# Efficient Query Retrieval from Social Data in Neo4j using LIndex

**Anita Brigit Mathew**
Department of Computer Science
National Institute of Technology, Calicut, India 679603
[e-mail: anita_p120024cs@nitc.ac.in]

*Received April 8, 2017; revised November 10, 2017; accepted January 13, 2018; published May 31, 2018*

## *Abstract*

The unstructured and semi-structured big data in social network poses new challenges in query retrieval. This requirement needs to be met by introducing quality retrieval time measures like indexing. Due to the huge volume of data storage, there originate the need for efficient index algorithms to promote query processing. However, conventional algorithms fail to index the huge amount of frequently obtained information in real time and fall short of providing scalable indexing service. In this paper, a new LIndex algorithm, which is a heuristic on Lucene is built on Neo4jHA architecture that holds the social network Big data. LIndex is a flexible and simplified adaptive indexing scheme that ascendancy decomposed shortest paths around term neighbors as basic indexing unit. This newfangled index proves to be effectual in query space pruning of graph database Neo4j, scalable in index construction and deployment. A graph query is processed and optimized beyond the traditional Lucene in a time-based manner to a more efficient path method in LIndex. This advanced algorithm significantly reduces query fetch without compromising the quality of results in time. The experiments are conducted to confirm the efficiency of the proposed query retrieval in Neo4j graph NoSQL database.

***Keywords:*** Database, data processing, indexing, information retrieval, social networks

## 1. Introduction

Blooming size and variety of relational data in social networks have inspired extensive research in supporting effective query retrieval methods. Neo4j, which is a graph NoSQL database, is used to store the unstructured social network big data in a structured form, thereby creating a route to process the queries. Current work has studied the problem of query retrieval in Neo4j graph NoSQL database connected via social network like Facebook or Twitter. Social network dataset like FIFA 2015, Ego Network, Northwind, Social Circle, Movie dataset, Galaxy, and Library are selected for data storage in Neo4j. Due to the scarcity of graph indexing mechanisms in Neo4j [1-3] and cost-efficacious query optimizers, it becomes difficult, otherwise impossible to query and analyze any social network data [4]. Therefore, there is a growing demand and strong motivation to take advantage of well-studied database index schemes and query optimization techniques to address the graph query problem in Neo4j when used by any social network scenario. Neo4j graph NoSQL database stores data in the form of records. Each of these records has a specific structure, which is detailed later in this paper. During query retrieval, data is driven from tables stored in Neo4j. Due to the limitation of record level index and Skip List index [5], this paper presents a heuristic on Lucene called LIndex, which is resolved for efficient query retrieval from Neo4j connected to a social network.

Skip List [6] index algorithm partitions the terms in a hierarchical tree structure, and organizes the leaf partitions using a List-based distributed data structure. This underlying Skip List, only supports one-dimensional term related to files in Neo4j-NSR. LIndex algorithm provides an efficient approach, where the user gets specific about the desired level of query accuracy. The drawback of Skip List approach was due to time overhead in query caused by the existence of too many duplicate entries. This is resolved by applying the heuristic Lucene. LIndex technique is derived incorporating Lucene index with heuristic. LIndex maintains Neo4j-NSR structure for each vertex of the social network. Vector space model [7,8] an effective tool for information retrieval is analyzed further and incorporated for efficient query retrieval in Neo4j (heuristic based Lucene) called LIndex. LIndex mainly focus on query over large data graph G. Given a query, how a resultant graph can be engendered from a large graph? For this, there is a need to reduce the search space solemnly, by transforming the vertices into points in vector space using graph-embedding techniques. These techniques help to transfigure the query into a distance based multi-way joins over the converted vector space.

The rest of the paper is organized as follows: Section 2 summarizes the preliminaries of query retrieval, storage of data in Neo4j graph database with pictorial representation of Neo4j architecture. Section 3 reviews on Problem definition. Search algorithms for colossal social network with different classes of data and theoretical view of LIndex are illustrated in section 4. Section 5 talks about experimental implementation conducted on the proposed model using different database dataset. The conclusion of the proposed work and the future enhancements to be carried out are presented in section 6.

## 2.Related work

Many researchers have studied the problems of query retrieval over relational databases. Sabina Alkire et al. [9] suggest a Multidimensional Poverty Index (MPI) on social media data. This index reflects on the depreciation of search procedure thereby increasing query performance. The MPI deeply constrained to single keyword data like names of people, countries and things. Here each dimension keyword is equally weighted still there is a lack of multi-keyword approach. How to query for keywords in a network, which is connected to data sources are covered by Xiaomeng Yi et al. [10]. They present two case studies of real BigData applications of social networks where there is a demand for faster and efficient search. Kim et al. [11] focus on the management of very large graphs such as social networks. Initially the paper explains main representations used to store the graph (dense/sparse native graphs, triple storage or relational layouts), the access patterns and typical queries considered (reachability or neighborhood queries, updates various reads, transactional requirements and graph consistency models). Later they describe how to map the data processing models to solid graph management data systems, highlighting target application domains, implementation techniques, scalability and workload requirements. A. B. Atkinson et al. [12, 13] reveals how multidimensional framework can be applied to social network data. This is a costly approach and it takes a lot of time during computation. Arijit Khan et al. [14] propose a neighborhood-based similarity measure to avoid costly graph isomorphism and edit distance computation in graph data management. Based on this they found an information propagation model that converts a huge network into groups of multidimensional vectors in which primary indexing and similarity-based search algorithms are available. The proposed method, called Ness (Neighborhood Based Similarity Search), can only be used for graphs with low automorphism and few edges between entities. This method cannot be used for social and network data interconnected via links. Lee et al. [15,16] suggests a new query engine based on algebra of operations on sets of key-value pairs. The new query engine amalgamates some regular relational database operations with some extensions oriented to collection processing and complex graph queries. They study the query plans of graph queries expressed in the new algebra, and find that most graph operations that can be efficiently expressed as semi-join programs. The above-mentioned works does not consider how query retrieval is performed in social network efficiently.

## 3. Preliminaries

Before advancing further, this section furnishes a brief preface about why query retrieval on social network data in Neo4j graph NoSQL database. Furthermore, how the data is stored in Neo4j by establishing a master-slave setup within the social network, followed by the mode of processing the data stored.

### 3.1. Query Retrieval

Social networks [16-19] like Facebook and Twitter manage a query as a term and update the results in real time basis. Any query with multiple terms like different posts, tweets etc. need to be yet considered. There is need for efficient query retrieval algorithm considering time dimension. After studying the users query retrieval etiquette in social networks, the need for sophisticated query schemes are found to be very essential.

Let us consider social data of FIFA 2015: this is defined to be the data arising in the angle of World Cup that includes both semi-structural and unstructured information generated by the users. FIFA 2015 continuously generates an immense mass of social data that encompass refurbish which are associated with the nodes or the edges of the network. This work aims at ingesting, managing and querying at a faster pace on such continuously generated real-time data of FIFA 2015 and other datasets by using efficient index mechanisms and reducing the indexing cost. To make the discourse solid and formal, let $G_t$ symbolize the underlying social graph of the user at time t, with $N_t$ and $E_t$ signify the set of nodes and edges at time t respectively. In general, $G_t$ is a heterogeneous social network, a multi-relational directed graph that contains nodes representing the user and its followers, properties of users and their relationships. Similarly, $E_t$ includes relationships. The information analogous to the nodes and edges can be apprehended through a set of attribute-value pairs associated with them. For this we have taken Neo4j graph database and why Neo4j is illustrated in our previous papers [2-3]. A query processing task is carried out using Cypher Query Language, which is the query language used by Neo4j. How this information is stored in the Neo4j graph database is explained in next section Data Storage.

