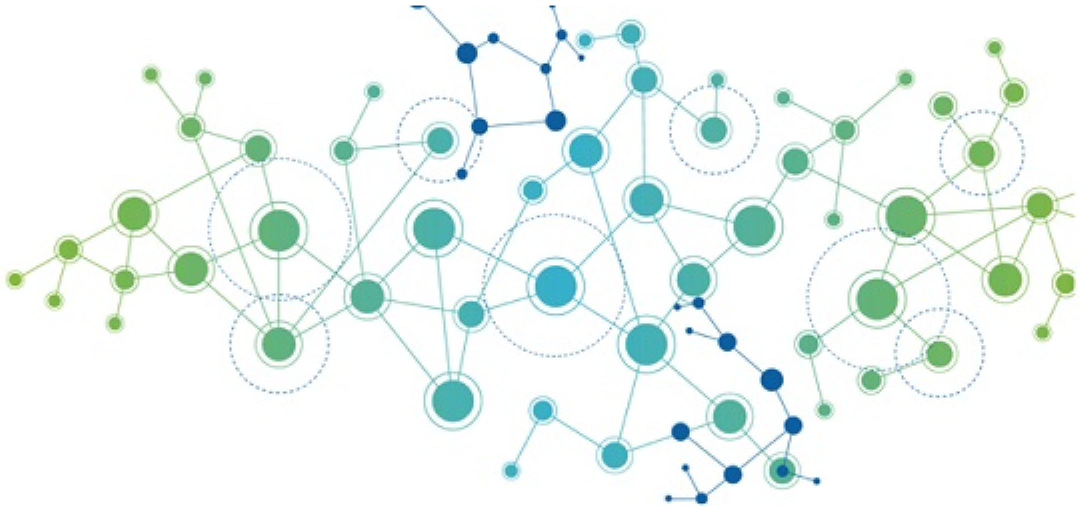
SRID-Report of Intern Shaurya Jayprakash Sinha, Cloud Dev Group

