

SCALABLE AND ACCURATE GRAPH REASONING WITH LLM-BASED MULTI-AGENTS

Anonymous authors

Paper under double-blind review

ABSTRACT

Recent research has explored the use of Large Language Models (LLMs) for tackling complex graph reasoning tasks. However, due to the intricacies of graph structures and the inherent limitations of LLMs in handling long text, current approaches often fail to deliver satisfactory accuracy, even on small-scale graphs and simple tasks. To address these challenges, we introduce GraphAgent-Reasoner, a fine-tuning-free framework that utilizes a multi-agent collaboration strategy for explicit and precise graph reasoning. Inspired by distributed graph computation theory, our framework decomposes graph problems into smaller, node-centric tasks that are distributed among multiple agents. The agents collaborate to solve the overall problem, significantly reducing the amount of information and complexity handled by a single LLM, thus enhancing the accuracy of graph reasoning. By simply increasing the number of agents, GraphAgent-Reasoner can efficiently scale to accommodate larger graphs with over 1,000 nodes. Evaluated on the GraphInstruct dataset, our framework demonstrates near-perfect accuracy on polynomial-time graph reasoning tasks, significantly outperforming the best available models, both closed-source and fine-tuned open-source variants. Our framework also demonstrates the capability to handle real-world graph reasoning applications such as webpage importance analysis.

1 INTRODUCTION

Graphs, as a crucial data structure for modeling complex real-world relationships, are ubiquitous across various scenarios, *e.g.* citation networks, recommendation networks. Many important applications like drug discovery (Stokes et al., 2020), traffic forecasting (Jiang & Luo, 2022), and financial detection (Motie & Raahemi, 2024), require reasoning over graphs to be realized. Noticing the powerful general knowledge and language processing capabilities of Large Language Models (LLMs) (Brown et al., 2020), a significant amount of works have focused on using LLMs to perform various reasoning tasks, such as mathematical formula derivation (Meadows et al., 2023), commonsense reasoning (Madaan et al., 2022), and multi-hop question answering (Creswell et al., 2023). However, most of them primarily involve shallow or sequential reasoning. To bring the LLM reasoning closer to human thinking, it is necessary for LLMs to master deeper and more complex reasoning, such as graph reasoning.

Despite significant efforts by researchers to enable LLMs to memorize, comprehend, and perform basic reasoning on graph structures, several issues still persist: **1) The scale of graphs that can be handled is limited.** Describing graph structures in natural language inevitably leads to excessively long inputs. Due to context length limitations and the shortcomings of LLMs in handling lengthy text (Liu et al., 2023), previous works (Chai et al., 2023; Fatemi et al., 2024; Perozzi et al., 2024) could only handle graphs of very limited size (*e.g.* fewer than 20 nodes and 100 edges). **2) The performance on graph reasoning tasks is relatively poor.** Unlike text, which can tolerate some degree of semantic deviation, reasoning and computation on graphs must be highly precise. However, current works demonstrate poor accuracy (average 20~60%) in various graph reasoning tasks like connectivity and shortest path. **3) Lacking explicit reasoning paths.** Taking the shortest path as an example, the responses of existing models resemble a heuristic search approach to finding the shortest path on a graph, rather than strictly executing an algorithm. This makes it difficult to determine whether LLMs are genuinely deriving the answer through correct reasoning or merely making educated guesses. Although GraphWiz (Chen et al., 2024a) attempts to generate explicit reasoning

paths through fine-tuning, it often fails due to the presence of incomplete or wrong reasoning paths in its training data. Furthermore, GraphWiz exhibits overfitting, where it tends to treat new or unrelated questions as one of the fine-tuned problems, which will be detailed in Section 5.3.

Motivation. The ultimate goal of graph reasoning is to enable LLMs to leverage graph-related knowledge or algorithms to solve real-world graph problems. However, with the development of information science and hardware storage, the scale of graphs and information per node become too large for a single LLM to handle. To address this, a natural idea is to use distributed approaches, where a large graph is stored across multiple LLMs separately and compute collaboratively. Therefore, just as graph algorithms have generally evolved from non-distributed to distributed forms (Meng et al., 2024b)), we hope that LLMs can also learn the concept of distributed processing, thereby harnessing the power of swarm intelligence to solve graph problems in real-world scenarios.

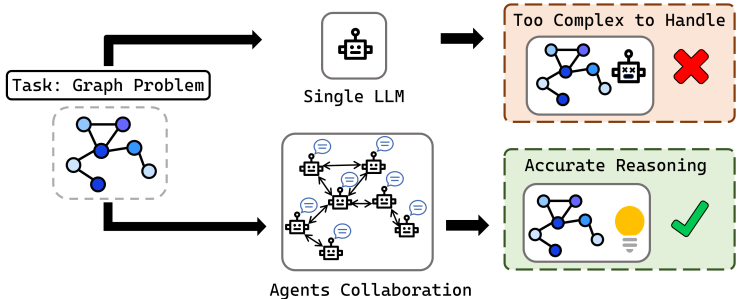


Figure 1: The current situation of LLMs in solving graph problems. Previous methods using a single LLM often failed due to the complex graph structures. In contrast, our approach leverages agents collaboration to effectively address graph problems.

Our Contribution. To address the above limitations, in this paper, we propose the GraphAgent-Reasoner(GAR) framework, which leverages the power of swarm intelligence to solve graph reasoning problems, as shown in Figure 1. We follow a node-centric approach, assigning an agent to each node, allowing it to focus on processing its own information and communicate with neighbors. Thus, we can easily scale up the size of graphs that can be processed by simply increasing the number of agents. At the same time, under the direction of a Master LLM, graph problems are decomposed into smaller, node-centric tasks, which are assigned to agents for collaborative resolution. This approach significantly reduces the scale and complexity of information each agent needs to process, thereby greatly improving the overall accuracy. Furthermore, since agents must clearly transmit the processed information to neighboring agents, the reasoning process becomes transparent, demonstrating the framework solves graph reasoning problems through clear and correct reasoning, rather than lucky guessing. In summary, our contributions are as follows:

- We propose GraphAgent-Reasoner, the first LLM-based multi-agents framework for graph reasoning, which requires no fine-tuning and can utilize any LLM as the underlying reasoning model. Our framework achieves near-perfect accuracy on various polynomial-time tasks, significantly surpassing the performance of existing methods.
- Our framework expands the scale of graph reasoning tasks handled by LLMs from 100 nodes to 1,000 nodes, demonstrating exceptional scalability. Furthermore, as the graph size increases, our framework does not exhibit the significant performance degradation seen in other methods and maintains robust accuracy.
- We explore the performance of our framework in real-world applications like webpage importance analysis, showcasing its potential for addressing complex graph reasoning problems in real-life situations.

2 PRELIMINARIES AND RELATED WORKS

Preliminaries. In general scenarios, when discussing LLMs solving graph reasoning problems, the input is a $(\mathcal{G}, \mathcal{Q})$ pair. \mathcal{G} is a graph represented as $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \{s_i\}, \{t_i\})$, where \mathcal{V} is the node set and \mathcal{E} , the edge set. For each node $v_i \in \mathcal{V}$, a sequential text node feature s_i is associated; similarly, for

each edge $e_i \in \mathcal{E}$, a sequential text edge feature t_i is assigned. The graph \mathcal{G} is described in natural language, typically using edge or adjacency list representation. \mathcal{Q} is a task-specific instruction or problem description. LLMs will process the $(\mathcal{G}, \mathcal{Q})$ pair and return an answer string A .

Large Language Models for Graph Reasoning. To further enhance the reasoning capabilities of LLMs, many works have attempted to improve the performance of LLMs in graph reasoning. Wang et al. (2023) first introduces the NLGraph Benchmark to evaluate the performance of LLMs on various graph reasoning tasks. Fatemi et al. (2024) explores the impact of different graph encoding methods and graph structure types on the performance of LLMs in graph reasoning tasks. Additionally, it introduces another benchmark called GraphQA. Considering the lengthy nature of describing graph structures in text, Chai et al. (2023) and Perozzi et al. (2024) respectively use Transformers and GNNs to encode graph structures and attempt to align them with LLMs. Inspired by how humans understand structural information through the visual modality, Wei et al. (2024) generates corresponding visual images based on graph structures and provides them to visual LLMs for graph reasoning. Chen et al. (2024a) conducted Supervised Fine-Tuning and Directly Preferred Optimization on LLMs, enhancing the performance of LLMs and encouraging them to output explicit reasoning paths.

Large Language Model based Multi-Agents. Recent advancements in LLMs have spurred interest in their application within multi-agent systems. LLM-based multi-agent frameworks leverage the natural language understanding and reasoning capabilities of LLMs to enable agents to collaborate, communicate, and solve complex tasks in a distributed manner. Existing multi-agents works for problem solving primarily focuses on applications such as Software Development (Dong et al., 2023; Hong et al., 2024; Qian et al., 2024), Embodied Agents (Zhang et al., 2024; Mandi et al., 2024; Chen et al., 2024b) and Science Debate (Xiong et al., 2023; Chan et al., 2024). However, using LLM-based multi-agents to handle graph data has been less explored, especially in the areas of graph reasoning and graph computation tasks. This may be due to the hallucination issue inherent in LLMs (Huang et al., 2023), where their responses are factually incorrect. This problem becomes more complex in a multi-agent setting, as the hallucinations of a single agent may propagate to other nodes by communication (Guo et al., 2024). This requires the performance of individual agents be sufficiently stable to ensure the correct operation of the entire multi-agent system.