### 3.2. Neo4j Data Storage

Neo4j is a reliable graph database that offers a disc-based, utterly optimized graph storage with high performance [20-23]. The architecture has a persistent store as a traditional database and an object cache. The internal model of data storage in Neo4j through CQL queries, start with transaction management, which plays a major role in accommodating the data to Neo4j. Data of any social network in Neo4j is modeled as a graph $G_D$(1) in case of offline datasets stored in Neo4j graph database and (2) in case of 2015 FIFA dataset because on a real-time basis the data arrives.

$$G_D = (V_D, E_D, I_D) \tag{1}$$

$$G_{Dt} = (V_{Dt}, E_{Dt}, I_{Dt}) \tag{2}$$

$V_D$ or $V_{Dt}$denotes the vertex set of nodes in $G_D$ as (3)

$$U_{i=1}^{\alpha} F_i \times (r_{ij})_{j=1}^{\beta} \tag{3}$$

In **Fig. 1**. Master-Slave Framework of Neo4j with Thread Manager and $E_D$ or $E_{Dt} \subseteq$ V×V represent the edge set of the graph database Neo4j. The symbol $l_{Dt}$ or $l_D$: E→STR indicates the label associated between the edges to show the relationships between nodes. STR represents any arbitrary string or numeral. In Neo4j graph NoSQL database D, data is stored in the form of files as represented in (4).Each of these files is a collection of records denoted as (5). These records contain fields indicated in (6).

$$D = (F_1, F_2 ... F_\alpha) = (F_i)_{i=1}^{\alpha} \tag{4}$$

$$F_i = (r_{i1}, r_{i2} ... r_{i\beta(i)}) = (r_{ij})_{j=1}^{\beta} \tag{5}$$

$$r_{ij} = (f_{ij1}, f_{ij2} \cdots f_{ij\gamma(i,j)}) = (f_{ijk})_{k=1}^{\gamma(i,j)} \tag{6}$$

Whenever a data arrives or data is driven from the dataset, in both the cases data has to be stored based on file-record field structure in neo4j database. After the storage of the data in Neo4j,NSR structure is called and committed. Nodes are kept in a file called NSR whose position is determined by the node name identifier. This node name identifier is the name associated to each file as shown in (4) in Neo4j database.

### 3.3 Data Processing in Neo4j

Processing of query is accomplished by traversal framework in Neo4j [24-25]. The best way in the selection of start nodes for traversals depends on the first term questioned in CQL of Neo4j. After structured storage in Neo4j, there comes the issue of efficient indexing schemes to speed-up query process. Neo4j by default uses a sequential store model that can be considered as a primary index structure on the record storage disc of Neo4j. This primary index is invoked by transaction management via the traversal framework of Neo4j. The sequential structure primary index takes specific approximately (7) time to traverse the required data from a machine.Each of these parameters is explained in (1) to (6).

$$O((f_{ijk})_{k=1}^{\gamma(i,j)} U(r_{ij})_{j=1}^{\beta} U(F_i)_{i=1}^{\alpha}) \tag{7}$$

This steered to the need of efficient index mechanism, in the subject to this Skip List was introduced. This index was based on labels that indicated file names constituting node names in Neo4j represented in (4). Skip List [4,5,20,26] indexing scheme was incorporated in transaction management so that it could skip multiple terms and only select the vital terms that exactly match the input query information. But since our social network dataset size was too large, it failed to accommodate in Neo4j single architecture hence Neo4j HA enterprise edition [27,28] was introduced. Due to indexing in a distributed architecture Skip List performance downgraded because initially only file names (e.g. NSR) was indexed and corresponding to the filename records and labels are retrieved in a sequential manner. Due to this data structure model of indexing it became inappropriate in Neo4j distributed HA architecture, as a result a ,new indexing scheme Lucene had to be integrated. To execute the query from Neo4j HA architecture after storing across multiple adjacent machine slaves is a tedious task to be performed. HA transaction management of Neo4j controls the store and fetch of requested data through CQL

Each HA transaction management is associated with a thread manager. The primary role of a thread manager is resource management (retrieve data), tracking availability of query term in which machine. The traversal from one machine to another through master is controlled by thread manager. All slaves send a heartbeat message to the master via thread manager every three seconds to indicate they are alive with the data stored. If the master does not receive a heartbeat from a precise slave for 10 seconds, then it acknowledges that slave node to be dead or diseased or out of service and then it initiates the thread manager to connect some other slave node and move data from departed slave to the new alive slave node. Once when a query comes for retrieval, transaction management of HA master translate query using cypher queryparser. It checks the object cache and file system cache whether the query comes under the last 59 committed transaction or not. If so it takes the data from the cache, else it transmits the call to traversal API. The traversal process is illustrated in detail by the same authors in [26]. If the data could not be obtained from the master, the thread manager of master contacts the thread manager of the adjacent client and checks for the query term. The same process of master continues for the client. The process terminates once the data term searched is obtained or once all the clients are traversed. The above-stated process is graphically shown in **Fig. 1**.

When a social network user data is taken for storage using CQL, the transaction management invokes cypher parser and translates the query, followed by which the transaction management calls record file and creates record level sequential index [20,26] for the newly stored data to disc after an invocation is given to file system cache. Hence when it comes to ‘*m*’ machines, the overall time will be (8).

$$O(m*(f_{ijk})_{k=1}^{\gamma(i,j)} U(r_{ij})_{j=1}^{\beta} U(F_i)_{i=1}^{\alpha}) \tag{8}$$

This deals query Q over Neo4j graph NoSQL database D after index creation on transaction management before going to the direct control over. The solution defines *Q* to be a set of tuples r record files. From database *D* data is retrieved after Join/AND CQL semantics process. The tuples are translated using CQL in *Q*. For example, given a query *Q* over the database, a possible answer $q1 \rightarrow t1$ (mapping equals), which contains “product” in *t1* and “product” in *q1*, and answer $t1 \rightarrow q1$ is acceptable if answers are same. Lucene index framework is used over

Skip List as the indexing scheme because even though it has a space complexity given by (9) in storing data in the form of files, where files are indexed.

