MA-GTS: A Multi-Agent Framework for Solving Complex Graph Problems in Real-World Applications

Anonymous ACL submission

Abstract

Graph-theoretic problems arise in real-world applications like logistics, communication networks, and traffic optimization. These problems are often complex, noisy, and irregular, posing challenges for traditional algorithms. Large language models offer potential solutions but face several challenges, including limited accuracy, input length constraints, and suboptimal algorithm selection. To address these challenges, we propose **MA-GTS** (**M**ulti-**A**gent **G**raph **T**heory **S**olver), a multi-agent framework that decomposes these complex problems through agent collaboration. MA-GTS maps the implicitly expressed text-based graph data into clear, structured graph representations and dynamically selects the most suitable algorithm based on problem constraints and graph structure scale. We validate MA-GTS using the **G-REAL** dataset, a real-world-inspired graph theory dataset we created. Experimental results show that MA-GTS outperforms state-of-the-art methods in cost-effectiveness, accuracy, and scalability, achieving strong results on multiple benchmarks (G-REAL **93.6%**, GraCoRe **96.9%** NLGraph **98.4%**) with robust performance on both closed- and open-source base models.

1 Introduction

Graph-theoretic problems have extensive applications in domains such as logistics scheduling, communication networks, production planning, and traffic optimization (Li et al., 2023b). These problems typically involve a large number of nodes and edges, coupled with complex constraints and dynamic variations, making their solution highly challenging (Bondy and Murty, 2008). Despite significant advancements in graph theory and algorithmic design, traditional approaches remain computationally expensive and inefficient when handling large-scale, high-complexity problems. Existing methods, including exact algorithms, greedy strategies, and dynamic programming (Bellman, 1966), perform well on small-scale instances. However, as problem size increases, their computational complexity and memory requirements grow exponentially, rendering them impractical for real-world applications. While heuristic methods (Kokash, 2005) can improve performance under specific conditions, they often suffer from local optima and require extensive parameter tuning and model selection. Therefore, developing efficient and scalable solution frameworks capable of addressing the computational demands and structural variability of complex graph-theoretic problems remains a critical research challenge.

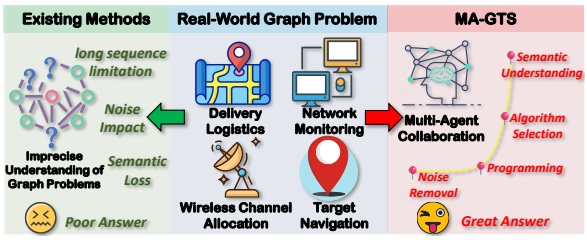

Figure 1: MA-GTS leverages multi-agent collaboration to overcome noise and semantic loss in real-world graph problems, leading to better answers.

Recent advancements in LLMs have spurred interest in their applications for graph-theoretic problems. Leveraging their natural language processing (NLP) capabilities, LLMs can serve as **scene interpreters** (mapping real-world problems to graph models), **graph extractors** (identifying graph structures from unstructured data), and **graph algorithm invokers** (assisting in solving and optimizing graph-based problems), addressing certain limitations of traditional algorithms. However, significant challenges remain in existing methods (LLMs and simple multi-agent framework). Figure 1 clearly illustrates the challenges existing methods face when addressing real-world graph problems. **Firstly**, LLMs rely on statistical pattern matching rather than strict mathematical computations, limiting their reasoning accuracy and making them unreliable for NP-hard prob-

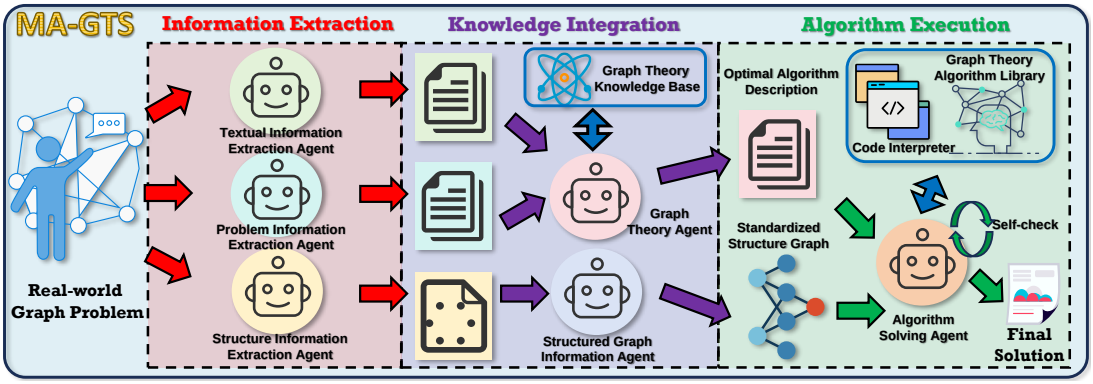

Figure 2: MA-GTS framework for solving real-world graph problems, consisting of three layers: Information Extraction, Knowledge Integration, and Algorithm Execution, each with specialized agents.

lems (Hochba, 1997). **Secondly**, their ability to handle large-scale graphs is limited by the Transformer (Vaswani, 2017) architecture's context window and computational complexity, which restricts their capacity to capture global information. **Finally**, LLMs lack the ability to decompose and map real-world graph theory problems, which often contain complex textual noise and implicit graph structures. In summary, existing methods struggle to effectively handle long texts and graph problems in real-world scenarios. Problems like disordered nodes, noisy text, and poor algorithm choices can all affect the quality of graph modeling, text understanding, and the interpretability of reasoning. These limitations highlight the inadequacy of existing methods for solving complex graph-theoretic problems in real-world applications and underscore the need for more efficient and scalable paradigms.

To tackle these challenges, we propose **MA-GTS**(**M**ulti-**A**gent **G**raph **T**heory **S**olver), an innovative multi-agent framework designed to address complex real world graph-theoretic problems through agent collaboration and competition. Figure 2 illustrates the framework, which incorporates a multi-agent coordination mechanism allowing agents to perform local searches independently while sharing information and cooperating, thus improving solution efficiency and accuracy. MA-GTS analyzes the original real-world problem textual data, filters out noise, and extracts key graph data and problem-specific details, reducing the text length that LLMs must process and enhancing reasoning efficiency. MA-GTS selects the optimal graph algorithm based on refined text and adjusts the graph's textual representation to match the algorithm, improving reasoning and solution quality. This coordination mitigates the limitations of LLMs in implicit graph structure modeling, ensuring efficient solutions for complex graph tasks. Additionally, dynamic agent interactions enable the framework to address large-scale problems and adapt to complex constraints and dynamic changes.

To validate the effectiveness of the multi-agent framework, we introduce the **G-REAL** dataset, designed to simulate complex graph theory problems relevant to real-world scenarios. Unlike traditional datasets that rely on simple textual descriptions of graph structures, G-REAL better reflects practical applications for large-scale models. Experiments comparing MA-GTS with state-of-the-art open-source and closed-source LLMs (including three closed-source and three open-source models), as well as with a general multi-agent framework and a graph-specific multi-agent framework, show that MA-GTS significantly outperforms existing LLMs and multi-agent frameworks in terms of efficiency and accuracy, under both direct reasoning and Chain of Thought (CoT) (Wei et al., 2022) reasoning settings. Notably, it excels in solving large-scale problems with complex constraints, offering superior scalability, robustness, and cost-effectiveness. The primary contributions of this study are as follows:

- First, we propose an innovative multi-agent framework, MA-GTS, which overcomes the limitations of traditional graph theory algorithms in large-scale complex problems, achieving state-of-the-art performance in our tests.
- Second, we constructed a real-world graph theory dataset, G-REAL, that aligns with practical needs, providing the necessary data support for validating the effectiveness of the algorithm.
- Finally, by introducing novel collaboration mechanisms and strategies, we achieve efficient and precise graph theory problem-solving within the multi-agent system, demonstrating its substantial potential in real-world application scenarios.