3 LIMITATIONS OF SINGLE LLM IN GRAPH REASONING

Although LLMs exhibit strong language processing and logical reasoning capabilities, problems with the Transformer architecture and Attention mechanism (Vaswani et al., 2017) still limit the scale and accuracy when they process graph problems. There are two primary limitations:

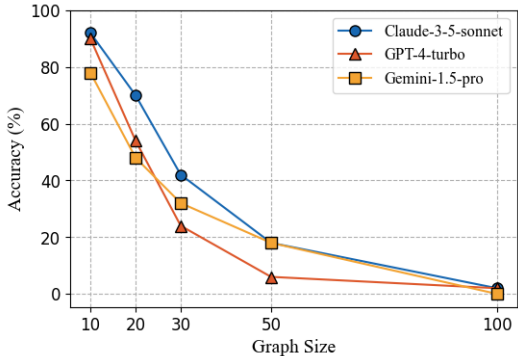


Figure 2: The performance of a single LLM in memorizing first-order neighboring nodes. As the number of nodes increases, all models exhibit significant memory errors.

The graph structure is too complex to memorize and understand for a single LLM. Using adjacency or edge lists to describe graph structures in natural language is the most intuitive and direct method, facilitating the processing of graph data by LLMs through text. However, this approach inevitably leads to a lengthy context, as the number of edges can grow quadratically with the number of nodes. As the graph scales up and becomes denser, the graph structure becomes highly complex, requiring a large amount of tokens to describe the edge relationships. When the text becomes too

lengthy, it becomes difficult for LLMs to properly allocate attention, and they may even struggle with simple tasks such as key-value pair matching Liu et al. (2023). This presents significant challenges for LLMs in identifying key information for graph reasoning tasks from the lengthy context. Figure 2 shows the performance of a single LLM in memorizing one-hop neighbor nodes. We observe that as the number of nodes in the graph increases, various LLMs exhibit a significant decline in accuracy. If a single LLM cannot even correctly recall basic graph structural information like node neighbors, it becomes difficult to proceed with more complex graph reasoning or computation.

Furthermore, the graph structure is described in a sequential manner. LLMs have to identify implicit graph structures from sequential text. Since the processing of LLMs is a black-box operation, it is difficult to assert that they truly construct graph structures implicitly and thereby understand them. Huang et al. (2024) conducted extensive experiments to explore whether LLMs treat the input prompts as graphs or merely as paragraphs with keywords on TAGs. The results show that the performance of LLMs in handling TAGs primarily stems from the context rather than the graph structure. LLMs tend to process the graph description as linearized paragraphs rather than graphs.

A single LLM struggles to solve reasoning problems in real-world scenarios. Researchers train LLMs on graph reasoning tasks to empower them to utilize learned graph-related knowledge or algorithms to tackle real-world graph problems. However, in practical scenarios, the amount of information associated with each node can be enormous. Take citation networks as an example: a single node represents a paper, and its node information includes the title, abstract, and references, which could amount to several thousand tokens. In addition to the complexity of graph structures, the need to handle a large amount of node information further exacerbates the burden on a single LLM and highlights its shortcomings in processing long contexts. Moreover, using a single LLM to handle the entire network is inefficient, as it cannot coherently process the entire network’s problems. Typically, it is necessary to manually compress or summarize the information for each node and then feed local subgraphs to the LLM for processing (Guo et al., 2023; Chen et al., 2023).

Furthermore, many current works (Chen et al., 2024a; Perozzi et al., 2024) require training GNNs or fine-tuning LLMs on individual or multiple graph reasoning tasks. However, when transferring to other graph tasks, a certain degree of performance degradation occurs, and retraining or fine-tuning for new graph tasks consumes a significant amount of time and resources. Whether LLMs can apply the graph knowledge and algorithms learned during the training process to actual graph reasoning also remains an open question. We explored this question in 5.3 and observed significant overfitting in LLMs fine-tuned on specific graph reasoning tasks. Therefore, the ideal solution would be to leverage the powerful general knowledge acquired during the pre-training phase of LLMs through an appropriate approach, enabling them to handle graph reasoning tasks as naturally as they do with natural language problems.

4 GRAPHAGENT-REASONER

To solve the limitations above, we propose a novel framework based on multi-agent collaboration called GraphAgent-Reasoner as shown in Figure 3, aiming to solve graph reasoning problems explicitly and correctly. The interface of the framework is a Master LLM, which is responsible for processing the textual input of graph problems, constructing the agent network, directing them to collaboratively solve the problem, and finally aggregating the states of all agents to derive the solution. Its implementation is based on the React Agent proposed by Yao et al. (2023), which is capable of reasoning based on the environment and executing corresponding actions, as detailed later. The pipeline of GAR consists of four steps: Graph Construction, Algorithm Establishing, Distributed Execution and Master Summarization.

Graph Construction. Given an input pair $(\mathcal{G}, \mathcal{Q})$, the Master LLM first extracts the node and edge information from the textual description of graph \mathcal{G} . It then constructs an agent for each node and initializes the node’s state and neighbor information, forming an interconnected network of agents. Each agent independently maintains its state and neighbor data, communicates with adjacent agents based on instructions from the Master LLM, and updates its state in each round.

Algorithm Establishing. To accommodate diverse graph tasks and fully exploit the knowledge embedded in LLMs during pre-training, we propose a unified solution approach framed within a

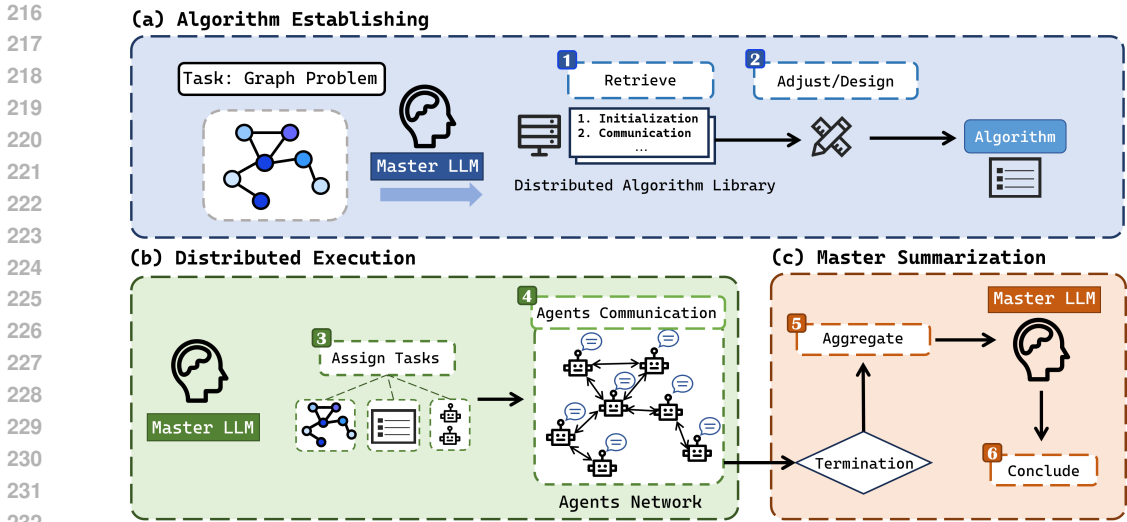


Figure 3: The framework of GraphAgent-Reasoner. Given a graph problem, the Master LLM will first construct agents network according to graph structures. It then sequentially performs Algorithm Establishing, Distributed Execution and Master Summarization, as detailed in this section.

distributed paradigm as shown in Algorithm 1. This approach requires the Master LLM to specify six core components for each problem: State, Message, Initialization, Send, Update, and Termination.

- **State:** The local information maintained by each node, representing its current state. This can include attributes like node features, labels, or any other task-specific data. The states evolve as nodes receive messages and update their information.
- **Message:** The data transmitted between nodes during the communication phase. Messages typically contain information that neighboring nodes need to perform updates, such as feature values, distances, or other task-relevant information.
- **Initialization:** At the start of the execution, each node initializes its state with predefined values, which may be based on node IDs, input features or task-specific requirements. This step ensures that the graph is ready to begin the communication process.
- **Send:** After initialization, each node generates messages based on its current state and sends them to its neighboring nodes. This step is repeated in each iteration, allowing nodes to continuously exchange information with their neighbors.
- **Update:** Upon receiving messages from its neighbors, each node updates its state by aggregating the incoming messages and combining them with its current state. This iterative process enables nodes to refine their information over time.
- **Termination:** The algorithm halts when a predefined stopping condition is met, such as reaching a fixed number of iterations, achieving convergence, or satisfying a task-specific criterion. Once the termination condition is reached, each node will send its final state to the Master LLM, and the execution terminates.

Since LLMs lack prior knowledge of this distributed paradigm, to facilitate the Master LLM’s understanding and application of the framework, we develop a distributed algorithm library that adheres to this distributed paradigm, from which the Master LLM can query relevant algorithm templates to generate distributed solutions within this paradigm. Specifically, we selected classic distributed graph algorithms and documented their implementations under this distributed paradigm. Some examples are presented in Appendix A.1. Drawing on prior work (Zheng et al., 2024; Meng et al., 2024a), we endeavor to write detailed reasoning steps of each part in the algorithm to encourage the agent to think step by step as much as possible, which plays an important role in enhancing the success rate of individual agents.

