GraphThought: Graph Combinatorial Optimization with Thought Generation

Anonymous ACL submission

Abstract

Graph combinatorial optimization (GCO) is critical across scientific and industrial domains. While Large Language Models (LLMs) show potential for structured reasoning in GCO, they often struggle with rigorous multi-step deduction and produce hallucinations. To address this, we first formalize the *Optimal Thoughts Design* (OTD) problem to provide structured guidance for intermediate reasoning. Building upon this formulation, we introduce *GraphThought*, a framework that generates high-quality thought sequences via either heuristic-guided *forward* search or solver-aligned *backward* reasoning. Fine-tuned on these sequences[1] yields Llama-GT (8B), which achieves state-of-the-art performance on the GraphArena benchmark, outperforming significantly larger models like DeepSeek-V3. Our results demonstrate that structured reasoning priors can significantly enhance LLM performance on GCO tasks without increasing model scale.

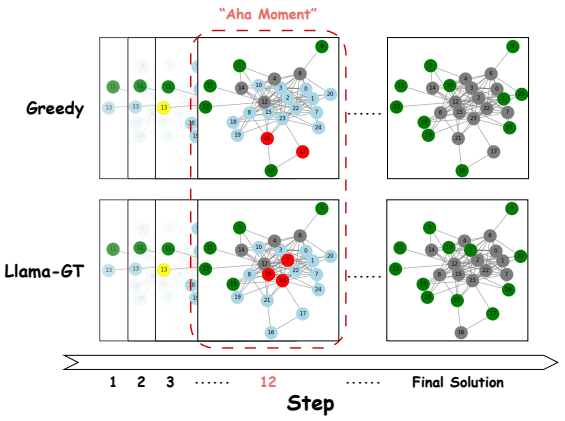

Figure 1: Comparative Analysis of Greedy Algorithm versus LLM-Generated Solution for MIS. Upper: Stepwise greedy selection process (e.g., selecting node 16, then removing neighbors 21 and 17), yielding an **11**-node solution. Lower: Llama-GT-generated solution employing adaptive heuristics (e.g., prioritizing node 18 and removing neighbors 2, 15, 23), achieving a superior **12**-node independent set. *Color legend:* yellow = isolated nodes; green = selected nodes; red = neighbors of current selection; gray = removed nodes.

1 Introduction

Graph combinatorial optimization problems constitute a broad class of computationally challenging tasks defined over graph structures, ranging from shortest-path and optimal tour routing to selecting maximum independent sets (MIS) or minimum vertex covers (MVC). These problems, which have driven fundamental advances in discrete mathematics and computer science for decades (Kuhn, 1955; Kruskal, 1956; Jr. and Fulkerson, 1956), manifest across diverse application domains including logistics optimization, electronic circuit design, bioinformatics, and social network analysis (Tang et al., 2025). Many canonical GCO tasks—most notably the traveling salesman problem (TSP) and MIS—remain NP-hard, motivating ongoing research into both exact and approximation techniques.

Traditional approaches to GCO have predominantly relied on human-designed algorithms and heuristics. Over decades, researchers have developed various methodologies including greedy algorithms, local search techniques, branch-and-bound approaches, and approximation schemes, which demonstrate notable effectiveness on moderate-scale instances (Kuhn, 1955; Jr. and Fulkerson, 1956; Kruskal, 1956; Paschos, 2014). While such manually engineered approaches often produce satisfactory solutions, they require substantial manual effort and domain-specific expertise. Furthermore, heuristics constrained by fixed patterns (e.g., selecting minimum-degree nodes in MIS) often fail to account for global optimality, potentially leading to suboptimal outcomes through locally optimal

[1]Dataset available at https://anonymous.4open.science/r/GraphThought-7CFE.

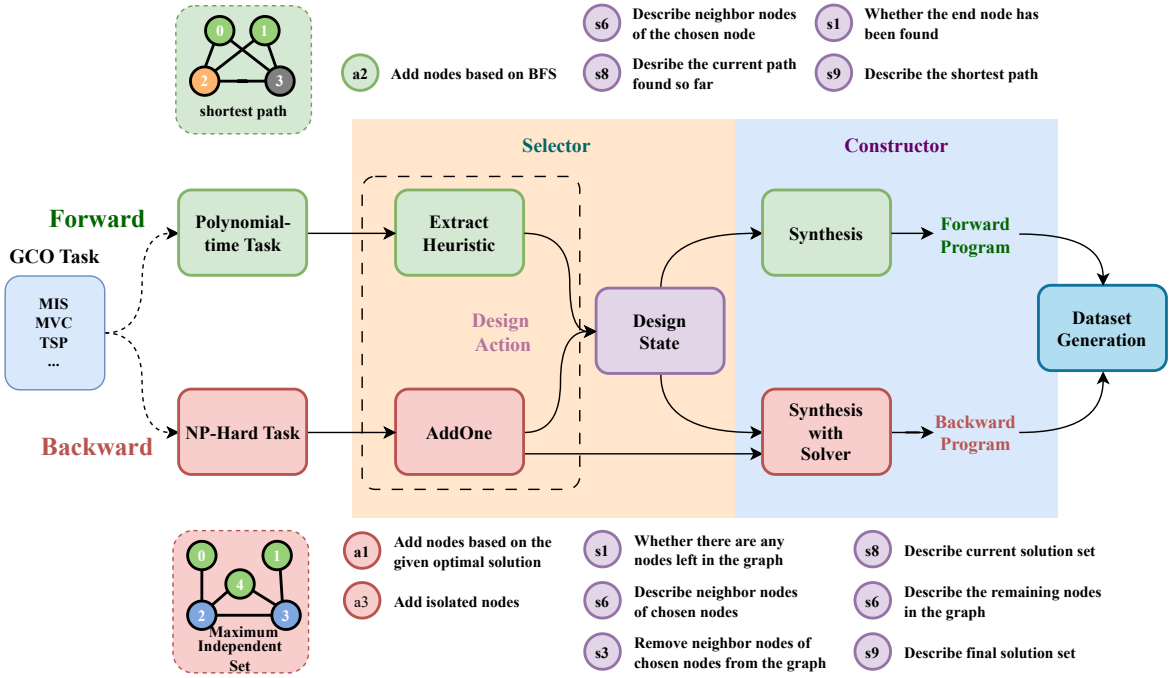

Figure 2: The *Forward* and *Backward* MTP frameworks for constructing the **Selector** and **Constructor** modules in GraphThought through distinct pathways. The **Selector** extracts action-state pairs from spaces $\mathbb{A}$ and $\mathbb{S}$, while the **Constructor** synthesizes an executable program. Two paradigm examples are shown: (1) Forward MTP for polynomial problems (e.g., shortest path) uses heuristic extraction of action thoughts (e.g., BFS-based node addition) to guide state selection; (2) Backward MTP for NP-hard problems (e.g., MIS) reconstructs reasoning steps through solver-derived solutions and reverse analysis. Both frameworks generate programs via systematic composition of thought sets, with backward MTP requiring combinatorial solvers.

choices, which can be observed in Figure 1.

In recent years, deep neural network–based approaches have attracted considerable attention for addressing GCO problems by learning solutions in a data-driven manner. Methods based on deep reinforcement learning and graph neural networks have demonstrated the ability to learn heuristics for specific tasks like TSP and MIS, often treating the problem solver as a black-box model optimized end-to-end (Jin et al., 2024b; Jiang et al., 2024b; Liu et al., 2023c; Iklassov et al., 2024; Lehnert et al., 2024b). However, a major drawback is that each problem typically requires a specialized network architecture or input encoding – for instance, a model architecture tailored for routing problems may not directly work for MIS, necessitating bespoke representation engineering per task (Jiang et al., 2024b; Jin et al., 2024b). Moreover, pure learning-based solvers often fail to exploit problem-specific priors that humans would incorporate.

Recent research leverages LLMs for GCO through natural language reasoning, utilizing embedded algorithmic knowledge for stepwise solution generation (Wang et al., 2022; Ren et al., 2024b; Li et al., 2024a; Liu et al., 2023b; Luo et al., 2024a). Although this paradigm offers flexibility and promising zero-shot performance on small-scale tasks (Li et al., 2024a; Wang et al., 2022), LLMs significantly trail dedicated solvers on NP-hard problems like MIS and graph coloring, often yielding suboptimal or invalid results (Tang et al., 2025; Wang et al., 2022). Persistent challenges include hallucinated reasoning and unreliable deduction despite structured prompting (Luo et al., 2024a; Ouyang et al., 2024a; Sanford et al., 2024b; Li et al., 2024b), leaving LLMs substantially inferior to commercial solvers like Gurobi (Jin et al., 2024a; Chen et al., 2024b; Liu et al., 2023a; Ren et al., 2024a). This motivates a fundamental question: *Can we incorporate the search principles of classical GCO solvers into the LLM's output process?*

Related ideas have been proposed to enhance the general problem-solving capabilities of LLMs, such as simulating human cognitive processes by generating intermediate thoughts prior to final responses. Methods like chain- (Wei et al., 2022), tree- (Yao et al., 2024) and graph-of-

thoughts (Besta et al., 2024a) prompting encourage step-by-step reasoning (Besta et al., 2024b). While these techniques are often effective, they can sometimes degrade performance due to self-enforcing (Huang et al., 2024). Moreover, techniques that work well on one dataset may not generalize to others due to variations in the type of reasoning involved (e.g., spatial reasoning vs. mathematical reasoning) (Lehnert et al., 2024a; Wu et al., 2024). Similar limitations appear in recent GCO-focused works (Luo et al., 2024b; Lehnert et al., 2024a; Chen et al., 2024a; Zhang et al., 2024; Ouyang et al., 2024b; Gandhi et al., 2024), which construct thoughts by unrolling search algorithms or generating them via LLMs, followed by supervised fine-tuning. However, thought generation in GCO presents a unique challenge: *For NP-hard or NP-complete problems, no efficient traditional (heuristic) search methods exist, rendering forward thought construction infeasible.*

To address these challenges, we formalize the **Optimal Thoughts Design (OTD)** problem, which systematically encodes search principles into LLM reasoning via action ($\mathbb{A}$) and state ($\mathbb{S}$) thought spaces. Building on this formulation, we propose **GraphThought**, a framework featuring dual generation paradigms: a forward mode utilizing heuristic-guided mechanisms for tractable problems and a backward mode implementing solver-guided backtracking for NP-hard challenges.

Furthermore, to demonstrate the practical efficacy of our approach, we develop **Llama-GT** by fine-tuning the Llama-3-8B-Instruct model on reasoning trajectories generated by GraphThought. Empirical evaluations across diverse GCO tasks demonstrate that Llama-GT significantly improves performance, approaching the optimality of commercial solvers like Gurobi while maintaining the flexibility of LLM-based reasoning.