Figure 3: This figure details the G-REAL dataset's composition and features, along with the full MA-GTS graph problem-solving pipeline, outlining each component's functions and input/output formats.

2 Related Work

LLMs for Graph: Recent advancements in LLMs for graph tasks have led to significant contributions in methodology and evaluation. These tasks are often classified into Enhancer, Predictor, and Alignment types (Li et al., 2023b). Notably, (Pan et al.) presents a roadmap for unifying LLMs with Knowledge Graphs (KGs), while (Chai et al., 2023) proposes an end-to-end method for solving graph-related problems, (Cao et al., 2024) improves LLMs' understanding of graph structures by addressing positional biases and incorporating an external knowledge base. On the evaluation front, several benchmarks have been introduced. NLGraph (Wang et al., 2024) offers a simple test dataset for graph tasks, and GPT4Graph (Guo et al., 2023) evaluates LLM capabilities on semantic tasks. GraCoRe (Yuan et al., 2025) comprehensively verifies the graph understanding and reasoning capabilities of LLM. In addition to these representative benchmarks, ProGraph (Li et al., 2024a), GraphArena (Tang et al., 2024), GLBench (Li et al., 2024c), etc. are also widely used. Other notable works include (Liu and Wu, 2023), which assesses LLMs in graph data analysis, and (Perozzi et al., 2024), which designs a hint method for graph tasks.

LLM Agents: Several multi-agent frameworks have been proposed to improve coordination and efficiency in complex tasks. MetaGPT (Hong et al., 2023) embeds human workflows into LLMs to reduce hallucinations. CAMEL (Li et al., 2023a) enables autonomous agent cooperation aligned with human goals, and its extension OWL (Hu et al., 2025) builds on this. AutoGen (Wu et al., 2023) offers a flexible framework for customizing agent interactions via natural language and code. Additionally, (Li et al., 2024b) addresses simple graph problems. Multi-agent frameworks like GraphTeam (Li et al., 2024b), GCoder (Zhang et al., 2024), and GraphAgent (Hu et al., 2024) can enhance the reasoning ability of LLMs through multiple interactions, but they are mainly applied to standard graph structures, and their effectiveness on real-world graph theory problems remains uncertain.

3 MA-GTS

We consider a real-world graph problem P, modeled as a graph G. The system uses a graph theory knowledge base $\mathcal{K}_{\mathcal{G}}$ and an algorithm library $\mathcal{L}_{\text{code}}$ to support problem understanding and solving. From P, it extracts the textual description $\mathcal{T}$, identifies the problem type $\mathcal{P}$, and constructs the graph structure G. The objective is to automatically select an appropriate algorithm Alg^*, apply it to the structured graph G', and iteratively optimize the solution S^n.

The MA-GTS framework adopts a hierarchical processing paradigm, comprising three layers: the **Information Extraction Layer(IEL)**, the **Knowledge Integration Layer(KIL)**, and the **Al-**

gorithm Execution Layer(AEL). These layers interact through a hierarchical collaborative communication mechanism, enabling an end-to-end pipeline that processes unstructured data and solves complex graph-theoretic problems. Additionally, to support the knowledge base of MA-GTS, we have constructed the Graph Theory Knowledge Base and Graph Theory Algorithm Library. More information about them in the Appendix A.

The IEL processes text and structured data to extract graph information and identify problem types for standardized input. The KIL builds structured graph data using graph theory and optimization to enhance accuracy and scalability. The AEL runs specified algorithms and performs self-checks to efficiently solve complex graph problems. Figure 3 shows each agent's function by layer.

By leveraging agent collaboration, MA-GTS ensures efficient problem-solving, high scalability, and adaptability to complex constraints, offering a novel solution for real-world graph-theoretic challenges. The specific functionalities of each agent are detailed as follows:

3.1 Information Extraction Layer (IEL)

The IEL extracts relevant information from text and unstructured data, structures it for downstream use, and filters out irrelevant content to sharpen problem-specific details. It also captures implicit graph structures to boost efficiency and reduce the effects of text length on LLMs inference.

Textual Information Extraction Agent (TIEA): The TIEA analyzes real-world graph problems to extract key textual information unrelated to graph structure or solution goals. Using NLP, it identifies and structures context, background, entities, concepts, and definitions, organizing semantic content to support later analysis. The output is standardized for downstream processing.

Graph Structure Information Extraction Agent (GSIEA): The GSIEA extracts implicitly embedded graph-structural information from text, particularly structured formats like tables, lists, adjacency matrices, or edge lists. It parses these inputs to identify nodes, edges, weights, and other topological properties, converting them into standardized graph representations (e.g., adjacency matrices, lists). This transformation enables downstream agents to efficiently use the extracted data for problem solving.

Problem Information Extraction Agent (PIEA): The PIEA leverages LLMs' problem classification capabilities to analyze real-world graph-theoretic problems, identify their types, and extract key components. It classifies problems (e.g., shortest path, network flow, graph matching), extracts relevant constraints and objectives, and outputs the information in a structured format. This guidance improves the accuracy and efficiency of downstream problem-solving agents. Formally,the operation of IEL is:

$$\begin{gathered}\mathcal{T} \leftarrow A_{TIEA}(P), \mathcal{P} \leftarrow A_{PIEA}(P), \\ G \leftarrow A_{GSIEA}(P), IEL_{output} = (\mathcal{T}, \mathcal{P}, G)\end{gathered} \tag{1}$$

where P is graph theory problem and A_* is a different agent in IEL, $(\mathcal{T}, \mathcal{P}, G)$ represent the extracted text information, question information and graph structure information respectively.

Algorithm 1 Pipeline of MA-GTS

Input: Real-world graph problem P, graph theory knowledge base $\mathcal{K}_{\mathcal{G}}$, graph theory algorithm library $\mathcal{L}_{code}$., self check number N_{check}
Output: Optimized solution S^n

Step 1: Information Extraction Layer
Extract textual information: $\mathcal{T} \leftarrow A_{TIEA}(P)$
Identify problem type: $\mathcal{P} \leftarrow A_{PIEA}(P)$
Extract graph structure: $G \leftarrow A_{GSIEA}(P)$
Generate extracted information set: $(\mathcal{T}, \mathcal{P}, G)$
Step 2: Knowledge Integration Layer
Select best algorithm:
  $\mathcal{L}_{\mathcal{P}} \leftarrow A_{GTA}(\mathcal{T}, \mathcal{P}, \mathcal{K}_{\mathcal{G}})$
  $Alg^* \leftarrow \arg\operatorname{opt}_{Alg_i \in \mathcal{L}_{\mathcal{P}}} A_{GTA}(Alg_i, \mathcal{T})$
Get structured graph: $G' \leftarrow A_{SGIA}(G)$
Define structured problem: (G', Alg^*)
Step 3: Algorithm Execution Layer
Load algorithm code:
  $Code_{Alg^*} \leftarrow A_{ASA}(Alg^*, \mathcal{L}_{code})$
Get algorithm output :
  $S_{code} \leftarrow A_{ASA}Coding(Code_{Alg^*}, G')$
Get optimized solution S^n :
  $S^0 \leftarrow A_{ASA}(S_{code}, Alg^*, G')$;
for $i = 1, 2, \cdots, N_{check}$ **do**
  $S^n \leftarrow A_{ASA}(S^{n-1}, Alg^*, G')$;
end for

3.2 Knowledge Integration Layer (KIL)

The primary objective of this layer is to construct structured graph data with high representational capacity and integrate graph-theoretic principles for advanced modeling, thus enhancing the efficiency of the solution and the quality of optimization.