When receiving a problem input, the Master LLM first retrieves the k algorithms most relevant to the problem description from the distributed algorithm library. If there are algorithms suitable for

Algorithm 1: Distributed Paradigm

270
271
272 1 **Input:** Agent Nodes \mathcal{A} , each agent $a \in \mathcal{A}$ maintains a state S_a , the maximum iterations I_{max}
273 given by the Master LLM.
274 2 **Output:** Final state S_a for each agent $a \in \mathcal{A}$
275 /* Initialization */
276 3 Each agent $a \in \mathcal{A}$ initializes its state S_a based on **Initialization** rules.
277 4 Each agent a sends an initial message $M_{a \rightarrow v}$ to each of its neighbors $v \in \text{Neighbors}(a)$ based on
278 its current state S_a and **Send** rules.
279 /* Communication */
280 5 **while** Iteration $i < I_{max}$ and **Termination** not met **do**
281 6 a. /* Receive */
282 7 Each agent a receives messages $M_{u \rightarrow a}$ from all neighboring agents u .
283 8 b. /* Update */
284 9 Each agent a updates its state S_a based on the received messages M and its own current state
285 S_a according to **Update** rules.
286 10 c. /* Send */
287 11 Each agent a sends updated messages $M_{a \rightarrow v}$ to each of its neighbors v based on the updated
288 state S_a according to **Send** rules.
289 12 **Return:** the final state S_a for all agents $a \in \mathcal{A}$

290
291 handling the problem, the Master LLM will adjust the algorithm according to the problem description,
292 such as changing the initialization and termination conditions (e.g., the source node in the shortest
293 path problem). If there are no appropriate algorithms, the Master LLM will design a distributed
294 algorithm following the distributed paradigm based on the examples of the retrieved algorithms. For
295 some generated examples, see Appendix A.2.

296 **Distributed Execution.** After the distributed algorithm is designed, the Master LLM will relay the
297 approach to each agent node for execution according to the process outlined in Algorithm 1. Each
298 agent will first initialize its state based on node information and algorithm rules and then send an
299 initial message to neighboring agents. Subsequently, each agent will iteratively execute the operations
300 of receiving messages, updating its state, and sending messages according to the algorithm rules,
301 synchronizing progress after each communication round. Communication will continue until the
302 maximum number of iterations is reached or the termination condition is met.

303 **Master Summarization.** Finally, the final state of all agent nodes will be aggregated to the Master
304 LLM, which will summarize the results conclude based on the problem and return the final answer in
305 natural language form.
306

307 5 EXPERIMENTS

308
309 In this section, we summarize the key experiments conducted with GAR. We begin by highlighting
310 some of the most exciting results from our analysis here:
311

- 312 • **R1:** GAR achieves **near-perfect accuracy** on polynomial-time graph reasoning problems, signifi-
313 cantly surpassing existing closed-source models and open-source models fine-tuned on extensive
314 data.
- 315 • **R2:** GAR maintains high accuracy on larger-scale graphs (**up to 1000 nodes**), demonstrating
316 superior scalability. In contrast, as the number of nodes increases, other models exhibit a significant
317 decline in performance or become incapable of handling the problem at all due to the context length
318 limitation.
- 319 • **R3:** GAR showcases a robust understanding and application of graph algorithms in real-world graph
320 reasoning scenarios, highlighting its potential for addressing complex graph problems encountered
321 in daily life. In contrast, other open-source models that have undergone extensive fine-tuning on
322 graph reasoning datasets fail to apply the learned graph reasoning knowledge when confronted
323 with rephrased real-world graph problems.

Datasets. We conduct our experiments on the graph reasoning tasks proposed in GraphInstruct (Chen et al., 2024a). This dataset contains nine graph reasoning problems with different time complexity, ranging from linear and polynomial complexity to NP-complete.

- **Linear.** Cycle Detection (Detect if a given graph \mathcal{G} contains any cycles), Connectivity (Assess if two nodes u and v in a given graph \mathcal{G} are connected via a path), Bipartite Graph Check (Judge if a given graph \mathcal{G} is bipartite), and Topological Sort (Find a topological ordering of vertices in a directed acyclic graph \mathcal{G}).
- **Polynomial.** Shortest Path (Compute the shortest path between two specific nodes u and v in a given graph \mathcal{G}), Maximum Triangle Sum (Find the maximum sum of weights for any connected triplet of vertices in a given graph \mathcal{G}), and Maximum Flow (Calculate the maximum flow from a source node s to a sink node t in a directed graph \mathcal{G}).

Due to the complexity of NP-complete problems, there are currently no mature exact distributed algorithms available for their solution. Consequently, the Master LLM is unable to design correct and effective distributed algorithms based on the knowledge acquired during pre-training. Therefore, in our experiments, we only consider linear and polynomial-time problems. Detailed information of the dataset and partial test results for NP-complete problems will be presented in Appendix B.

Setting. The underlying reasoning LLM of Agent Node used in our framework is ChatGPT-4o-mini-2024-07-18, and the base model of Master LLM is ChatGPT-4-turbo(OpenAI, 2023). The temperature is consistently set to 0. Our framework is built upon AgentScope (Gao et al. (2024)), an innovative platform to easily build reliable, high-performance multi-agent applications.

5.1 EXPERIMENT 1: PERFORMANCE ON GRAPHINSTRUCT

In this experiment, we evaluate the performance of GAR on polynomial-time tasks of the GraphInstruct dataset. The results are shown in Table 1. We see GAR exhibits near-perfect results on these tasks, significantly outperforming other models. Especially on shortest and triangle tasks with high time complexity, GAR substantially improves the performance of LLMs. Problems that a single LLM struggles to solve have been effectively resolved through collaboration by agents after being decomposed into smaller, node-centric tasks.

As the number of nodes increases, the graph structures become more complex, making the solution of graph problems increasingly difficult. To investigate how the performance of models varies with increasing problem complexity, we conduct experiments on cycle detection and shortest path problems, gradually increasing the number of nodes from 5 to 100. The results are presented in Figure 4.

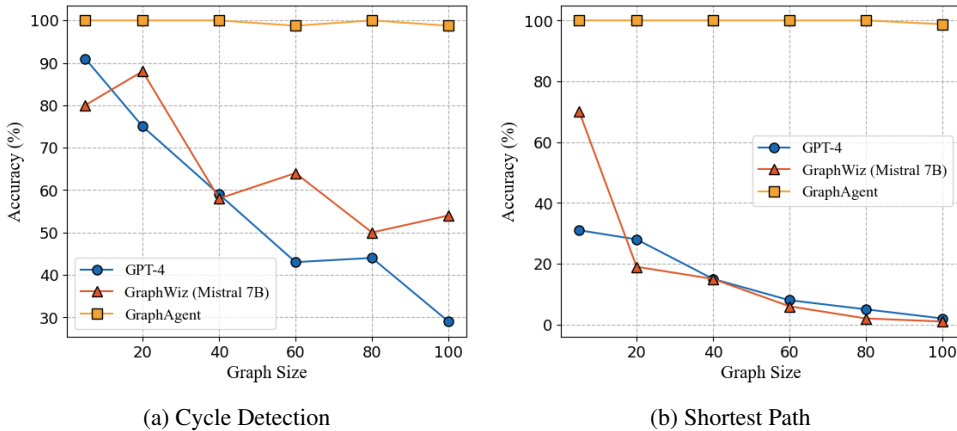


Figure 4: Performance of GraphAgent-Reasoner, GPT4(2 shot) and GraphWiz(Mistral 7B) on cycle detection and shortest path problems with different graph sizes.

We see with the number of nodes increasing, both ChatGPT-4 and Graphwiz exhibit a significant decline in performance. However, the accuracy of GAR remains stable, almost unaffected by the graph size, demonstrating robust scalability. Although the scale of the graph is increasing, the

information processed by each agent has not significantly increased. Each agent still only handles its own information and communicates with neighboring agents. We observe that GAR occasionally makes errors in specific cases, likely due to the increasing communication rounds as the number of nodes and edges grows. Even when handling simple node-centric tasks, a single agent still has the potential to make mistakes. Therefore, as the number of agents and communication rounds increases, the overall likelihood of errors also rises. This can be improved by enhancing the capability of individual agents (such as using stronger LLMs as the underlying reasoning model) or by more finely designed prompts.

Table 1: Performance of GraphAgent-Reasoner and other models on polynomial-time tasks of GraphInstruct test set. Each task contains 400 test cases, with a maximum of 100 nodes. The first best result for each task is highlighted in bold, and the second best result is highlighted underlined.

Models	Linear				Polynomial		Average
	cycle	connect	bipartite	topology	shortest	triangle	
Closed-source Models							
GPT-4 (zero-shot)	38.75	17.00	65.25	5.00	9.25	5.75	23.50
GhatGPT (2-shot)	51.25	43.75	70.75	4.50	3.50	17.25	31.83
GPT-4 (2-shot)	52.50	62.75	74.25	25.25	18.25	31.00	44.00
Fine-tuned Open-source Models							
Naive SFT (LLaMA 2-7B)	73.75	83.50	41.25	4.00	9.50	30.00	40.17
Naive SFT (Mistral-7B)	73.75	83.50	78.50	1.00	23.00	47.00	51.13
GraphWiz (LLaMA 2-7B)	91.50	87.00	74.00	18.00	28.00	38.25	56.13
GraphWiz (Mistral-7B)	92.00	89.50	72.00	19.00	<u>31.25</u>	38.75	57.08
GraphWiz-DPO (LLaMA 2-7B)	89.00	82.50	84.75	46.75	24.00	<u>52.75</u>	<u>63.29</u>
GraphWiz-DPO (Mistral-7B)	85.50	79.50	<u>85.50</u>	<u>85.25</u>	12.50	29.00	62.88
GraphAgent-Reasoner	99.50	100.00	100.00	96.50	99.75	93.25	98.00

5.2 EXPERIMENT 2: PERFORMANCE ON LARGE-SCALE GRAPHS

In this experiment, we evaluate the performance of current LLMs on large-scale graphs. The largest graph size handled by existing graph reasoning work is 100 nodes (Chen et al., 2024a), which is still far from sufficient for real-world graph reasoning scenarios. To evaluate the reasoning performance of existing models on larger graphs, we conduct shortest path experiments on graphs with 100, 200, 500, and 1000 nodes. Due to the excessively long input text (reaching 16,000 tokens for 1000 nodes) and the money cost, we only create 20 test samples for each graph size. The results are shown in Table 2.

