I Ashqar, Shadi Jaradat, Ahmed Jaber, Sebastien Glaser, and Andry Rakotonirainy. 2024. Eyeballing Combinatorial Problems: A Case Study of Using Multimodal Large Language Models to Solve Traveling Salesman Problems. *arXiv:2406.06865* (2024), 341–355.

[13] Mohammed Elhenawy, Ahmad Abutahoun, Taqwa I Alhadidi, Ahmed Jaber, Huthaifa I Ashqar, Shadi Jaradat, Ahmed Abdelhay, Sebastien Glaser, and Andry Rakotonirainy. 2024. Visual Reasoning and Multi-Agent Approach in Multimodal Large Language Models (MLLMs): Solving TSP and mTSP Combinatorial Challenges. *Machine Learning and Knowledge Extraction* 6, 3 (2024), 1894–1920.

[14] Shaodi Feng, Zhuoyi Lin, Jianan Zhou, Cong Zhang, Jingwen Li, Kuan-Wen Chen, Senthilnath Jayavelu, and Yew-Soon Ong. 2025. Lifelong Learner: Discovering Versatile Neural Solvers for Vehicle Routing Problems. *arXiv:2508.11679* (2025).

[15] Vincent Furnon and Laurent Perron. 2025. OR-Tools Routing Library. https://developers.google.com/optimization/routing/. Version 9.12, Google.

[16] Yuxiao Huang, Wenjie Zhang, Liang Feng, Xingyu Wu, and Kay Chen Tan. 2024. How multimodal integration boost the performance of llm for optimization: Case study on capacitated vehicle routing problems. *arXiv:2403.01757* (2024).

[17] Zangir Iklassov, Yali Du, Farkhad Akimov, and Martin Takac. 2024. Self-Guiding Exploration for Combinatorial Problems. *Advances in Neural Information Processing Systems* (2024).

[18] Xia Jiang, Yaoxin Wu, Minshuo Li, Zhiguang Cao, and Yingqian Zhang. 2025. Large Language Models as End-to-end Combinatorial Optimization Solvers. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*.

[19] Xia Jiang, Yaoxin Wu, Yuan Wang, and Yingqian Zhang. 2024. Bridging Large Language Models and Optimization: A Unified Framework for Text-attributed Combinatorial Optimization. *arXiv:2408.12214* (2024).

[20] Xia Jiang, Yaoxin Wu, Chenhao Zhang, and Yingqian Zhang. 2025. DRoC: Elevating large language models for complex vehicle routing via decomposed retrieval of constraints. In *International Conference on Learning Representations*.

[21] Sirui Li, Zhongxia Yan, and Cathy Wu. 2021. Learning to delegate for large-scale vehicle routing. In *Advances in Neural Information Processing Systems*, Vol. 34. 26198–26211.

[22] Zhuoyi Lin, Yaoxin Wu, Bangjian Zhou, Zhiguang Cao, Wen Song, Yingqian Zhang, and Senthilnath Jayavelu. 2024. Cross-problem learning for solving vehicle routing problems. In *Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence*. 6958–6966.

[23] Fei Liu, Xialiang Tong, Mingxuan Yuan, Xi Lin, Fu Luo, Zhenkun Wang, Zhichao Lu, and Qingfu Zhang. 2024. Evolution of heuristics: towards efficient automatic algorithm design using large language model. In *Proceedings of the 41st International Conference on Machine Learning* (Vienna, Austria) *(ICML'24)*. JMLR.org, Article 1304, 23 pages.

[24] Shengcai Liu, Caishun Chen, Xinghua Qu, Ke Tang, and Yew-Soon Ong. 2024. Large language models as evolutionary optimizers. In *2024 IEEE Congress on Evolutionary Computation (CEC)*. IEEE, 1–8.

[25] Yining Ma, Zhiguang Cao, and Yeow Meng Chee. 2023. Learning to search feasible and infeasible regions of routing problems with flexible neural k-opt. *Advances in Neural Information Processing Systems* 36 (2023), 49555–49578.

[26] Mazin Abed Mohammed, Mohd Khanapi Abd Ghani, Raed Ibraheem Hamed, Salama A Mostafa, Dheyaa Ahmed Ibrahim, Humam Khaled Jameel, and Ahmed Hamed Alallah. 2017. Solving vehicle routing problem by using improved K-nearest neighbor algorithm for best solution. *Journal of Computational Science* 21 (2017), 232–240.

[27] Laurent Perron and Frédéric Didier. 2025. CP-SAT Solver. https://developers.google.com/optimization/cp/cp_solver/. Version 9.12, Google.

[28] Laurent Perron and Vincent Furnon. 2025. OR-Tools. https://developers.google.com/optimization/. Version 9.12, Google.