Structured Graph Information Agent (SGIA): The SGIA standardizes graph data from the GSIEA for efficient, consistent, and usable output. It cleans, deduplicates, and optimizes raw data into formats compatible with diverse environments to ensure accuracy. Additionally, it optimizes data storage and indexing based on algorithm requirements, enhancing computational efficiency for

large-scale graphs. Without this agent, data inconsistencies, redundancy, and unoptimized structures could hinder algorithm performance. As a key component of MA-GTS, it ensures data standardization and optimization for efficient, scalable problem-solving.

Graph Theory Agent (GTA): The GTA integrates information from the TIEA and PIEA with a Graph Theory Knowledge Base to analyze graph problems and find optimal solutions, improving LLM inference efficiency. It models the input problem by extracting key features such as type, constraints, and structural complexity, then queries the **Graph Theory Knowledge Base** to select the most suitable solution method from classical algorithms (e.g., shortest path, maximum flow, graph matching) (Gallo and Pallottino, 1988; Goldberg and Tarjan, 1988) and heuristic techniques. By matching problems to algorithms, it reduces inefficient exhaustive searches, cutting computational costs and improving solution quality. Additionally, it also guides multi-agent collaboration, allowing the AEL to directly invoke optimal algorithms for efficient, scalable execution. Without it, LLMs risk poor strategy selection, high computation, and lower efficiency. As a key MA-GTS component, it ensures effective algorithm selection and inference in complex graph tasks. Formally,the operation of KIL is:

$$\begin{aligned} \mathcal{L}_{\mathcal{P}} &\leftarrow A_{GTA}(\mathcal{T}, \mathcal{P}, \mathcal{K}_{\mathcal{G}}), \\ Alg^* &\leftarrow \arg\operatorname{opt}_{Alg_i \in \mathcal{L}_{\mathcal{P}}} A_{GTA}(Alg_i, \mathcal{T}), \\ G' &\leftarrow A_{SGIA}(G), KIL_{output} = (G', Alg^*) \end{aligned} \tag{2}$$

where $\mathcal{L}_{\mathcal{P}}$ represents the set of graph theory algorithms selected by GTA based on textual and problem-specific information, $\mathcal{K}_{\mathcal{G}}$ denotes the Graph Theory Knowledge Base, Alg^* refers to the algorithm suitable for the given graph size, and G' stands for the normalized graph structure data.

3.3 Algorithm Execution Layer (AEL)

The primary goal of this layer is to integrate multiple algorithmic paradigms, ensuring efficient, scalable, and robust solutions under various constraints. Without it, the MA-GTS framework would rely solely on LLM-based inference, leading to high computational costs, instability, or suboptimal outcomes. As the computational core, the AEL enables the efficient solution of complex graph-theoretic problems across varying scales and complexities.

Algorithm Solving Agent (ASA): The ASA is the core computational unit of the AEL, responsible for solving problems by executing algorithmic functions based on the optimal strategy selected by the GTA and the structured graph data processed by the SGIA. It utilizes a **Graph Theory Algorithm Library** that integrates exact algorithms (Noto and Sato, 2000) and heuristic approaches, ensuring suitable solutions across various problem scenarios. After computation, the agent performs result integration and verification through cross-validation, error analysis, and constraint checking to ensure correctness. The ASA also offers explainable reasoning with inference paths, key decisions, and optimization steps for transparency. As MA-GTS's computational core, it delivers efficient, robust, and scalable solutions for complex graph problems. Formally,the operation of AEL is:

$$\begin{aligned} Code_{Alg^*} &\leftarrow A_{ASA}(Alg^*, \mathcal{L}_{code}), \\ S_{code} &\leftarrow A_{ASA}Coding(Code_{Alg^*}, G'), \\ S^0 &\leftarrow A_{ASA}(S_{code}, Alg^*, G'), \end{aligned} \tag{3}$$

where $\mathcal{L}_{code}$ represents the Graph Theory Algorithm Library, $Code_{Alg^*}$ denotes the code obtained after optimal algorithm matching by ASA, S_{code} refers to the output generated by running the code, and S^0 represents the interpretable output obtained by combining the code output with problem review. Finally, ASA undergoes n rounds of self-checking, ultimately producing the final suitable result, S^n.

4 G-REAL

Existing datasets for evaluating LLMs' understanding and reasoning on graph-structured data are explicitly constructed. However, real-world graph-theoretic problems often involve rich textual semantic information and implicitly structured representations. To assess the performance of the MA-GTS framework on practical problems, we introduce **G-REAL**, a dataset that captures real-world graph problems. This dataset comprises three commonly encountered graph-theoretic challenges: (1) the optimization of logistics and delivery routes, (2) wireless network channel allocation, and (3) network monitoring optimization. These correspond to four fundamental graph problems: the **Traveling Salesman Problem (TSP)**, the **Minimum Graph Coloring Problem**, the **Minimum Vertex Cover Problem** and the **Shortest Path Problem**, respectively (Hoffman et al., 2013; Jensen and Toft, 2011; Hochbaum, 1982). They correspond to four common problems in real-world scenarios, namely the

	G-REAL				GraCoRe	NLGraph	
	TSP	**Coloring**	**Vetex Cover**	**Shortest Path**	**TSP**	**Shortest Path**	**Cycle**
#Graph	900	900	900	900	360	380	1150
Node Range	8 to 25	8 to 25	8 to 25	8 to 25	8 to 25	5 to 20	5 to 15
Real-World Problem	Delivery Logistics	Wireless Channel Allocation	Network Monitoring	Target Navigation	✖	✖	✖
Text Noise	✔	✔	✔	✔	✖	✖	✖

Table 1: Differences between different datasets.

Delivery Logistics Problem, the **Wireless Channel Allocation Problem**, the **Network Monitoring Problem**, and the **Target Navigation Problem**. The composition of G-REAL can be seen briefly in Figure 3. In this section, we provide a detailed description of the dataset's composition and construction methodology. More detail about G-REAL in Appendix C.

4.1 Data Collection

To mitigate the risk of data contamination in LLMs, which could lead to biased test accuracy due to prior exposure to training data, G-REAL employs several techniques, including randomized node naming, synthetic node descriptions, added textual noise, and randomly structured graph representations. Node names are generated by randomly combining the 26 letters of the alphabet, and synthetic node descriptions are created with arbitrary textual representations. For example, a node may be described as: *"Amber Plaza: A bustling central square surrounded by cafes, boutiques, and street performers."* These fictional descriptions ensure that LLMs cannot leverage prior knowledge, maintaining the integrity of the evaluation.

To improve dataset realism and obscure graph structure, we introduce textual noise to each instance, simulating real-world graph problems embedded in unstructured text. Graph structures are randomly generated, with each node assigned a unique name to reduce prior LLM exposure. Optimal and approximate solutions are generated for each problem type using established algorithms, providing benchmarks for evaluating both LLM and MA-GTS performance.

4.2 Data Statistics

To evaluate our framework's effectiveness in real-world graph-theoretic problems, we construct test datasets with graph sizes from 8 to 25 nodes for each problem type. Each sub-dataset includes 50 instances with distinct structures, offering both optimal and approximate solutions for a comprehensive assessment of robustness and generalization. A statistical summary is provided in Table 1.

4.3 Evaluation

For the TSP, Minimum Graph Coloring, Minimum Vertex Cover, and Shortest Path problems, the output includes both selected nodes and the final solution, requiring dual evaluation. To fully assess LLMs' graph reasoning, both output types are used as evaluation metrics. The model's performance is measured by verifying the accuracy of both the selected node set and the computed solution. The methodology for calculating the final accuracy is as follows: $ACC_{ALL} = 0.5 \cdot ACC_{\text{nodes}} + 0.5 \cdot ACC_{\text{result}}$, where ACC_{nodes} and ACC_{result} represent the accuracy of the node set and the predicted values, respectively, with a value of 1 for correct predictions and 0 for incorrect ones.