Table 2: Performance on large-scale graphs dealing with shortest path problems. x/20 indicates that out of 20 test samples, x samples are correct. NA signifies that testing could not be conducted due to the fact that the context length limit is exceeded.

Graph Size	100	200	500	1000
Graphwiz (LLaMA 2-7B)	0/20	0/20	NA	NA
Graphwiz (LLaMA 2-7B-DPO)	0/20	0/20	NA	NA
Chatgpt-3.5-turbo-16k	0/20	0/20	0/20	0/20
Chatgpt-4-32k	0/20	1/20	0/20	0/20
GraphAgent-Reasoner	20/20	20/20	20/20	18/20

We see the two GraphWiz models fine-tuned on the LLaMA2-7B (Touvron et al., 2023) base model are unable to handle graphs with 500 or more nodes due to the context length limitation (the context length limit for Llama2 is 4096 tokens). Although ChatGPT-3.5-turbo-16k and ChatGPT-4-32k can manage longer contexts, they output wrong answers in almost all test samples, with only ChatGPT-4-32k being correct in one 200 nodes test sample. In contrast, GAR maintains a high accuracy in large-scale graph, only failed in two 1000-node test samples, further demonstrating its robust scalability.

5.3 EXPERIMENT 3: CASE STUDY

In this experiment, we explore the application of two graph reasoning models, Graphwiz and GAR, in real-world graph reasoning scenarios. We present a case study of webpage importance analysis in Figure 5.

```

(G,Q): For a webpage network graph, each node represents a webpage, and a directed edge (i->j) means
that webpage i has a link to webpage j. Now given a webpage network graph G, you need to analyze the
importance of each webpage and give the top three webpage ids. G: The nodes are numbered from 1 to 20,
and the edges are: (1->5) (1->9) (2->6) (2->17) (3->8) (3->12) (4->2) (4->19) (5->14) (6->1) (7->10)
(7->3) (8->13) (9->4) (10->18) (11->7) (12->15) (13->20) (14->11) (15->2) (16->8) (16->5) (17->16)
(18->14) (19->6) (20->1) (5->7) (10->12) (15->19) (6->11) (8->2) (12->4) (14->9) (3->20) (7->15) (13-
>18) (9->17) (2->13) (11->16) (4->10).
-----
GraphWiz(LLaMA 2-7B): \n\n### The output is: ### The output is: The graph has 21 nodes and 21 edges, so
the number of edges is equal to the number of nodes. Therefore, the graph is connected and there is a path
between any two nodes. The graph is not bipartite, so there is no way to divide the nodes into two groups
such that no two nodes in the same group are directly connected. X
GraphWiz(LLaMA 2-7B-DPO): The graph has 21 nodes and 19 edges, and we can use a similar strategy as
the previous examples to find the top three nodes ... Finally, we can select node 6, which has the lowest
degree (one) among the remaining nodes. \nThis way, we obtain a valid topological sorting of the graph that
respects the direction of the edges. \n### The top three nodes are 1, 2, and 6. X
GraphAgent-Reasoner: To analyze the importance of each webpage in the given directed graph, we can use a
distributed algorithm based on the concept of PageRank. Below is the structured algorithm design:
### State:
1. `rank`: Represents the importance of the webpage, initialized to 1.0.
### Initialization
Step 1: Set `rank` to 1.0 for each node (webpage).
### Message:
1. `contribution`: Current rank of the node divided by its number of outgoing links.
Distributed_Algorithm_Designer: ### Send
Step 1: Calculate `contribution` as `rank / out_degree`, where `out_degree` is the total number of outgoing
links.
Step 2: Send `contribution` to each connected neighbor (Node A and Node B).
### Update
Step 1: Sum all received `contribution` values from messages.
Step 2: Update `rank` using the formula:
- `new_rank = damping_factor * (sum_of_received_contributions) + (1 - damping_factor) * initial_rank`.
Step 3: Set the new `rank` as the current state.

<Distributed_Execution>

[RESULT]: Node: 1 State: 1. rank: 1.1194682167142727
Node: 2 State: 1. rank: 1.1624009025870747
...
Node: 19 State: 1. rank: 0.662517899831644
Node: 20 State: 1. rank: 0.5902983815488694

Master: {'thought': 'The PageRank values for each node have been calculated. The top three important
webpages are those with the highest ranks.', 'speak': 'The top three important webpages are 16, 14, and 5
based on their PageRank values.'}

```

Figure 5: The importance analysis in webpage network. While the GraphWiz fails due to incorrect graph assessments, GAR correctly uses the PageRank algorithm to identify nodes 16, 14, and 5 as the most important.

Although GraphWiz performed well on fine-tuned tasks, it exhibits severe overfitting when faced with real-world graph problems, failing to apply the graph reasoning knowledge learned during the fine-tuning phase. Since GraphWiz uses a consistent graph node description, the sentence "The nodes are numbered from 0 to ..." appears across all datasets during the mixed-task instruction tuning. When the actual problem has nodes numbered from 1 to 20, it still assumes the existence of node 0. As a result, both GraphWiz models first output that the graph has 21 nodes and an incorrect number of edges. Furthermore, neither of the two GraphWiz models recognizes that this is a problem associated with web page importance ranking. Instead, they approach it as the bipartite graph check or topological sort problems they had been fine-tuned on. Additionally, neither model generates an explicit and correct reasoning path. These observations indicate that there is still a significant gap between excelling in classic graph reasoning tasks and effectively solving real-world graph reasoning problems. In contrast, GAR correctly identifies that the problem should be solved using knowledge related to PageRank (Yang et al., 2024) and designs an algorithm that adhered to the distributed

paradigm (Note: the distributed algorithm library does not contain a PageRank algorithm template). GAR then assigns the algorithm to agent nodes for execution, ultimately obtaining the PageRank value for each node and arriving at the correct conclusion. Through the distributed paradigm, GAR effectively bridges the powerful knowledge learned by LLMs with the solving of real-world graph reasoning problems, which enables it to flexibly handle practical issues in a distributed manner. This case study demonstrates the feasibility of using GAR to solve real-world graph reasoning problems, indicating its substantial practical applicability and offering researchers and practitioners a powerful framework to address such tasks.

6 CONCLUSION

We first summarize three key issues faced by existing LLMs in graph reasoning tasks: limited graph scale, poor performance, and the lack of explicit reasoning paths. We then reflect on the limitations of a single LLM in addressing graph reasoning problems, such as the graph structures being too complex to memorize and understand and the overwhelming information in real-world graph reasoning scenarios. To address these challenges, we propose GraphAgent-Reasoner, a framework based on multi-agent collaboration to solve graph reasoning problems. This framework demonstrates superior accuracy and scalability, significantly surpassing existing closed-source and fine-tuned open-source models. Our experiments show its robust scalability, maintaining high accuracy on large graphs (up to 1,000 nodes). Our case study on webpage importance analysis further illustrates its capability to handle real-world graph reasoning problems. Future work will focus on designing more accurate and scalable LLM-based multi-agent graph reasoning frameworks, aiming to apply them to larger and more complex real-world reasoning scenarios.

REFERENCES

- Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. In Hugo Larochelle, Marc Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin (eds.), *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020. URL <https://proceedings.neurips.cc/paper/2020/hash/1457c0d6bfc4967418bfb8ac142f64a-Abstract.html>.
- Ziwei Chai, Tianjie Zhang, Liang Wu, Kaiqiao Han, Xiaohai Hu, Xuanwen Huang, and Yang Yang. Graphllm: Boosting graph reasoning ability of large language model. *CoRR*, abs/2310.05845, 2023. URL <https://doi.org/10.48550/arXiv.2310.05845>.
- Chi-Min Chan, Weize Chen, Yusheng Su, Jianxuan Yu, Wei Xue, Shanghang Zhang, Jie Fu, and Zhiyuan Liu. Chateval: Towards better llm-based evaluators through multi-agent debate. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024. URL <https://openreview.net/forum?id=FQepisCUWu>.
- Nuo Chen, Yuhan Li, Jianheng Tang, and Jia Li. Graphwiz: An instruction-following language model for graph computational problems. In Ricardo Baeza-Yates and Francesco Bonchi (eds.), *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, KDD 2024, Barcelona, Spain, August 25-29, 2024*, pp. 353–364. ACM, 2024a. URL <https://doi.org/10.1145/3637528.3672010>.
- Yongchao Chen, Jacob Arkin, Yang Zhang, Nicholas Roy, and Chuchu Fan. Scalable multi-robot collaboration with large language models: Centralized or decentralized systems? In *IEEE International Conference on Robotics and Automation, ICRA 2024, Yokohama, Japan, May 13-17, 2024*, pp. 4311–4317. IEEE, 2024b. URL <https://doi.org/10.1109/ICRA57147.2024.10610676>.
- Zhikai Chen, Haitao Mao, Hang Li, Wei Jin, Hongzhi Wen, Xiaochi Wei, Shuaiqiang Wang, Dawei Yin, Wenqi Fan, Hui Liu, and Jiliang Tang. Exploring the potential of large language models (llms) in learning on graphs. *SIGKDD Explor.*, 25(2):42–61, 2023. URL <https://doi.org/10.1145/3655103.3655110>.