The main contributions are as follows: 1) We formalize the OTD problem with state thought space $\mathbb{S}$ and action thought space $\mathbb{A}$, which facilitates the systematic generation of reasoning thoughts. 2) Within the GraphThought framework, we propose dual thought generation frameworks: a forward (heuristic-guided) one and a backward (solver-guided) one for GCO problems. 3) The state-of-the-art performance of our approach is evidenced by the fine-tuning of Llama-3-8B-Instruct, which achieves superior accuracy on GraphArena, outperforming both proprietary and open-source LLMs by significant margins. and approaching the performance of the commercial solver Gurobi.

Remark. While direct end-to-end processing of massive graphs is currently constrained by LLM context windows and potential information loss in embedding-based approaches (Li et al., 2025; Wei et al., 2025), we posit that the future of large-scale GCO lies in treating graphs as interactive environments where agents utilize tools for exploration. In this context, *GraphThought* addresses a fundamental necessity: it instills the rigorous structural priors and reasoning logic—essentially the "cognitive engine"—that future tool-augmented agents require to effectively navigate and optimize within large-scale graph environments.

2 Problem Formulation

Combinatorial optimization (CO) involves identifying optimal solutions from discrete candidate sets. Traditional approaches typically utilize handcrafted heuristics derived from rigorous analysis to balance solution quality with computational efficiency. A key challenge, however, is to integrate these heuristic principles into the reasoning processes of LLMs, facilitating knowledge transfer without compromising algorithmic efficiency.

We focus on GCO problems, where graphs $G = (V, E)$ naturally model complex discrete structures with nodes $V = \{v_1, \ldots, v_n\}$ and edges $E = \{e_1, \ldots, e_m\}$. Formally, let $\mathcal{F} : (\mathrm{LLM}, D) \to \mathrm{LLM}_D$ represent the supervised fine-tuning process that maps a foundation LLM to a task-specific model LLM_D using training data D. Performance is then assessed via an evaluation metric $\mathcal{M} : \mathrm{LLM}_D \to \mathbb{R}^+$. The core optimization challenge is formalized as:

$$D^* = \arg\min_D \mathcal{M}\big(\mathcal{F}(\mathrm{LLM}, D)\big), \qquad (1)$$

where D^* is the optimal training dataset given LLM, $\mathcal{F}$, and $\mathcal{M}$. The formulation (1) constitutes a CO challenge over the exponentially large space of possible training datasets D. Three key difficulties emerge: 1) The discrete solution space prohibits gradient-based optimization; 2) The black-box nature of $\mathcal{M}$ prevents analytical evaluation; 3) The computational cost of evaluating $\mathcal{M}$ grows superlinearly with instance size. It becomes important to efficiently compute high-quality approximate optimal solutions of (1).

Thoughts of solving specific instances of math problems can be viewed as high-quality training

data for fine-tuning LLMs and thus developing enhanced thoughts is critical for the creation of superior training datasets. In the following, we will propose a series of thought generation methods to establish thoughts as one kind of approximate suboptimal solutions for the formulation (1).

The representation and design of thoughts are typically a promising research direction in order to enhance the reasoning capabilities of LLMs. The thought generation process can be modeled as to solve an approximate optimal solution $D' \subset \hat{D}$ for the formulation (1), where $\hat{D}$ denotes the set of all training data with thoughts as content. For GCO problems, there are mainly two kinds of thoughts. One kind is to decide the following available action executed on the instance and the other kind is to show the current solving state. A hierarchical decision problem in generating these kinds of thoughts is govern as:

$$\begin{cases} \text{Choose optimal } A^* \subset \mathbb{A}, S^* \subset \mathbb{S}, \\ \text{Constructing program } \mathcal{P} \text{ by } A^*, S^*. \end{cases} \tag{2}$$

where $\mathbb{A}$ and $\mathbb{S}$ denote the action thought space and state thought space for a GCO problem, respectively. The first item is to choose optimal A^* from $\mathbb{A}$ and optimal S^* from $\mathbb{S}$ according to the characteristics of the GCO problem, where A encapsulates core algorithmic operations while S maintains dynamic problem-solving states. The second item is to incorporate all chosen thoughts of A^* and S^* into a program template $\mathcal{P}$, which is used to generate specific thoughts in solving a GCO instance.

In summary, the OTD problem is to generate the optimal thought set A^* and S^* to construct a program $\mathcal{P}$ for a GCO problem, which is the key problem to be addressed in this work. To emphasize, the optimal thoughts dataset in OTD might not be the optimal solution of the formulation (1), but it can be considered as a high-quality approximate solution.

3 The GraphThought Framework

3.1 The Overall Framework

GraphThought is introduced as a novel framework for fine-tuning LLMs through reasoning thought generation, as formalized in Algorithm 1. The architecture of the framework comprises two core modules that collaboratively generate task-specific reasoning programs by selecting state and action thoughts and constructing programs, denoted as **Selector** and **Constructor**, respectively.

- **Selector**: For a given task τ, this module performs dynamic selection of *action-state pairs* (A, S) from predefined action and state thought space $\mathbb{A}$ and $\mathbb{S}$. The selection mechanism adaptively adjusts its strategies to capture the essential characteristics of τ, ensuring context-aware component selection;
- **Constructor**: This module operates as a program synthesis engine that methodically assembles the selected (A, S) into an executable program $\mathcal{P}$.

For the predefined action thought space $\mathbb{A}$ and state thought space $\mathbb{S}$, we systematically derive nine state representations and sixteen action operators through rigorous analysis of task-solving processes. These elements constitute the fundamental components of our framework, where $\mathbb{A}$ primarily represents operations on the solution set (e.g., adding nodes or edges), while $\mathbb{S}$ captures both graph operations (e.g., node insertion/removal) and state descriptions (e.g., remaining nodes/edges, current solution status). Complete specifications are provided in Appendix G.

The synthesized program $\mathcal{P}$ serves dual objectives: 1) as an algorithmic solver for task τ, and 2) as a structured data generator for creating reasoning thoughts. Through iterative execution on instances set $\tilde{\mathcal{D}}_\tau$, where $|\tilde{\mathcal{D}}_\tau| = n$, $\mathcal{P}$ produces reasoning thoughts set $T = \{T_i\}_{i=1}^n$ that form the training corpus for LLM fine-tuning.

Algorithm 1 Thought-Enhanced LLM Fine-Tuning

Require: Target task τ, problem instances $\tilde{\mathcal{D}}_\tau$, $\mathbb{A}$, $\mathbb{S}$, foundation model $\tilde{\text{LLM}}$;

```
(A, S) ← Selector(τ, 𝔸, 𝕊);   ▷ Action/State set selection
P ← Constructor(τ, A, S);   ▷ Program synthesis
Initialize thought corpus T ← ∅;
for i = 1 to |D̃_τ| do
    Select an instance I_i ← D̃_τ[i];
    Generate thought T_i ← P(I_i);   ▷ Program execution
    T ← T ∪ {T_i};
end for
return LLM_fine-tuned ← F(LL̃M, T).
```

To achieve the functionalities of **Selector** and **Constructor**, we propose **Meta-Thought Programming (MTP)**, a systematic methodology for generating $(A, S, \mathcal{P})$ triples through structured knowledge extraction. For GCO problems, we develop two MTP frameworks as follows:

- **Forward MTP**: decomposes classical heuristic algorithms to extract fundamental reasoning patterns through constructive forward-chaining;
- **Backward MTP**: discovers implicit reasoning principles via backward analysis of high-quality solutions.

The GraphThought framework, designed to address the OTD problem, embodies the dual-process framework depicted in Figure 2. The proposed framework consists of two independent components: the forward one illustrated in the upper half of the figure, whereas the backward one is correspondingly demonstrated in the lower half. The architectural components will be discussed in the following subsections, collectively establishing the solution of OTD problem.

3.2 Forward MTP Framework

The green-highlighted modules in Figure 2 formally establish the forward MTP framework. For a given GCO task τ, this framework systematically extracts several classical heuristic algorithms. These algorithms are systematically combined to distill fundamental operations from $\mathbb{A}$, thereby constructing the action thought set A. Concurrently, the corresponding state thought set S is axiomatically derived from $\mathbb{S}$ with A and τ. Following this procedural logic, the target program $\mathcal{P}$ is synthesized via categorical composition of selected A-S pairs. The complete generation mechanisms of A and S is rigorously detailed in Appendix F.

A basic program template of $\mathcal{P}$ is structured in Program Template 1. It sequentially applies action thoughts in A followed by displaying each state of S. This template may require domain-specific adaptations depending on the task τ. An application of the forward MTP framework for the connected components problem is introduced in Appendix H.1.

3.3 Backward MTP Framework

High-quality GCO solutions often embed intelligent problem solving strategies. Although some established knowledge has guided heuristic and approximation algorithm design, many complex patterns remain undiscovered due to analytical complexity. The representational capacity of deep networks enables them to encode such optimal solution patterns, making it promising to generate thoughts for LLMs using approximation or optimal solvers' guidance.

Program 1 A Forward MTP Program Template

Require: Instance I, state and action thought set S, A.

Initialize an empty solution $x \leftarrow \emptyset$
Initialize a solving flag $flag \leftarrow False$
while $flag$ **do**
  for $a \in A$ **do**
    Update solution x according to a; ▷ Action thought application with heuristic methods
    for $s \in S$ **do**
      Display state s; ▷ State thought application
    end for
    Update instance I;
  end for
  Update $flag$; ▷ Update the iteration condition
end while

The red-highlighted modules in Figure 2 structurally define the backward MTP framework. This architecture employs high-quality solvers (including both exact and approximate methods) to obtain high-quality solutions from which we can extract solution construction patterns. The action thought set is constrained to a single operator, AddOne, which incrementally incorporates solution elements into the current partial solution, thereby generating stepwise trajectories. The state design methodology maintains consistency with the forward framework in Section 3.2. The program $\mathcal{P}$ integrates a task-specific solver $\mathcal{X}$ with the thought-generation process through the Program Template 2. An application of the backward MTP framework for the MIS problem is shown in Appendix H.2.

4 Experiments

4.1 Experiments Setting

With generated reasoning datasets, we fine-tuned Meta-Llama-3-8B-Instruct[2] using Low-Rank Adaptation (LoRA) via the llama-factory framework[3], naming the resulting model **Llama-GT** (GT means GraphThought). The training process is conducted with 30K instruction-following examples. Full hyperparameters are placed in Appendix B.

During the inference phase, we deployed the base untrained and trained model using the vllm framework, including Llama-GT, the original Meta-Llama-3-8B-Instruct model and the reasoning mod-

[2] https://huggingface.co/meta-llama/Meta-Llama-3-8B-Instruct

[3] https://github.com/hiyouga/LLaMA-Factory

Program 2 A Backward MTP Program Template

Require: Instance I, action thought set A, state thought set S, a solver $\mathcal{X}$ of τ.

Initialize an optimal or suboptimal solution $\hat{x} \leftarrow \mathcal{X}(I)$
Initialize an empty solution $x \leftarrow \emptyset$
Initialize a solving flag $flag \leftarrow False$
while $flag$ **do**
  $e \leftarrow \text{AddOne}(\hat{x})$; ▷ Add element from standard solution
  $x \leftarrow x \cup \{e\}$;
  for $s \in S$ **do**
    Display state s; ▷ State thought application
  end for
  Update instance I;
  Update $flag$; ▷ Update the iteration condition
end while

els (QWQ-32B[4] and DeepSeek-R1-Distill-Llama-8B[5]). For deployed models, we use `vLLM` [6] to accelerate. Other models utilized API-based inference (including Deepseek-V3, llama-3.3-70b and so on).

