Graph Databases

Using Neo4j and Amazon SNAP Dataset

Database Management Systems Laboratory (CS39202)

Team CheeseCake

Shivam Choudhury

Ayush Mundada

Tanishka Rahate

Omkar Bhandare

Jadhav Udaykiran

Problem Statement

The objective of this project is to process large graphs using a graph database. The core tasks include:

i. Install any graph processing system, such as ApacheGraph, Pregel (GoldenOrb), Giraph, Neo4j, or Stanford GPS.

ii. Load a large graph from the Stanford SNAP large graph repository.

iii. Provide an interface to run simple graph queries. Bonus marks may be awarded for implementing PageRank.

iv. Profile and evaluate the performance of the system.

Background

Graph databases represent a powerful paradigm in database technology that focuses on relationships between data entities. Graph databases are designed to efficiently store, process, and query highly connected data using graph structures with nodes, edges, and properties.

They consist of three fundamental elements:

- **Nodes**: Represent entities or instances such as products, categories, or other items. They function similar to records, like in relational databases.
- **Edges**: Connect nodes, representing the relationships between entities. They can either be directed or undirected, and they often carry properties that describe the nature of the relationship.
- **Properties**: Information associated with nodes and edges that describe their characteristics and attributes.

At its core, a graph database models data in a way that mirrors real-world relationships. For example, in a social network like Twitter, each user is a node, and "follows" connections are represented as edges. This intuitive structure makes graph databases ideal for handling complex, interconnected data.

Built on graph theory principles, they abstract away mathematical complexities, offering a natural way to represent relationships. Languages like Cypher—developed for Neo4j—make it easy to express complex queries through readable syntax.

Graph databases are especially well-suited for e-commerce applications such as product recommendation systems, like the one implemented in this project. Their strength in capturing relationships makes them valuable in domains like fraud detection, social analytics, and customer behavior modeling. With flexible schemas and efficient traversal, graph databases have become a foundational technology for modern, scalable, and adaptive systems.

Objective

This project aims to create a robust graph-based product recommendation system using the Amazon SNAP dataset and Neo4j. It enhances user experience with intuitive navigation, advanced search, and data-driven recommendations, leveraging product data relationships. The project highlights graph databases as a powerful alternative to traditional recommendation systems.

To achieve this goal, the project focuses on the following key objectives:

1. **Efficient Product Discovery via Hierarchical Structures**: Implement mechanisms for seamless product discovery by utilising categorical and group-based relationships inherent in the dataset.
2. **Customised PageRank Algorithm for Product Relevance**: Develop and integrate a tailored PageRank-inspired algorithm to rank products based on relevance, popularity, and contextual importance within the graph.
3. **Advanced Multi-Category Search Functionality**: Enable flexible and precise search across multiple product categories simultaneously, allowing for complex and user-specific query resolution.
4. **Performance Monitoring Framework**: Establish a comprehensive performance evaluation and monitoring framework to ensure system scalability, reliability, and responsiveness under varying loads.
5. **User Influence Scoring Mechanism**: Introduce a user influence scoring model that quantifies the impact of user behaviour on product visibility and recommendation outcomes.
6. **Temporal Analysis of Product Trends**: Analyse temporal dynamics in product popularity and user interaction to uncover evolving trends and seasonality within the dataset.
7. **Visualisation of Product Networks**: Design interactive visualisations to repre-

sent the structure and relationships among products, categories, and user interactions within the graph.

8. **Integration of a RAG-Based Natural Language Query System**: Incorporate a Retrieval-Augmented Generation (RAG) based natural language processing interface to facilitate intuitive and conversational product discovery.

Methodology

The development of the graph-based product recommendation system follows a structured and multi-faceted approach, encompassing key stages such as data modelling, algorithm design, system implementation, and user interface development. Each phase is carefully designed to align with the overall objective of leveraging graph structures for enhanced product discovery and recommendation. The methodology ensures a coherent integration of data engineering, graph analytics, and user-centric design to deliver a scalable and effective recommendation platform.

Data Schema Design

The underlying data schema is designed to effectively represent the complex relationships among products, consumers, categories, and groups within the Amazon SNAP dataset. This schema serves as the structural backbone of the recommendation system, enabling efficient querying, relationship traversal, and algorithmic computations.