- 540 Antonia Creswell, Murray Shanahan, and Irina Higgins. Selection-inference: Exploiting large
541 language models for interpretable logical reasoning. In *The Eleventh International Conference on*
542 *Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net, 2023.
543 URL <https://openreview.net/forum?id=3Pf3Wg6o-A4>.
- 544 Yihong Dong, Xue Jiang, Zhi Jin, and Ge Li. Self-collaboration code generation via chatgpt. *CoRR*,
545 abs/2304.07590, 2023. URL <https://doi.org/10.48550/arXiv.2304.07590>.
- 547 Bahare Fatemi, Jonathan Halcrow, and Bryan Perozzi. Talk like a graph: Encoding graphs for large
548 language models. In *The Twelfth International Conference on Learning Representations, ICLR*
549 *2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024. URL <https://openreview.net/forum?id=IuXRlCCrSi>.
- 551 Dawei Gao, Zitao Li, Weirui Kuang, Xuchen Pan, Daoyuan Chen, Zhijian Ma, Bingchen Qian,
552 Liuyi Yao, Lin Zhu, Chen Cheng, Hongzhu Shi, Yaliang Li, Bolin Ding, and Jingren Zhou.
553 Agentscope: A flexible yet robust multi-agent platform. *CoRR*, abs/2402.14034, 2024. URL
554 <https://doi.org/10.48550/arXiv.2402.14034>.
- 555 Jiayan Guo, Lun Du, and Hengyu Liu. Gpt4graph: Can large language models understand graph
556 structured data ? an empirical evaluation and benchmarking. *CoRR*, abs/2305.15066, 2023. URL
557 <https://doi.org/10.48550/arXiv.2305.15066>.
- 559 Taicheng Guo, Xiuying Chen, Yaqi Wang, Ruidi Chang, Shichao Pei, Nitesh V. Chawla, Olaf Wiest,
560 and Xiangliang Zhang. Large language model based multi-agents: A survey of progress and
561 challenges. *CoRR*, abs/2402.01680, 2024. URL [https://doi.org/10.48550/arXiv.](https://doi.org/10.48550/arXiv.2402.01680)
562 [2402.01680](https://doi.org/10.48550/arXiv.2402.01680).
- 563 Sirui Hong, Mingchen Zhuge, Jonathan Chen, Xiawu Zheng, Yuheng Cheng, Jinlin Wang, Ceyao
564 Zhang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, Chenyu Ran, Lingfeng
565 Xiao, Chenglin Wu, and Jürgen Schmidhuber. Metagpt: Meta programming for A multi-
566 agent collaborative framework. In *The Twelfth International Conference on Learning Rep-*
567 *resentations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024. URL
568 <https://openreview.net/forum?id=VtmBAGCN7o>.
- 569 Jin Huang, Xingjian Zhang, Qiaozhu Mei, and Jiaqi Ma. Can llms effectively leverage graph
570 structural information through prompts, and why? *Trans. Mach. Learn. Res.*, 2024, 2024. URL
571 <https://openreview.net/forum?id=L2jRavXRxs>.
- 572 Lei Huang, Weijiang Yu, Weitao Ma, Weihong Zhong, Zhangyin Feng, Haotian Wang, Qianglong
573 Chen, Weihua Peng, Xiaocheng Feng, Bing Qin, and Ting Liu. A survey on hallucination in large
574 language models: Principles, taxonomy, challenges, and open questions. *CoRR*, abs/2311.05232,
575 2023. URL <https://doi.org/10.48550/arXiv.2311.05232>.
- 576 Weiwei Jiang and Jiayun Luo. Graph neural network for traffic forecasting: A survey. *Expert Syst.*
577 *Appl.*, 207:117921, 2022. URL <https://doi.org/10.1016/j.eswa.2022.117921>.
- 580 Nelson F. Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni,
581 and Percy Liang. Lost in the middle: How language models use long contexts. *Transactions*
582 *of the Association for Computational Linguistics*, 12:157–173, 2023. URL [https://api.](https://api.semanticscholar.org/CorpusID:259360665)
583 [semanticscholar.org/CorpusID:259360665](https://api.semanticscholar.org/CorpusID:259360665).
- 584 Aman Madaan, Shuyan Zhou, Uri Alon, Yiming Yang, and Graham Neubig. Language models of
585 code are few-shot commonsense learners. In Yoav Goldberg, Zornitsa Kozareva, and Yue Zhang
586 (eds.), *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing,*
587 *EMNLP 2022, Abu Dhabi, United Arab Emirates, December 7-11, 2022*, pp. 1384–1403. Associa-
588 tion for Computational Linguistics, 2022. URL [https://doi.org/10.18653/v1/2022.](https://doi.org/10.18653/v1/2022.emnlp-main.90)
589 [emnlp-main.90](https://doi.org/10.18653/v1/2022.emnlp-main.90).
- 590 Zhao Mandi, Shreeya Jain, and Shuran Song. Roco: Dialectic multi-robot collaboration with large
591 language models. In *IEEE International Conference on Robotics and Automation, ICRA 2024,*
592 *Yokohama, Japan, May 13-17, 2024*, pp. 286–299. IEEE, 2024. URL [https://doi.org/10.](https://doi.org/10.1109/ICRA57147.2024.10610855)
593 [1109/ICRA57147.2024.10610855](https://doi.org/10.1109/ICRA57147.2024.10610855).

- 594 Jordan Meadows, Marco Valentino, and André Freitas. Generating mathematical derivations with
595 large language models. *CoRR*, abs/2307.09998, 2023. URL [https://doi.org/10.48550/
596 arXiv.2307.09998](https://doi.org/10.48550/arXiv.2307.09998).
- 597 Lingkai Meng, Yu Shao, Long Yuan, Longbin Lai, Peng Cheng, Xue Li, Wenyuan Yu, Wenjie Zhang,
598 Xuemin Lin, and Jingren Zhou. A survey of distributed graph algorithms on massive graphs. *CoRR*,
599 abs/2404.06037, 2024a. URL <https://doi.org/10.48550/arXiv.2404.06037>.
- 600 Lingkai Meng, Yu Shao, Long Yuan, Longbin Lai, Peng Cheng, Xue Li, Wenyuan Yu, Wenjie Zhang,
601 Xuemin Lin, and Jingren Zhou. A survey of distributed graph algorithms on massive graphs. *CoRR*,
602 abs/2404.06037, 2024b. URL <https://doi.org/10.48550/arXiv.2404.06037>.
- 603 Soroor Motie and Bijan Raahemi. Financial fraud detection using graph neural networks: A systematic
604 review. *Expert Syst. Appl.*, 240:122156, 2024. URL [https://doi.org/10.1016/j.eswa.
605 2023.122156](https://doi.org/10.1016/j.eswa.2023.122156).
- 606 OpenAI. GPT-4 technical report. *CoRR*, abs/2303.08774, 2023. URL [https://doi.org/10.
607 48550/arXiv.2303.08774](https://doi.org/10.48550/arXiv.2303.08774).
- 608 Bryan Perozzi, Bahare Fatemi, Dustin Zelle, Anton Tsitsulin, Seyed Mehran Kazemi, Rami Al-Rfou,
609 and Jonathan Halcrow. Let your graph do the talking: Encoding structured data for llms. *CoRR*,
610 abs/2402.05862, 2024. URL <https://doi.org/10.48550/arXiv.2402.05862>.
- 611 Chen Qian, Wei Liu, Hongzhang Liu, Nuo Chen, Yufan Dang, Jiahao Li, Cheng Yang, Weize
612 Chen, Yusheng Su, Xin Cong, Juyuan Xu, Dahai Li, Zhiyuan Liu, and Maosong Sun. Chatdev:
613 Communicative agents for software development. In Lun-Wei Ku, Andre Martins, and Vivek
614 Srikumar (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational
615 Linguistics (Volume 1: Long Papers), ACL 2024, Bangkok, Thailand, August 11-16, 2024*, pp.
616 15174–15186. Association for Computational Linguistics, 2024. URL [https://doi.org/10.
617 18653/v1/2024.acl-long.810](https://doi.org/10.18653/v1/2024.acl-long.810).
- 618 Jonathan M Stokes, Kevin Yang, Kyle Swanson, Wengong Jin, Andres Cubillos-Ruiz, Nina M
619 Donghia, Craig R MacNair, Shawn French, Lindsey A Carfrae, Zohar Bloom-Ackermann, et al. A
620 deep learning approach to antibiotic discovery. *Cell*, 180(4):688–702, 2020.
- 621 Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, et al. Llama 2: Open foundation and
622 fine-tuned chat models. *CoRR*, abs/2307.09288, 2023. URL [https://doi.org/10.48550/
623 arXiv.2307.09288](https://doi.org/10.48550/arXiv.2307.09288).
- 624 Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez,
625 Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In Isabelle Guyon, Ulrike von
626 Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman
627 Garnett (eds.), *Advances in Neural Information Processing Systems 30: Annual Conference on
628 Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pp.
629 5998–6008, 2017. URL [https://proceedings.neurips.cc/paper/2017/hash/
630 3f5ee243547dee91fbd053c1c4a845aa-Abstract.html](https://proceedings.neurips.cc/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html).
- 631 Heng Wang, Shangbin Feng, Tianxing He, Zhaoxuan Tan, Xiaochuang Han, and Yulia Tsvetkov.
632 Can language models solve graph problems in natural language? In Alice Oh, Tris-
633 tan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine (eds.), *Ad-
634 vances in Neural Information Processing Systems 36: Annual Conference on Neural Infor-
635 mation Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 -
636 16, 2023*, 2023. URL [http://papers.nips.cc/paper_files/paper/2023/hash/
637 622afc4edf2824alb6aaf5afe153fa93-Abstract-Conference.html](http://papers.nips.cc/paper_files/paper/2023/hash/622afc4edf2824alb6aaf5afe153fa93-Abstract-Conference.html).
- 638 Yanbin Wei, Shuai Fu, Weisen Jiang, James T. Kwok, and Yu Zhang. Gita: Graph to visual
639 and textual integration for vision-language graph reasoning. 2024. URL [https://api.
640 semanticscholar.org/CorpusID:267413180](https://api.semanticscholar.org/CorpusID:267413180).
- 641 Kai Xiong, Xiao Ding, Yixin Cao, Ting Liu, and Bing Qin. Examining inter-consistency of large
642 language models collaboration: An in-depth analysis via debate. In Houda Bouamor, Juan Pino,
643 and Kalika Bali (eds.), *Findings of the Association for Computational Linguistics: EMNLP 2023,
644 Singapore, December 6-10, 2023*, pp. 7572–7590. Association for Computational Linguistics,
645 2023. URL <https://doi.org/10.18653/v1/2023.findings-emnlp.508>.