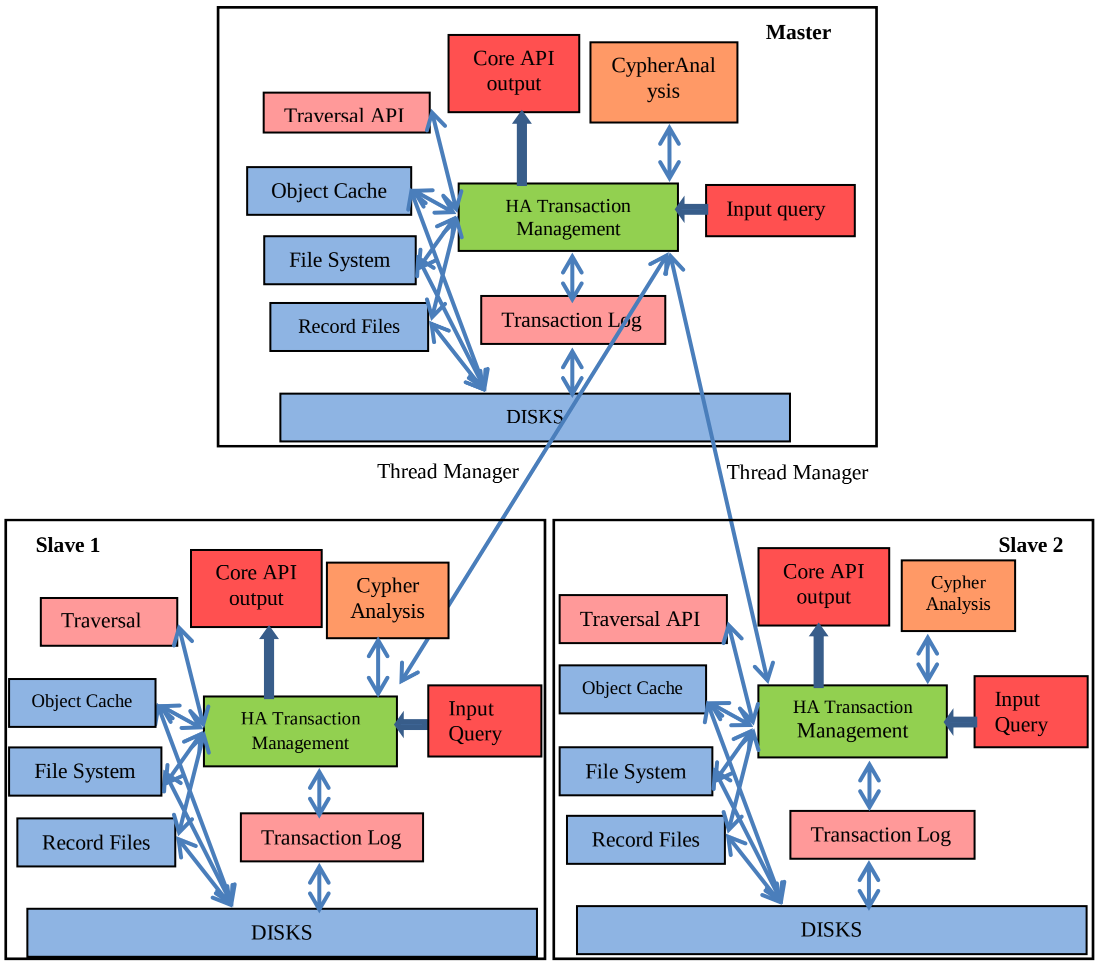


**Fig. 1.** Master-slave Framework of Neo4j with Thread Manager

SkipList in worst case takes a complexity of (10) and (11) for accessing and storing the files, this needed to be improved in Neo4j. Hence Doug Cutting Lucene Inverted Index [29] technique was used, and moreover Lucene Inverted Index helps for a search in distributed architecture where the data in the graph database can be shared and indexed across multiple adjacent data nodes.

$$O((f_{ijk})_{k=1}^{\gamma(i,j)} U(r_{ij})_{j=1}^{\beta} U(F_i)_{i=1}^{\alpha}) \tag{9}$$

$$O((\alpha \log \alpha) U(r_{ij})_{j=1}^{\beta} (f_{ijk})_{k=1}^{\gamma(i,j)}) \tag{10}$$

$$O((f_{ijk})_{k=1}^{\gamma(i,j)} U(r_{ij})_{j=1}^{\beta} U(F_i)_{i=1}^{\alpha}) \tag{11}$$

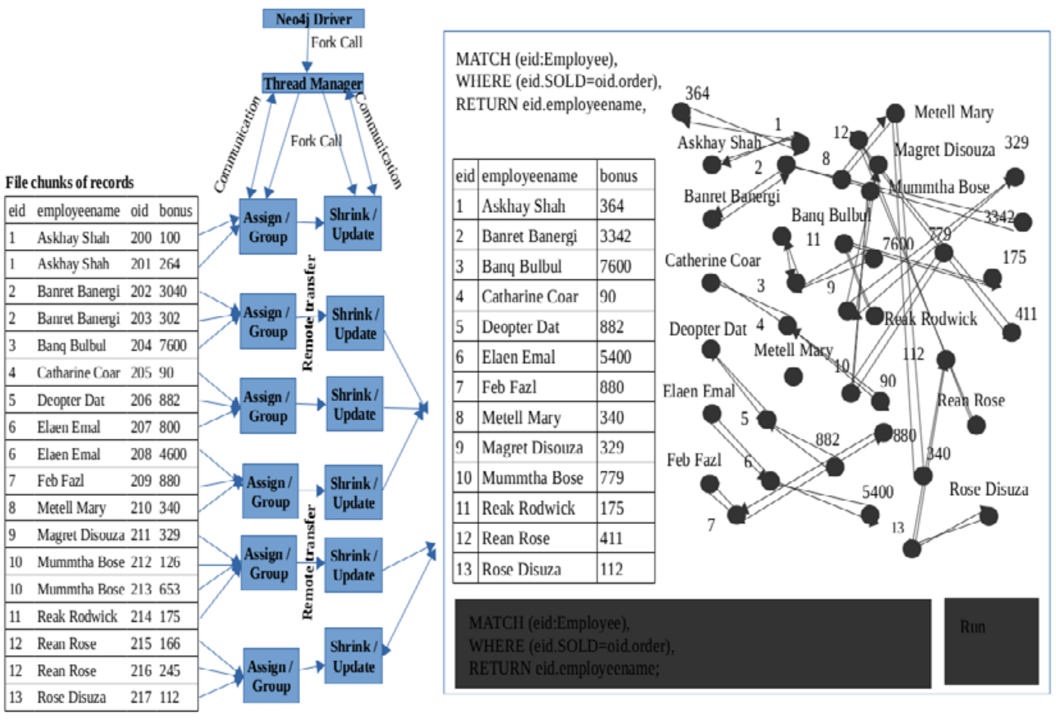
## 3.4. Example

Here how data processing can be done with the proposed LIndex in Northwind Database in Neo4j is detailed with an example. The assign, grouping of the associative list, shrinking and updating modules working is shown in **Fig. 3**. Out of the many files, few files like employee, session, and order are selected based on the multi-keyword query to be processed. Each file is associated with a key and each record in the file is assigned with the key. If any two record name matches then same keys are assigned by the grouping of associative list phase. Consider a query "List the bonus obtained by the employees". This query is first written in CQL form as

```
MATCH (eid:Employee),
WHERE (eid.SOLD=oid.order),
RETURN eid. employee name,
COUNT(distinct sum (eid.order)) AS total Orders;
```

Query joins employee and order records and computes the required employee's bonus.

Before these two records are joined in grouping of, associative list a pair of assigners and shrinkers first processes them. The flow of retrieval is pictorially shown in **Fig. 2**. Shrink phase then adds up the bonus of employees with respect to all sessions he has worked and sorts them. Finally, updating of records using append, sort emit are done based on algorithm 3, illustrated in search algorithms subsection. After the updating process algorithm 4 is called where the frequency of terms is computed using VSM cosine similarity and prediction based probability function detailed in section, theoretical view of LIndex. **Fig. 3** gives a detail view of how the query is processed in LIndex structure.