Node Types (along with the attributes)

- **Product Node**: Id, ASIN, title, salesrank, avg_rating, intrinsic_score, total_score
- **Group Node**: name
- **Category Node**: name
- **Consumer Node**: Id, Customer, name, score

Edge Types

- **BELONGS_TO:** Product Node → Category Node
- **PART_OF:** Product Node → Group Node
- **REVIEWED:** Consumer Node → Product Node
  Attributes: Date, Rating, Votes, Helpful

- **SIMILAR:** Product Node → Product Node
- **COPURCHASED_WITH:** Product Node → Product Node
  Attributes: Frequency

Indexing Strategy

Given the dataset's variety and volume of queries, implementing appropriate indexes is critical for ensuring efficient data access and optimal performance. The following indexes were established, each tailored to anticipated query patterns:

Name	Type	Entity Type	Labels or Types	Properties
`category_name_index`	RANGE	NODE	{Category}	{name}
`consumer_customer_index`	RANGE	NODE	{Consumer}	{Customer}
`group_name_index`	RANGE	NODE	{Group}	{name}
`product_asin_index`	RANGE	NODE	{Product}	{ASIN}
`product_id_index`	RANGE	NODE	{Product}	{Id}

Table 1: Indexing Strategy

Custom PageRank Algorithm

To accurately reflect the dynamics of product relevance and interconnectedness within the Amazon SNAP dataset, we designed and implemented a novel two-phase PageRank algorithm tailored specifically for graph-based product recommendation. Unlike traditional PageRank—which primarily relies on nodes' structural connectivity—our approach integrates product-specific attributes and co-purchase behaviour to derive a more context-aware and impactful relevance score.

Phase 1: Computing the Intrinsic Score

$$\text{intrinsic\_score} = (2^{\text{avg\_rating}}) \cdot \left(\frac{\log_{10}(1 + \text{sum of all helpful})}{\log_{10}(2)} \right) \tag{1}$$

This formulation provides a meaningful balance: exponential scaling of the rating gives prominence to top-rated products. In contrast, logarithmic scaling of helpful votes accounts for user trust without allowing it to dominate the score.

Phase 2: Incorporating Network Influence via Co-Purchase Relationships

$$\text{total\_score} = \text{intrinsic\_score} + 0.5 \cdot \sum \left(\frac{\text{frequency}_j}{\text{total\_frequency}} \cdot \text{total\_score}_j \right) \tag{2}$$

Note: Only co-purchased products with frequency ≥ 2 are considered in the summation.

Only co-purchase edges with frequency ≥ 2 are considered, effectively filtering out noise and incidental relationships. The final `total_score` represents a balanced combination of intrinsic product quality and network influence, with a damping factor (0.5) applied to avoid disproportionate influence from the co-purchase network.

To ensure stable convergence of scores across the network, we performed five iterative refinements of the total score calculation. This iterative process was inspired by the Newton-Raphson method, a numerical technique used for root-finding, adapted here to iteratively approximate a stable state in score propagation. This helped achieve smooth and reliable convergence, allowing the influence of network structure to be gradually incorporated without overamplifying noisy relationships.

User Influence Scoring

To assess the influence of individual reviewers within the system, we developed a custom algorithm that calculates a user-specific influence score based on the helpfulness of their reviews. This approach prioritises users who consistently contribute meaningful and helpful reviews across various products while mitigating the potential impact of outliers.

Formula

The user influence score is calculated as follows:

$$\text{score} = \sum \left(\min \left(\frac{\log_{10}(1 + \text{helpful})}{\log_{10}(2)},\ 5.0 \right) \right) \tag{3}$$

For each consumer, the algorithm computes a base score for each review using logarithmic scaling, caps this score at 5.0 to prevent exceptionally high helpfulness values from disproportionately affecting the overall influence score, and sums up these capped scores across all reviews to derive the final influence score. This method ensures that users who consistently contribute helpful reviews across different products are rewarded, while logarithmic scaling prevents a few highly helpful reviews from dominating the score.

Trending Product Analysis

We developed a system to identify trending products by analyzing review activity over a specified time frame. The algorithm ranks products based on review volume as the

primary factor and PageRank-derived score as the secondary factor. This dual-ranking system prioritizes products with a surge in reviews and high recommendation scores, offering a strong measure of recent popularity driven by customer engagement.

