Design and Analysis of Algorithms final project

Student:

Ali Kashi Pazha (40224641)

Instructor:

Dr. Pishgoo

GitHub Repository:

Link

Project:

Dynamic GraphRAG for Contract Risk Analysis

Part I: Analysis and Design

1. Problem Definition & Architecture

The goal of this project is to develop a **Dynamic GraphRAG** system capable of inferring contractual risks based on real-time news events. The system architecture is built upon two key principles: **Multi-Agent Extraction** and **Graph Forest Architecture**.

1.1. Multi-Agent Extraction Pipeline

The conversion of raw text into a Knowledge Graph is handled by a pipeline of specialized LLM Agents (simulated in the current dataset, but designed as follows):

1. **Entity Extractor Agent:** Parses text (contract or news) to identify key Nodes with specific types (e.g., `Company`, `Location`, `RiskEvent`).
2. **Relation Classifier Agent:** Analyzes the interaction between entities to define Edges and their types (e.g., `SUPPLIES_TO`, `LOCATED_IN`, `BLOCKS`).
3. **Graph Validator Agent:** Performs logical checks on the graph (e.g., ensuring a “Location” cannot “Manufacture” unless specified) and outputs the final JSON.

1.2. Graph Forest Architecture

Instead of constantly merging sub-graphs into a monolithic database, we utilize a **Forest** architecture:

- **Base Graph (G_{Base}):** Derived from the contract text. This graph is **Immutable**, ensuring the stability of the contract analysis.
- **News Graphs (G_{News}):** Each news item acts as a standalone micro-graph.
- **Dynamic Linking:** News graphs are “linked” to the Base Graph, not merged. This connection occurs via **Bridge Nodes** (e.g., a node Taiwan in a news graph links to the node Taiwan in the contract graph).

This structure allows us to preserve the temporal context of events and trace Ripple Effects through inter-graph hops.

---

2. Mathematical Model

We define the problem space as a directed graph $G = (V, E)$.

- **Vertex Set (V):** Comprises contract entities (V_C) and news entities (V_N).

$$V = V_C \cup V_N$$

- **Edge Set (E):** Comprises intra-contract relations (E_C), intra-news relations (E_N), and virtual connecting edges (E_{Bridge}).

$$E = E_C \cup E_N \cup E_{Bridge}$$

Inference (Reasoning) Definition:

Risk identification is equivalent to finding a directed **Path** P:

$$P = \langle v_{start}, \dots, v_{bridge}, \dots, v_{end} \rangle$$

Where:

1. $v_{start} \in V_N$(Rooted in a news event, e.g., “Typhoon”).
2. $v_{bridge} \in V_N \cap V_C$(The semantic bridge connecting News to Contract).
3. $v_{end} \in V_C$(Target node in the contract representing breach or penalty, e.g., “Penalty”).

---

3. Reasoning Algorithm: Causal Chain Discovery

The inference engine employs a **Hybrid** approach: **Graph Search (BFS/DFS)** for structural traversal and **LLM** for semantic validation of paths.

3.1. Pseudocode

```
FUNCTION DiscoverRisk(contract_graph, news_graph_forest):
    risks_found = []
    FOR EACH news_graph IN news_graph_forest:
        # 1. Identify Entry Points (Events in News)
        events = [node for node in news_graph.nodes if node.type in ["Event", "Risk"]]
        FOR event IN events:
            # 2. Initialize Traversal (Queue for BFS)
            queue = [(event, path=[event])]
            visited = {event}
            WHILE queue IS NOT EMPTY:
                current_node, path = queue.pop()
                # 3. Check for Bridge to Contract
                IF current_node IN contract_graph.nodes:
                    # Switch context to Contract Graph
                    extended_paths = TraverseContract(contract_graph, current_node, path)
                    risks_found.extend(extended_paths)
                # 4. Traverse within News Graph
                neighbors = GetNeighbors(news_graph, current_node)
                FOR neighbor IN neighbors:
                    IF neighbor NOT IN visited:
                        visited.add(neighbor)
                        queue.push((neighbor, path + [neighbor]))
    RETURN risks_found
FUNCTION TraverseContract(graph, start_node, current_path):
    # Continue traversal inside the contract to find impact
    # Look for specific target types (e.g., "Penalty", "Clause")
    ...
```

3.2. Big-O Complexity Analysis

- Let V be the total nodes and E the total edges.
- In the worst-case scenario, BFS traverses all nodes and edges: $O(V + E)$.
- Since contract and news graphs are generally **Sparse** and the causal chain depth is limited (typically < 10 hops), this algorithm is highly optimized for Real-time applications.