File chunks of records

| eid | employeename | oid | bonus |
|---|---|---|---|
| 1 | Askhay Shah | 200 | 100 |
| 1 | Askhay Shah | 201 | 264 |
| 2 | Banret Banergi | 202 | 3040 |
| 2 | Banret Banergi | 203 | 302 |
| 3 | Banq Bulbul | 204 | 7600 |
| 4 | Catharine Coar | 205 | 90 |
| 5 | Deopter Dat | 206 | 882 |
| 6 | Elaen Emal | 207 | 800 |
| 6 | Elaen Emal | 208 | 4600 |
| 7 | Feb Fazl | 209 | 880 |
| 8 | Metell Mary | 210 | 340 |
| 9 | Magret Disouza | 211 | 329 |
| 10 | Mummtha Bose | 212 | 126 |
| 10 | Mummtha Bose | 213 | 653 |
| 11 | Reak Rodwick | 214 | 175 |
| 12 | Rean Rose | 215 | 166 |
| 12 | Rean Rose | 216 | 245 |
| 13 | Rose Disuza | 217 | 112 |

```
MATCH (eid:Employee),
WHERE (eid.SOLD=oid.order),
RETURN eid.employeename,
```

| eid | employeename | bonus |
|---|---|---|
| 1 | Askhay Shah | 364 |
| 2 | Banret Banergi | 3342 |
| 3 | Banq Bulbul | 7600 |
| 4 | Catharine Coar | 90 |
| 5 | Deopter Dat | 882 |
| 6 | Elaen Emal | 5400 |
| 7 | Feb Fazl | 880 |
| 8 | Metell Mary | 340 |
| 9 | Magret Disouza | 329 |
| 10 | Mummtha Bose | 779 |
| 11 | Reak Rodwick | 175 |
| 12 | Rean Rose | 411 |
| 13 | Rose Disuza | 112 |

```
MATCH (eid:Employee),
WHERE (eid.SOLD=oid.order),
RETURN eid.employeename;
```

**Fig. 2.** Query Retrieval example to find employee bonus obtained from orders

## 4. Search Algorithms for Colossal Social Network Data

This section talks about the proposed algorithm LIndex and how it works

### 4.1Problem Description

Consider any social network data stored in Neo4j graph NoSQL database D is a directed simple graph G with edges constituting relationships and nodes representing the union of file and record names denoted as $G_{Dt}$ or $G_D$ in Neo4j. We aspire to perform query retrieval $Q$ (12) using CQL

$$Q \in U_{i=1}^{\alpha} f_i U U_{i=1}^{\alpha} U_{k=1}^{\gamma(i,2)} ( f_{i2k} ) U U_{k'=1}^{\gamma(i,3)} f_{i3k'} \tag{12}$$

The parameters of (12) are explained in (1) to (6). The query problem is to find an exact match of $Q$ in $G$ and retrieve $Q$ from $G$ provided query execution time is reduced without compromising the quality of the query search.

### 4.2 Proposed Algorithm- LIndex

Algorithm 1 illustrates how the query retrieval process is done. When a query arrives the transaction management is called, and the cypher parser is invoked for query translation in Neo4j graph database. After the query gets translated, first it looks in the transaction log followed by object cache and file system cache, if the data requested for search is obtained then it retrieves otherwise it calls the search in Neo4j record storage structure. Algorithm 2 discuss how search operation is done with the captured *node_value*, where,*node_value* $\in Q$ (12). Once the essential node is found, then the corresponding node's property and relation index is scrutinized across and the required data is returned. Once when the query search starts algorithm 3 is called. Here each NSR is search based on the index structure. By default, it takes the first record of our proposed LIndex and further traverses the entire structure till the desired search item obtained. LIndex our proposed algorithm 4 first calls the algorithm 3 were it talks about the involvement of Lucene Index in Neo4j in order to get a faster query retrieval result. Algorithm 3also maintains a new record frequency weight of term $t_{inr}$ and term inverse record frequency weight is also calculated. Frequently retrieved multi-keyword query terms are identified based on Cosine Similarity Heuristic when applied to Lucene Vector Space Model (VSM) and predicted the probability.

### 4.3 Theoretical view of LIndex

LIndex is interpreted as a view based on query term frequency to inverse record frequency. Initially, it is assumed that record is a collection of terms denoted as labels. Let $\beta$ and $\gamma$ denote the total number of records and terms/labels, equation (5-6) illustrates $\beta$ and $\gamma$. In adjustment to feasibility view, $r_{ij}$ denotes a query retrieval of finding a record from $\beta$. Likewise $f_{ijk}$ denotes query retrieval of finding a term/label. Now $\beta$ and $\gamma$ are terms that denote query retrieval defined over (5) and (6) respectively. Introduction of $\beta$ and $\gamma$ portray how query constitute as a feasibility distribution over $\beta$, and retrieval shows the feasibility distribution over $\gamma$. The main objective is to perform efficient search of multi-keyword within $\beta$ and $\gamma$ in Neo4j graph database connected to social network.

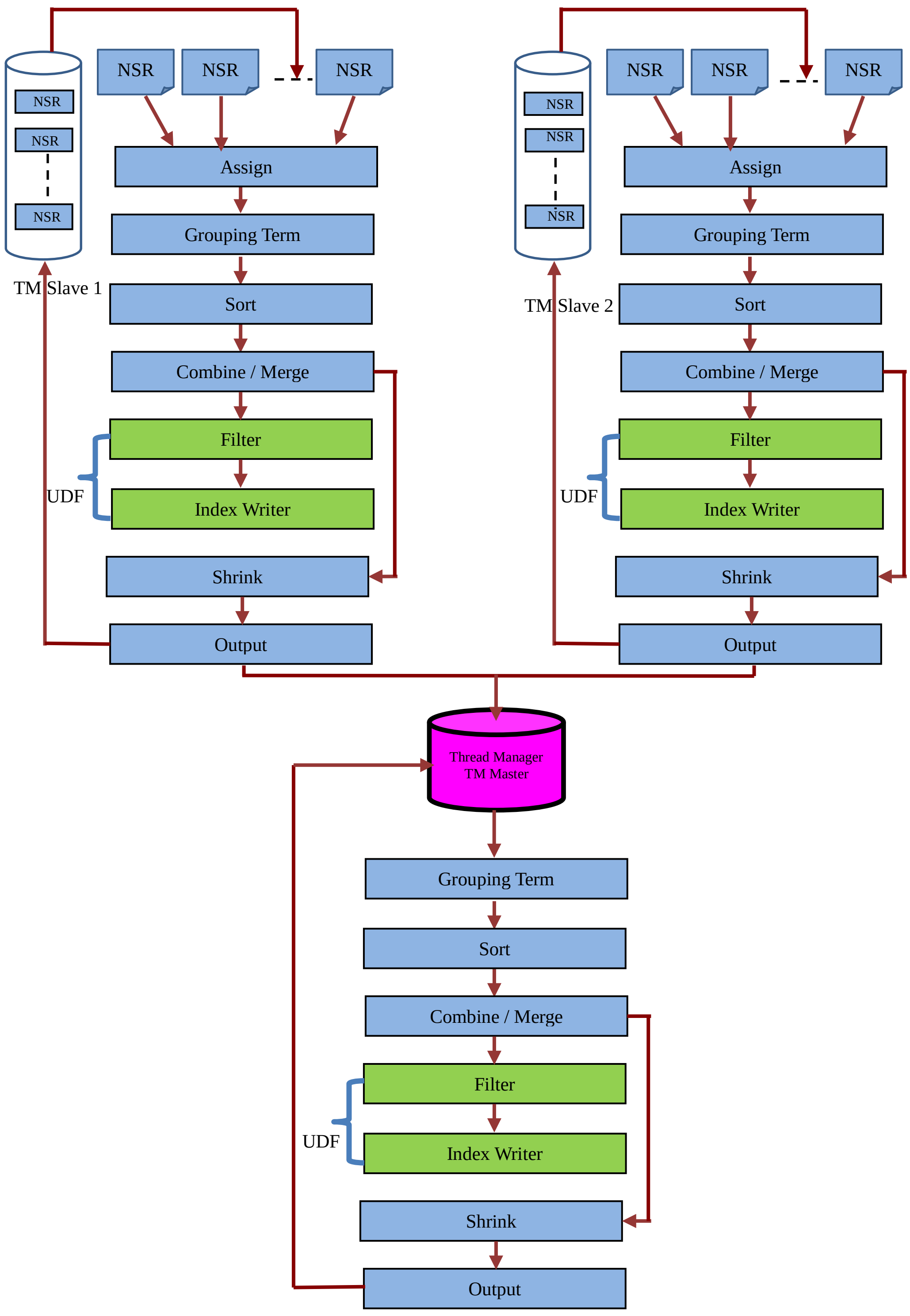


**Fig. 3.** LIndex Setup in Neo4j HA Architecture

**Algorithm 1: Query Retrieval**

```
Input: Enter the Cypher Query to search for
begin
1. q ← CypherQuery/*Multikeyword Query q to be search is dispatched
   by Thread Manager to Transaction Management of Master Neo4j HA*/
2. Call Transaction Management/*Transaction Management transmit q
   Cypher*/
3. Call Cypher Parser/*Q translated and processed for search*/
4. Call Transaction Log, Object Cache, File System Cache
/*Once search successful return to Transaction Management*/
5.if CypherQuery found
6. Retrieve query
7. else
6. Call Search in Neo4j Record Storage
end
```