[29] David Pisinger and Stefan Ropke. 2018. Large neighborhood search. In *Handbook of metaheuristics*. Springer, 99–127.

[30] Eduardo Queiroga, Ruslan Sadykov, Eduardo Uchoa, and Thibaut Vidal. 2022. 10,000 optimal CVRP solutions for testing machine learning based heuristics. In *AAAI-22 Workshop on Machine Learning for Operations Research (ML4OR)*.

[31] Gerhard Reinelt. 1991. TSPLIB—A traveling salesman problem library. *ORSA journal on computing* 3, 4 (1991), 376–384.

[32] Nizar Rokbani, Raghvendra Kumar, Ajith Abraham, Adel M Alimi, Hoang Viet Long, Ishaani Priyadarshini, and Le Hoang Son. 2021. Bi-heuristic ant colony optimization-based approaches for traveling salesman problem. *Soft Computing* 25, 5 (2021), 3775–3794.

[33] Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Matej Balog, M Pawan Kumar, Emilien Dupont, Francisco JR Ruiz, Jordan S Ellenberg, Pengming Wang, Omar Fawzi, et al. 2024. Mathematical discoveries from program search with large language models. *Nature* 625, 7995 (2024), 468–475.

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

[35] Zhiqing Sun and Yiming Yang. 2023. DIFUSCO: Graph-based Diffusion Solvers for Combinatorial Optimization. In *Advances in Neural Information Processing Systems*.

[36] Eduardo Uchoa, Diego Pecin, Artur Pessoa, Marcus Poggi, Thibaut Vidal, and Anand Subramanian. 2017. New Benchmark Instances for the Capacitated Vehicle Routing Problem. *European Journal of Operational Research* 257, 3 (2017), 845–858.

[37] Heng Wang, Shangbin Feng, Tianxing He, Zhaoxuan Tan, Xiaochuang Han, and Yulia Tsvetkov. 2024. Can language models solve graph problems in natural language? *Advances in Neural Information Processing Systems* 36 (2024).

[38] Segev Wasserkrug, Leonard Boussioux, Dick den Hertog, Farzaneh Mirzazadeh, Ilker Birbil, Jannis Kurtz, and Donato Maragno. 2024. From Large Language Models and Optimization to Decision Optimization CoPilot: A Research Manifesto. *arXiv:2402.16269* (2024).

[39] Michael Wilson, Jackson Petty, and Robert Frank. 2023. How Abstract Is Linguistic Generalization in Large Language Models? Experiments with Argument Structure. *Transactions of the Association for Computational Linguistics* 11 (Nov. 2023), 1377–1395.

[40] Ziyang Xiao, Dongxiang Zhang, Yangjun Wu, Lilin Xu, Yuan Jessica Wang, Xiongwei Han, Xiaojin Fu, Tao Zhong, Jia Zeng, Mingli Song, et al. 2023. Chain-of-Experts: When LLMs Meet Complex Operations Research Problems. In *International Conference on Learning Representations*.

[41] Chengrun Yang, Xuezhi Wang, Yifeng Lu, Hanxiao Liu, Quoc V Le, Denny Zhou, and Xinyun Chen. 2024. Large Language Models as Optimizers. In *International Conference on Learning Representations*.

[42] Haoran Ye, Jiarui Wang, Zhiguang Cao, Federico Berto, Chuanbo Hua, Haeyeon Kim, Jinkyoo Park, and Guojie Song. 2025. ReEvo: large language models as hyper-heuristics with reflective evolution. In *Proceedings of the 38th International Conference on Neural Information Processing Systems* (Vancouver, BC, Canada) *(NIPS '24)*. Curran Associates Inc., Red Hook, NY, USA, Article 1381, 38 pages.

[43] Jie Zhao, Tao Wen, and Kang Hao Cheong. 2025. Can Large Language Models Be Trusted as Evolutionary Optimizers for Network-Structured Combinatorial Problems? *arXiv:2501.15081* (2025).

[44] Jianan Zhou, Yaoxin Wu, Wen Song, Zhiguang Cao, and Jie Zhang. 2023. Towards omni-generalizable neural methods for vehicle routing problems. In *Proceedings of the 40th International Conference on Machine Learning (ICML'23)*.

[45] Jianghan Zhu, Yaoxin Wu, Zhuoyi Lin, Zhengyuan Zhang, Haiyan Yin, Zhiguang Cao, Senthilnath Jayavelu, and Xiaoli Li. 2025. Bridging Synthetic and Real Routing Problems via LLM-Guided Instance Generation and Progressive Adaptation. In *The 40th Annual AAAI Conference on Artificial Intelligence*.