Formula

$$\text{TrendingScore}(p) = \text{sort}\left(\text{review\_count}_{[t_1,t_2]}(p),\ \text{total\_score}(p)\right) \tag{4}$$

where $[t_1, t_2]$ *represents the time window within which reviews are counted. Products are sorted first by review count (descending), and then by total score (descending) in case of ties.*

Multi-Category Search

We implemented a Multi-Category Search feature to refine product discovery, allowing users to filter across multiple overlapping categories. To simplify category selection, especially with uncertain spelling or phrasing, we integrated fuzzy matching using TheFuzz library, enabling users to search with approximate or partial terms.

RAG-Based Chatbot Implementation

We built a RAG-based chatbot for enhanced product discovery using embeddings and similarity search. The 'BAAI/beg-small-en-v1.5' model generates embeddings for product titles, groups, and categories, which are indexed with FAISS for fast similarity search. The system translates user queries into embeddings, searches for relevant products, and returns top matches with a similarity threshold of 0.3, ensuring accurate recommendations.

The RAG engine integrates multiple components to process user queries efficiently:

- **Text Cleaning and Preprocessing:** User input is first cleaned using the `cleantext` library, removing noise such as URLs, emails, and unnecessary punctuation. A stopword filtering process is also applied to ensure that only the most relevant terms remain in the query.
- **Embedding-Based Search:** After cleaning the query, it is passed through an embedding engine to obtain a vector representation. The system then searches for similar products, groups, and categories and ranks them based on their cosine similarity to the query vector.

- **Results Filtering and Response Generation:** The top matching products, categories, and groups are retrieved and enriched with additional product details (e.g., ASIN, title, sales rank). A well-structured response is generated to ensure an engaging user experience, highlighting the most relevant results.

Performance Monitoring Framework

We have integrated a comprehensive performance monitoring framework to ensure optimal system efficiency and transparency across our application. This framework provides consistent and real-time tracking of key system metrics for every route and query execution.

Key Metrics Tracked

- **Query Execution Time (in seconds):** Measures the duration to execute individual route handlers and database interactions.
- **CPU Usage (percentage):** Captures the CPU consumption during processing.
- **Memory Consumption (in megabytes):** Tracks the memory footprint before and after route execution to detect potential memory leaks or spikes.

This monitoring system was implemented using a custom decorator, `@log_performance`, which wraps all relevant route handler functions. This approach ensures minimal intrusion into the application logic while enabling uniform performance tracking across the board.

User Interface Components

The user interface has been thoughtfully designed to provide an intuitive, flexible, and engaging browsing experience across multiple dataset dimensions. Below is a detailed overview of the key components:

- **Home Page:** Displays all available Groups, serving as the starting point for product exploration.
- **Products by Group:** Lists products under a selected Group, sorted by total score, with options to sort by title, avg rating, or sales rank, plus filters like minimum rating.

- **Products by Categories:** Shows products in a selected Category, default sorted by total score with interactive sorting and filtering.
- **Products by Multiple Categories:** Enables product discovery across multiple categories with fuzzy-matching search and sorting/filtering options.
- **Product Detail Page:** Provides detailed product info, including categories, co-purchased/similar items, and user reviews.
- **Top Reviewers Page:** Ranks reviewers based on a custom scoring algorithm to highlight trusted voices.
- **Trending Products Page:** Displays trending products determined by review activity or popularity spikes.
- **User Reviews Page:** Showcases helpful reviews from top reviewers, ranked by helpful count for informed decision-making.

Query Complexity Analysis

Let the number of nodes and edges of defined types be as follows:

Node Types:	Category = N, Consumer = M, Group = G, Product = P
Edge Types:	BELONGS_TO = B, COPURCHASED_WITH = C, PART_OF = X, REVIEWED = R, SIMILAR = S