**Algorithm 2: Search in Neo4j Record Storage**

```
Input: Search nodevalue in NSR, where NSR ← $(r_{ij})_{j=1}^{\beta}$

/* Check whether terms in q matchesNSR of any node*/
begin
1. while (node$_{value}$ ≠NSR)

2. while  $((f_{ijk})_{k=1}^{\gamma(i,j)}{}_{index}\, ptr \neq null)$ /*NSR LIndex traversed */

3. Call Function LIndex Creation
4. if  $((f_{ijk})_{k=1}^{\gamma(i,j)}{}_{index} = node_{value})$

5. then get Node node$_{property}$ $(f_{ijk})_{k=1}^{\gamma(i,j)}{}_{index}$

6. get   relation$(f_{ijk})_{k=1}^{\gamma(i,j)}{}_{index}$  relation$_{property}$
7. end
8. return    node    $(f_{ijk})_{k=1}^{\gamma(i,j)}{}_{index}$, relation
9. else
10. end while
11. end while
12. if node$_{value}$ = NSR
13. then goto step4
      end
```

**Algorithm 3: Simple Lucene Index Creation**

```
Input: Assign each NSR record with keyFunction insertQuery (q[i])
begin
1. Function Assign($nsr_{no}$ n, NeoNodeStore nsr)/*New AssociativeList
   maintained for NSR*/
2. RL ←new AssociativeList/*Taking each term uniquely from NSR */
3. for all term qt ∊ NeoNodeStore nsr do
4. $RL_{qt}$ ← $RL_{qt}$ + 1
5. for all term qt ∊ RL do /* Assignment */
6. Emit(term qt, assign n, $RL_{qt}$)
end
begin
1. Function Shrink(term qt, assign[(n1,f1)...])/* Grouping equivalent
   terms */
2. L new List
3. for all assign n, f ∊ assign [n1, f1....] do/* Append assigned grouped
   term to List */
4. L.Append(n,f)/* Sorting the recorded new list */
6. L.Sort()
7. Emit(term t, assign L)
end
```

**Algorithm 4: LIndex Creation**

```
Input: NSR record nsr = nsr1, nsr2, ...........nsrn,
Total number of records N, terms extracted from recordsof nsr t = t1,
t2.....tr, Q = q1, q2,........qt
begin
1. Call Function Assign($nsr_{no}$ n, NeoNodeStore nsr)
2. Call Function Shrink (term qt, postings[(n1,f1)...])
3. /*Calculate Frequency weight of query term qt in record r of file
   f */
4. repeat
5. for j ← 1 to r do
6. $fw_{(qt;r)} \leftarrow 1 + log_{10} tf_{(qt,r)};$ if $tf_{(qt,r)} > 0$
7. $fw_{qt,r} \leftarrow 0$, otherwise
8. score ← $\Sigma qt \in q \cap r (1 + \log tf_{(qt,r)})$
9. end for
10./*Calculate Inverse Record Frequency weight of term t in r */
11. repeat
12. for j ← 1 to r do
13. $irf_{qt} \leftarrow log_{10} \frac{N}{rf_{qt}}$
14. end for
15./* Calculate Term frequency - Inverse Record Frequency weight */
16.repeat
17. for j ← 1 to r do
18.$fw_{qt,r} \leftarrow 1 + log10\ tf_{(qt,r)} * irf_{qt}$
19. end for
20. /*Calculate of Cosine Similarity of Query */
21. repeat
```

```
22. for j ← 1 to n do
23.     $\|q\| \leftarrow \sqrt{\sum_{j=1}^{n} q_j^2}$
24.     $\cos\theta = \frac{r_2 \cdot q}{\|r_2\|\|q\|}$
25. end for
26. */Calculate of Predicted Probability of Query*/
27. repeat
28. for i ← 1 to α do
29. for j ← 1 to βdo
30. for k ← 1 to γdo
31.     $\sum_{f_{ijk}} \in \gamma \sum_{r_{ij}} \in \beta pf_{f_{ijk}} / PF\log(\beta/\beta_i)$
32. end for
33. end for
34. end for
35. Call Function Insert in Neo Store Record
end
```

Assume all records have an equal possibility in the initial phase hence P($r_{ij}$)=1/$\beta$ for all $r_{ij} \in \alpha$. This indicates the volume of information for each record given by *-log(1/α)*. The degree of randomness of variable $\beta$ is denoted as (13).

$$H(\beta)\sum_{r_{ij}\in\beta} P(r_{ij})\log P(r_{ij}) = -\beta(1/\beta)\log(1/\beta) = -\log(1/\beta) \tag{13}$$

Consider next what happens when records $r_{ij}$containing specified terms $f_{ijk} \in \gamma$ are known ahead of query retrieval. Let $\alpha_i$ be the number of records from the set. Assuming αi records will happen, amount of data observed for each record occurrence is *-log*(*1/*$\beta_i$). In this case degree of randomness of $\beta$ given $f_{ijk}$ becomes (14).

$$H(\beta / f_{ijk}) = -\sum_{r_{ij}\in\beta} P(r_{ij})\log P(r_{ij} / f_{ijk}) = -\beta_i(1/\beta_i)\log(1/\beta_i) = -\log(1/\beta_i) \tag{14}$$

Some records may not have $f_{ijk}$occurrence query term, therefore, the factor of these records become $\beta$ to $\beta_i$. Let us assume multikeyword query random appearance of terms $qt_i$from the whole recordset. The frequency of query term $qt_i$within $r_{ij}$ as $pf_{ij}$, similarly frequency of $f_{ijk}$ in the whole record as $pf_{f_{ijk}}$ and the total frequency of all other multi-keywords appearing is *PF*. The frequency of specific term $f_{ijk}$ selected is (15).

$$\sum_j pfij / PF = pf_{f_{ijk}} / PF \tag{15}$$

Then, the predicted probability is calculated based on Kullback–Leibler Divergence [17] as

$$
\begin{aligned}
D_{KL}(\beta,\gamma) = H(\beta) - H(\beta|\gamma) &= \sum_{f_{ijk}\in\gamma} P(f_{ijk})H(\beta) - H(\beta|f_{ijk}) \\
&= \sum_{f_{ijk}\in\gamma} pf_{f_{ijk}} / PF(-\log(1/\beta) + \log(1/\beta_i)) \quad (16) \\
&= \sum_{f_{ijk}\in\gamma} pf_{f_{ijk}} / PF(\log(\beta/\beta_i)) \\
&= \sum_{f_{ijk}\in\gamma} \sum_{r_{ij}\in\beta} pf_{f_{ijk}} / PF(\log(\beta/\beta_i))
\end{aligned}
$$

Term frequency sum of products is denoted as $pf_{ij}$ or $pf_{f_{ijk}}$, and the inverse record frequency (irf) divided by constant factor of PF is shown in (16). Therefore to culminate in theoretic perspective, tf-irf can be explained as the information required for calculating (16) The irf gives the difference in information after detecting on terms and the tf denotes the probability approximate of term observed. From (16) two aspects of tf-irf can be viewed. When tf mentions $pf_{f_{ijk}}$, tf-irf is taken as a measure for term selection. Similarly, from(16) the value over all the existing terms, indicate the specificity of all records in the retrieval system. When tf represent $pf_{ij}$, tf-irf is a taken to measure weighting of terms among all records inorder to decrease the unpredictability of records based on the submitted query.