GRAPH DATABASE

Social Media Friend Recommender



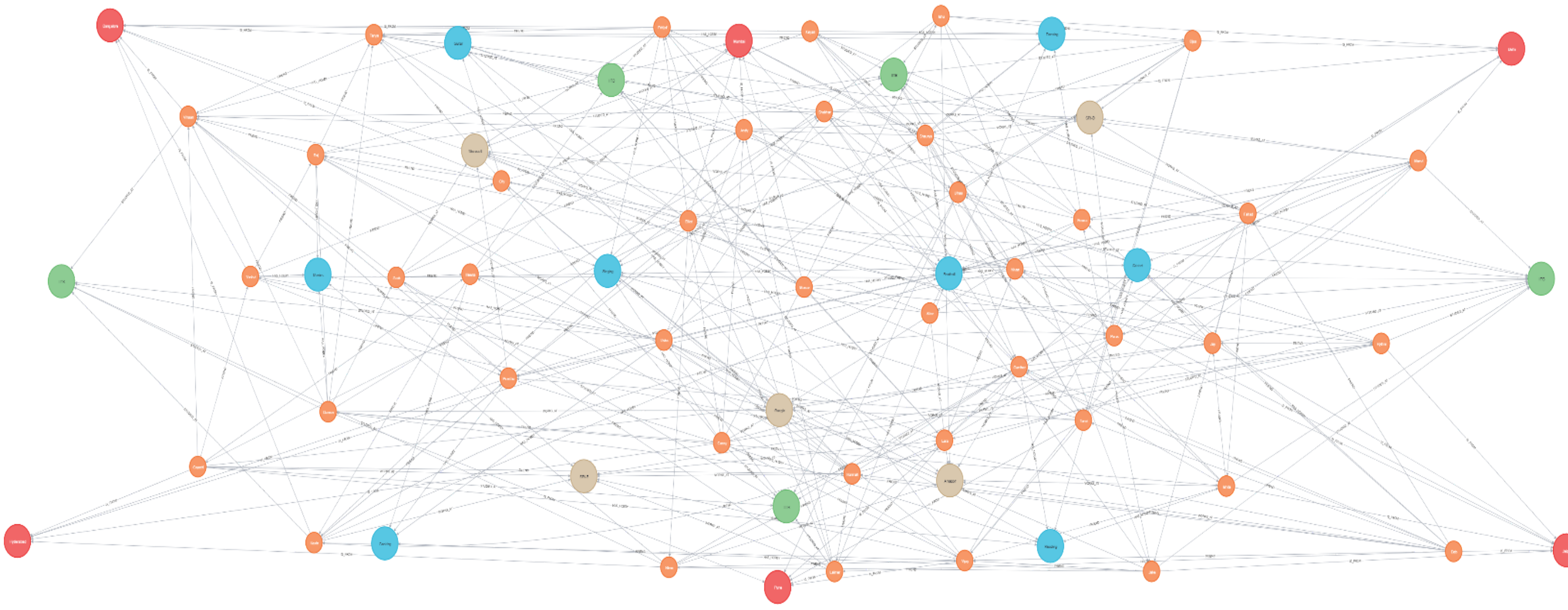
Introduction

The focus of this report is a **friend recommendation system** for a generic social media use case. The Report describes a development of a friend recommender system using techniques based on users' *mutual friendships*, common attributes such as *Workplace College, Hobbies* and *Place of Belonging* by combining a **weighted average** of all these factors. The final recommender system is implemented using the **Neo4j** graph database using the **Cypher** Query Language .

Problem-Use Case Definition

There are many USERs in our social media environment. They are represented as nodes. Besides this, we have various other types of nodes: WORKPLACE, NATIVEPLACE, HOBBIES and COLLEGE.

USER nodes have the relationship FRIEND with other USER nodes
USER node has relationship WORKS_AT with WORKPLACE node.
USER node has relationship STUDIED_AT with COLLEGE node.
USER node has relationship IS_FROM with NATIVEPLACE node.
USER node has relationship HAS_HOBBY with HOBBIES node.