5 Experiments Setup

5.1 Datasets

To evaluate the reasoning capabilities of the MA-GTS framework across various graph-theoretic problem types, complexities, and domains, we used the **G-REAL** dataset alongside two benchmark datasets, **GraCoRe** (Yuan et al., 2025) and **NLGraph** (Wang et al., 2024), covering seven distinct graph-theoretic tasks. We selected three sub-tasks for evaluation: the TSP, shortest path problem, and Cycle problem in GraCoRe and NLGraph. Notably, both GraCoRe and G-REAL include TSP instances, both NLGraph and G-REAL include Shortest Path instances; however, the G-REAL TSP and Shortest Path is more complex and reflects real-world scenarios with implicit graph structure data. By comparing performance on these two types instances, we assess the model's ability to handle more intricate problems. The simpler tasks in NLGraph evaluate the generalization and robustness of MA-GTS. A summary of the differences between these datasets is provided in Table 1.

5.2 Baselines and Foundation Model

We compared three of OpenAI's latest closed-source models: *o3-mini*, *GPT-4o-mini*, and *GPT-3.5* (Achiam et al., 2023). Additionally, we evaluated three of the most recent open-source models: *Llama3-7b* (Touvron et al., 2023), *Qwen2.5-7b* (Bai et al., 2023) and Deepseek-V3-660B (Liu et al., 2024). For the evaluation methodology, we adopted both direct inference and CoT reasoning approaches. For the foundation model, we selected the GPT-4o-mini and Deepseek-V3-660B model, they respectively represent some of the more ad-

Model	Method	G-REAL				GraCoRe	NLGraph	
		Delivery Logistics Problem (TSP)	Wireless Channel Allocation Problem (Coloring)	Network Monitoring Problem (Vetex Cover)	Target Navigation Problem (Shortest Path)	TSP	Shortest Path	Cycle
o3-mini	**Direct**	11.8%	80.1%	68.7%	47.1%	79.7%	**100.0%**	97.3%
	CoT	12.9%	83.1%	72.8%	41.8%	80.0%	98.4%	97.8%
GPT-4o-mini	**Direct**	2.5%	23.4%	0.3%	7.1%	1.1%	27.3%	50.9%
	CoT	3.1%	25.1%	0.0%	6.4%	1.1%	27.6%	51.1%
GPT-3.5	**Direct**	0.1%	0.7%	2.5%	4.0%	1.9%	30.5%	50.0%
	CoT	2.1%	7.6%	4.8%	3.6%	1.6%	34.7%	49.9%
Qwen2.5-7B	**Direct**	0.6%	16.2%	17.4%	4.6%	3.8%	22.1%	49.6%
	CoT	0.6%	8.8%	8.5%	5.8%	3.0%	27.3%	52.7%
Llama3-7B	**Direct**	3.6%	10.1%	7.2%	4.3%	0.3%	12.6%	53.7%
	CoT	4.1%	14.3%	6.7%	4.2%	0.3%	19.4%	50.9%
Deepseek-V3-660B	**Direct**	4.9%	27.2%	21.1%	11.4%	10.5%	50.8%	78.1%
	CoT	5.5%	28.3%	22.2%	32.2%	18.8%	92.9%	77.8%
OWL (GPT-4o-mini)	**Multi-Agent**	10.2%	47.4%	7.8%	19.1%	4.4%	36.3%	49.7%
GraphTeam (GPT-4o-mini)	**Multi-Agent**	8.8%	90.0%	12.0%	87.7%	84.4%	98.4%	**100.0%**
MA-GTS (Deepseek-V3-660B)	**Multi-Agent**	76.2%	88.2%	**99.1%(↑26.3%)**	88.2%	93.1%	93.8%	**100.0%**
MA-GTS (GPT-4o-mini)	**Multi-Agent**	**94.9%(↑82%)**	**94.5%(↑4.5%)**	93.2%	**91.7%(↑4%)**	**96.9%(↑12.5%)**	97.8%(↓2.2%)	98.9%

Table 2: The performance comparison of LLMs and MA-GTS on G-REAL and two benchmarks is shown. Red text indicates MA-GTS's accuracy improvement over the best LLM, while green text highlights the opposite. GPT-4o-mini was used as the base model for MA-GTS.

vanced open-source and closed-source models. Furthermore, we conducted a comparative analysis of the performance of OWL (Hu et al., 2025) and GraphTeam (Li et al., 2024b), which respectively represent a general-purpose multi-agent framework and a graph-theoretic multi-agent framework. Regarding the final test results, for each task, we used the accuracy of the final computed solution as the primary evaluation metric. More details about models in Appendix B.

6 Results and Analysis

We evaluate the performance of our framework against other LLMs on graph theory problems, with results presented in Table 2. MA-GTS outperforms all baselines, achieving state-of-the-art results and matching the performance of the leading o3-mini model on simpler problems. We also assess the MA-GTS framework from multiple perspectives.

6.1 Performance on real-world problems

As shown in Table 2, G-REAL provides four real-world graph theory problems, with TSP being the most complex. Based on the results from these problems, MA-GTS demonstrates superior performance, achieving an accuracy rate exceeding 90% across all tests. Notably, in the case of the TSP, MA-GTS outperforms the o3-mini model by 82%. Even when built upon the open-source DeepSeek model, MA-GTS still achieves strong performance. Furthermore, when compared to the GPT-4o-mini model, MA-GTS significantly improves its performance from 3.1% to 94.9%, marking a substantial increase. This clearly underscores the effectiveness of our framework. Additionally, it is evident that, aside from the o3-mini model, other models exhibit subpar performance on the G-REAL dataset. It is particularly interesting that the performance gap between the two open-source and two closed-source models is minimal, suggesting that the complexity of the problems may lead to a consistent decline in performance, an issue that warrants further investigation. Overall, MA-GTS stands out for its advanced capabilities and generalization when handling complex graph theory problems.

6.2 Performance on simple problem

Table 2 shows that for simpler graph theory problems, such as the Shortest Path and Cycle problems from the NLGraph dataset, the o3-mini model performs exceptionally well, with MA-GTS also showing strong results. Specifically, for Shortest Path problem, the gap between MA-GTS and o3-mini is just 2.2%, and MA-GTS performs equally well on the Cycle problem. In contrast, other models perform less satisfactorily. The MA-GTS framework, based on the GPT-4o-mini model, significantly enhances the accuracy of the 4o model, bringing it on par with the o3-mini. Overall, MA-GTS demonstrates excellent performance across diverse textual descriptions and graph structures, highlighting its remarkable generalization capabilities.

6.3 G-REAL effectiveness analysis

To evaluate the performance of LLMs and MA-GTS on real-world graph theory problems, we constructed the G-REAL dataset. As shown in Table 2, the performance of existing LLMs on the G-REAL dataset is suboptimal. To validate the effectiveness of this dataset, we compared it with the TSP problem from the GraCoRe Benchmark, testing problems with node sizes ranging from 8 to 25, consistent with the scale of G-REAL. From this comparison, we observe that on the G-REAL dataset, which includes text complexity, added text noise, and node name shuffling, the o3-mini model

	TSP			Coloring			VertexCover			Shortest Path		
	Inp.Tokens(k)	Out.Tokens(k)	Price($)	Inp.Tokens(k)	Out.Tokens(k)	Price($)	Inp.Tokens(k)	Out.Tokens(k)	Price($)	Inp.Tokens(k)	Out.Tokens(k)	Price($)
o3-mini	2.31	4.98	0.0244	0.69	6.83	0.0309	0.6	9.73	0.0443	0.87	3.12	0.0147
MA-GTS(GPT-4o-mini)	13.32	**4.56**(↓8.4%)	**0.0047**(↓80.7%)	6.79	**2.57**(↓62.4%)	**0.0025**(↓91.9%)	6.39	**2.31**(↓76.2%)	**0.0023**(↓94.8%)	7.36	**2.42**(↓22.4%)	**0.0025**(↓82.9%)

Table 3: Comparison of inference costs between MA-GTS and o3-mini model on G-REAL.