Coming to LIndex algorithm analysis, the duplicate values are combined to the associative list by assigning links to existing ones and assign unique key for identification of record field. Hence only one entry for each data item is obtained with additional pointers (ptrs) to locations that hold the item (Algorithm 3). These reduce the storage space and improve access efficiency. Heuristic based Lucene called LIndex is called where the basic units of information record are indexed and stored for retrieval. Therefore each record of NSR has connection to its corresponding sub-records of itsNeo4j architecture. Thus, using LIndex indexing features each record is considered as a term-vector. Initially, a simple weight frequency count based on the distance from source is initiated for each record. Then a Heuristic - Vector Space Model is used.

Neo4j records and queries are represented as vectors. Equations (1) to (6) and (12) denotes how each file, record are represented as terms of vectors. MATCH Query q represents (12) .If a data/term occurs in the node record (NR), its value in the vector is non-zero. The values can be computed by using (term/data) associated with NSR records. The tf-irf weighting scheme can be used here. Ranking of records called heuristic based cosine similarity in MATCH query search for Neo4j is done, based on the similarity of record with query. Similarity theory details how to compare the angle of deviations between each record vector and the MATCH query vector where the query is constituted in the same form as a vector. To calculate the cosine angle between the vectors is shown in (17).

$$\cos\theta = \frac{\mathrm{r}_2 \cdot \mathrm{q}}{\|\mathrm{r}_2\|\|\mathrm{q}\|} \quad (17)$$

Where r2·q is the intersection (dot product) of the record r2 and the query term q from match query of Neo4j. Vector $\|\mathrm{r}_2\|$ is the norm of vector r2, and $\|\mathrm{q}\|$ is the norm of vector q. The norm of a vector is calculated as (18).

$$\|q\| = \sqrt{\sum_{i=1}^{n} q_i^2} \tag{18}$$

All vectors taken under the consideration of our Neo4j LIndex model should be non-negative, a cosine value of zero indicates the query and record vector are orthogonal and have no match from cosine similarity formulae. The cosine of two vectors can be obtained from Euclidean dot product formula (19).Here two vectors are record *r2* and query *q*, the cosine similarity, *cos*($\theta$), is represented using a dot product and magnitude as (20).

$$\mathrm{r2}\cdot \mathrm{q} = \|\mathrm{r\ 2}\|\ \|\mathrm{q}\|\ cos(\theta) \tag{19}$$

$$\text{similarity} = \cos(\theta) = \frac{\mathrm{r}_2 \cdot \mathrm{q}}{\|\mathrm{r}_2\|\|\mathrm{q}\|} = \frac{\sum_{i=1}^{n} r2_i q_i}{\sqrt{\sum_{i=1}^{n} r2_i^2}\sqrt{\sum_{i=1}^{n} q_i^2}} \tag{20}$$

Where $r2_\mathrm{i}$ and $q_i$are components of vector r2 and q respectively.

The outcome similarity ranges from -1 meaning no match and1 meaning match, with 0 no match, and in between values indicate intermediate similarity or dissimilarity. For each query in Neo4j using CQL the attribute vectors *r2* and q are usually the term frequency vectors of the records in each file of Neo4j.The cosine similarity can be seen as a method of normalizing the record length of each file during comparison. In the case of query search, the cosine similarity of two records will range from 0 to 1, since the term frequencies (tf-irf weights) cannot be negative. The angle between two term frequency vectors cannot be greater than $90^0$. If the attribute vectors are standardized by subtracting the vector means (r2 - $\bar{r2}$), the measure is called centered cosine similarity. Equations (16), (17) and (18) together constitute the proposed LIndex based search for multi-keyword queries in Neo4j integrated with social network.

## 5. Implementation

The proposed system is implemented on Neo4j HA architecture enterprise edition. The algorithms described in this paper are developed in Java with Neo4j version 3.0.0 graph database. The Neo4j graph database after storage of the datasets is analyzed. LuceneIndex is created on a specific directory of Neo4j version 3.0.0 with a specific analyzer Version.Neo4j.LUCENE (3.0.0). The directory is a distributed memory across master-slave architecture since most indices are too large to store in a particular memory location of a physical machine. The Lucene Analyzer parses data that is added to the index into terms so that it can be searchable.

Neo4j database access can occur using JDBC driver or with Neo4j libraries imported with built-in dependencies. For connection through JDBC we must import JDBC Neo4j connection driver to start the Neo4j service on machine. In some case master and slave can act on the same machine depending on the size of dataset whereas in others they have a distributed multi node architecture. Before the JDBC driver starts its activation driver, Neo4j should login to your any social network account example Twitter, Facebook or LinkedIn. In our case we have used Facebook and Twitter accounts for login. The JDBC driver asks for connecting the machines we are using. In our case dell precision tower 5810 workstation, Fujitsu, two i3 physical machines are connected to Twitter or Facebook. Once connected to Twitter or Facebook you can access only your personal details through the Neo4j-Twitter/Facebook api. Then we can establish the localhost (slave) connection through port 3857. For each slave we have different ports. Once if localhost connection is done then we can import real-time data of 2015 FIFA World Cup or any other data from the dataset. After importing this data we store the same in the record storage structure format of Neo4j that is the NSR class format as already discussed in the above Neo4j data storage section of Neo4j HA database. After the storage of 2015 FIFA or any other data, we again go for CREATE command to store the *followers, following*. Once if this is done we can get access to the query terms of the particular user whose Twitter/Facebook account is linked to Neo4j.

Thus, when a cypher query request for search is made, accessing of the query terms is done only after the connection established setup of local machine to master and the thread manager gets activated of master. The thread manager invokes fork call to transaction management of Neo4j master to start its role of query execution. The transaction management invokes the cypher parser, where the query translation takes place. This process denotes a shorter duration in query translation compared to the process using Embedded Database method of GraphDatabaseFactory. Here the Class Highly Available GraphDatabaseFactory is used as a file depository in disc which store the record data managed of Neo4j. If once the Highly Available GraphDatabaseFactory is opened and if any other application tries to establish connection the attempt will result in a error failure connection. Even though it is a good option to use JDBC driver due to the improvement in response time, some methods belonging to Neo4J library might not be present. This problem is resolved by the use of Class Highly Available GraphDatabaseFactory with JDBC driver.

After the translation of query by Cypher parser, the transaction log is called and checked. Transaction log of Neo4j should have been set as true before the master-slave setup. For "conf/neo4j.conf" need to be edited as setting the "neo4j.dbms.logs.query.parameterloggingenabled = true". If once the transaction log was enabled then when the same query arrives that was traversed previously, the result is processed from transaction log. Else the object cache is checked followed by file system cache. If the data is not obtained, then the record file is invoked from which we get the records of query terms asked. After the retrieval of query terms, the required result is obtained and sends to the transaction management.

