📊 Project 3 – Graph Databases with Neo4j

Course: Big Data Systems
Dataset: MOOC User Action Dataset
Tools: Python, Pandas, Neo4j, Cypher, NetworkX, Matplotlib
Author: Nikos Mitsakis
Due Date: June 8, 2024

🚀 Project Overview

The objective of this project was to transform the MOOC User Action dataset into a graph representation using Neo4j, execute Cypher queries to extract meaningful insights, and visualize a portion of the graph.

🗃️ Dataset Description

We used the **MOOC Action Dataset**, consisting of:

- `mooc_actions.tsv`: records of user actions with `user_id`, `target_id`, `action_id`, `timestamp`.
- `mooc_action_features.tsv`: 4 numerical features per action.
- `mooc_action_labels.tsv`: one label (integer 0 or 1) per action.

Each action thus becomes a relationship in the graph, enriched with metadata (features, label, timestamp).

🧱 Data Preprocessing

☑️ Merged and Cleaned Data

```
# Merged all TSV files into one DataFrame
combined_df = pd.concat([actions_df, features_df, labels_df], axis=1)
```

☑️ Exported and Split Data

- Users and Targets saved as CSVs.
- Actions split into 10,000-row chunks for batch import.

```
# Save processed files
users_df.to_csv("users.csv")
targets_df.to_csv("targets.csv")
```

🧠 Neo4j Database Setup

🛠️ Environment

- **Neo4j Desktop** version 1.6.1
- Local DB instance configured to accept CSV imports
- Neo4j Bolt connection on `localhost:7687`

☑️ Database Initialization

- Database reset using:

```
:use system
DROP DATABASE neo4j IF EXISTS;
CREATE DATABASE neo4j;
```

- Driver connected and tested via Python.

📥 Data Loading into Neo4j

🧩 Schema Design

- **Nodes**:
  - `(:User {id})`
  - `(:Target {id})`
- **Relationship**:
  - `(:User)-[:PERFORMS {action_id, timestamp, feature1-4, label}]->(:Target)`

☑️ Bulk Load with Transactions

```
CALL {
  LOAD CSV WITH HEADERS FROM 'file:///actions_part_X.csv' AS row
  MATCH (u:User {id: row.user_id})
  MATCH (t:Target {id: row.target_id})
  CREATE (u)-[:PERFORMS {
    action_id: row.action_id,
    timestamp: row.timestamp,
    feature1: toFloat(row.feature1),
    feature2: toFloat(row.feature2),
    feature3: toFloat(row.feature3),
    feature4: toFloat(row.feature4),
    label: toInteger(row.label)
  }]->(t)
} IN TRANSACTIONS OF 1000 ROWS
```

📋 Cypher Queries

All required Cypher queries were stored in `cypher_queries.txt` and executed using a Python automation script. Results were saved in the `results/` directory.