---

4. Data Structure (JSON Schema)

Based on the `contracts_and_news.json` dataset, the system input structure is standardized as follows:

```
[
  {
    "contract_id": "String (e.g., C01)",
    "title": "String",
    "contract_text": "String (Raw Text)",
    "base_graph": {
      "entities": [{"id": "String", "type": "String"}],
      "relations": [{"source": "String", "target": "String", "type": "String"}]
    },
    "news_sequence": {
      "entities": [{"id": "String", "type": "String"}],
      "relations": [{"source": "String", "target": "String", "type": "String"}]
    }
  }
]
```

---

5. Repository Structure

To maintain project discipline and adhere to GitHub best practices, the following folder structure is mandated for Phase 1 & 2:

```
Algorithm_Project_SemanticGraph_Group9/
├── data/
│   └── raw/
│       └── contracts_and_news.json
├── src/
│   ├── __init__.py
│   ├── data_loader.py
│   ├── show_graphs.py
│   ├── show_graphs.py
│   ├── graph_rag_engine.py
│   ├── neo4j_manager.py
│   └── reasoning_engine.py
├── outputs/
│   ├── graphs/
│   │   ├── c0.html
│   │   ├── c1.html
│   │   ├── c2.html
│   │   └── ...
│   ├── main_output.txt
│   └── main_neo4j_output.txt
├── tests/
│   ├── test_loader.py
│   └── test_reasoning.py
├── docs/
│   └── report/
│       └── Algorithm_Project_SemanticGraph_Group9.pdf
├── .gitignore
├── main_neo4j.py
├── main.py
├── requirements.txt
└── README.md
```

Part II: Dynamic GraphRAG for Contract Risk Analysis

Abstract

Global supply chains and contractual obligations are highly sensitive to external disruptions such as natural disasters, geopolitical conflicts, and labor strikes. Traditional text-based search methods fail to capture the structural dependencies between contract entities and dynamic news events. This project presents a **Semantic Graph System** that employs two distinct algorithmic approaches to analyze risk: (1) A deterministic **NetworkX-based Causal Chain Discovery** algorithm, and (2) A generative **Graph Retrieval-Augmented Generation (GraphRAG)** system using Neo4j and Large Language Models (LLMs). The system dynamically links static contract graphs with streaming news graphs to identify, trace, and explain risk propagation paths.

1. Introduction

Modern contracts are not isolated documents; they exist within a complex network of dependencies (suppliers, logistics, locations, resources). When an external event occurs (e.g., a Typhoon in Taiwan), it triggers a cascade of effects that may breach specific contract clauses (e.g., shipment delays).

The objective of this project is to automate the detection of these risks. We model contracts as **Knowledge Graphs** where nodes represent entities (Companies, Products, Locations) and edges represent relations (PRODUCES, SHIPPED_FROM). We compare two methods:

1. **Algorithmic Graph Traversal:** A strict, path-finding approach to detect causal links.
2. **LLM-driven GraphRAG:** A semantic approach converting natural language queries into Cypher queries to extract and summarize insights.

2. Methodology and Data Structure

2.1. Dataset and Graph Modeling

The dataset (contracts_and_news.json) consists of paired sub-graphs:

- **Base Graph (Static):** Represents the contract.
  - $G_B = (V_B, E_B)$ where V_B includes nodes like TechCore_Inc, SiliconFoundry, Port_of_Kaohsiung.
- **News Graph (Dynamic):** Represents a sequence of real-world events.
  - $G_N = (V_N, E_N)$ where V_N includes nodes like Typhoon_Krathon, Kaohsiung_Region.

2.2. Bridge Node Identification

The core algorithmic challenge is connecting disconnected graphs. We define **Bridge Nodes** (V_{bridge}) as the intersection of entities present in both the contract and the news:

$$V_{bridge} = V_B \cap V_N$$

For example, if a contract mentions "Port_of_Kaohsiung" and the news mentions the port closing, this node acts as the bridge allowing risk propagation.