To ensure reproducibility, we set the temperature to 0.1 for all models and performed single-pass inference for each test instance. For Best-of-N (BoN) experiments investigating whether increased inference-time computation enhances model performance, we adjusted the temperature to 1.0 and the batch size to 16/32 to maximize response diversity.

We evaluate performance using the *optimality* metric across all ten tasks predefined in the `GraphArena` benchmark (Tang et al., 2025). In accordance with the benchmark's protocol, graphs are categorized into small and large scales, with comprehensive task descriptions, scale definitions, and optimality criteria provided in Appendix B.2.

4.2 Main Results

We evaluate **Llama-GT** against four groups of baselines across all GraphArena tasks: (i) the original GraphArena results (incl. DeepSeek-V2-Coder, GPT-4o-Coder, Qwen2-7B-SFT), (ii) few-shot thought prompting, (iii) iterative self-improvement via STaR (Zelikman et al., 2022), and (iv) ablations without the thought mechanism. Unless otherwise stated, all models are tested on 500 instances per task using vLLM under identical settings. Full quantitative results appear in Table 3.

Comparison with GraphArena Llama-GT significantly outperforms GraphArena baselines, particularly on small graphs where it achieves near-perfect performance on polynomial tasks such as Neighbor (1.0), Distance (1.0), and Diameter (0.954). On large-scale NP-hard tasks, Llama-GT demonstrates a 60–90% improvement over GPT-4o (e.g., MIS: 0.900 vs. 0.034). While code-augmented models like DeepSeek-V2-Coder and GPT-4o-Coder show intermediate capability on basic tasks, they struggle with complex challenges like Diameter (0.334 optimality). Furthermore, the failure of Qwen2-7B-SFT on MIS (0.054) highlights the limitations of standard supervised fine-tuning compared to thought-enhanced paradigm.

Comparison with Few-shot Thought Prompting While few-shot prompting improves performance, Llama-GT shows greater consistency, especially on large graphs. DeepSeek-V3 achieves perfect scores on simple tasks like Neighbor (1.0) but struggles with MVC (0.366). Llama-GT outperforms DeepSeek-V3 on NP-hard tasks like MVC (0.744 vs 0.120) and MIS (0.900 vs 0.304), demonstrating the superiority of the GraphThought framework over few-shot exemplars.

Comparison with STaR Framework Comparison with STaR (Zelikman et al., 2022) framework's self-generated thought approach reveals critical insights. Using STaR with Llama-3-8B-Instruct under two configurations[7]: (1) few-shot prompting with thoughts generated with GraphThought framework and (2) with `LLM`-generated thoughts. Llama-GT substantially outperforms both variants across all graph sizes, showing 2-3x improvements on large graphs (MVC: 0.744 vs 0.360, MIS: 0.900 vs 0.216). The gap persists even when STaR uses thought-enhanced examples, emphasizing the limits of iterative self-improvement approaches. In addition, STaR(w/GT) performs better than STaR(w/LLM) across nearly all tasks and various graph sizes, further underscoring the importance of thought paradigm over automated generation for training.

Ablation of Thought Mechanism Impact The thought mechanism significantly improves performance on NP-hard tasks. Llama-GT without thought integration still performs well on poly-

[4] https://huggingface.co/Qwen/QwQ-32B
[5] https://huggingface.co/deepseek-ai/DeepSeek-R1-Distill-Llama-8B
[6] https://github.com/vllm-project/vllm
[7] More detailed configuations see appendix B.

Table 1: Performance comparison of optimal solution rates (%) across 10 graph tasks, evaluated on small and large graphs, each has 500 instances. Results are shown for: (1) Original inference results of LLMs from GraphArena (Claude3-haiku, GPT-4o, etc.); (2) Code-augmented models (DeepSeek-V2-Coder, GPT-4o-Coder); (3)Supervised Fine-Tuned Model provided by GraphArena (Qwen2-7B-SFT);(4) Few-shot thought prompting variants (Deepseek-V3, Llama-3.3-70B, etc.); (5) STaR framework implementations with different fewshot strategies; and (6) Our Llama-GT model trained with/without thought mechanisms. Metrics include polynomial tasks (Neighbor, Distance, Connected, Diameter), NP-hard tasks (MVC, MIS, MCP, TSP, MCS, GED). * indicates that the model in the GraphArena paper was trained on fewer data compared to the data used in our model.

Graph Task (Small Graphs)										
Model	Neighbor	Distance	Connected	Diameter	MVC	MIS	MCP	TSP	MCS	GED
Claude3-haiku	0.768	0.580	0.260	0.116	0.336	0.450	0.482	0.242	0.282	0.216
gpt-4o-2024-0513	0.860	0.796	0.794	0.426	0.326	0.518	0.528	0.404	0.406	0.268
Llama3-70b-Instruct	0.674	0.894	0.632	0.248	0.420	0.368	0.428	0.232	0.442	0.316
Llama3-8b-Instruct	0.368	0.412	0.248	0.114	0.318	0.282	0.280	0.162	0.096	0.072
DeepSeek-V2-Coder	0.816	0.894	0.586	0.142	0.176	0.482	0.498	0.276	0.228	0.214
GPT-4o-Coder	0.808	0.654	0.712	0.334	0.296	0.530	0.644	**0.490**	0.508	0.320
Qwen2-7b-SFT*	0.966	0.912	0.888	0.608	0.548	0.702	0.696	0.368	0.000	0.054
+Few-shot Thought										
Deepseek-V3	**1.000**	0.988	**1.000**	0.850	0.366	0.642	0.754	0.370	**0.544**	0.308
QwQ-32B-Preview	0.962	0.848	0.696	0.514	0.262	0.482	0.560	0.310	0.444	0.318
Llama3-8b-Instruct	0.700	0.480	0.502	0.070	0.108	0.248	0.258	0.190	0.222	0.450
+SFT										
STaR(w/ GT)	0.648	0.910	0.522	0.466	0.722	0.640	0.676	0.352	0.308	0.370
STaR(w/ LLM)	0.296	0.662	0.440	0.380	0.688	0.654	0.288	0.282	0.328	0.258
Llama-GT(w/o Thought)	0.988	0.990	0.906	0.820	0.930	0.972	0.906	0.366	0.538	**0.608**
Llama-GT	**1.000**	**1.000**	0.996	**0.954**	**0.972**	**0.994**	**0.952**	0.392	0.496	0.460
Graph Task (Large Graphs)										
Model	Neighbor	Distance	Connected	Diameter	MVC	MIS	MCP	TSP	MCS	GED
Claude3-haiku	0.406	0.358	0.052	0.002	0.076	0.014	0.090	0.000	0.000	0.018
gpt-4o-2024-0513	0.674	0.550	0.370	0.032	0.102	0.034	0.102	0.004	0.000	0.018
Llama3-70b-Instruct	0.434	0.530	0.264	0.034	0.114	0.026	0.106	0.000	0.000	0.008
Llama3-8b-Instruct	0.118	0.234	0.022	0.002	0.064	0.010	0.022	0.000	0.000	0.002
DeepSeek-V2-Coder	0.672	0.632	0.206	0.008	0.080	0.028	0.072	0.000	0.022	0.014
GPT-4o-Coder	0.868	0.684	0.378	0.112	0.110	0.072	0.222	0.028	**0.036**	0.018
Qwen2-7b-SFT*	0.790	0.570	0.230	0.092	0.156	0.054	0.136	0.000	0.000	0.032
+Few-shot Thought										
DeepSeek-V3	**0.992**	0.942	**0.932**	0.448	0.120	0.304	0.290	0.020	0.012	0.020
QwQ-32B-Preview	0.912	0.504	0.498	0.164	0.058	0.106	0.124	0.000	0.002	0.020
Llama3-8b-Instruct	0.604	0.220	0.132	0.002	0.028	0.010	0.038	0.000	0.002	0.026
+SFT										
STaR(w/ GT)	0.374	0.618	0.124	0.080	0.360	0.216	0.134	0.018	0.006	0.026
STaR(w/ LLM)	0.320	0.332	0.090	0.038	0.126	0.124	0.030	0.002	0.004	0.024
Llama-GT(w/o Thought)	0.804	0.864	0.340	0.302	0.652	0.652	0.370	0.024	0.020	**0.068**
Llama-GT	0.988	**0.984**	0.836	**0.600**	**0.744**	**0.900**	**0.634**	**0.036**	**0.036**	0.008

nomial tasks (e.g., Neighbor: 0.988 small, 0.804 large). With thought integration, MVC increases from 0.930 to 0.972 (small) and 0.652 to 0.744 (large), MIS from 0.972 to 0.994 (small) and 0.652 to 0.900 (large). However, performance drops on GED (small: 0.608 to 0.460; large: 0.068 to 0.008). We posit that potential causes underlying this phenomenon will be explored in the later discussion.

4.3 Performance Comparison on Reasoning

We evaluate Llama-GT against QWQ-32B and DeepSeek-R1-Distill-Llama-8B using 500 instances per task via vLLM. As shown in Figure 3, Llama-GT achieves comparable or superior performance in polynomial-time and specific NP-hard tasks, such as MVC and MIS. Crucially, Llama-GT offers a significant efficiency advantage: while conventional reasoningmodels often generate redundant reasoning steps that increase latency and exceed token limits, Llama-GT constrains these processes into structured patterns to accelerate inference without sacrificing quality. Performance on complex CO tasks (e.g., TSP, GED, and MCS) is further analyzed in Appendix J.

4.4 BoN-Enhanced Optimality Rates versus Heuristics

To better evaluate the quality of the obtained solutions, we introduce a generalized optimality ratio metric. For each problem instance I, the ground-truth optimal solution x^* is computed using the Gurobi optimizer, which guarantees exact optimality. The optimality ratio for a solution x on the instance I is formally defined as:

$$\text{Rate}(I, x) = \min\left\{\frac{\phi(x^*)}{\phi(x)}, \frac{\phi(x)}{\phi(x^*)}\right\}, \qquad (3)$$

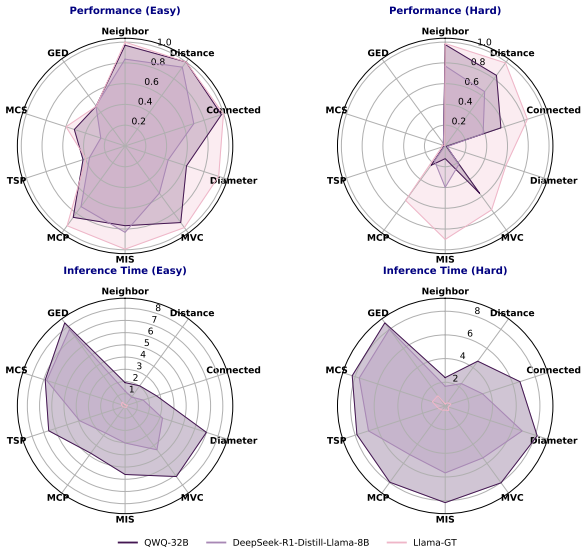

Figure 3: Performance and time cost of inference model and Llama-GT on ten graph tasks of GraphArena benchmark. (Upper-Left) Performance on small graph instances. (Upper-Right) Performance on large ones. (Bottom-Left) Inference time cost on small ones. (Bottom-Right) Inference time cost on large ones.

where $\phi(\cdot)$ is the evaluation function of the solution. This symmetric ratio works for both maximization and minimization tasks. For maximization tasks (e.g., MIS), where $\phi(x) \leq \phi(x^*)$, the ratio $\frac{\phi(x)}{\phi(x^*)}$ directly measures approximation quality within $[0, 1]$. For minimization problems (e.g., MVC), the inverse ratio $\frac{\phi(x^*)}{\phi(x)}$ appropriately penalizes suboptimal solutions while maintaining the same normalized range. This metric enables cross-problem performance comparisons while maintaining the interpretability of near-optimal solutions.