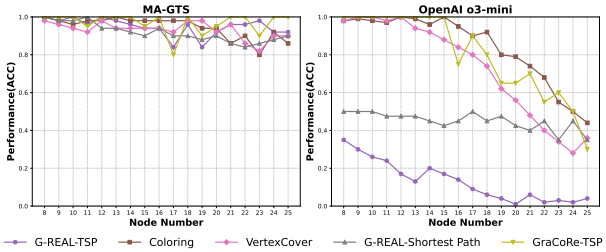

Figure 4: Performance of different problems across varying node numbers (MA-GTS v.s. o3-mini).

	G-REAL			Average error rate
	TSP	Coloring	Vetex Cover	
GPT-4o-mini(Tool use)	30.8%	39.0%	4.6%	75.0%
w/o IEL	12.5%	42.2%	14.6%	19.4%
w/o KIL	7.8%	37.1%	12.8%	1.0%
w/o AEL	4.6%	32.1%	7.4%	3.2%
MA-GTS(GPT-4o-mini)	**94.9%**	**94.5%**	**93.2%**	**0.5%**

Table 4: Ablation Experiments for Each Layer of MA-GTS ("Tool use" refers to the utilization of only the algorithm library we have constructed).

performs poorly, with its accuracy dropping from 79.7% in GraCoRe to 11.8%. In contrast, the MA-GTS framework appears unaffected by the complexities of real-world graph theory problems, maintaining performance above 90%. This result indirectly supports the validity of the G-REAL dataset and shows the stability of the MA-GTS.

6.4 Impact of Node Size

To evaluate the impact of node scale on LLMs in complex graph theory problems, we tested the performance of MA-GTS and the o3-mini model on four complex graph problem datasets, with node sizes ranging from 8 to 25. The results, shown in Figure 4, clearly demonstrate that as the number of nodes increases, the performance of the o3-mini model deteriorates, particularly in the TSP problem from G-REAL. For node sizes greater than 20, the o3-mini model is unable to produce correct answers. In contrast, under the MA-GTS, the effect of node size is less pronounced. Even with more than 20 nodes, MA-GTS maintains high prediction accuracy and stability. It highlights both the effectiveness and superiority of MA-GTS. Performance of MA-GTS on larger node scales is discussed in the Appendix D.

6.5 Cost Analysis

Since MA-GTS requires multiple agent calls to model APIs for inference, cost considerations arise. To address this, we compared the inference costs of MA-GTS based on the GPT-4o-mini model with the o3-mini model, as shown in Table 3. Surprisingly, MA-GTS incurs significantly lower costs than the o3-mini model. The o3-mini model, in contrast, has hidden reasoning tokens during inference, leading to long, concealed reasoning processes even in direct inference scenarios. As shown in the table, the inference cost of MA-GTS is about one-tenth to one-twentieth of the o3-mini model, requiring far fewer inference tokens. Moreover, MA-GTS achieves far better results than o3-mini, demonstrating its high cost-effectiveness in delivering more accurate outcomes at a lower cost. Runtime efficiency is discussed in the Appendix D.

6.6 Ablations Studies and Analyses

To validate the effectiveness of each layer in MA-GTS, we conducted ablation experiments, with results shown in Table 4. It demonstrates that each layer is crucial, and removing any layer significantly affects the final results. Although the IEL layer has the smallest impact on accuracy, its absence leads to a substantial increase in error rate (19%), highlighting its role in maintaining stability. The absence of the AEL layer results in the greatest accuracy loss. Even when a module is removed, MA-GTS still improves the accuracy of the base model, validating the framework's effectiveness. Additionally, when inference is performed using only the GPT-4o-mini model with the constructed algorithm library, accuracy improves, but the error rate remains high (75%). For graph sizes larger than 10 nodes, the model struggles to correctly invoke algorithms, further demonstrating the robustness and generalizability of MA-GTS.

7 Conclusion

We introduces MA-GTS, a Multi-Agent Framework for solving real-world graph theory problems, validated using the G-REAL dataset. Performance comparisons across various LLMs show that MA-GTS achieves high accuracy, stability, and cost-effectiveness, excelling in both complex and simpler graph problems. With accuracy consistently above 90%, MA-GTS outperforms existing methods, maintaining stability across different problem scales and being well-suited for larger graphs. Future work will focus on scaling to even larger problems and improving cost-efficiency.

Limitations

Although the MA-GTS framework demonstrates significant advantages in addressing complex graph-theoretic problems, several limitations remain. First, while the G-REAL dataset provides valuable support for validating the framework's effectiveness, it may not fully capture the diversity of real-world graph problems, thus limiting the generalizability of the framework. Second, the MA-GTS framework may still require substantial computational resources when handling large-scale problems, particularly in resource-constrained environments. Moreover, despite the improvements made in enhancing LLMs' graph structure modeling capabilities, LLMs may still encounter performance bottlenecks when dealing with graphs that exhibit highly dependent relationships or specialized structures. Finally, the current capabilities of open-source model invocation tools are insufficient, which may impact the stability of the MA-GTS framework.

References

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

Jinze Bai, Shuai Bai, Yunfei Chu, Zeyu Cui, Kai Dang, Xiaodong Deng, Yang Fan, Wenbin Ge, Yu Han, Fei Huang, et al. 2023. Qwen technical report. *arXiv preprint arXiv:2309.16609*.

Richard Bellman. 1966. Dynamic programming. *science*, 153(3731):34–37.

John Adrian Bondy and Uppaluri Siva Ramachandra Murty. 2008. *Graph theory*. Springer Publishing Company, Incorporated.

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 preprint 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 preprint arXiv:2310.05845*.

Giorgio Gallo and Stefano Pallottino. 1988. Shortest path algorithms. *Annals of operations research*, 13(1):1–79.

Andrew V Goldberg and Robert E Tarjan. 1988. A new approach to the maximum-flow problem. *Journal of the ACM (JACM)*, 35(4):921–940.

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 preprint arXiv:2305.15066*.

Dorit S Hochba. 1997. Approximation algorithms for np-hard problems. *ACM Sigact News*, 28(2):40–52.

Dorit S Hochbaum. 1982. Approximation algorithms for the set covering and vertex cover problems. *SIAM Journal on computing*, 11(3):555–556.

Karla L Hoffman, Manfred Padberg, Giovanni Rinaldi, et al. 2013. Traveling salesman problem. *Encyclopedia of operations research and management science*, 1:1573–1578.

Sirui Hong, Xiawu Zheng, Jonathan Chen, Yuheng Cheng, Jinlin Wang, Ceyao Zhang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, et al. 2023. Metagpt: Meta programming for multi-agent collaborative framework. *arXiv preprint arXiv:2308.00352*.

Mengkang Hu, Yuhang Zhou, Wendong Fan, Yuzhou Nie, Bowei Xia, Tao Sun, Ziyu Ye, Zhaoxuan Jin, Yingru Li, Zeyu Zhang, Yifeng Wang, Qianshuo Ye, Ping Luo, and Guohao Li. 2025. Owl: Optimized workforce learning for general multi-agent assistance in real-world task automation.

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

Tommy R Jensen and Bjarne Toft. 2011. *Graph coloring problems*. John Wiley & Sons.