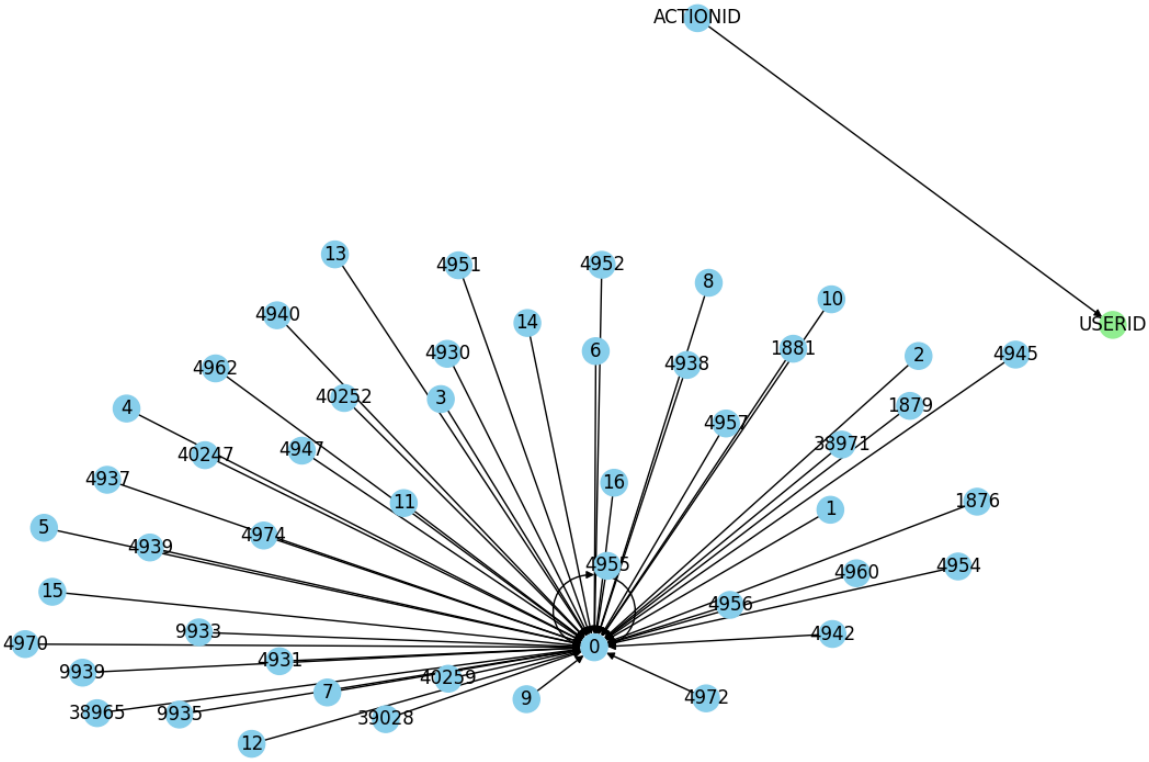
🔍 Sample Queries Executed

Description	Cypher
Show small portion of graph	`MATCH (u:User)-[r:PERFORMS]->(t:Target) RETURN ... LIMIT 25`
Count all users	`MATCH (u:User) RETURN COUNT(u)`
Count all targets	`MATCH (t:Target) RETURN COUNT(t)`
Count all actions	`MATCH ()-[r:PERFORMS]->() RETURN COUNT(r)`
Actions/targets of specific user	`MATCH (u:User {id: ...})-[r:PERFORMS]->(t:Target) RETURN ...`
Count actions per user	`MATCH (u:User)-[r:PERFORMS]->() RETURN ...`
Count users per target	`MATCH ()-[r:PERFORMS]->(t:Target) RETURN ...`
Average actions per user	`MATCH (u:User)-[r:PERFORMS]->() ... RETURN AVG(action_count)`
Users/targets with positive Feature2	`MATCH (u:User)-[r:PERFORMS]->(t:Target) WHERE r.feature2 > 0`
Actions with label=1 per target	`MATCH ()-[r:PERFORMS]->(t:Target) WHERE r.label = 1 ...`

➡️ Full list is available in `cypher_queries.txt`

📊 Graph Visualization

A sample of 50 `(User)-[:PERFORMS]->(Target)` relationships was visualized using NetworkX.



- **Blue nodes**: Users
- **Green nodes**: Targets
- **Directed edges**: Actions performed

Environment & Dependencies

All dependencies are listed in `requirements.txt`. Key packages:

- `pandas`, `numpy`, `neo4j`, `networkx`, `matplotlib`, `tqdm`

To reproduce the environment:

```
pip install -r requirements.txt
```

Key Outcomes

- ~250k actions ingested into Neo4j
- Cypher queries automated and saved
- Interactive graph visualization created
- Schema successfully mapped: user–action–target

Notes

- CSVs must be manually placed in Neo4j Desktop's `import/` folder.
- The database reset includes a 30-second wait time to avoid premature connections.
- Ensure Neo4j Desktop is **running** before executing Python scripts.

Potential Extensions

- Add time-based exploration (e.g., activity per month)
- Community detection on users via target overlap
- Feature-based link prediction or graph embeddings

Contact

Nikos Mitsakis
AUEB Informatics, Big Data Systems (2024)