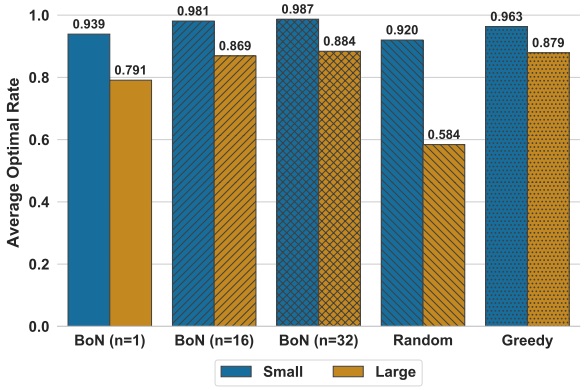

Figure 4: Average performance of Llama-GT with BoN strategy and classic solvers on 6 NP-Hard Tasks for small and large graphs. The values represent the average optimality ratio across different tasks.

Figure 4 demonstrates the average optimality ratio through comprehensive evaluations across $1,000$ graph instances (500 small and 500 large) spanning 6 NP-Hard tasks: GED, MCP, MCS, MIS, MVC and TSP. Our base model ($n = 1$, without Best-of-N enhancement) achieves remarkable optimality ratios of 93.9% on small graphs and 79.1% on large graphs, outperforming random baselines by significant margins of 1.9% and 20.7% respectively. These results demonstrate the inherent effectiveness of our Llama-GT model in generalizing across problem scales and types.

We enhance solution quality with BoN strategy, which generates n solutions per instance and selects the best. With BoN(n=32), optimality improves to 98.7% on small graphs and 88.4% on large graphs, surpassing greedy algorithms (96.3% and 87.9%, respectively). This technique significantly narrows the performance gap between data-driven approaches and manual heuristics, achieving near-optimal results.

A case study in the MIS problem (Figure 1) highlights our approach's superiority. While the greedy algorithm selects low-degree nodes (e.g., node 16), our method uses adaptive heuristics to prioritize high-impact nodes (e.g., node 18) and optimize selections based on evolving graph structures, achieving a 12-node MIS compared to the greedy algorithm's 11. This demonstrates how our framework avoids the myopic decisions of traditional heuristics, leading to higher-quality solutions.

5 Conclusion

In this work, We introduce **GraphThought**, a structured framework designed to enhance LLM reasoning in Graph Combinatorial Optimization (GCO). By formalizing the OTD problem, we develop dual data generation strategies—heuristic-driven forward and solver-aligned backward—to construct high-quality reasoning trajectories. This approach yields Llama-GT, a compact 8B model that achieves state-of-the-art performance on GraphArena, significantly outperforming larger baselines. Our findings validate the efficacy of incorporating structured reasoning into LLM training and pave the way for hybrid models that synergize traditional solvers with neural heuristics.

Limitations

Despite the promising results achieved by the *GraphThought* framework, several key limitations

persist: (i) its performance markedly deteriorates on highly complex or knowledge-scarce NP-hard graph optimization tasks—such as the Traveling Salesman Problem, Graph Edit Distance, and Maximum Common Subgraph—where the absence of explicit combinatorial priors often causes convergence to sub-optimal solutions and the generation of invalid intermediate reasoning steps (see Section J); (ii) relying on large language models as combinatorial solvers entails substantial computational overhead, yielding higher inference latency and greater resource consumption than classical optimization engines like Gurobi or specialized heuristics, thus constraining practical deployment in latency-sensitive scenarios; and (iii) the verbose, sequential nature of natural-language reasoning prevents efficient scaling to large graphs with thousands or millions of nodes, whose size exceeds current LLM context windows, leading to fragmented or incomplete reasoning that degrades overall solution quality and validity.

References

Maciej Besta, Nils Blach, Ales Kubicek, Robert Gerstenberger, Michal Podstawski, Lukas Gianinazzi, Joanna Gajda, Tomasz Lehmann, Hubert Niewiadomski, Piotr Nyczyk, and 1 others. 2024a. Graph of thoughts: Solving elaborate problems with large language models. In *AAAI*.

Maciej Besta, Florim Memedi, Zhenyu Zhang, Robert Gerstenberger, Guangyuan Piao, Nils Blach, Piotr Nyczyk, Marcin Copik, Grzegorz Kwasniewski, Jürgen Müller, and 1 others. 2024b. Demystifying chains, trees, and graphs of thoughts. *arXiv:2401.14295*.

Yukun Cao, Shuo Han, Zengyi Gao, Zezhong Ding, Xike Xie, and S Kevin Zhou. 2024. Graphinsight: Unlocking insights in large language models for graph structure understanding. *arXiv:2409.03258*.

Ziwei Chai, Tianjie Zhang, Liang Wu, Kaiqiao Han, Xiaohai Hu, Xuanwen Huang, and Yang Yang. 2023. Graphllm: Boosting graph reasoning ability of large language model. *arXiv:2310.05845*.

Lijun Chang, Wei Li, and Wenjie Zhang. 2017. Computing a near-maximum independent set in linear time by reducing-peeling. In *Proceedings of the 2017 ACM International Conference on Management of Data*, pages 1181–1196.

Nuo Chen, Yuhan Li, Jianheng Tang, and Jia Li. 2024a. Graphwiz: An instruction-following language model for graph computational problems. In *KDD*.

Zhikai Chen, Haitao Mao, Hang Li, Wei Jin, Hongzhi Wen, Xiaochi Wei, Shuaiqiang Wang, Dawei Yin, Wenqi Fan, Hui Liu, and 1 others. 2024b. Exploring the potential of large language models (LLMs) in learning on graphs. *ACM SIGKDD Explorations Newsletter*, 25(2):42–61.

Xinnan Dai, Qihao Wen, Yifei Shen, Hongzhi Wen, Dongsheng Li, Jiliang Tang, and Caihua Shan. 2024. Revisiting the graph reasoning ability of large language models: Case studies in translation, connectivity and shortest path. *arXiv:2408.09529*.

Darko Drakulic, Sofia Michel, and Jean-Marc Andreoli. 2024. Goal: A generalist combinatorial optimization agent learner. *arXiv:2406.15079*.

Mohammed Elhenawy, Ahmed Abdelhay, Taqwa I Alhadidi, Huthaifa I Ashqar, Shadi Jaradat, Ahmed Jaber, Sebastien Glaser, and Andry Rakotonirainy. 2024a. Eyeballing combinatorial problems: A case study of using multimodal large language models to solve traveling salesman problems. *arXiv:2406.06865*.

Mohammed Elhenawy, Ahmad Abutahoun, Taqwa I Alhadidi, Ahmed Jaber, Huthaifa I Ashqar, Shadi Jaradat, Ahmed Abdelhay, Sebastien Glaser, and Andry Rakotonirainy. 2024b. Visual reasoning and multi-agent approach in multimodal large language models (mllms): Solving tsp and mtsp combinatorial challenges. *arXiv:2407.00092*.

Bahare Fatemi, Jonathan Halcrow, and Bryan Perozzi. 2023. Talk like a graph: Encoding graphs for large language models. *arXiv:2310.04560*.

Yifan Feng, Chengwu Yang, Xingliang Hou, Shaoyi Du, Shihui Ying, Zongze Wu, and Yue Gao. 2024. Beyond graphs: Can large language models comprehend hypergraphs? *arXiv:2410.10083*.

Hamed Firooz, Maziar Sanjabi, Wenlong Jiang, and Xiaoling Zhai. 2024. Lost-in-distance: Impact of contextual proximity on llm performance in graph tasks. *arXiv:2410.01985*.

Kanishk Gandhi, Denise Lee, Gabriel Grand, Muxin Liu, Winson Cheng, Archit Sharma, and Noah D Goodman. 2024. Stream of search (sos): Learning to search in language. *arXiv:2404.03683*.

Jiayan Guo, Lun Du, Hengyu Liu, Mengyu Zhou, Xinyi He, and Shi Han. 2023. Gpt4Graph: Can large language models understand graph structured data? an empirical evaluation and benchmarking. *arXiv:2305.15066*.

Yuwei Hu, Runlin Lei, Xinyi Huang, Zhewei Wei, and Yongchao Liu. 2024. Scalable and accurate graph reasoning with llm-based multi-agents. *arXiv:2410.05130*.

Jie Huang, Xinyun Chen, Swaroop Mishra, Huaixiu Steven Zheng, Adams Wei Yu, Xinying Song, and Denny Zhou. 2024. Large language models cannot self-correct reasoning yet. In *ICLR*.

S. Iklassov, J. Smith, and K. Lee. 2024. Deep learning approaches for graph combinatorial optimization problems. *Journal of Artificial Intelligence Research*, 70:123–145.

Xia Jiang, Yaoxin Wu, Yuan Wang, and Yingqian Zhang. 2024a. Unco: Towards unifying neural combinatorial optimization through large language model. *arXiv:2408.12214*.

Zhengdao Jiang, Zhen Zhang, Yujia Li, and Jure Leskovec. 2024b. Graph neural networks for combinatorial optimization: A survey. In *Proceedings of the 38th AAAI Conference on Artificial Intelligence*.

Bowen Jin, Gang Liu, Chi Han, Meng Jiang, Heng Ji, and Jiawei Han. 2024a. Large language models on graphs: A comprehensive survey. *IEEE Transactions on Knowledge and Data Engineering*.

Xiaoyang Jin, Yujie Fan, Yujia Li, and Jure Leskovec. 2024b. Learning to solve combinatorial optimization problems on graphs via reinforcement learning. In *Proceedings of the 38th AAAI Conference on Artificial Intelligence*.

L. R. Ford Jr. and D. R. Fulkerson. 1956. Maximal flow through a network. *Canadian Journal of Mathematics*, 8:399–404.

Joseph B. Kruskal. 1956. On the shortest spanning subtree of a graph and the traveling salesman problem. *Proceedings of the American Mathematical Society*, 7(1):48–50.