Initially, Neo4j did not have an index structure. It used a default record storage architecture format. This default record storage structure when traversed each time for query retrieval by Neo4j HA took tremendous time(shown in Basic Structure blue colour of **Fig. 4**).Therefore the proposed LIndex scheme improved the performance of query processing as shown in **Fig. 4**. As the implementation of index on Neo4j HA does affect the query performance but rather improves the same. This new LIndex on the transaction management of Neo4j HA does not need any further integrations or layered approaches as the traversal phase is replaced by the interrupt call to LIndex on the neostore record structure. The time needed to create the index initially in case of LIndex it takes time in minutes compared to default time it takes is seconds but once when index created the query time for LIndex is in negligible seconds whereas for **Fig. 4**. FIFA Twitter Database **Fig. 5**. FIFA Database record storage structure traversal is in seconds. The algorithms 1, 2, 3 and 4 are tested based on the datasets specified and results are shown in experimental results subsection.

### 5.1. Experimental Results

Seven datasets namely FIFA 2015, Galaxy, Ego Network, Northwind, Social Circle, Movie and Library were tested on Neo4j 3.0.0 version. Each dataset with different sizes was separately stored on dell precision tower 5810 workstation, dual core 2.68GHz Intel Xeon processor Fujitsu, two i3 physical machines. Dell precision tower 5810 workstation and dual core 2.68GHz Intel Xeon processor Fujitsu running in CentOS X86-64 with 8 GB of memory, striped RAID array 2 disk 1 TB was used. Each dataset is separately categorized to different classes of nodes properties and edges based on Neo4j HA architecture.

Example: FIFA 2015 has 1632803 nodes 306225641 relations and 4912441252488 properties. Similarly, all other datasets are also categorized.

After the categorization Neo4j HA master-slave started and connection established to social networks as stated in the above section followed by which different queries depending on datasets are executed. The results obtained after the execution of queries are plotted in graph in **Fig. 4.** The comparison of time in seconds seized by the queries can be analyzed after index construction query retrieval. In the graphs plotted x-axis indicates the queries and y-axis the time in seconds for each query with respect to index algorithms at the time of search. The time taken for index construction inNeo4j HA graph database is given in **Fig. 5** of the appendix. Some of the queries retrieved in Neo4j HA from different social datasets are provided in **Table 1** to **Table 5** of the appendix. The results obtained after the execution of queries are provided in **Table 6.**

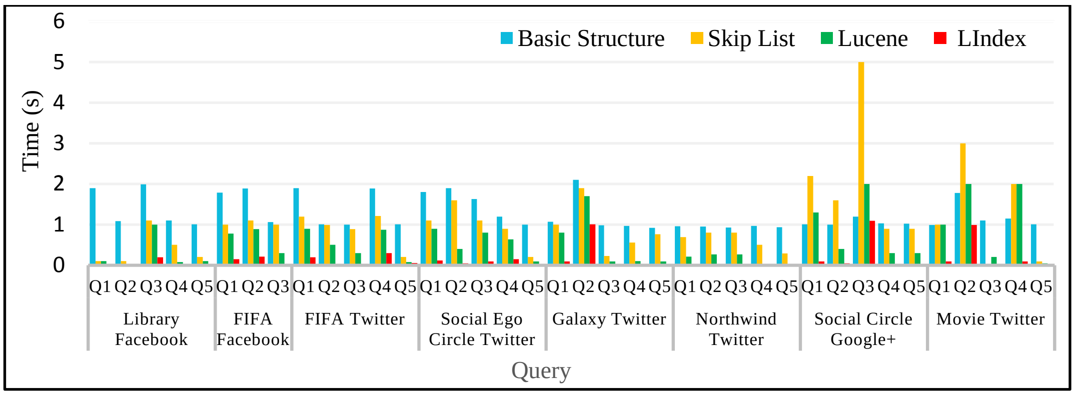


**Fig. 4.** Comparison of time taken in seconds for query retrieval

## 6. Conclusion

This work demonstrates how query retrieval can be done efficiently on graph NoSQL database Neo4j with social network data. For experimental evaluation and exhibition inNeo4j, dataset like FIFA 2015, Galaxy, Ego Network, Northwind, Social Circle, Movie and Library are used along with Twitter or Facebook social network after establishing a connection with the same. This dataset was stored in record structure of Neo4j 3.0.0 version graph database. The connection establishment, how data is stored and retrieved are observed and experimented. The query retrieval process was checked by using CQL by posting queries on Neo4j terminal. The newfangled algorithm, heuristic-based Lucene;LIndex was tested on the same set of datasets in Neo4j and compared the dissimilitude in the amount of time drawn during processing of a query. It was found that after executing the queries with LIndex there is an enormous growth in performance in query retrieval. Here, the time build performance is observed for each of the query retrieved. Even though experimental results showed better performance during query retrieval process in Neo4j integrated with Twitter or Facebook using various datasets, if we can cluster the frequently occurring terms into sets, then the query process retrieval within the set can be carried out with lesser time. This is considered as a future enhancement to be carried out.

## References

[1] J. Webber, "A programmatic introduction to Neo4j," in *Proc. of the 3rd annual conference on Systems, programming, and applications: software for humanity - SPLASH '12*, pp. 217-218, 2012. Article (CrossRef Link)
[2] Ferreira D. R. G., "Using Neo4J geospatial data storage and integration," *Dissertation*, University of Madeira, 2014. Article (CrossRef Link)
[3] Rabuzin, Kornelije, "Deductive Graph Database-Datalog in Action," in *Proc. of* IEEE *Int. Conf. on Computational Science and Computational Intelligence (CSCI)*, pp. 114-118, 2015. Article (CrossRef Link)
[4] Mathew Anita Brigit, "Comparison of Search Techniques in SocialGraph Neo4j", in *Proc. of 3rdInt. Symposiumon BigData and Cloud Computing Challenges (ISBCC-16)*, pp. 293-305, 2016. Article (CrossRef Link)
[5] Mathew Anita Brigit and Kumar SD Madhu, "Novel research frame work on SN"s NoSQL databases for efficient query processing," *International Journal of Reasoning-based Intelligent Systems*, vol. 7, no.3, pp. 330-338, 2015. Article (CrossRef Link)
[6] Mathew Anita Brigit and Madhu Kumar S D, "Analysis of data management and query handling in social networks using NoSQL databases," in *Proc. of Int. Conf. on Advances in Computing, Communications and Informatics (ICACCI)*, pp. 800-806, May 2015. Article (CrossRef Link)
[7] Yaroslav, K., VladimirTarasenko, Julia Boyarinova, and YakovKalynovskiy. "Vector Functionally-Oriented Processors with Vertical Parallelism for Operations on Quaternions," *Journal of Qafqaz University*, vol. 1, no. 2, pp.83-90, 2013. Article (CrossRef Link)
[8] Liu, Lu, and Tao Peng. "Post-processing of deep web information extraction based on domain ontology," *Advances in Electrical and Computer Engineering*, vol. 13, no. 4, pp. 25-32, 2013. Article (CrossRef Link)
[9] Alkire, Sabina, "The missing dimensions of poverty data: Introduction to the special issue," *Oxford development studies*, vol. 35, no. 4, pp. 347-359, 2013. Article (CrossRef Link)
[10] Yi, Xiaomeng and Liu, Fangming and Liu, Jiangchuan and Jin, Hai, "Building a network highway for big data: architecture and challenges," *IEEE Network*, vol. 28, no.4, pp. 5-13, 2014. Article (CrossRef Link)
[11] Kim, Kyoungsook, MoonsukYeon, ByeongsooJeong, and Kwanghoon Kim, "A ConceptualApproach for Discovering Proportions of Disjunctive Routing Patterns in a Business ProcessModel," *KSII Transactions on Internet & Information Systems*, vol.11, no. 2, 2017. Article (CrossRef Link)
[12] Atkinson, Anthony B, "Multidimensional deprivation: contrasting social welfare and counting approaches" The Journal of Economic Inequality, vol.1, no. 1, pp. 51-65, 2013. Article (CrossRef Link)
[13] Batrinca, Bogdan and Treleaven, Philip C, "The G graph database: efficiently managing large distributed dynamic graphs," *Distributed and Parallel Databases*, vol. 33, no. 4, pp. 479-514, 2015.Article (CrossRef Link)
[14] Khan, Arijit and Li, Nan and Yan, Xifeng and Guan, Ziyu and Chakraborty, Supriyo and Tao, Shu, "Neighborhood based fast graph search in large networks," in *Proc. of Int. Conf. on Management of data, ACM SIGMOD*, pp. 901-912, 2011. Article (CrossRef Link)