- 648 Mingji Yang, Hanzhi Wang, Zhewei Wei, Sibow Wang, and Ji-Rong Wen. Efficient algorithms for
 649 personalized pagerank computation: A survey. *IEEE Trans. Knowl. Data Eng.*, 36(9):4582–4602,
 650 2024. URL <https://doi.org/10.1109/TKDE.2024.3376000>.
 651
- 652 Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R. Narasimhan, and Yuan Cao.
 653 React: Synergizing reasoning and acting in language models. In *The Eleventh International Confer-*
 654 *ence on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net,
 655 2023. URL https://openreview.net/forum?id=WE_vluYUL-X.
- 656 Hongxin Zhang, Weihua Du, Jiaming Shan, Qinhong Zhou, Yilun Du, Joshua B. Tenenbaum, Tianmin
 657 Shu, and Chuang Gan. Building cooperative embodied agents modularly with large language
 658 models. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna,*
 659 *Austria, May 7-11, 2024*. OpenReview.net, 2024. URL [https://openreview.net/forum?](https://openreview.net/forum?id=EnXJfQqy0K)
 660 [id=EnXJfQqy0K](https://openreview.net/forum?id=EnXJfQqy0K).
- 661 Xin Zheng, Qiming Zhu, Hongyu Lin, Yaojie Lu, Xianpei Han, and Le Sun. Executing natural
 662 language-described algorithms with large language models: An investigation. In Nicoletta Calzolari,
 663 Min-Yen Kan, Véronique Hoste, Alessandro Lenci, Sakriani Sakti, and Nianwen Xue (eds.),
 664 *Proceedings of the 2024 Joint International Conference on Computational Linguistics, Language*
 665 *Resources and Evaluation, LREC/COLING 2024, 20-25 May, 2024, Torino, Italy*, pp. 6752–6837.
 666 ELRA and ICCL, 2024. URL <https://aclanthology.org/2024.lrec-main.596>.
 667

669 A DISTRIBUTED ALGORITHMS UNDER THE DISTRIBUTED PARADIGM

671 A.1 EXAMPLE OF DISTRIBUTED ALGORITHMS IN DISTRIBUTED ALGORITHM LIBRARY

672 **Shortest Path:** See Figure 6.

673 **Connectivity:** See Figure 7.

674 A.2 EXAMPLE OF DISTRIBUTED ALGORITHMS DESIGNED BY THE MASTER LLM

675 **PageRank:** See Figure 8.

676 **Hamilton Path:** See Figure 9.

677 **Subgraph Matching:** See Figure 10.

682 B THE GRAPHINSTRUCT DATASET

683 The statistics and detailed information of GraphInstruct are shown in Table 3. Hamilton Path and
 684 Subgraph Matching are NP-complete problems.

685 **Hamilton Path Execution Example.**

686 Problem Description:

687 Determine whether or not there is a Hamiltonian path in an undirected graph. In an undirected graph,
 688 (i,j) means that node i and node j are connected with an undirected edge. Given a graph, you need
 689 to output Yes or No, indicating whether there is a Hamiltonian path in the graph. Q: The nodes are
 690 numbered from 0 to 5, and the edges are: (0, 3) (0, 1) (0, 2) (0, 4) (1, 5) (1, 4) (1, 2) (1, 3) (2, 4) (2, 5)
 691 (3, 5) (3, 4). Is there a Hamiltonian path in this graph?

692 Execution Process:

693 `### State`

- 694 1. `'visited'`: Boolean indicating if the node has been visited.
- 695 2. `'path_length'`: Integer representing the current length of the path.
- 696 3. `'max_path_length'`: Integer tracking the longest path found.

697 `### Initialization`

698 Step 1: Set `'visited = False'`, `'path_length = 0'`, and `'max_path_length = 1'` for all n

702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755

```

Shortest Path

State:
1. `distance`: Known shortest distance from source node
Initialization:
For each node, the input is:
Node Id: <node_id>
---
### Initialization
Step 1: If Node Id is 14, set `distance` to 0; otherwise, set `distance` to infinity.
Message:
1. `new_distance`: Computed minimum distance from node 2 to the sending node.
Send:
For each node, the input is:
Node Id: <node_id>
State: 1. distance: <distance>
Neighbor Information:
Connected to:
Node A with weight <weight1>
Node B with weight <weight2>
---
### Send
Step 1: For each neighbor node, calculate `new_distance` = <distance> + <weight>
Step 2: Construct Message sent to Node xx: 1. new_distance: <new_distance>
### Example
## Input
Node Id: 5
State: 1. distance: 2
Neighbor Information:
Connected to:
Node A with weight 3
Node B with weight 5
## Process
For Node A with weight 3,
Step 1: calculate `new_distance` = 2 + 3 = 5
Step 2: construct Message sent to Node A: 1. new_distance: 5
For Node B with weight 5,
Step 1: calculate `new_distance` = 2 + 5 = 7
Step 2: construct Message sent to Node B: 1. new_distance: 7
## Output
Message sent to Node A: 1. new_distance: 5
Message sent to Node B: 1. new_distance: 7
Update:
For each node, the input is:
Node Id: <node_id>
State: 1. distance: <distance>
Received Messages:
Message: 1. new_distance: <new_distance>
Message: 1. new_distance: <new_distance>
---
### Update
Step 1: For each received message, check if `new_distance` is less than current `distance`.
Step 2: If it is, update `distance` to `new_distance`; otherwise, keep `distance` unchanged.
### Example
## Input
Node Id: 5
State: 1. distance: 15
Received Messages:
Message: 1. new_distance: 12
Message: 1. new_distance: 17
## Process
For Message: 1. new_distance: 12,
Step 1: `new_distance` = 12 is less than current `distance` = 15.
Step 2: update `distance` to 12. State: 1. distance: 12
For Message: 1. new_distance: 17
Step 1: `new_distance` = 17 is not less than current `distance` = 12.
Step 2: no update needed. State remains: 1. distance: 12
## Output
State: 1. distance: 12

```

Figure 6: Distributed algorithm for shortest path problem under the distributed paradigm.

756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809

```

Connectivity
State:
1. `Component_Id`: unique identifier for the connected component.
Message:
1. `Sender_Component_Id`: the current component identifier of the sender node.
Initialization:
For each node, the input is:
Node Id: <node_id>
---
### Initialization
Step 1: set Component_Id = <node_id>
Send:
For each node, the input is:
Node Id: <node_id>
State: 1. Component_Id: <component_id>
Neighbor Information:
Connected to:
Node A
Node B
---
### Send
Step 1: Construct Message sent to Node xx: 1. Component_Id: <component_id>
### Example
## Input
For each node, the input is:
Node Id: 5
State: 1. Component_Id: 5
Neighbor Information:
Connected to:
Node A
Node B
## Process
For Node A,
Step 1: Construct Message sent to Node A: 1. Component_Id: 5
For Node B,
Step 1: Construct Message sent to Node B: 1. Component_Id: 5
## Output
Message sent to Node A: 1. Component_Id: 5
Message sent to Node B: 1. Component_Id: 5
Update:
For each node, the input is:
Node Id: <node_id>
State: 1. Component_Id: <component_id>
Receive Messages:
Message: 1. Sender_Component_Id: <sender_component_id>
Message: 1. Sender_Component_Id: <sender_component_id>
---
### Update
Step 1: For each received message, check if <sender_component_id> is less than current
<component_id>
Step 2: If it is, update <component_id> to <sender_component_id>; otherwise, keep
<component_id> unchanged.
### Example
## Input
For each node, the input is:
Node Id: 5
State: 1. Component_Id: 5
Receive Messages:
Message: 1. Sender_Component_Id: 2
Message: 1. Sender_Component_Id: 7
## Process
For Message: 1. Sender_Component_Id: 2,
Step 1: `Sender_Component_Id` = 2 is less than current `Component_Id` = 5.
Step 2: update `Component_Id` to 2. State: 1. Component_Id: 2
For Message: 1. Sender_Component_Id: 7,
Step 1: `Sender_Component_Id` = 7 is not less than current `Component_Id` = 2.
Step 2: no update needed. State remains: 1. Component_Id: 2
## Output
State: 1. Component_Id: 2'''

```

Figure 7: Distributed algorithm for connectivity problem under the distributed paradigm.