Graph Representation of the entire Database

USER

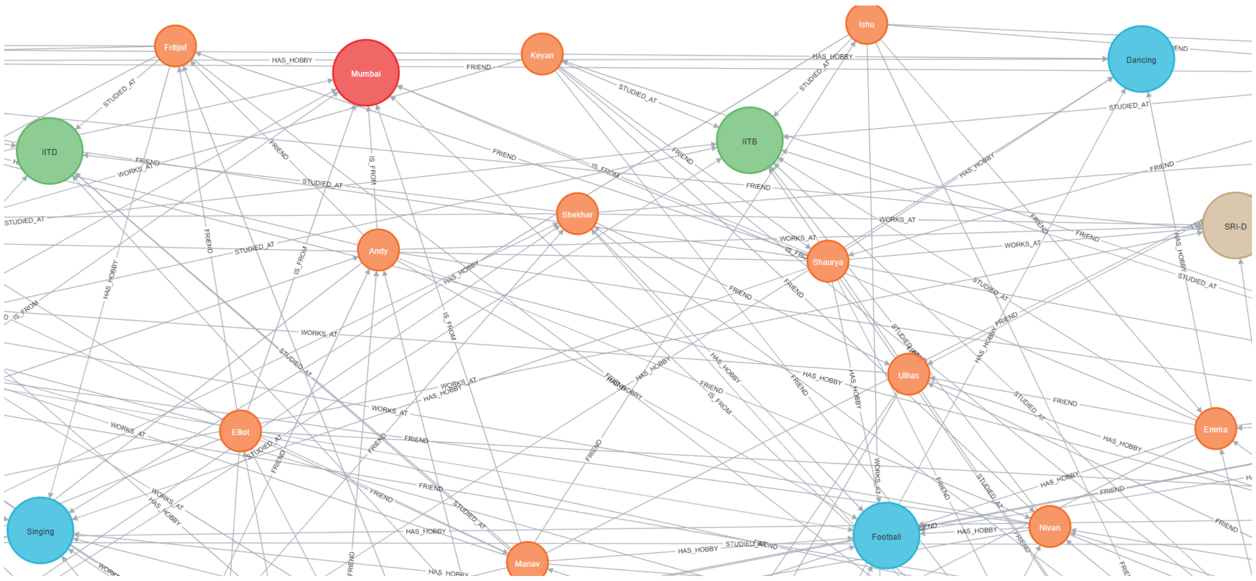
HOBBIES

COLLEGE

WORKPLACE

NATIVEPLACE

Enlarged image of the above Graph representation is in the following page.



Enlarged image of the above Graph representation

ALGORITHM

The most common approach is a recommendation based on friends of friends. The algorithm recommends users who are common to another user. The Cypher query in source code 1.1 selects friends of friends and sorts them by their number of occurrence; the most mutual friends of friends are on the top. The query selects the first 10 users. The algorithm skips users who are already friends with the given user **a** or if it is **a** itself.

```
MATCH (a:User {id: {actorId}})
-[:HAS_FRIEND]-(:User)
-[:HAS_FRIEND]-(b:User)
WHERE NOT (a)-[:HAS_FRIEND]-(b) AND a <> b
RETURN DISTINCT b.id AS user_id, count(b) AS score
ORDER BY count(b) DESC
LIMIT 10
```

Code 1.1: Friends of Friends selected and sorted according to frequency of occurrence

Similarly we can select 2 users having the same College, Workplace, Nativeplace or Hobby. Each of these can come out to be either 1 or 0, depending on whether they have the common attribute, or they don't.

The codes for each of the cases is as follows:

```
MATCH (a:USER)-[:HAS_HOBBY]-(:HOBBY)-[:HAS_HOBBY]-(b:USER)
WHERE a.name="Gunther" AND a<>b AND NOT (a)-[:FRIEND]-(b)
RETURN DISTINCT b.name as Recommended
LIMIT 10;
```
Code 1.2: User having common Hobby gets returned as a score of 1 else 0

```
MATCH (a:USER)-[:WORKS_AT]-(:WORKPLACE)-[:WORKS_AT]-(b:USER)
WHERE a.name="Gunther" AND a<>b AND NOT (a)-[:FRIEND]-(b)
RETURN DISTINCT b.name as Recommended
LIMIT 10;
```
Code 1.3: User having common Workplace gets returned as a score of 1 else 0

```
MATCH (a:USER)-[:IS_FROM]-(:NATIVEPLACE)-[:IS_FROM]-(b:USER)
WHERE a.name="Gunther" AND a<>b AND NOT (a)-[:FRIEND]-(b)
RETURN DISTINCT b.name as Recommended
LIMIT 10;
```
Code 1.4: User having common Nativeplace gets returned as a score of 1 else 0

```
MATCH (a:USER)-[:STUDIED_AT]-(:COLLEGE)-[:STUDIED_AT]-(b:USER)
WHERE a.name="Gunther" AND a<>b AND NOT (a)-[:FRIEND]-(b)
RETURN DISTINCT b.name as Recommended
LIMIT 10;
```
Code 1.5: User having common College gets returned as a score of 1 else 0

We then combine all the above queries into 1 single query and perform weighted average on the scores. This way we can give priority to certain relations over others and obtain a final score.

```
MATCH (a:USER),
(a)-[rel:FRIEND|HAS_HOBBY|IS_FROM|WORKS_AT|STUDIED_AT]-()-[]-(b:USER)
WHERE a.name="Shaurya" AND NOT (a)-[:FRIEND]-(b) AND a <> b
WITH DISTINCT b.name AS Recommendations, CASE
WHEN type(rel) = "FRIEND" THEN count(rel)*0.2
WHEN type(rel) = "HAS_HOBBY" THEN count(rel)*0.1
WHEN type(rel) = "IS_FROM" THEN count(rel)*0.2
WHEN type(rel) = "WORKS_AT" THEN count(rel)*0.3
WHEN type(rel) = "STUDIED_AT" THEN count(rel)*0.2
END AS score
WITH DISTINCT Recommendations, sum(score) as score
RETURN Recommendations, score
ORDER BY score DESC
LIMIT 10;
```
Code 1.6: The actual algorithm used, which combines all the above 5 codes through weights

