Assignment - 3 Big Data Analysis

Akshat Raj Saxena

Abstract

This projects deals with the json file named "train.json". So first i filter the json file and make the entries comma seperated in order to make simple for Neo4j to identify different entries. Then, the modified json file is uploaded to Neo4j to create a citation graph where "paper" represents the node id of the citing paper, and the list corresponding to "reference" represents the node ids of the cited papers. Following that, i ran SimRank algorithm on this graph using Apache Spark to analyze similarities between nodes (papers).

1. Data Preprocessing for Neo4j

The dataset provided in 'train.json' contains information on research papers, which includes identifiers, venues, text, and references. I filtered the json file to extract necessary fields and converted it into a format that Neo4j might be feasible with.

- **Paper ID**: Unique identifier for each paper.
- **Venue**: Conference or journal where the paper was published.
- **Text**: Abstract or text data of the paper.
- **References**: List of other papers cited by this paper.

2. Generating Graph Using Neo4j

After preprocessing, the modified file is uploaded to Neo4j Graph Database to create a directed citation graph. Each paper is represented as a node, and each citation is represented as a directed edge between two nodes. Figure 1 represents the output.

Code for Graph Creation

The following code snippet shows the structure of the Neo4j query used to create nodes and relationships:

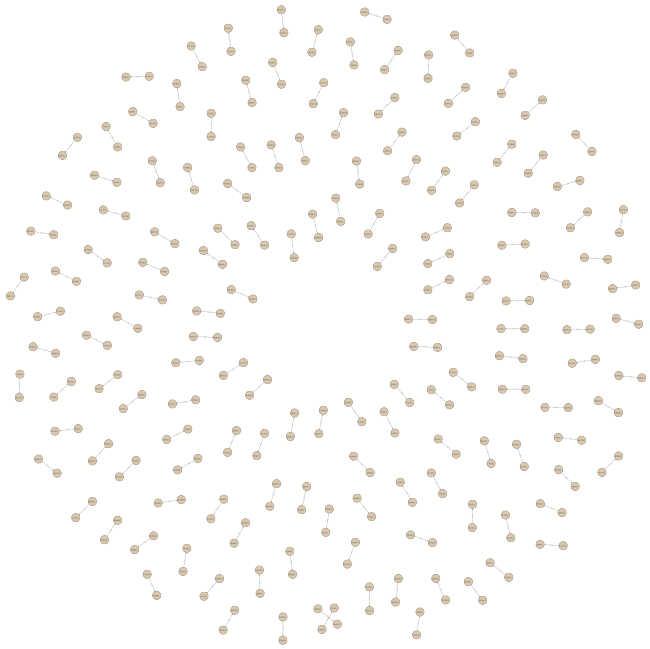

Figure 1: Sample of the filtered JSON data for Neo4j

```
def create_citation_graph(tx, paper_id, venue, text, references):
    tx.run("MERGE (p:Paper {id: $paper_id}) SET p.venue = $venue, p.text = $te
           paper_id=paper_id, venue=venue, text=text)
    for ref_id in references:
        tx.run("MERGE (ref:Paper {id: $ref_id})"
               "MERGE (p:Paper {id: $paper_id})-[:CITES]->(ref)",
               paper_id=paper_id, ref_id=ref_id)
```

The Neo4j database was built using this code, with nodes and edges representing papers and their citations, respectively.

Command on Neo4j Cipher to Generate Graph

```
MATCH (p:Paper)-[:CITES]->(ref:Paper)
RETURN p, ref
```

3. Exporting Graph to CSV

The Neo4j graph was exported to a CSV file, with each row containing a source node and a target node representing a citation relationship. This format is compatible with NetworkX, enabling us to analyze it with SimRank.

Exporting the csv file of the Generated Graph

```
MATCH (p:Paper)-[:CITES]->(ref:Paper)
RETURN p.id AS source, ref.id AS target
```

4. Running SimRank Using Apache Spark

SimRank is an iterative algorithm used to compute similarities between nodes in a graph. In my code, the similarity based on the concept that "two nodes are similar if they are referenced by similar nodes."