810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863

```

PageRank

State:
1. `rank`: Represents the importance of the webpage, initialized to 1.0.

Initialization:
Step 1: Set `rank` to 1.0 for each node (webpage).
Step 2: (No additional steps required).

Message:
1. `contribution`: Current rank of the node divided by its number of outgoing links.

Send:
Step 1: Calculate `contribution` as `rank / out_degree`, where `out_degree` is the total
number of outgoing links.
Step 2: Send `contribution` to each connected neighbor (Node A and Node B).
### Example
## Input
Node Id: 1
State: 1. rank: 2.0
Neighbor Information:
Connected to:
Node A
Node B
## Process
1. Calculate `out_degree` for Node 1:
  - Outgoing links = 2 (to Node A and Node B).
2. Calculate `contribution`:
  - `contribution = rank / out_degree = 2.0 / 2 = 1.0`.
3. Send `contribution` to neighbors:
  - Message to Node A: `contribution = 1.0`.
  - Message to Node B: `contribution = 1.0`.
## Output
Message sent to Node A: 1. contribution: 1.0
Message sent to Node B: 1. contribution: 1.0

Update:
Step 1: Sum all received `contribution` values from messages.
Step 2: Update `rank` using the formula:
  - `new_rank = damping_factor * (sum_of_received_contributions) + (1 - damping_factor) *
initial_rank`.
Step 3: Set the new `rank` as the current state.
### Example
## Input
Node Id: 2
State: 1. rank: 1.5
Received Messages:
Message: 1. contribution: 0.8
Message: 2. contribution: 1.2
## Process
1. Sum all received `contribution` values:
  - `sum_of_received_contributions = 0.8 + 1.2 = 2.0`.
2. Update `rank` using the formula:
  - `new_rank = damping_factor * (sum_of_received_contributions) + (1 - damping_factor) *
initial_rank`.
  - Assuming `damping_factor = 0.85`:
  - `new_rank = 0.85 * 2.0 + 0.15 * 1.5 = 1.7 + 0.225 = 1.925`.
3. Set the new `rank` as the current state:
  - `rank = 1.925`.
## Output
State: 1. rank: 1.925

```

Figure 8: Distributed algorithm for pagerank calculation under the distributed paradigm.

864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917

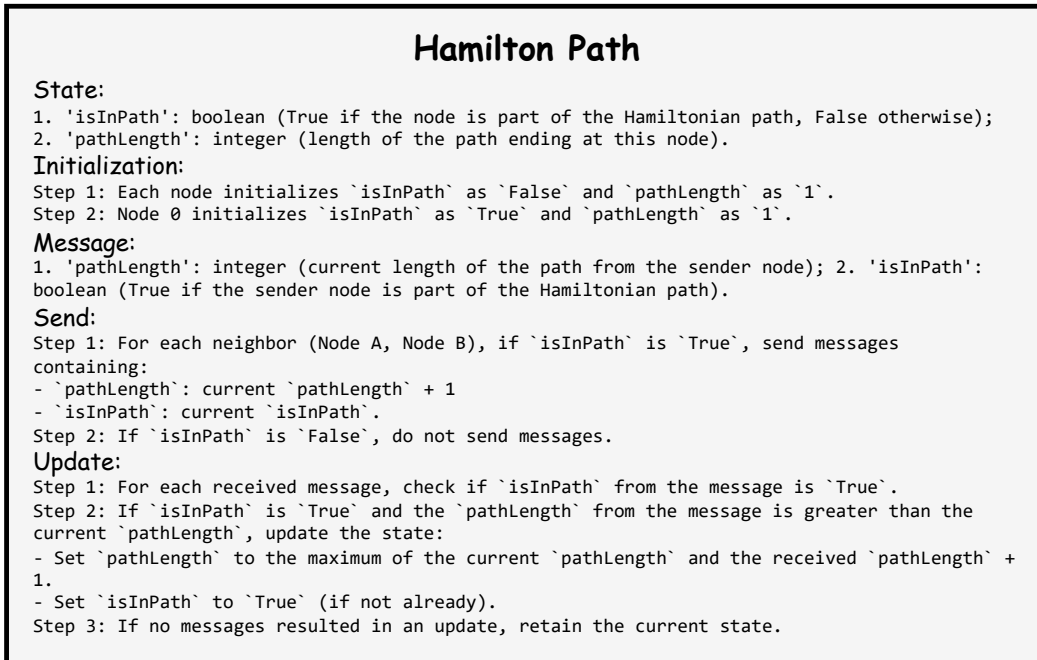


Figure 9: Distributed algorithm for hamilton path problem under the distributed paradigm.

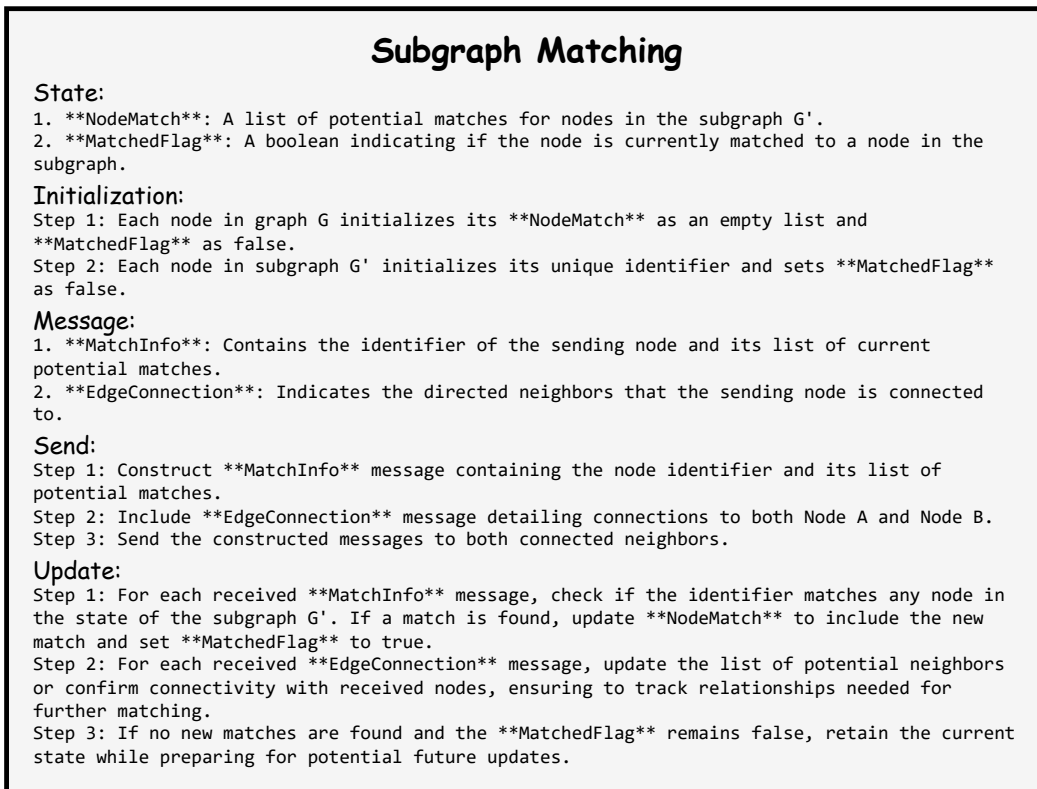


Figure 10: Distributed algorithm for subgraph matching problem under the distributed paradigm.

Table 3: The detailed information of GraphInstruct dataset.

Problem	Definition	Node Range	Test Size
Cycle Detection	Detect if a given graph \mathcal{G} contains any cycles.	[2, 100]	400
Connectivity	Assess if two nodes u and v in a given graph \mathcal{G} are connected via a path.	[2, 100]	400
Bipartite Graph Check	Judge if a given graph \mathcal{G} is bipartite.	[2, 100]	400
Topological Sort	Find a topological ordering of vertices in a directed acyclic graph \mathcal{G} .	[2, 50]	400
Shortest Path	Compute the shortest path between two specific nodes u and v in a given graph \mathcal{G} .	[2, 100]	400
Maximum Triangle Sum	Find the maximum sum of weights for any connected triplet of vertices in a given graph \mathcal{G} .	[2, 25]	400
Maximum Flow	Calculate the maximum flow from a source node s to a sink node t in a directed graph \mathcal{G} .	[2, 50]	400
Hamilton Path	Determine if a given graph \mathcal{G} has a Hamiltonian path that visits each vertex exactly once.	[2, 50]	400
Subgraph Matching	Verify if there exists a subgraph in \mathcal{G} that is isomorphic to a given graph \mathcal{G}' .	[2, 30]	400

Step 2: For the initial node (e.g., Node Id: 0), set `visited = True` and `path_length = 1`.

Distributed_Algorithm_Designer: ### Message

1. `path_length`: The current path length from the sender.
2. `max_path_length`: The maximum path length known to the sender.
3. `visited_flag`: Boolean indicating whether the sender has visited the node.

Send

Step 1: For each neighbor (Node A, Node B), if `visited` is `False`, construct the message.

- Message to Node A: `(Node Id: <node_id>, path_length + 1, max(max_path_length, path_length))`

- Message to Node B: `(Node Id: <node_id>, path_length + 1, max(max_path_length, path_length))`

Step 2: Send the constructed messages to both neighbors.

Example

Input

Node Id: 0

State:

1. visited: False
2. path_length: 2
3. max_path_length: 3

Neighbor Information:

Connected to:

Node A

Node B

Process

Step 1: Check neighbors Node A and Node B. Since `visited` is `False`, proceed to construct messages.

- For Node A:

- Calculate new `path_length`: `2 + 1 = 3`

- Calculate `max_path_length`: `max(3, 2 + 1) = 3`

- For Node B:

- Same calculations apply as for Node A.