Harold W. Kuhn. 1955. The hungarian method for the assignment problem. In *Naval Research Logistics Quarterly*, volume 2, pages 83–97.

Lucas Lehnert, Sainbayar Sukhbaatar, Paul McVay, Michael Rabbat, and Yuandong Tian. 2024a. Beyond a*: Better planning with transformers via search dynamics bootstrapping. In *ICLR Workshop on Large Language Model (LLM) Agents*.

T. Lehnert, A. Müller, and Y. Zhang. 2024b. Graph neural networks for solving np-hard problems: A review. *IEEE Transactions on Neural Networks and Learning Systems*, 35(4):789–805.

H. Li, S. Wang, and T. Zhang. 2024a. Text-based approaches to graph combinatorial problems using large language models. *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 456–467.

J. Li, N. Chen, and Q. Zhang. 2024b. Optimal thoughts design for graph combinatorial optimization. *arXiv preprint arXiv:2406.07890*.

Xijun Li, Jiexiang Yang, Jinghao Wang, Bo Peng, Jianguo Yao, and Haibing Guan. 2025. Strcmp: Integrating graph structural priors with language models for combinatorial optimization. *arXiv preprint arXiv:2506.11057*.

Xin Li, Qizhi Chu, Yubin Chen, Yang Liu, Yaoqi Liu, Zekai Yu, Weize Chen, Chen Qian, Chuan Shi, and Cheng Yang. 2024c. Graphteam: Facilitating large language model-based graph analysis via multi-agent collaboration. *arXiv:2410.18032*.

Jiawei Liu, Cheng Yang, Zhiyuan Lu, Junze Chen, Yibo Li, Mengmei Zhang, Ting Bai, Yuan Fang, Lichao Sun, Philip S Yu, and 1 others. 2023a. Towards graph foundation models: A survey and beyond. *arXiv:2310.11829*.

Y. Liu, X. Chen, and Y. Zhao. 2023b. Benchmarking large language models on graph problems. *arXiv preprint arXiv:2309.01234*.

Yujie Liu, Yujia Li, and Jure Leskovec. 2023c. Combinatorial optimization with graph neural networks: A survey. *arXiv preprint arXiv:2301.11270*.

Michael Luby. 1986. A simple parallel algorithm for the maximal independent set problem. *SIAM Journal on Computing*, 15(4):1036–1053.

Z. Luo, X. Song, H. Huang, J. Lian, C. Zhang, J. Jiang, X. Xie, and H. Jin. 2024a. Graphinstruct: Empowering large language models with graph understanding and reasoning capability. *arXiv preprint arXiv:2403.04483*.

Zihan Luo, Xiran Song, Hong Huang, Jianxun Lian, Chenhao Zhang, Jinqi Jiang, Xing Xie, and Hai Jin. 2024b. Graphinstruct: Empowering large language models with graph understanding and reasoning capability. *arXiv:2403.04483*.

L. Ouyang, J. Wu, and Y. Zhang. 2024a. Enhancing llms for graph reasoning tasks. *arXiv preprint arXiv:2402.09876*.

Sheng Ouyang, Yulan Hu, Ge Chen, and Yong Liu. 2024b. Gundam: Aligning large language models with graph understanding. *arXiv:2409.20053*.

Vangelis Th Paschos. 2014. *Applications of Combinatorial Optimization*. John Wiley & Sons.

Bryan Perozzi, Bahare Fatemi, Dustin Zelle, Anton Tsitsulin, Mehran Kazemi, Rami Al-Rfou, and Jonathan Halcrow. 2024. Let your graph do the talking: Encoding structured data for LLMs. *arXiv:2402.05862*.

Xubin Ren, Jiabin Tang, Dawei Yin, Nitesh Chawla, and Chao Huang. 2024a. A survey of large language models for graphs. In *KDD*.

Y. Ren, L. Zhao, and M. Chen. 2024b. Large language models for combinatorial optimization: Opportunities and challenges. *arXiv preprint arXiv:2401.12345*.

Clayton Sanford, Bahare Fatemi, Ethan Hall, Anton Tsitsulin, Mehran Kazemi, Jonathan Halcrow, Bryan Perozzi, and Vahab Mirrokni. 2024a. Understanding transformer reasoning capabilities via graph algorithms. *arXiv:2405.18512*.

J. Sanford, D. Lee, and H. Kim. 2024b. Meta-thought programming: A framework for dual-mode reasoning in llms. *arXiv preprint arXiv:2405.06789*.

Konstantinos Skianis, Giannis Nikolentzos, and Michalis Vazirgiannis. 2024. Graph reasoning with large language models via pseudo-code prompting. *arXiv:2409.17906*.

Jianheng Tang, Qifan Zhang, Yuhan Li, Nuo Chen, and Jia Li. 2025. Grapharena: Evaluating and exploring large language models on graph computation. In *The Thirteenth International Conference on Learning Representations*.

Heng Wang, Shangbin Feng, Tianxing He, Zhaoxuan Tan, Xiaochuang Han, and Yulia Tsvetkov. 2024a. Can language models solve graph problems in natural language? In *NeurIPS*.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2022. Self-consistency improves chain of thought reasoning in language models. *arXiv preprint arXiv:2203.11171*.

Yiming Wang, Ziyang Zhang, Hanwei Chen, and Huayi Shen. 2024b. Reasoning with large language models on graph tasks: The influence of temperature. In *ICCEA*.

Chunyu Wei, Wenji Hu, Xingjia Hao, Xin Wang, Yifan Yang, Yueguo Chen, Yang Tian, and Yunhai Wang. 2025. Graphchain: Large language models for large-scale graph analysis via tool chaining. *arXiv preprint arXiv:2511.00457*.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, and Denny Zhou. 2022. Chain-of-thought prompting elicits reasoning in large language models. In *NeurIPS*.

Yanbin Wei, Shuai Fu, Weisen Jiang, James T Kwok, and Yu Zhang. 2024. Gita: Graph to visual and textual integration for vision-language graph reasoning. *arXiv:2402.02130*.

Tianhao Wu, Janice Lan, Weizhe Yuan, Jiantao Jiao, Jason Weston, and Sainbayar Sukhbaatar. 2024. Thinking llms: General instruction following with thought generation. *arXiv:2410.10630*.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. 2024. Tree of thoughts: Deliberate problem solving with large language models. In *NeurIPS*.

Zike Yuan, Ming Liu, Hui Wang, and Bing Qin. 2024. Gracore: Benchmarking graph comprehension and complex reasoning in large language models. *arXiv:2407.02936*.

Eric Zelikman, Yuhuai Wu, Jesse Mu, and Noah Goodman. 2022. Star: Bootstrapping reasoning with reasoning. In *NeurIPS*.

Yizhuo Zhang, Heng Wang, Shangbin Feng, Zhaoxuan Tan, Xiaochuang Han, Tianxing He, and Yulia Tsvetkov. 2024. Can llm graph reasoning generalize beyond pattern memorization? *arXiv:2406.15992*.

Zefang Zong, Xiaochen Wei, Guozhen Zhang, Chen Gao, Huandong Wang, and Yong Li. 2024. Solving diverse combinatorial optimization problems with a unified model. Under review.

A More Related Work

A.1 LLMs for Graph Combinatorial Optimization

The integration of Large Language Models (LLMs) into solving Graph Combinatorial Optimization (GCO) problems has garnered significant attention in recent years. Several works have attempted to leverage LLMs for GCO problems from various perspectives. Jin et al. (Jin et al., 2024a) provided a comprehensive survey on the application of LLMs on graphs, categorizing potential scenarios into pure graphs, text-attributed graphs, and text-paired graphs. They discussed techniques such as using LLMs as predictors, encoders, and aligners. Chen et al. (Chen et al., 2024b) explored the potential of LLMs in graph machine learning, especially for node classification tasks, and investigated two pipelines: LLMs-as-Enhancers and LLMs-as-Predictors. Liu et al. (Liu et al., 2023a) introduced the concept of Graph Foundation Models (GFMs) and classified existing work into categories based on their dependence on graph neural networks and LLMs. Ren et al. (Ren et al., 2024a) conducted a survey on LLMs for graphs, proposing a taxonomy to categorize existing methods based on their framework design.

A.1.1 Prompt Engineering and Benchmark Development

Recent advances in LLM applications for graph problems have focused on prompt engineering strategies and comprehensive benchmarking frameworks. Wang et al. (Wang et al., 2024a) introduced the NLGraph benchmark with Build-a-Graph prompting and algorithmic optimizations for graph reasoning tasks. Tang et al. (Tang et al., 2025) developed GraphArena as a standardized benchmarking platform, while Yuan et al. (Yuan et al., 2024) proposed GraCoRe for assessing graph comprehension capabilities. Skianis et al. (Skianis et al., 2024) demonstrated the effectiveness of pseudo-code prompting for enhancing graph problem performance.

A.1.2 Architectural and Framework Innovations

Several architectural innovations have emerged to enhance LLM capabilities in graph processing. Li et al. (Li et al., 2024c) introduced GraphTeam as a multi-agent collaborative system, complemented by Hu et al.'s GraphAgent-Reasoner framework (Hu et al., 2024). Perozzi et al. (Perozzi et al., 2024) developed GraphToken for explicit structured data representation, while Cao et al. (Cao et al., 2024) proposed GraphInsight to improve structural comprehension.

A.1.3 Graph Representation and Encoding Strategies

Innovative graph encoding methods have been crucial for bridging the gap between LLMs and graph structures. Fatemi et al. (Fatemi et al., 2023) pioneered text-based graph encoding, while Elhenawy et al. (Elhenawy et al., 2024a,b) explored multimodal visual reasoning for TSP solutions. Feng et al. (Feng et al., 2024) extended these approaches to hypergraphs through LLM4Hypergraph.

A.1.4 Empirical Analysis and Performance Factors

Comprehensive empirical studies have revealed critical insights into LLM capabilities and limitations. Guo et al. (Guo et al., 2023) conducted large-scale evaluations on graph-structured data, while Wang et al. (Wang et al., 2024b) analyzed temperature's impact on reasoning performance. Firooz et al. (Firooz et al., 2024) investigated contextual proximity effects, and Dai et al. (Dai et al., 2024) provided case studies on graph reasoning limitations. Sanford et al. (Sanford et al., 2024a) analyzed transformer architectures through graph algorithmic lenses.

Other works have focused on training graph foundation models using dense embeddings from LLM pretraining. Drakulic et al. (Drakulic et al., 2024) proposed GOAL, a generalist model for solving multiple combinatorial optimization problems. Jiang et al. (Jiang et al., 2024a) introduced UNCO, a unified framework for solving various COPs using LLMs. Zong et al. (Zong et al., 2024) proposed a unified model for diverse CO problems using a transformer backbone. Chai et al. (Chai et al., 2023) introduced GraphLLM to boost the graph reasoning ability of LLMs. Wei et al. (Wei et al., 2024) proposed GITA, a framework integrating visual and textual information for graph reasoning.