Implementation

We implemented a memory-efficient version of SimRank in Python, using Apache Spark and NetworkX to handle large graphs.

```
def batch_simrank_similaritydef batch_simrank_similarity(G, source=None, importa
    """Memory-efficient implementation of SimRank algorithm using batched proces
    nodes = list(G.nodes())
    n = len(nodes)
    node_to_idx = {node: i for i, node in enumerate(nodes)}
    logger.info(f"Processing graph with {n} nodes")
    if source is not None:
        source_idx = node_to_idx[source]
        sim_curr = lil_matrix((n, 1), dtype=np.float64)
        sim_curr[source_idx, 0] = 1.0
        sim_prev = lil_matrix((n, 1), dtype=np.float64)
        for iteration in range(max_iterations):
            logger.info(f"Iteration {iteration + 1}/{max_iterations}")
            sim_prev = sim_curr.copy()
            sim_curr = lil_matrix((n, 1), dtype=np.float64)
            sim_curr[source_idx, 0] = 1.0
            for start_idx in range(0, n, batch_size):
                end_idx = min(start_idx + batch_size, n)
                batch_nodes = nodes[start_idx:end_idx]
                for v_node in batch_nodes:
                    v = node_to_idx[v_node]
                    if v == source_idx:
                        continue
                    u_neighbors = list(G.predecessors(nodes[source_idx]))
                    v_neighbors = list(G.predecessors(v_node))

                    if not u_neighbors or not v_neighbors:
                        sim_curr[v, 0] = 0
                        continue
```

```
                    sum_sim = 0
                    for u_nbr, v_nbr in product(u_neighbors, v_neighbors):
                        u_nbr_idx = node_to_idx[u_nbr]
                        v_nbr_idx = node_to_idx[v_nbr]
                        sum_sim += sim_prev[v_nbr_idx, 0]
                    sim_curr[v, 0] = (importance_factor * sum_sim) / (len(u_neig
            if np.abs(sim_curr - sim_prev).max() < tolerance:
                logger.info(f"Converged after {iteration + 1} iterations")
                break
        return {nodes[i]: sim_curr[i, 0] for i in range(n)}
    else:
        raise ValueError("Full matrix computation not supported in memory-efficie
```

Output Interpretation

The output of the SimRank algorithm shows the top 10 most similar papers to each query node, based on citation patterns. These similarities are ranked, revealing which papers are conceptually closest based on citations. Figure 2 below represents the output that i am getting.

5. Insights

This projects gives me the insights about:

- How to preprocess structured JSON data for use in Neo4j.
- Exposure of Neo4j Graph Database.
- The implementation of graph similarity algorithms, such as SimRank, for citation analysis.

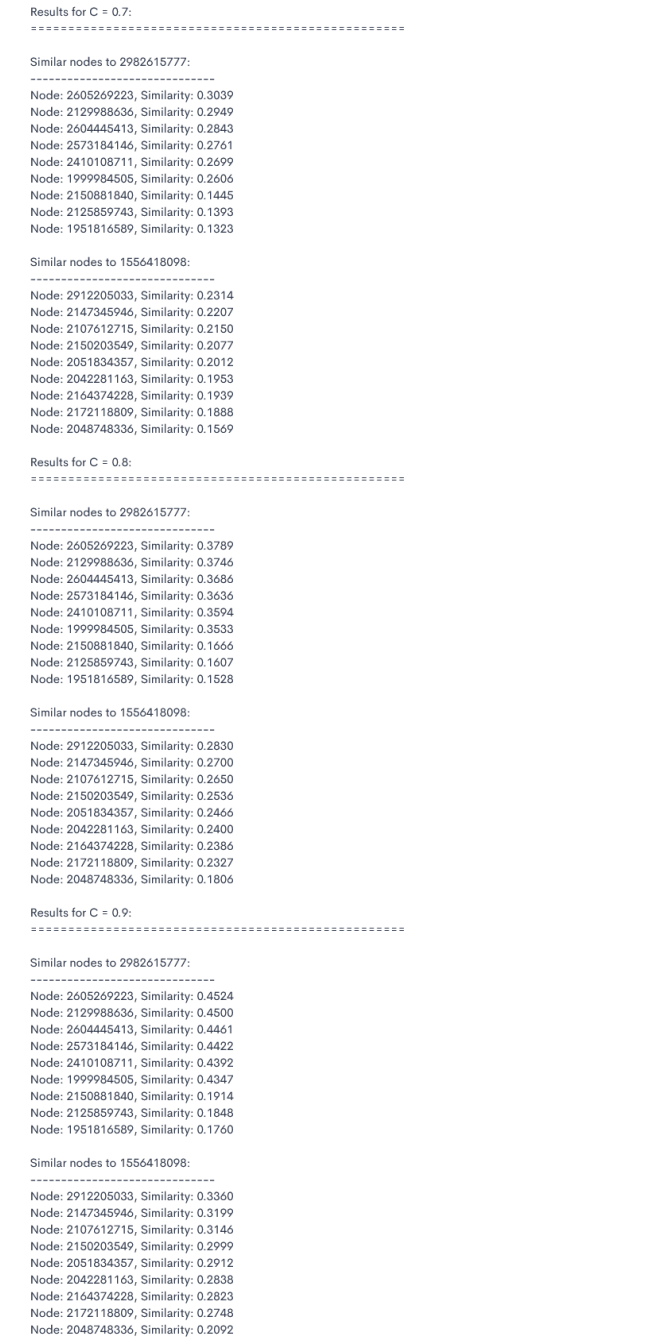

Figure 2: SimRank similarity results between selected papers