Step 2: Construct messages:

- Message to Node A: `(Node Id: 0, 3, 3)`

- Message to Node B: `(Node Id: 0, 3, 3)`

Output

Message sent to Node A:

1. path_length: 3
2. max_path_length: 3
3. visited_flag: False

```

972 Message sent to Node B:
973 1. path_length: 3
974 2. max_path_length: 3
975 3. visited_flag: False
976 ### Update
977 Step 1: For each received message, check the 'visited_flag'. If it is
978 'False', proceed to the next steps.
979 Step 2: Update 'visited' to 'True' for the current node.
980 Step 3: Update 'path_length' to the maximum of its current value and
981 the received 'path_length' from the message, incrementing by 1 if the
982 message indicates a potential continuation of the path.
983 Step 4: Update 'max_path_length' to the maximum of its current value
984 and the received 'max_path_length' from the message.
985 Step 5: If any messages were processed, trigger a re-evaluation of
986 outgoing messages to neighbors, if applicable.
987 ### Example
988 ## Input
989 Node Id: 1
990 State:
991 1. visited: False
992 2. path_length: 1
993 3. max_path_length: 2
994 Received Messages:
995 Message 1:
996 1. path_length: 2
997 2. max_path_length: 3
998 3. visited_flag: False
999 Message 2:
1000 1. path_length: 3
1001 2. max_path_length: 4
1002 3. visited_flag: True
1003 ## Process
1004 Step 1: Process Message 1. Since 'visited_flag' is 'False', proceed:
1005 - Update 'visited' to 'True'.
1006 - Update 'path_length': 'max(1, 2) + 1 = 3'.
1007 - Update 'max_path_length': 'max(2, 3) = 3'.
1008 Step 2: Process Message 2. Since 'visited_flag' is 'True', ignore this
1009 message and do not make any changes.
1010 ## Output
1011 State:
1012 1. visited: True
1013 2. path_length: 3
1014 3. max_path_length: 3
1015 ### Termination
1016 The algorithm continues until:
1017 - A node reaches a 'path_length' equal to the total number of nodes
1018 (indicating a Hamiltonian path exists).
1019 - Or until a predefined number of iterations equal to the total number
1020 of nodes ('N') is reached without discovering a Hamiltonian path.
1021 Master: ##### ITER 3, STEP 1: REASONING #####
1022 Master: {'thought': "The algorithm is designed to check for a
1023 Hamiltonian path. Now let's execute it to see if such a path exists.",
1024 'speak': "The algorithm is ready. Let's execute it to determine if there is
1025 a Hamiltonian path.", 'function': [{'name': 'Distributed_Execution',
1026 'arguments': {}}]}
1027 Master: ##### ITER 3, STEP 2: ACTING #####
1028 0: State: 1. visited: True 2. path_length: 1 3. max_path_length: 1
1029 2: State: 1. visited: False 2. path_length: 0 3. max_path_length: 1
1030 5: State: 1. visited: False 2. path_length: 0 3. max_path_length: 1

```

```

1026 3: State: 1. visited: False 2. path_length: 0 3. max_path_length: 1
1027 4: State: 1. visited: False 2. path_length: 0 3. max_path_length: 1
1028 1: State: 1. visited: False 2. path_length: 0 3. max_path_length: 1
1029 Node 5 Send Message to Node 1: 1. path_length: 1 2. max_path_length:
1030 1 3. visited_flag: False
1031 Node 5 Send Message to Node 2: 1. path_length: 1 2. max_path_length:
1032 1 3. visited_flag: False
1033 Node 5 Send Message to Node 3: 1. path_length: 1 2. max_path_length:
1034 1 3. visited_flag: False
1035 ...
1036 All agents' state unchanged, terminating early...
1037 >>> END
1038 system: 1. Execute function Distributed_Execution
1039 [ARGUMENTS]:
1040 [STATUS]: SUCCESS
1041 [RESULT]: Node: 0 State: 1. visited: True 2. path_length: 5 3. max_path_length: 5
1042 Node: 1 State: 1. visited: True 2. path_length: 2 3. max_path_length: 1
1043 Node: 2 State: 1. visited: True 2. path_length: 4 3. max_path_length: 3
1044 Node: 3 State: 1. visited: True 2. path_length: 5 3. max_path_length: 3
1045 Node: 4 State: 1. visited: True 2. path_length: 4 3. max_path_length: 3
1046 Node: 5 State: 1. visited: True 2. path_length: 5 3. max_path_length: 3
1047 Master: ##### ITER 4, STEP 1: REASONING #####
1048 Master: {'thought': 'Since the maximum path length found is 5, which is
1049 less than the total number of nodes (6), there is no Hamiltonian path in
1050 the graph.', 'speak': 'No, there is no Hamiltonian path in this graph.',
1051 'function': []}
1052
1053 C EXECUTION EXAMPLES OF GRAPHAGENT-REASONER
1054
1055 Shortest Path Execution Example.
1056
1057 Problem Description:
1058 Find the shortest distance from a source node to other nodes in an undirected graph. In an undirected
1059 graph, (i,j,k) means that node i and node j are connected with an undirected edge with weight k. The
1060 graph has 8 nodes, and the edges are: (0,7,9) (0,1,7) (0,4,9) (1,7,1) (2,7,7) (2,6,5) (2,5,8) (3,5,9)
1061 (3,4,8) (3,6,1) (4,7,7) (4,5,6) (5,7,6). Give the weight of the shortest distance from node 1 to other
1062 node.
1063 Execution Process:
1064
1065 Master: ##### ITER 1, STEP 1: REASONING #####
1066 Master: {'thought': 'I need to find the shortest distances from node 1 to all
1067 other nodes in the given undirected graph. First, I will build the graph using
1068 the provided edges and then design a distributed algorithm to find the
1069 shortest paths.', 'speak': 'Building the graph and designing the distributed
1070 algorithm to find the shortest distances from node 1.', 'function': '[{"name":
1071 "Build_Graph", "arguments": {"Edges": [[0, 7, 9], [0, 1, 7], [0, 4, 9], [1, 7, 1], [2,
1072 7, 7], [2, 6, 5], [2, 5, 8], [3, 5, 9], [3, 4, 8], [3, 6, 1], [4, 7, 7], [4, 5, 6],
1073 "Nodes_Num": 8, "Directed": false}]}'
1074 Master: ##### ITER 1, STEP 2: ACTING #####
1075 >>> Edges: [[0, 7, 9], [0, 1, 7], [0, 4, 9], [1, 7, 1], [2, 7, 7], [2, 6, 5], [2,
1076 >>> Nodes_Num: 8
1077 >>> Directed: False
1078 >>> END
1079 system: 1. Execute function Build_Graph
1080 [ARGUMENTS]:
1081 Edges: [[0, 7, 9], [0, 1, 7], [0, 4, 9], [1, 7, 1], [2, 7, 7], [2, 6, 5], [2,

```

```

1080         5, 9], [3, 4, 8], [3, 6, 1], [4, 7, 7], [4, 5, 6], [5, 7, 6]]
1081         Nodes_Num: 8
1082         Directed: False
1083     [STATUS]: SUCCESS
1084 Master: ##### ITER 2, STEP 1: REASONING #####
1085 Master: {'thought': 'The graph has been successfully built. Now, I will design
1086 a distributed algorithm to find the shortest distances from node 1 to all
1087 other nodes.', 'speak': 'Designing a distributed algorithm to find the
1088 shortest distances from node 1.', 'function': '[{"name":
1089 "Design_Distributed_Algorithm", "arguments": {"Problem_Str": "Find the
1090 shortest distance from node 1 to all other nodes in an undirected graph
1091 with 8 nodes and given edges."}]'}
1092 Master: ##### ITER 2, STEP 2: ACTING #####
1093 >>> END
1094 Master: ##### ITER 3, STEP 1: REASONING #####
1095 Master: {'thought': 'The distributed algorithm has been successfully
1096 designed. Now, I will execute the algorithm to find the shortest distances
1097 from node 1 to all other nodes.', 'speak': 'Executing the distributed
1098 algorithm to calculate the shortest distances from node 1.', 'function':
1099 '[{"name": "Distributed_Execution", "arguments": {}}]'}
1100 Master: ##### ITER 3, STEP 2: ACTING #####
1101 Initialization:
1102 1: State: 1. distance: 0
1103 0: State: 1. distance: \infinity
1104 2: State: 1. distance: \infinity
1105 4: State: 1. distance: \infinity
1106 5: State: 1. distance: \infinity
1107 7: State: 1. distance: \infinity
1108 3: State: 1. distance: \infinity
1109 6: State: 1. distance: \infinity
1110 Node 6 Send Message to Node 2: 1. new_distance: \infinity
1111 Node 6 Send Message to Node 3: 1. new_distance: \infinity
1112 Node 1 Send Message to Node 0: 1. new_distance: 7
1113 Node 1 Send Message to Node 7: 1. new_distance: 1
1114 Node 3 Send Message to Node 5: 1. new_distance: \infinity
1115 ....
1116 All agents' state unchanged, terminating early...
1117 >>> END
1118 system: 1. Execute function Distributed_Execution
1119 [ARGUMENTS]:
1120 [STATUS]: SUCCESS
1121 [RESULT]: Node: 0 State: 1. distance: 7
1122 Node: 1 State: 1. distance: 0
1123 Node: 2 State: 1. distance: 8
1124 Node: 3 State: 1. distance: 14
1125 Node: 4 State: 1. distance: 8
1126 Node: 5 State: 1. distance: 7
1127 Node: 6 State: 1. distance: 13
1128 Node: 7 State: 1. distance: 1
1129 Master: ##### ITER 4, STEP 1: REASONING #####
1130 The shortest distances from node 1 are as follows: Node 0: 7, Node 1: 0,
1131 Node 2: 8, Node 3: 14, Node 4: 8, Node 5: 7, Node 6: 13, Node 7: 1.Master:
1132 {'thought': 'The distributed algorithm has been successfully executed, and I
1133 have obtained the shortest distances from node 1 to all other nodes.',
1134 'speak': 'The shortest distances from node 1 are as follows: Node 0: 7, Node
1135 1: 0, Node 2: 8, Node 3: 14, Node 4: 8, Node 5: 7, Node 6: 13, Node 7: 1.',
1136 'function': []}

```