Natallia Kokash. 2005. An introduction to heuristic algorithms. *Department of Informatics and Telecommunications*, pages 1–8.

Guohao Li, Hasan Abed Al Kader Hammoud, Hani Itani, Dmitrii Khizbullin, and Bernard Ghanem. 2023a. Camel: Communicative agents for "mind" exploration of large language model society. In *Thirty-seventh Conference on Neural Information Processing Systems*.

Xin Li, Weize Chen, Qizhi Chu, Haopeng Li, Zhaojun Sun, Ran Li, Chen Qian, Yiwei Wei, Chuan Shi, Zhiyuan Liu, et al. 2024a. Can large language models analyze graphs like professionals? a benchmark, datasets and models. *Advances in Neural Information Processing Systems*, 37:141045–141070.

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

Yuhan Li, Zhixun Li, Peisong Wang, Jia Li, Xiangguo Sun, Hong Cheng, and Jeffrey Xu Yu. 2023b. A survey of graph meets large language model:

Progress and future directions. *arXiv preprint arXiv:2311.12399*.

Yuhan Li, Peisong Wang, Xiao Zhu, Aochuan Chen, Haiyun Jiang, Deng Cai, Victor W Chan, and Jia Li. 2024c. Glbench: A comprehensive benchmark for graph with large language models. *Advances in Neural Information Processing Systems*, 37:42349–42368.

Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, et al. 2024. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*.

Chang Liu and Bo Wu. 2023. Evaluating large language models on graphs: Performance insights and comparative analysis. *arXiv preprint arXiv:2308.11224*.

Masato Noto and Hiroaki Sato. 2000. A method for the shortest path search by extended dijkstra algorithm. In *Smc 2000 conference proceedings. 2000 ieee international conference on systems, man and cybernetics.'cybernetics evolving to systems, humans, organizations, and their complex interactions'(cat. no. 0*, volume 3, pages 2316–2320. IEEE.

Shirui Pan, Linhao Luo, Yufei Wang, Chen Chen, Jiapu Wang, and Xindong Wu. Unifying large language models and knowledge graphs: A roadmap, 2023. *arXiv preprint arXiv:2306.08302*.

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 preprint arXiv:2402.05862*.

Jianheng Tang, Qifan Zhang, Yuhan Li, and Jia Li. 2024. Grapharena: Benchmarking large language models on graph computational problems. *arXiv preprint arXiv:2407.00379*.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. 2023. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*.

A Vaswani. 2017. Attention is all you need. *Advances in Neural Information Processing Systems*.

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.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837.

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Shaokun Zhang, Erkang Zhu, Beibin Li, Li Jiang, Xiaoyun Zhang, and Chi Wang. 2023. Autogen: Enabling next-gen llm applications via multi-agent conversation framework. *arXiv preprint arXiv:2308.08155*.

Zike Yuan, Ming Liu, Hui Wang, and Bing Qin. 2025. Gracore: Benchmarking graph comprehension and complex reasoning in large language models. In *Proceedings of the 31st International Conference on Computational Linguistics*, pages 7925–7948.

Qifan Zhang, Xiaobin Hong, Jianheng Tang, Nuo Chen, Yuhan Li, Wenzhong Li, Jing Tang, and Jia Li. 2024. Gcoder: Improving large language model for generalized graph problem solving. *arXiv preprint arXiv:2410.19084*.

A MA-GTS Details

A.1 Graph Theory Knowledge Base

The Graph Theory Knowledge Base is a graph theory problem database that we have constructed, containing a wide range of common graph theory problems encountered in daily life, including both complex and simple ones. Each problem is associated with multiple optimal or approximate solution algorithms. For each algorithm, we provide a detailed description of its complexity, applicable conditions, and parameter settings, though it does not include corresponding code. This database can serve as a reference book for agents in graph theory. A specific example can be seen in Figure 5.

A.2 Graph Theory Algorithm Library

The Graph Theory Algorithm Library is a Python code repository that we have constructed, containing code corresponding to the graph theory algorithms in the Graph Theory Knowledge Base. This ensures the correctness of input parameters and helps maintain the stability of the MA-GTS framework. Each code snippet is accompanied by detailed parameter descriptions and is designed to accommodate various types of graph structure representations. A specific example can be seen in Figure 6.

A.3 Prompt Templates

In this section, I will introduce the prompts for each agent, which will be displayed in Figures 7 to 12.

B Details on baseline models

We evaluated 6 of the latest LLMs, including OpenAI o3-mini reasoning model, launched on January 31, 2025 and the latest open-source model, DeepSeek-V3. Table 6 presents more details on the models and their versions.

C Details on G-REAL

Existing graph theory benchmarks do not align with real-world scenarios. To better evaluate the ability of MA-GTS in solving graph theory problems in practical contexts and to test the performance gap between LLMs on structured textual graph data and implicit representations, we constructed the G-REAL dataset. This dataset contains three common real-world problems, with detailed information provided in the G-REAL section. It generates problem graphs of varying scales by randomly encoding node names and structures, with the naming conventions and sample problems illustrated in Figures 13 to 16.

D More experimental analysis

D.1 Large-scale node analysis

Why 8-25 enough: In our work, we chose to focus on graphs with 8–25 nodes, primarily due to the complexity and reasoning difficulty posed by realistic tasks. Unlike large-scale but structurally explicit graphs, the G-REAL dataset introduces substantial textual noise, implicit graph structures, and randomly named nodes. These factors make the problem setting significantly closer to real-world semantic reasoning scenarios and increase the overall problem difficulty. This differs from existing benchmarks, which typically construct graph-structured data using explicitly defined and concise textual descriptions.

More experiments: We have extended our experiments to include larger graph sizes. As shown in Table 7, we tested the TSP and Graph Coloring problems with 25, 30, 35, and 40-node graphs, with 5 instances evaluated for each size. The results show that even on larger graphs, our framework maintains high accuracy and stability. We plan to include more experiments on even larger graph sizes in future versions of the paper to further validate the scalability of our approach.

	25	**30**	**35**	**40**
TSP	0.8	0.6	0.8	0.8
Coloring	1	0.6	1	0.6

Table 5: This table shows the results of experiments conducted on TSP and Coloring problems with extended graph sizes of 25, 30, 35, and 40 nodes. For each size, 5 problem instances were tested.

D.2 Runtime analysis

G-REAL focuses on graph reasoning under complex semantic conditions, which more closely resemble real-world user scenarios. These tasks often contain intricate semantic information and irrelevant noise, posing significant challenges for LLMs. For instance, the Delivery Logistics Problem in G-REAL is a TSP instance. In contrast to existing TSP benchmarks—where nodes are ordered, connections are explicitly stated, and the problem type is clearly defined—G-REAL requires the model to infer all of this information from natural language.

This increases the difficulty of graph construction and makes reasoning more error.

We conducted a supplementary evaluation on the G-REAL-TSP task by randomly selecting five graphs with 15-node scales (Table 6). We measured the average solution time per problem instance, including task decomposition, tool invocation, and result verification. Compared to existing multi-agent frameworks, MA-GTS demonstrates a clear advantage in time efficiency. These results highlight that our framework is capable of maintaining high accuracy while keeping inference time relatively low, further validating its practical applicability.