OUTPUT

The resulting output is a table with ‘Recommendations’ against a decimal value ‘score’ sorted in the descending order of the score, which means higher up names are to be suggested first by the recommendation system. We limit the number of results to 10 to avoid 0 score results in the display.

Recommended	score
"Manvi"	0.9
"Raj"	0.7
"Manav"	0.7
"Fahad"	0.6
"Shekhar"	0.6
"Andy"	0.6
"Heena"	0.5
"Paras"	0.5
"Damon"	0.4
"Preetha"	0.4

Result 1.1: Users Recommended for Query in Code 1.6 by the Recommender for the User named ‘Shaurya‘ with the scores as an indicator of likelihood of successful recommendation.(not a percentage, but a numeric metric which might exceed 1)

Further improvements:

- Other measures such as measure of centrality- betweenness and closedness can be used to further make the algorithm richer
- We can incorporate the scores of the neighborhood while calculating the score of the target node with respect to a given node. This will be a non ending iterative process and will be more costly operation. For Huge Graphs with many nodes, this can be an effective strategy.
- The distance calculated here was taken to be 1. If we also consider other distances like 2 3 , we can make a complex algorithm, with very slight improvement in recommendations.
- All missing link prediction algorithms for graph should work well for this problem as we can generalise this use case in that manner.

What other fields can Graph Databases can be used.

1. ***Money Laundering***: By detecting zero sum Loops in the graph, can flag potential money laundering cases and sift doubtful cases for manual scrutiny.
2. ***Linkedin***: LinkedIn can show all your 1st, 2nd, and 3rd -degree connections, and the mutual contacts with your 2nd level contacts in real-time. The answer is: because LinkedIn organizes its entire contact network of 660+ million users with a graph!
3. ***Fraud and anomalies****:* there might be many e-commerce purchases from different accounts -- but all from the same IP address or cluster of IP addresses. Or there may be several cash withdrawals for the same amount of money from the same neighborhood, followed by cash deposits the same day to different accounts.
4. ***Real-Time Recommendation Engines***: Real-time product and ecommerce recommendations provide a better user experience while maximizing profitability. Notable cases include Netflix, eBay, and Walmart.
5. ***Routing:*** Information travels through a network by finding optimal paths makes graph databases the perfect choice for routing
6. ***Context-aware services.*** Graphs help provide services related to natural disaster warnings, traffic updates, or product recommendations and for a given location, graph databases offer a logical solution to real-life circumstances.

My Take-Away from this Project

Graph database was a new field for me to explore. It is a top notch skill in todays world, as can be seen from the above fields of use. I learnt many things through this experience,

- Graph databases are highly Relational Databases. They are quicker than normal Relational Databases on Queries that involve multiple joins in RDBMS because joins are quite expensive operations, and in Graph database, these are simple edge traversals. Graph algorithms can be efficiently run on Graph databases whereas the same is not true for simple RDBMS. At the same time, it is Worth noting that Graph database might not perform better on each and every kind of query, some places simple RDBMS might be better.
  Graph database shows us how the method of storage of data is so important in performance of the database.
- The various fields in which graph databases can be used, on which I plan to further research on my own. I also plan to explore other tools like Amazon Neptune (AWS) and check out GQL(Graph Query Language).
- I am comfortable with Neo4j and the corresponding Cypher Query Language, Thanks to this experience.
- I gained multiple insights on theory and practical aspects with the Help of my Mentor, *Deepak Soni*. I would like to express my gratitude for this smooth and highly enjoyable, yet knowledgeable experience of Interning at SRID.