Finding all Groups (sorted): $O(G)$

Products in the same Group (sorted): $O(1 + X + P \cdot \log P)$

Products in the same Category (sorted): $O(1 + B + P \cdot \log P)$

Products in multiple selected Categories: $O(N \cdot P + P \cdot \log P)$

Trending Products: $O(R + P \cdot \log P)$

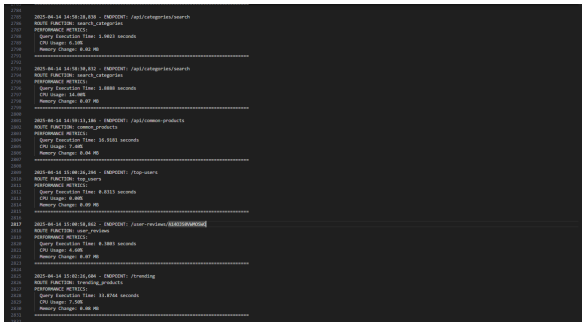
Top Reviewers (with score calculation): $O(M \cdot \log M + R + M)$

PageRank score calculation: $O(P + R + P + C) = O(P + R + C)$

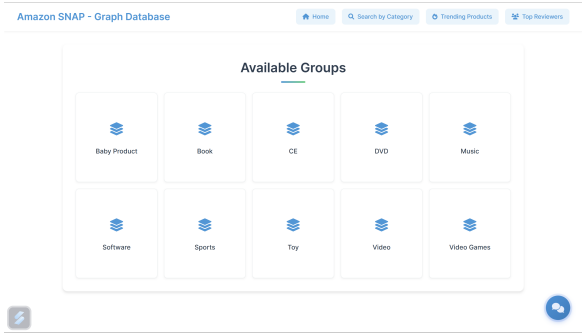
Reviews of a selected top Reviewer: $O(R + R \cdot \log R)$

Selected Product data (all the linked info): $O(1+N+S+S \cdot \log S+C+C \cdot \log C+R + R \cdot \log R)$