A.2 Chain-of-Thought, Tree-of-Thought, and Graph-of-Thought Methods

The Chain-of-Thought (CoT) prompting technique, introduced by Wei et al. (Wei et al., 2022), demonstrates that generating intermediate reasoning steps can significantly improve LLMs' performance on complex reasoning tasks. This method was further extended by Yao et al. (Yao et al., 2024), who proposed the Tree of Thoughts (ToT) framework, enabling exploration over coherent units of text (thoughts) to enhance problem-solving abilities. The ToT approach allows LMs to consider multiple reasoning paths and self-evaluate choices, significantly improving performance on tasks requiring non-trivial planning or search. Besta et al. (Besta et al., 2024a) introduced the Graph of Thoughts (GoT), which advances prompting capabilities by modeling LLM-generated information as an arbitrary graph. This approach enables combining arbitrary LLM thoughts into synergistic outcomes and enhances thoughts using feedback loops. Besta et al. (Besta et al., 2024b) further demystified the concepts of chains, trees, and graphs of thoughts, providing a taxonomy of structure-enhanced LLM reasoning schemes. These studies highlight the importance of structured reasoning topologies in improving LLMs' problem-solving abilities.

A.3 Self-Correction, Planning, and Search Strategy Learning

Huang et al. (Huang et al., 2024) examined the role of self-correction in LLMs, finding that intrinsic self-correction without external feedback often fails to improve reasoning accuracy. In contrast, Lehnert et al. (Lehnert et al., 2024a) proposed the Searchformer model, which predicts the search dynamics of the A* algorithm, significantly outperforming traditional planners on complex decision-making tasks. Gandhi et al. (Gandhi et al., 2024) introduced the Stream of Search (SoS) approach, teaching language models to search by representing the process as a flattened string. This method significantly improved search accuracy and enabled flexible use of different search strategies.

B Experiments Setting

B.1 Training Parameter

Our framework is implemented based on LLaMA-Factory. We perform supervised fine-tuning (SFT)

on the Meta-Llama-3-8B-Instruct[8] model using LoRA adaptation. The training process, conducted with 30K instruction-following examples, required approximately 138,828 seconds (38 hours) on a single NVIDIA H800 PCIe 80GB GPU. Key hyperparameters include a cosine learning rate scheduler with initial value 8e-4, 4 training epochs, and gradient accumulation over 8 steps. The complete configuration details are presented in Table 2.

Table 2: Detailed training configuration for supervised fine-tuning.

Category	Setting
Model Configuration	
Base Model	Meta-Llama-3-8B-Instruct
Fine-tuning Method	LoRA
Dataset Size	30,000 samples
Validation Split	10%
Max Sequence Length	3,000 tokens
Training Parameters	
Epochs	4
Learning Rate	8e-4
Batch Size (per device)	8
Gradient Accumulation Steps	8
Optimizer	AdamW
Learning Rate Scheduler	Cosine
Warmup Steps	0
Max Gradient Norm	1.0
LoRA Configuration	
LoRA Rank	8
LoRA Alpha	16
LoRA Dropout	0

When comparing the Star(Zelikman et al., 2022) method, We follow the dataset construction methodology described in the original STaR paper. Specifically, we first perform inference using either an untrained or partially trained model, then conduct rationalization on unsuccessful cases that failed to reach the optimal solution. After collecting these rationalized examples to construct the training dataset, we subsequently perform supervised fine-tuning (SFT) on the base model. For the base model, we consistently employ Llama-3-8B-Instruct with identical hyperparameter settings as listed in the configuration Table 2. The complete STaR process undergoes four iterative cycles to progressively enhance model performance.

B.2 Evaluation Dataset

We conduct evaluations using GraphArena's original test datasets with preserved graph size definitions ("small" vs "large") from their benchmark.

[8] https://huggingface.co/meta-llama/Meta-Llama-3-8B-Instruct

The 10 tasks are categorized by time complexity and specified as follows:

- **Common Neighbor (Polynomial)**: For a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ and a node pair (v_1, v_2), identify the set of mutual neighbors $S = \{u \in \mathcal{V} \mid (u, v_1) \in \mathcal{E} \wedge (u, v_2) \in \mathcal{E}\}$. The objective is to maximize the cardinality $|S|$. **Neighbor** for short.

- **Shortest Distance (Polynomial)**: Determine the minimum path length between nodes v_1 and v_2 in $\mathcal{G}$. Optimality is defined by minimizing the geodesic distance $\ell(p_{v_1 \to v_2})$. **Distance** for short.

- **Connected Component (Polynomial)**: Identify a set of representative nodes such that all disjoint components in $\mathcal{G}$ are covered. Optimality requires full coverage of the component set $\mathcal{C}$. **Connected** for short.

- **Graph Diameter (Polynomial)**: Compute the maximum shortest path distance between any pair of nodes in $\mathcal{V}$, maximizing the value $\max_{u,v} d(u, v)$. **Diameter** for short.

- **Maximum Clique Problem (NP-hard)**: A clique is a complete subgraph where every pair of distinct vertices is adjacent. The task is to identify the largest such subgraph $\mathcal{C} \subseteq \mathcal{V}$ in $\mathcal{G}$. A solution is optimal if it maximizes $|\mathcal{C}|$ (i.e., for any other clique $\mathcal{C}' \subseteq \mathcal{V}$, $|\mathcal{C}'| \leq |\mathcal{C}|$). **MCP** for short.

- **Maximum Independent Set (NP-hard)**: Select the largest possible subset of nodes $\mathcal{S} \subseteq \mathcal{V}$ such that no two nodes in $\mathcal{S}$ are adjacent. Optimality is achieved by maximizing $|\mathcal{S}|$. **MIS** for short.

- **Minimum Vertex Cover (NP-hard)**: Determine the smallest vertex subset $\mathcal{S}$ that covers all edges in $\mathcal{E}$. The optimal solution minimizes the number of nodes $|\mathcal{S}|$. **MVC** for short.

- **Maximum Common Subgraph (NP-hard)**: Identify the largest node-induced subgraph $\mathcal{S}$ common to both $\mathcal{G}$ and $\mathcal{H}$ that preserves edge topology. The goal is to maximize the number of shared vertices $|V(\mathcal{S})|$. **MCS** for short.

- **Graph Edit Distance (NP-hard)**: Calculate the minimum cost of edit operations (including node/edge insertions and deletions) required to transform $\mathcal{G}$ into $\mathcal{H}$. Optimal solutions minimize the total edit cost. **GED** for short.

- **Traveling Salesman Problem (NP-hard)**: In a complete weighted graph, find the shortest Hamiltonian cycle. The optimal route minimizes the total edge weight $\sum w(e_i)$. **TSP** for short.

In alignment with the original study's framework, we maintain the original node range definitions across task categories: (1) Neighbor/Distance tasks operate with small graphs (4-19 nodes) and large graphs (20-50 nodes); (2) Component/Diameter measurements alongside combinatorial problems (MCP/MIS/MVC) utilize small-scale graphs (4-14 nodes) contrasting with large-scale counterparts (15-30 nodes); (3) MCS/GED/TSP adhere to the established size parameters of small (4-9 nodes) versus large (10-20 nodes) instances. By employing the benchmark's test datasets with unmodified size criteria, we preserve experimental continuity and facilitate meaningful cross-study comparisons.

To address more general graph-theoretic challenges and facilitate algorithmic processing, we systematically convert the input representations of these tasks into a unified and structured schema.

C License for GraphArena benchmark

We utilize the **GraphArena** benchmark to evaluate large language models on graph computational problems solely for academic research purposes. GraphArena is open-sourced under the **Creative Commons Attribution 4.0 International (CC BY 4.0)** license, as indicated in the official publication (Tang et al., 2025).

The datasets incorporated within GraphArena originate from various sources, each with its respective licensing terms:

- **DBLP**: CC0 1.0 Public Domain Dedication
- **Social Network**: CC BY-SA 3.0 License
- **DBpedia**: CC BY-SA 3.0 License
- **OpenFlights**: Database Contents License (DbCL) v1.0
- **PubChemQC (PCQM4Mv2)**: CC BY 4.0 License

All datasets are publicly available and have been utilized in accordance with their respective licenses. The datasets employed, including those within the GraphArena benchmark, are either synthetic or derived from publicly available sources with appropriate anonymization measures in place.

D Comprehensive Experimental Results

In this section, we provide the complete documentation of our experimental results across all evaluated tasks. Table 3 present the full suite of performance metrics, facilitating a detailed comparison between our proposed framework and the baseline models across various graph optimization benchmarks.

E Inference Cost

The inference time per instance is approximately 0.0694 seconds for the Llama-3-8B-Instruct model and 0.0703 seconds for the fine-tuned model (Llama-GT). Notably, when using the fine-tuned model with Best-of-N (N=16), the inference time increases to approximately 0.1643 seconds per instance. This increase is not proportional to the number of queries (N=16) due to batch processing, which reduces computational overhead while generating more diverse responses to select the optimal answer.

Table 4: Inference Time Cost of the Original Model, Llama-GT, and Llama-GT with Best-of-N (N=16)

Model	Time (s)
Llama-3-8B-Instruct	0.0694
Llama-GT	0.0703
Llama-GT (N=16)	0.1643

F Methods for constructing action thoughts and state thoughts for GCO problem

F.1 Action thoughts generation methods

For a GCO problem, there are mainly three classes of methods to construct action thought set A with heuristics.

- **Ordinary Generation Rule**: Foundational strategies (e.g., greedy selection, random sampling) provide baseline mechanisms for generating thoughts. These rule-based approaches offer broad applicability across diverse problem domains through their simplicity.
- **Simple Heuristic Thoughts:** Heuristics of GCO problems leverage structural properties of target problems to enhance operational efficiency. Such methods typically derive from simple reasoning conclusions of the problem.
- **Complex Heuristic Thoughts:** Complex heuristics involve a broader set of operational primitives, presenting two fundamental challenges. First, their

Table 3: Performance comparison of optimal solution rates (%) across 10 graph tasks, evaluated on small and large graphs, each has 500 instances. Results are shown for: (1) Original inference results of LLMs from GraphArena (Claude3-haiku, GPT-4o, etc.); (2) Code-augmented models (DeepSeek-V2-Coder, GPT-4o-Coder); (3)Supervised Fine-Tuned Model provided by GraphArena (Qwen2-7B-SFT);(4) Few-shot thought prompting variants (Deepseek-V3, Llama-3.3-70B, etc.); (5) STaR framework implementations with different fewshot strategies; and (6) Our Llama-GT model trained with/without thought mechanisms. Metrics include polynomial tasks (Neighbor, Distance, Connected, Diameter), NP-hard tasks (MVC, MIS, MCP, TSP, MCS, GED). * indicates that the model in the GraphArena paper was trained on fewer data compared to the data used in our model.