	MA-GTS	**Graphteam**	**OWL**
Time use (s)	148.48	251.34	139.39
ACC (%)	100	0	0

Table 6: The table presents the results of testing on the G-Real-TSP problem using 5 randomly selected graphs with 15 nodes each.Base model is GPT-4o-mini.

```
{
  "graph_theory_problems": {
   "TSP(Traveling Salesman Problem)": [
    {
     "algorithm": "Brute Force",
     "solution_type": "Optimal",
     "description": "By exhaustively checking all possible paths, it finds the shortest route.",
     "suitable_graph_size": "Suitable for small graphs (up to 10 nodes) due to factorial time complexity
(O(n!)), as the computation time increases drastically with more nodes.",
     "time_complexity": "O(n!)",
     "input": {
      "graph": "A complete weighted graph represented as an adjacency matrix or edge list.",
      "start_node": "The starting node for the traveling salesman problem."
     }
    },
    {
     "algorithm": "Dynamic Programming (Held-Karp Algorithm)",
     "solution_type": "Optimal",
     "description": "Uses dynamic programming to reduce repeated calculations, building the global
solution from subproblems.",
     "suitable_graph_size": "Suitable for medium-sized graphs (up to 50 nodes). This algorithm has
higher time complexity, so it’ s more suitable for smaller to medium-sized instances.",
     "time_complexity": "O(n^2 * 2^n)",
     "input": {
      "graph": "A complete weighted graph represented as an adjacency matrix or edge list.",
      "start_node": "The starting node for the traveling salesman problem."
     }
    }, ... ...
```


Figure 5: Details of Graph Theory Knowledge Base

```
def transform_dict(input_dict):
    output_dict = {}
    for key, value in input_dict.items():
        new_list = []
        for item in value:
            for sub_key, sub_value in item.items():
                new_list.append((int(sub_key), int(sub_value)))
        output_dict[int(key)] = new_list
    return output_dict

def tsp_dynamic_programming(adjacency_list): ...

def tsp_greedy_nearest_neighbor(adjacency_list): ...

def graph_coloring_backtracking(adjacency_list): ...

def graph_coloring_greedy(adjacency_list):  ...

def vertex_cover_brute_force(adjacency_list): ...
```

Figure 6: Details of Graph Theory Algorithm Library

```
TIEA_SYS_PROMPT =
"""
Your task is to extract textual information from the input real-world graph theory
problem. This information should include background descriptions, context,
definitions of entities or concepts, and any other details not directly related to
graph structure or problem objectives. Output the results as a dictionary in the
following format:

{
   "context": "The background and contextual description of the problem",
   "entities": "A list of all entities or concepts mentioned",
   "definitions": "Definitions and explanations of terms involved"
}
Based on the input, complete the extraction and ensure the format is clear.
"""
```

Figure 7: Details of TIEA

```
PIEA_SYS_PROMPT =
"""
Your task is to extract the problem objectives and related details from the input real-
world graph theory problem. Clearly state the problem's goal (e.g., shortest path,
maximum flow, graph coloring), any constraints, and potential optimization
objectives.You need to explain in detail what the goal of the problem is. If you are
looking for a path, you need to give the starting and ending nodes. Output the results
as a dictionary in the following format:

{
   "objective": "The goal of the problem",
   "constraints": "Any constraints associated with the problem",
   "optimization": "Any explicit optimization objectives, if applicable"
}
Based on the input, complete the extraction and ensure the format is clear.
"""
```

Figure 8: Details of PIEA

```
GSIEA_SYS_PROMPT =
"""
Your task is to extract graph structure information from the input real-world graph theory problem. Ensure the
information is complete and concise, even if there are many nodes or edges. Follow these steps:

1. **Nodes**: List all nodes. If the number of nodes is too large, group them logically (e.g., by properties or
categories) and explain the grouping.

2. **Edges**: List all edges in a simplified format as tuples:
   - Each tuple contains the two connected nodes and, if applicable, essential attributes (e.g., weight,
direction).
   If the edges are too many, group them logically (e.g., by node, weight range) and explain the grouping.

3. **Graph Type**: Specify the type of graph (e.g., undirected, directed, weighted).

Output the results as a dictionary in the following format:
{
   "nodes": ["Node1", "Node2", "Node3", ...],
   "edges": [
       ("Node1", "Node2", {"weight": 5}),
       ("Node2", "Node3", {"direction": "one-way"}),
   ],
   "graph_type": "Type of the graph (e.g., undirected, directed, weighted)"
}
If grouping is applied, clearly state the grouping method and ensure **all information is complete**.
"""
```

Figure 9: Details of GSIEA

```
SGIA_SYS_PROMPT = """
You will receive a textual graph structure data, which contains the information of the
nodes and edges of the graph. Please convert it into a digital graph structure data in
a standard graph representation format. Note that you can only call the tool once.
You can use appropriate tools or codes to complete this task. You need to use the
"generate_adjacency_list" tool to convert the text into an adjacency list. Output the
results as a dictionary in the following format:
{
  "graph_type": "directed" or "undirected",
  "adjacency_list": {
    node_number: [(neighbor_number, weight)]
  },
  "node_mapping": {
    node_name: node_number
  }
}

**The output "adjacency_list" should be exactly the same as the output of the tool.**
"""
```

Figure 10: Details of SGIA

```
GTA_SYS_PROMPT = """
You are an expert in graph theory algorithms, and you have access to a comprehensive library of graph algorithms. Given the following two pieces of information:
1. **Text Information**: This includes details about the graph, such as its structure, number of nodes, number of edges, sparsity, and other properties. Based on this information, you should assess the scale and characteristics of the graph.
2. **Problem Information**: This defines the specific graph theory problem to solve (e.g., shortest path, graph connectivity, minimum spanning tree, maximum flow, graph coloring, etc.). You should choose the most appropriate algorithm to solve the problem based on its type.
3. **Graph Theory Algorithm Library:**: A library of graph theory algorithms, including the problem and graph size that each algorithm is suitable for.

Your task is to:
- Analyze the graph's scale and characteristics (e.g., small vs large graph, sparse vs dense).
- Choose the most suitable graph algorithm based on the problem type and graph properties (considering time and space complexity). In particular, the algorithm to be used is determined based on the number of nodes obtained based on the graph structure information.
- The algorithm function to be used is determined according to the **suitable_graph_size** description in the algorithm.
- Output a dictionary that includes:
   - **problem type**: Types of graph theory problems.
   - **algorithm**: The name of the selected algorithm.
   - **parameters**: The parameters required for the algorithm.(You only need to tell the retriever to retrieve the parameter name, not the entire parameter input data.)
   - **complexity**: The time complexity of the selected algorithm (brief description).
   - **description**: A brief explanation of why this algorithm is the best choice for the given problem.
Output the results as a dictionary in the following format:
{
   "problem": "Types of graph theory problems.",
   "algorithm": "The name of the selected algorithm.",
   "parameters": "The parameters required for the algorithm.",
   "complexity": "The time complexity of the selected algorithm (brief description).",
   "description": "A brief explanation of why this algorithm is the best choice for the given problem."
}

"""
```

Figure 11: Details of GTA

```
AGENT_ASA_SYS_PROMPT =  """
You are tasked with solving a graph-related problem using the provided input data. The input specifies the graph type, adjacency list, node mapping, problem type, and the algorithm to use. Please use the tools according to the given algorithm to get the final answer.

Your task:
1. Identify the algorithm to use from the "algorithm" key.
2. Extract the required inputs based on the algorithm's parameters. Ensure the inputs strictly follow the parameter requirements and format.
3. Use the appropriate algorithm tool to solve the problem.
4. Analyze the tool's output and summarize the final answer.

**Instructions for using the tool**:
- Identify the algorithm name from the input (e.g., Dijkstra, BFS).
- Use the parameters required for the algorithm tool exactly as described in the "algorithm" input.
- Ensure the input format matches the tool's strict parameter requirements.

**Output Requirements**:
1. Summarize the problem and the algorithm used.
2. Display the tool's output clearly.
3. Finally, you need to analyze the output of the tool, combine it with the node mapping information and question text information, and give the final appropriate answer.
"""
```

Figure 12: Details of ASA

Model	Version	Model Link
OpenAI o3-mini	o3-mini	https://platform.openai.com/docs/models/o1#o3-mini
GPT-4o-mini	gpt-4o-mini	https://platform.openai.com/docs/models/gpt-4o-mini
GPT-3.5	gpt-3.5-turbo	https://platform.openai.com/docs/models/gpt-3-5-turbo
Llama3-ins-8b	Meta-Llama-3-8B-Instruct	https://huggingface.co/meta-llama/Meta-Llama-3-8B-Instruct
Qwen2.5-7b-ins	Qwen2.5-7B-Instruct	https://huggingface.co/Qwen/Qwen2-7B-Instruct
Deepseek-V3	DeepSeek-V3-0324-660B	https://huggingface.co/deepseek-ai/DeepSeek-V3-0324

Table 7: More details about models.