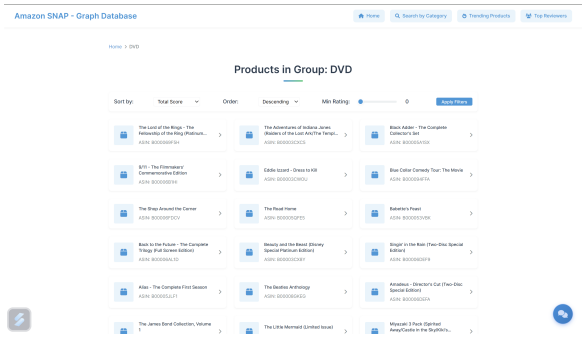
Screenshots of the Demonstration



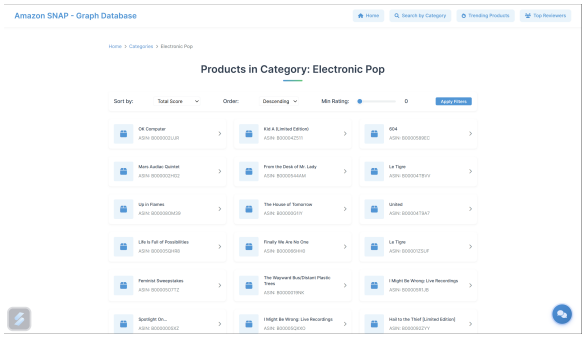
Performance Log Entries



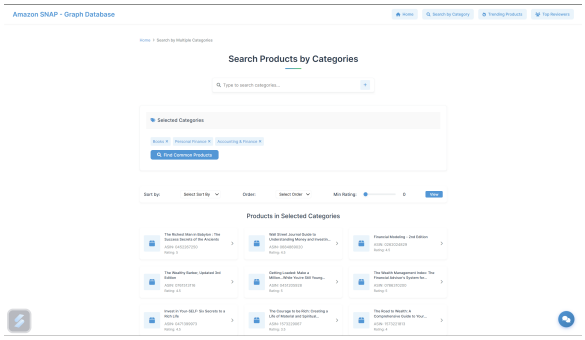
Home Page



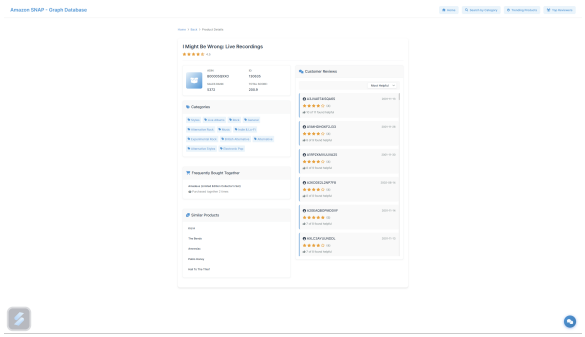
Products in selected Group



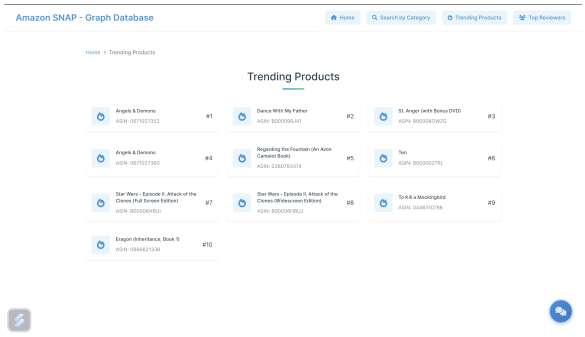
Products in selected Category



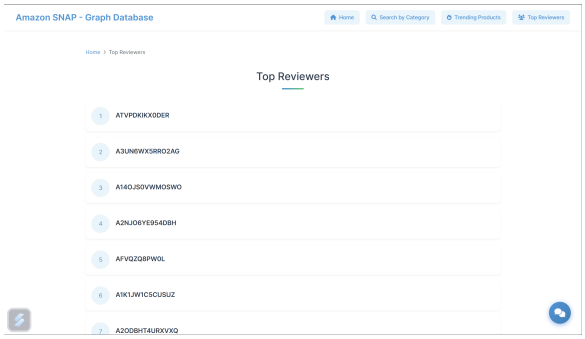
Products in selected Categories



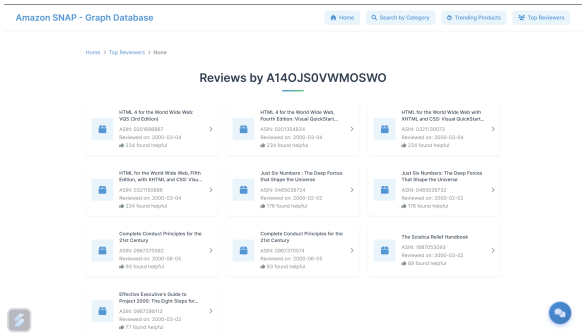
Product Details



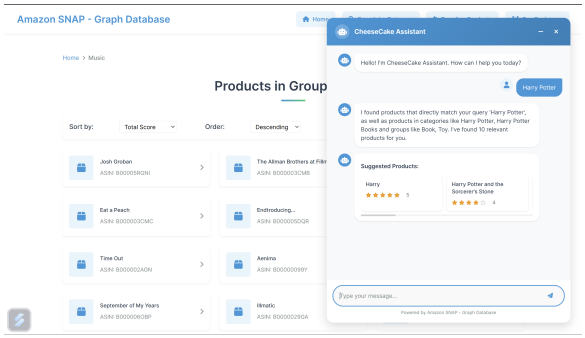
Trending Products



Top Reviewers



Best Reviewed Products of Top Reviewers



Integrated RAG-based ChatBox

Future Scope

1. Advanced Hybrid Recommendation Techniques

-Add knowledge graph reasoning for explainable recommendations.

-Integrate real-time session-based recommendations using streaming graph updates.

2. Enhanced Graph Learning

-Implement Graph Neural Networks (GNNs) for dynamic relationship modelling.

-Add temporal graph networks for dynamic tracking of evolving product popularity.

-Explore heterogeneous graph transformers for multi-modal data.

3. Scalability Improvements

-Implement Graph Partitioning for distributed Neo4j clusters.

-Add incremental graph updates for real-time score recalculations.

-Explore vector-graph hybrid search combining FAISS with Cypher.

4. RAG System Enhancements

-Integrate multi-modal embeddings.

-Add conversational memory for chat-based recommendations.

-Implement query intent analysis using graph-aware LLMs.

5. Social Network Integration

-Add social graph features for friend-based recommendations.

-Track cross-product network effects through user communities.

References

https://aws.amazon.com/nosql/graph/

https://neo4j.com/docs/getting-started/

https://neo4j.com/docs/getting-started/appendix/tutorials/guide-cypher-basics/