Graph Task (Small Graphs)										
Model	Neighbor	Distance	Connected	Diameter	MVC	MIS	MCP	TSP	MCS	GED
Claude3-haiku	0.768	0.580	0.260	0.116	0.336	0.450	0.482	0.242	0.282	0.216
DeepSeek-V2	0.540	0.814	0.474	0.226	0.376	0.360	0.400	0.368	0.236	0.282
Gemma-7b	0.410	0.496	0.014	0.086	0.128	0.212	0.254	0.134	0.020	0.028
gpt-3.5-turbo-0125	0.562	0.572	0.096	0.130	0.376	0.184	0.400	0.230	0.176	0.144
gpt-4o-2024-0513	0.860	0.796	0.794	0.426	0.326	0.518	0.528	0.404	0.406	0.268
Llama3-70b-Instruct	0.674	0.894	0.632	0.248	0.420	0.368	0.428	0.232	0.442	0.316
Llama3-8b-Instruct	0.368	0.412	0.248	0.114	0.318	0.282	0.280	0.162	0.096	0.072
Mixtral-8x7b	0.530	0.566	0.286	0.130	0.130	0.116	0.278	0.188	0.098	0.124
Qwen1.5-72b-Chat	0.572	0.478	0.394	0.118	0.224	0.386	0.388	0.228	0.298	0.174
Qwen1.5-8b-Chat	0.138	0.266	0.022	0.048	0.138	0.118	0.216	0.170	0.062	0.058
DeepSeek-V2-Coder	0.816	0.894	0.586	0.142	0.176	0.482	0.498	0.276	0.228	0.214
GPT-4o-Coder	0.808	0.654	0.712	0.334	0.296	0.530	0.644	**0.490**	0.508	0.320
Qwen2-7b-SFT*	0.966	0.912	0.888	0.608	0.548	0.702	0.696	0.368	0.000	0.054
+Few-shot Thought										
Deepseek-V3	**1.000**	0.988	**1.000**	0.850	0.366	0.642	0.754	0.370	**0.544**	0.308
gpt-4o-mini	0.980	0.902	0.760	0.336	0.100	0.710	0.578	0.312	0.360	0.280
Llama-3.3-70B	0.970	0.970	0.954	0.674	0.206	0.648	0.554	0.292	0.484	0.326
Mixtral-8x7b	0.580	0.536	0.378	0.156	0.148	0.326	0.490	0.168	0.196	0.262
QwQ-32B-Preview	0.962	0.848	0.696	0.514	0.262	0.482	0.560	0.310	0.444	0.318
Llama3-8b-Instruct	0.700	0.480	0.502	0.070	0.108	0.248	0.258	0.190	0.222	0.450
+SFT										
STaR(w/ GT)	0.648	0.910	0.522	0.466	0.722	0.640	0.676	0.352	0.308	0.370
STaR(w/ LLM)	0.296	0.662	0.440	0.380	0.688	0.654	0.288	0.282	0.328	0.258
Llama-GT(w/o Thought)	0.988	0.990	0.906	0.820	0.930	0.972	0.906	0.366	0.538	**0.608**
Llama-GT	**1.000**	**1.000**	0.996	**0.954**	**0.972**	**0.994**	**0.952**	0.392	0.496	0.460
Graph Task (Large Graphs)										
Model	Neighbor	Distance	Connected	Diameter	MVC	MIS	MCP	TSP	MCS	GED
Claude3-haiku	0.406	0.358	0.052	0.002	0.076	0.014	0.090	0.000	0.000	0.018
DeepSeek-V2	0.278	0.534	0.154	0.022	0.064	0.032	0.032	0.006	0.000	0.014
Gemma-7b	0.116	0.246	0.002	0.004	0.014	0.006	0.016	0.000	0.000	0.002
gpt-3.5-turbo-0125	0.412	0.322	0.012	0.008	0.082	0.008	0.052	0.000	0.000	0.006
gpt-4o-2024-0513	0.674	0.550	0.370	0.032	0.102	0.034	0.102	0.004	0.000	0.018
Llama3-70b-Instruct	0.434	0.530	0.264	0.034	0.114	0.026	0.106	0.000	0.000	0.008
Llama3-8b-Instruct	0.118	0.234	0.022	0.002	0.064	0.010	0.022	0.000	0.000	0.002
Mixtral-8x7b	0.232	0.282	0.040	0.000	0.036	0.008	0.034	0.000	0.000	0.006
Qwen1.5-72b-Chat	0.236	0.258	0.068	0.006	0.038	0.018	0.022	0.000	0.000	0.010
Qwen1.5-8b-Chat	0.026	0.156	0.002	0.000	0.002	0.000	0.016	0.000	0.000	0.006
DeepSeek-V2-Coder	0.672	0.632	0.206	0.008	0.080	0.028	0.072	0.000	0.022	0.014
GPT-4o-Coder	0.868	0.684	0.378	0.112	0.110	0.072	0.222	0.028	**0.036**	0.018
Qwen2-7b-SFT*	0.790	0.570	0.230	0.092	0.156	0.054	0.136	0.000	0.000	0.032
+Few-shot Thought										
DeepSeek-V3	**0.992**	0.942	**0.932**	0.448	0.120	0.304	0.290	0.020	0.012	0.020
gpt-4o-mini	0.920	0.550	0.284	0.016	0.050	0.226	0.146	0.000	0.006	0.014
Llama-3.3-70B	0.952	0.866	0.856	0.290	0.118	0.254	0.136	0.000	0.012	0.012
Mixtral-8x7b	0.466	0.282	0.108	0.002	0.032	0.024	0.056	0.000	0.000	0.022
QwQ-32B-Preview	0.912	0.504	0.498	0.164	0.058	0.106	0.124	0.000	0.002	0.020
Llama3-8b-Instruct	0.604	0.220	0.132	0.002	0.028	0.010	0.038	0.000	0.002	0.026
+SFT										
STaR(w/ GT)	0.374	0.618	0.124	0.080	0.360	0.216	0.134	0.018	0.006	0.026
STaR(w/ LLM)	0.320	0.332	0.090	0.038	0.126	0.124	0.030	0.002	0.004	0.024
Llama-GT(w/o Thought)	0.804	0.864	0.340	0.302	0.652	0.652	0.370	0.024	0.020	**0.068**
Llama-GT	0.988	**0.984**	0.836	**0.600**	**0.744**	**0.900**	**0.634**	**0.036**	**0.036**	0.008

intricate implementation mechanisms lack intuitive interpretability. Second, the expanded solution space necessitates consideration of diverse operational combinations. These characteristics hinder LLMs from effectively discerning the underlying principles learning such heuristics.

- **Mixed Heuristic Thoughts:** The former strategies possess distinct advantages, hybrid approaches demonstrate superior efficacy in specific problem contexts through strategic combination of complementary strategies.

F.2 State thoughts generation methods

In solving a GCO problem, maintaining instance state descriptions is essential for two reasons. First, the GCO instance evolves with each reasoning step, requiring the removal of redundant elements. Second, explicitly tracking instance states prevents hallucinations in LLMs. The following state thought generation methods are defined.

- **Instance Description:** Recording the current GCO instance information forms the foundational state representation. For example describe the node set V and edge set E of a graph G.

- **Instance Simplification:** Pruning redundant elements after every reasoning step. For example, in maximum independent set (MIS) problems, removing all neighbors of the current MIS set from the graph G.

- **Instance Reduction:** Transforming instances into equivalent problem spaces. For instance, solving MIS of G can be reduced to finding minimum vertex cover (MVC) of G's complement graph $\bar{G}$.

- **Solution Description:** The solution information not only emphasizes the importance of the solution but also shows the changes of the solution. This prompts the LLM to explicitly track how selected actions influence the current solution.

- **Solving flag:** Formal stopping criteria marking solution completion through state indicators.

G Usual action thoughts and state thoughts for GCO problem

In graph optimization problems, the fundamental heuristic operations primarily comprise four canonical primitives: node addition, node deletion, edge addition, and edge deletion. Formally, let $\mathbb{A}$ denote the action thought space for graph optimization, which includes the following important actions:

- Add Nodes Based on the Given Optimal Solution (a_1): Add one or more nodes to the current solution set based on the given optimal solution.
- Add Nodes Based on Rules (a_2): Add one or more nodes to the current solution set using rules such as greedy or random selection.
- Add Nodes Based on Simple Prior Knowledge (a_3): Add one or more nodes to the current solution set using simple prior knowledge.
- Add Nodes Based on Complex Prior Knowledge (a_4): Add one or more nodes to the current solution set using complex prior knowledge.
- Remove Nodes Based on the Given Optimal Solution (a_5): Remove one or more nodes from the current solution set based on the given optimal solution.
- Remove Nodes Based on Rules (a_6): Remove one or more nodes from the current solution set using rules such as greedy or random selection.
- Remove Nodes Based on Simple Prior Knowledge (a_7): Remove one or more nodes from the current solution set using simple prior knowledge.
- Remove Nodes Based on Complex Prior Knowledge (a_8): Remove one or more nodes from the current solution set using complex prior knowledge.
- Add Edges Based on the Given Optimal Solution (a_9): Add one or more edges to the current solution set based on the given optimal solution.
- Add Edges Based on Rules (a_{10}): Add one or more edges to the current solution set using rules such as greedy or random selection.
- Add Edges Based on Simple Prior Knowledge (a_{11}): Add one or more edges to the current solution set using simple prior knowledge.
- Add Edges Based on Complex Prior Knowledge (a_{12}): Add one or more edges to the current solution set using complex prior knowledge.
- Remove Edges Based on the Given Optimal Solution (a_{13}): Remove one or more edges

from the current solution set based on the given optimal solution.

- Remove Edges Based on Rules (a_{14}): Remove one or more edges from the current solution set using rules such as greedy or random selection.

- Remove Edges Based on Simple Prior Knowledge (a_{15}): Remove one or more edges from the current solution set using simple prior knowledge.

- Remove Edges Based on Complex Prior Knowledge (a_{16}): Remove one or more edges from the current solution set using complex prior knowledge.

The fundamental state thought space $\mathbb{S}$ for graph optimization problems is defined through node and edge set representations. Formally, the main canonical state components are structured as follows:

- Solving State (s_1): Describes the solving state to determine whether the process is complete.

- Add Nodes to the Graph (s_2): Describes the operation of adding nodes to the original graph for subsequent solving.

- Remove Nodes from the Graph (s_3): Describes the operation of removing nodes from the original graph for subsequent solving.

- Add Edges to the Graph (s_4): Describes the operation of adding edges to the original graph for subsequent solving.

- Remove Edges from the Graph (s_5): Describes the operation of removing edges from the original graph for subsequent solving.

- Graph Node Set (s_6): Describes the set of all nodes or partial nodes in the original graph.

- Graph Edge Set (s_7): Describes the set of all edges or partial edges in the original graph.

- Current Solution Set (s_8): Describes the set of elements in the current solution.

- Final Solution Set (s_9): Describes the set of elements in the final solution.