PLACE = {
"Amber Plaza": "A bustling central square surrounded by cafes, boutiques, and street performers.",
"Beacon Tower": "The tallest building in the city, offering panoramic views and a rotating rooftop restaurant.",
"Cobalt Market": "A vibrant marketplace where merchants sell exotic goods and fresh produce from all over.",
"Duskwood Park": "A sprawling urban park filled with dense trees, walking trails, and a serene lake.",
"Echo Station": "The city′ s largest transportation hub, always alive with the sound of trains and announcements.",
"Flare Alley": "A narrow, colorful street lined with neon-lit bars and underground clubs.",
"Gilded Archway": "A historic landmark leading to the city′ s oldest district, adorned with intricate carvings.",
"Haven Docks": "The city′ s bustling port area, filled with cargo ships, seafood stalls, and lively taverns.",
"Ironbridge Crossing": "A massive steel bridge connecting the industrial zone with the city center.",
"Jade Fountain": "A tranquil plaza centered around a beautiful fountain made of green stone.",
"King′ s Row": "A luxurious shopping street lined with high-end stores and designer boutiques.",
"Lighthouse Point": "A scenic overlook by the bay with a historic lighthouse and picnic spots.",
"Moonlit Promenade": "A romantic walkway along the riverbank, lit by soft lanterns at night.",
"Nimbus Plaza": "A futuristic square surrounded by glass skyscrapers and interactive digital art installations.",
... ...

Figure 13: Details of Random Places

Our company handles deliveries across a busy urban area, and today we have 7 distinct delivery points to cover. The delivery driver will start from our central warehouse and needs to drop off packages at each location before returning to the warehouse. Since these delivery points are scattered throughout different parts of the city, we′ re looking to find the most efficient route to minimize the total distance traveled. This will help us save on fuel, reduce delivery times, and improve our overall efficiency.
The warehouse, is located near the city center. Each location represents a different type of business or residential area with unique delivery requirements:
Zenith Arena: A state-of-the-art stadium for concerts, sports events, and major public gatherings.
Pennywhistle Arcade: A vintage entertainment district with old-style theaters, arcades, and street performers.
Gilded Archway: A historic landmark leading to the city′ s oldest district, adorned with intricate carvings.
Primrose Boulevard: A tree-lined street with boutique stores, local bakeries, and street performers.
Temple Square: A historic site featuring a grand temple surrounded by artisan shops and open courtyards.
Lunar Pier: A picturesque wooden pier with food stalls, fishing spots, and a small amusement park.
Jade Fountain: A tranquil plaza centered around a beautiful fountain made of green stone.
Each pair of points has a different travel distance between them, based on city traffic patterns and street layouts. Here is the distance table showing the approximate distance (in kilometers) between each pair of locations:
Distances from Warehouse to each delivery point: Warehouse to Zenith Arena is 9 km, Warehouse to Pennywhistle Arcade is 8 km, Warehouse to Gilded Archway is 3 km, Warehouse to Primrose Boulevard is 5 km, Warehouse to Temple Square is 6 km, Warehouse to Lunar Pier is 3 km, Warehouse to Jade Fountain is 10 km.
Distances from Delivery Zenith Arena to each delivery point: Zenith Arena to Pennywhistle Arcade is 10 km, Zenith Arena to Gilded Archway is 1 km, Zenith Arena to Primrose Boulevard is 6 km, Zenith Arena to Temple Square is 6 km, Zenith Arena to Lunar Pier is 8 km, Zenith Arena to Jade Fountain is 4 km.
Distances from Delivery Pennywhistle Arcade to each delivery point: Pennywhistle Arcade to Gilded Archway is 8 km, Pennywhistle Arcade to Primrose Boulevard is 6 km, Pennywhistle Arcade to Temple Square is 5 km, Pennywhistle Arcade to Lunar Pier is 9 km, Pennywhistle Arcade to Jade Fountain is 8 km.
Distances from Delivery Gilded Archway to each delivery point: Gilded Archway to Primrose Boulevard is 3 km, Gilded Archway to Temple Square is 3 km, Gilded Archway to Lunar Pier is 8 km, Gilded Archway to Jade Fountain is 1 km.
Distances from Delivery Primrose Boulevard to each delivery point: Primrose Boulevard to Temple Square is 3 km, Primrose Boulevard to Lunar Pier is 10 km, Primrose Boulevard to Jade Fountain is 7 km.\nDistances from Delivery Temple Square to each delivery point: Temple Square to Lunar Pier is 10 km, Temple Square to Jade Fountain is 9 km.
Distances from Delivery Lunar Pier to each delivery point: Lunar Pier to Jade Fountain is 10 km.
Based on this distance table, we need to determine the optimal delivery route that allows the driver to start from the warehouse, visit each delivery point exactly once, and return to warehouse with the shortest possible total distance.

Figure 14: Details of TSP

I am designing a public Wi-Fi network for my city, with the goal of providing free high-speed internet access across various public areas. The network will cover 4 major locations in the city: Maplewood Conservatory, Moonlit Promenade, Shadowbridge Arcade and Pennywhistle Arcade.
Each of these locations will have a Wi-Fi base station, but the stations are located at varying distances from one another, and some may have overlapping coverage areas. The main issue I face is how to allocate frequencies to these base stations in a way that minimizes interference. I know that if two adjacent stations use the same frequency, their signals will interfere with each other, which will affect the network′ s stability and speed.
The interference relationships between the base stations are as follows:
The Maplewood Conservatory has overlapping signal areas with Pennywhistle Arcade.
The Moonlit Promenade has overlapping signal areas with Pennywhistle Arcade.
The Shadowbridge Arcade has overlapping signal areas with Pennywhistle Arcade.
I need to assign frequencies to the stations in such a way that no two adjacent stations use the same frequency, ensuring minimal interference. The ideal solution is to minimize the number of frequencies needed, as this would lower both the infrastructure costs and the ongoing maintenance expenses.
Can you help me come up with a solution for frequency allocation to ensure stable and reliable network performance across all locations?

Figure 15: Details of Coloring Problem

Our company has 7 computers connected by several communication links. These computers are named: Server Bluewave, Server Skyhawk, Server Glacierpeak, Server Stealthwind, Server Oceanview, Server Ghostwind and Server Stormbreaker.
To ensure network security, we need to install monitoring devices (such as firewalls or intrusion detection systems) on some of these computers so that all communication links are monitored. Assume that the connections between the computers (i.e., the communication links) are bidirectional. This means that information can flow in both directions across any link. Our goal is to deploy monitoring devices in a way that ensures all communication links are covered by at least one monitoring device.
Problem: How can we select the minimum number of computers to deploy monitoring devices, such that every communication link is monitored by at least one device?
Communication links as follows: :
Server Bluewave is connected with Server Skyhawk, Server Glacierpeak, Server Stealthwind, Server Oceanview. \nServer Skyhawk is connected with Server Glacierpeak, Server Ghostwind. Server Stealthwind is connected with Server Oceanview, Server Ghostwind, Server Stormbreaker.

Figure 16: Details of Vertex Cover Problem