---

3. Algorithmic Approach (Branch: main)

This approach utilizes the ReasoningEngine class implemented using the **NetworkX** library.

3.1. Algorithm: Causal Chain Discovery

The problem is modeled as a search problem on a directed graph.

1. **Graph Composition:** We construct a composite graph $G_{total} = G_B \cup G_N$.
2. **Root Cause Identification:** The node in the News Graph with an in-degree of 0 is selected as the Start_Event (e.g., Typhoon_Krathon).
3. **Target Identification:** We define a set of target node types $T = \{Risk, Penalty, Obligation, Product\}$.
4. **Traversal (Modified BFS):**
   We employ a Breadth-First Search (BFS) strategy with a depth limit to trace the path from $Start\_Event$ to any $t \in T$.
   - *Constraint:* To capture upstream dependencies (e.g., A Factory depends on a Country), the traversal considers both successors and predecessors in the graph structure (successors(u) + predecessors(u)), effectively treating the graph as undirected for connectivity analysis while preserving edge semantics for reporting.

3.2. Complexity Analysis

- **Time Complexity:** $O(V + E)$, where V and E are the vertices and edges in the composed graph. Since contract graphs are sparse, this is highly efficient.
- **Space Complexity:** $O(V)$ to store the graph and the visited queue.

---

4. GraphRAG Approach (Branch: feature/neo4j-llm)

This approach utilizes **Neo4j**, **LangChain**, and **Google Gemini LLM**.

4.1. Architecture

1. **Ingestion (neo4j_manager.py):** The JSON graphs are ingested into a Neo4j graph database. Nodes are labeled as Entity with dynamic types (e.g., :Company, :WeatherEvent).
2. **Text-to-Cypher Translation:** The GraphRAGEngine uses an LLM to convert a natural language question (e.g., "Analyze the impact of Typhoon Krathon") into a structured Cypher query.
   - *Prompt Engineering:* We guide the LLM to generate queries that search for paths of length 1 to 3 (-[*1..3]-) to find indirect connections.
3. **Result Synthesis:** The graph paths returned by Neo4j are fed back to the LLM to generate a natural language summary of the risk.

4.2. Advantage

Unlike the rigid BFS approach, GraphRAG can handle semantic variations and complex query logic (e.g., "Find all suppliers affected by floods") without hardcoded target types.

---

5. Experimental Results

Case Study 1: Semiconductor Supply Chain (Contract C01)

- **Event:** Typhoon Krathon hits Taiwan.
- **Algorithmic Output (main.py):**
  - Identified Path: `Typhoon_Krathon --(FLOODS)--> Kaohsiung_Region --(CLOSES)--> Port_of_Kaohsiung <--(SHIPPED_FROM)-- GPU_Chips`
  - Result: Detected risk to GPU_Chips shipment.
- **GraphRAG Output (main_neo4j.py):**
  - LLM Response: "`Typhoon Krathon is approaching Taiwan... causing closure of Port of Kaohsiung... SiliconFoundry produces GPU Chips... contract with TechCore Inc.`"
  - Result: Provided a narrative explanation of the penalty clause.

Case Study 2: Automotive Manufacturing (Contract C03)

- **Event:** Trade Union GDL Strike.
- **Algorithmic Output:**
  - Identified Path: `Trade_Union_GDL --(STRIKES_AGAINST)--> Rail_Cargo_DB <--(MOVED_BY)-- Steel_Sheets`
- **GraphRAG Output:**
  - LLM Response: "`The Trade Union GDL is impacting... Rail Cargo DB. This strike disrupts the movement of Steel Sheets, which directly affects SteelWorks Gmbh.`"

6. Conclusion

The project demonstrates that representing contracts as semantic graphs allows for efficient risk analysis.

- The **Algorithmic Approach** (Main Branch) is deterministic, fast, and ideal for automated alerting systems where specific logic (e.g., "Path to Penalty") is required.
- The **GraphRAG Approach** (Feature Branch) offers superior interpretability and flexibility, allowing users to query the system in natural language.

A hybrid system—using algorithms for detection and LLMs for explanation—provides the optimal solution for Algorithm-aided Legal Risk Analysis.