Algorithm 2 Breadth-First Search(G,u)

Require: Graph G, start node u
Ensure: Connected component C
1: Action: Start BFS at node u of G;
2: $L \leftarrow \{u\}$; ▷ Nodes waiting to be visited
3: $C \leftarrow \emptyset$; ▷ Visited nodes
4: **while** $L \neq \emptyset$ **do**
5: $v \leftarrow$ PopFrom(L); ▷ Select unvisited node
6: Action: add v to the current component;
7: $C \leftarrow C \cup \{v\}$;
8: **for** $w \in$ Neighbor(G,v) **do**
9: **if** $w \notin C \wedge w \notin L$ **then**
10: AddTo(w, L); ▷ Record unvisited nodes
11: State: add an unvisited neighbor w to L;
12: **end if**
13: **end for**
14: State: show the current visited nodes C;
15: **end while**
16: State: finished, show the connected component C;
17: **return** C;

H Concrete Applications of Forward and Backward MTP Frameworks

H.1 A Forward MTP for the Connected Components Problem

The Connected Components (CC) problem requires identifying all maximally connected subgraphs in an undirected graph G. Formally, given $G = (V, E)$, the goal is to partition V into disjoint subsets $\{C_1, \ldots, C_k\}$ where each C_i forms a connected subgraph.

Action Thought Generation Methods:

- **A simple heuristic thought: Breadth-First Search (BFS)**, systematically explores node neighborhoods through queue-based traversal (lines 1 and 6 of Algorithm 2).

- **An ordinary generation rule: Random Selection**, chooses initial nodes for BFS through random sampling (line 3 of Algorithm 3).

These action thoughts in the three lines constitute the set A.

State Thought Generation Methods:

- **Instance Simplification**: Add an unvisited node to the queue of nodes to be visited (line 11 of Algorithm 2).

- **Solution Description**: Current connected component being explored (line 14 of Algorithm 2, line 9 of Algorithm 3).
- **Solving Flag**: Indicator for algorithm completion (line 16 of Algorithm 2).

These state thoughts in the four lines constitute the set S.

A forward MTP for CC is shown in Algorithm 3. RandomSelect(G) select randomly a node u of G.

In BFS, the queue L stores nodes awaiting visitation, while set C maintains all visited nodes. When $L \neq \emptyset$, the algorithm selects a node v of L to visit, then adds v's unvisited neighbors to L. PopFrom(L) pops the first node of L. Neighbor(G,v) returns all neighbor node of v in G. AddTo(w, L) appends w to the end of L.

Algorithm 3 A Thoughts Template for Connected Component Problem

Require: Graph $G = (V, E)$
$CC \leftarrow \emptyset$; ▷ Initialize connected components
while $V \neq \emptyset$ **do**
  Action: choose a node for BFS randomly;
  $u \leftarrow$ RandomSelect(G); ▷ Random node selection
  $C_u \leftarrow$ BFS(G, u); ▷ Component discovery
  $CC \leftarrow CC \cup \{C_u\}$;
  $V \leftarrow V \setminus C_u$; ▷ Graph simplification
end while
State: describe connected components in CC;
return CC;

H.2 A Backward MTP for the MIS Problem

The MIS problem identifies the largest subset of non-adjacent nodes. The AddOne thought comprises two specialized operations:

- **AddOne: Add Isolated Nodes**, immediately incorporates all isolated nodes into the current solution MIS (line 4 of Algorithm 4).
- **AddOne: Add Optimal Nodes**, selectively integrates one node from solver outputs into the current solution MIS (line 6 of Algorithm 4).

The first action atomically adds all isolated nodes to the current solution because all isolated nodes belong to the optimal MIS solution. These action thoughts in the two lines constitute the set A.

State Thought Generation Methods:

- **Instance Description**: Maintains graph G's current structure (line 13 of Algorithm 4)
- **Instance Simplification**: Records graph simplification operations (line 10 of Algorithm 4)
- **Solution Description**: Tracks current/final solution candidates (lines 9 & 15 of Algorithm 4)
- **Solving Flag**: Monitors termination conditions (line 15 of Algorithm 4)

These state thoughts in the four lines constitute the set S.

A backward MTP for MIS is shown in Algorithm 4. An integer programming model of MIS problem is put into Gurobi, which serves as an optimal solution solver. Isolated(G) returns all nodes of G without neighbors.

Algorithm 4 A Thought Template for MIS

Require: Graph $G = (V, E)$, a mis solver $\mathcal{X}$.
$OPT \leftarrow \mathcal{X}(G)$; ▷ Compute optimal MIS
$MIS \leftarrow \emptyset$; ▷ Initialize solution
while $V \neq \emptyset$ **do**
  Action: add all isolated nodes to MIS;
  $MIS \leftarrow MIS \cup$ Isolated(G); ▷ Add all isolated nodes
  Action: add one node of OPT to MIS;
  $u \leftarrow$ AddOne(OPT); ▷ Add optimal node
  $MIS \leftarrow MIS \cup \{u\}$;
  State: describe the current MIS;
  State: delete all neighbors of u in G;
  $V \leftarrow V \setminus$ (Isolated(G) $\cup$ Neighbor(G, u) $\cup$ $\{u\}$); ▷ Remove useless nodes
  $OPT \leftarrow OPT \setminus MIS$;
  State: describe the current G;
end while
State: finished, describe the solution MIS;
return MIS.

I Ablation and Supplementary Experiments

I.1 Impact of Thought Types: State vs. Action

We conduct a targeted ablation study to analyze the relative contributions of state and action representations within the reasoning chain. As shown in Table 5, relying solely on action thoughts results in performance degradation compared to the no-thought baseline, suggesting that action-only reasoning leads to unstructured and less interpretable

logic. Conversely, the integration of state-based thoughts provides a critical structural scaffold, facilitating more systematic reasoning and significantly enhancing performance across both polynomial and NP-hard task categories.

Table 5: Ablation study on the impact of different thought types.

Method	Polynomial Tasks	NP-Hard Tasks
Llama3-8B (w/o thoughts)	0.752	0.509
Llama3-8B (action-only)	0.741	0.488
Llama3-8B (full thoughts)	**0.920**	**0.552**

I.2 Comparison with Classical Heuristics on MIS

We benchmarked Llama-GT against classical baselines including Greedy, Luby's Algorithm(Luby, 1986), and BDOne(Chang et al., 2017). As the Best-of-N (BoN) value increases, Llama-GT demonstrates significant performance improvements. On both easy and hard instances, Llama-GT with higher BoN values not only surpasses heuristic approaches but also achieves performance comparable to the Gurobi solver, particularly excelling in high-difficulty settings.

Table 6: Comparison on MIS Problem (Average Solution Size)

Method	MIS_easy	MIS_hard
Llama-GT (N=1)	4.472	11.844
Llama-GT (N=16)	4.502	12.568
Llama-GT (N=32)	4.502	**12.570**
Greedy	4.502	12.562
Luby	4.502	12.568
BDOne	4.502	12.568
Gurobi	4.502	**12.570**

I.3 Relative Improvements Across Different Model Scales

Due to computational constraints, we could not fine-tune the Llama3-70B model, but instead evaluated generalization trends using smaller-scale models. Our analysis reveals that the relative performance gain from thought integration diminishes with increasing model size, suggesting that larger models inherently possess stronger reasoning capabilities. Nevertheless, thought guidance remains valuable—it significantly enhances the capabilities of smaller models, potentially providing a more computationally efficient alternative to pure model scaling. We recommend that future research investigate this hypothesis across a broader range of model sizes to better understand the scaling dynamics.

Table 7: Performance Across Model Scales, *inf means the pre-finetuning result is 0.0

Task	**Llama3.2-3B**[9]	**Llama3-8B**
MIS	0.988 (+766.7%)	0.994 (+300.8%)
Diameter	0.872 (+2625.0%)	0.954 (+1262.9%)
MCS	0.394 (+310.4%)	0.496 (+123.4%)
MIS (Hard)	0.870 (+21650.0%)	0.900 (+8900.0%)
Diameter (Hard)	0.358 (inf)	0.600 (+29900.0%)
MCS (Hard)	0.018 (inf)	0.036 (+1700.0%)

I.4 Automated Thought Dataset Synthesis via LLM-Driven Code Generation

We created a template to automate dataset generation for ten tasks using a LLM(Qwen2.5-Coder-32B-Instruct[10]). This automated approach generated valid code for the tasks, creating datasets of equivalent size to fine-tune the Meta-Llama-3-8B-Instruct model. We compared four configurations: 1) Origin: The base model, 2) w/o Thought: Fine-tuned with direct-answer supervision, 3) Llama-GT (w/ LLM-Design): Fine-tuned on datasets generated by LLM-synthesized code, and 4) Llama-GT (w/ Human-Design): Fine-tuned on datasets generated by human-designed code.

As shown in Figure 5, Llama-GT (w/ LLM-Design) significantly improves reasoning capabilities over the Origin model, outperforming w/o Thought on Polynomial tasks. However, on NP-hard tasks, the LLM-generated code struggles to produce high-quality thought sequences, resulting in performance similar to w/o Thought (0.509 vs. 0.508). While Llama-GT (w/LLM-Design) lags behind Llama-GT (w/Human-Design) in both categories, it offers advantages like lower construction costs for unseen problems and better compatibility with optimization methods.

J Performance Degradation on Certain NP-hard Problems

While our method demonstrates strong performance on specific NP-hard tasks such as MIS, achieving significant improvements over baseline models, we observe relatively modest gains for

[10] https://huggingface.co/Qwen/Qwen2.5-Coder-32B-Instruct

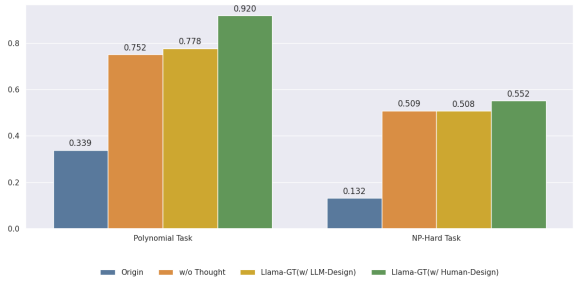

Figure 5: Performance comparison of Meta-Llama-3-8B-Instruct variants on Polynomial and NP-hard tasks: Base model (Origin), fine-tuned with direct-answer datasets (w/o Thought),datasets generated using LLM-synthesized code Llama-GT (w/LLM-Design), and datasets generated using human-designed code Llama-GT (w/Human-Design). The values represent the average ratio of the number of optimal solutions achieved across different tasks.

tasks like TSP, MCS, and GED. In certain instances, performance metrics even regressed below those of baseline models trained without thought mechanisms, particularly for the GED task.

We posit two principal factors contributing to this performance disparity:

First, these tasks inherently exhibit **higher computational complexity** compared to problems like MIS or MCP, presenting greater challenges for LLM-based solutions.

Second, the thought construction process for TSP, MCS, and GED primarily utilizes a single action type (e.g., "Add Nodes Based on the Given Optimal Solution," detailed in Appendix G). This monolithic approach operates as a **black-box mechanism**, limiting the incorporation of domain-specific prior knowledge or structural knowledge. Conversely, tasks like MIS benefit from diverse action types (e.g., both "Add Nodes Based on the Given Optimal Solution" and "Add Nodes Based on Simple Prior Knowledge") and state transformations (e.g., "Remove Nodes from the Graph"). These mechanisms effectively **embed structural priors** (e.g., mandatory inclusion of isolated nodes in independent sets) and **systematically prune the search space**, thereby reducing hallucination risks while enhancing solution quality.