[15] Martinez-Bazan, Norbert and Dominguez-Sal, David, "Using semijoin programs to solve traversal queries in graph databases," in *Proc. of Workshop on Graph Data management Experiences and Systems*, ACM, pp. 1-6, 2014.Article (CrossRef Link)

[16] Li, Hongwei, Yi Yang, Mi Wen, HongweiLuo, and Rongxing Lu. "EMRQ: An Efficient Multikeyword Range Query Scheme in Smart Grid Auction Market," *TIIS*, vol. 8, no. 11, pp.3937-3954, 2014. Article (CrossRef Link)

[17] Otte, Evelien and Rousseau, Ronald, "Social network analysis: a powerful strategy, also for the information sciences," *Journal of information Science*, vol. 28, no. 6, pp. 441-453, 2002. Article (CrossRef Link)

[18] Batrinca, Bogdan and Treleaven, Philip C, "Social media analytics: a survey of techniques, tools and platforms," *AI & SOCIETY*, vol.30, no.1, pp. 89-116, 2015.Article (CrossRef Link)

[19] Morris, Meredith Ringel and Teevan, Jaime and Panovich, Katrina, "What do people ask their social networks, and why?: a survey study of status message q&a behavior," in *Proc. of the SIGCHI conf. on Human Factors in Computing Systems*, 42, ACM, (1), 2010, pp. 1739-1748. Article (CrossRef Link)

[20] Mathew Anita Brigit and Pattnaik, Priyabrat and Madhu Kumar S D, "Efficient information retrieval using Lucene, LIndex and HIndexinHadoop," in *Proc. of 11th Int. Conf. on Computer Systems and Applications (AICCSA)*, pp. 333-340, Nov. 2015. Article (CrossRef Link)

[21] Li, Hongwei, Yi Yang, Mi Wen, HongweiLuo, and Rongxing Lu, "EMRQ: An Efficient Multikeyword Range Query Scheme in Smart Grid Auction Market," *TIIS*, vol. 8, no. 11, pp.3937-3954, 2014Article (CrossRef Link)

[22] Selim, Haysam and Zhan, Justin, "Towards shortest path identification on large networks", *Journal of Big data*, vol.3, no.1, pp. 1-10, 2016.Article (CrossRef Link)

[23] Patino Mart´?nez, Marta and Sancho, Diego and Jim´enezPeris, RicardoandBrondino, Ivan and Vianello, Valerio and Dhamane, Rohit, "Snap shot isolation for Neo4j," *OpenProceedings.org*, 2016. Article (CrossRef Link)

[24] Shojafar M., Abawajy J. H., Delkhah, Z. et al., "An efficient and distributed file search in unstructured peer-to-peer networks," *Peer to-Peer Networking and Applications*, vol. 8, no.1, pp. 120-136, 2015.Article (CrossRef Link)

[25] Sakr, Sherif and Liu, Anna and Batista, Daniel M and Alomari, Mohammad, "A survey of large scale data management approaches in cloud environments," *IEEE Communications Surveys & Tutorials*, vol.13, no.3, pp. 311-336, 2011. Article (CrossRef Link)

[26] Mathew Anita Brigit and Madhu Kumar S D, "An Efficient Index based Query handling model for Neo4j," *International Journal of Advances in Computer Science and Technology*, vol. 3, no. 2, pp. 12-18, 2014. Article (CrossRef Link)

[27] CiroCattuto, Marco Quaggiotto, Andre Panisson, Alex Averbuch, "Time-varying social networks in a graph database: a Neo4j use case," in *Proc. of 1st Int. Workshop on* Graph Data Management Experiences and Systems, June 23-23, pp.1-6, New York, 2013. Article (CrossRef Link)

[28] Mussarat, Yasmin, Sharif Muhammad, MohsinSajjad, and IrumIsma. "Content based image retrieval using combined features of shape, color and relevance feedback." *KSII Transactions on internet and information systems*, vol.7, no. 12, pp.3149-3165, 2013. Article (CrossRef Link)

[29] Wan, Jiafu, Hehua Yan, HuiSuo, and Fang Li. "Advances in Cyber-Physical Systems Research," *TIIS*, vol.5, no. 11, pp.1891-1908, 2011.Article (CrossRef Link)

## Appendix

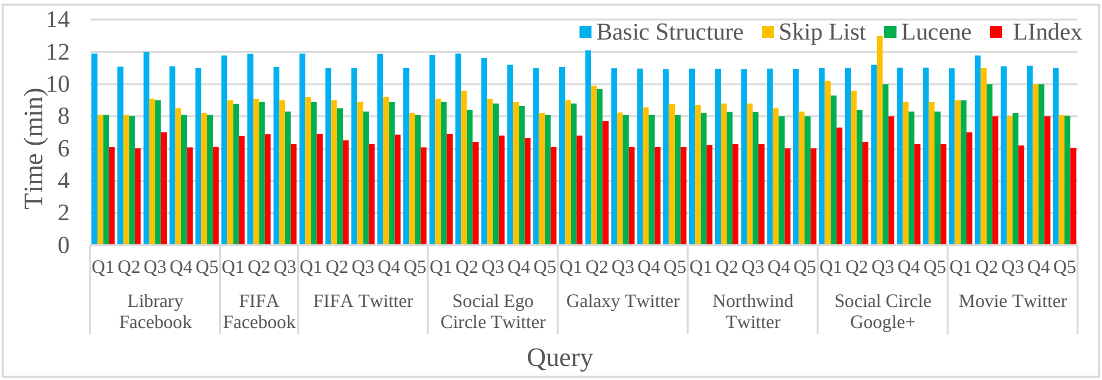


**Fig. 5.** Comparison of time taken in minutes for Index Construction inNeo4j graph data base

**Table 1.** Neo4j-Facebook 2015 FIFA database

1) Query 1 : Matches played on 16 September 2015
MATCH (matchid:Matchmatchid.date: ’16 Septiembre’),(yearid:Year’2015’),
RETURN matchid.date,matchid.mname, matchid.score;
(MI) Nodes: 120 Relations: 9900 Properties: 172312
(SI) Nodes: 108 Relations: 9799 Properties: 26991
(LI) Nodes: 11 Relations: 99 Properties: 432
(LII) Nodes: 11 Relations: 99 Properties: 432
2) Query 2 : Search for Matchday 4 and 7 details in FIFA 2015
MATCH (matchid:Match),
WHERE matchid.matchday = ’4’ AND matchid.matchday = ’7’,
RETURN matchid.matchday,matchid.mname,matchid.score;
(MI) Nodes: 1201 Relations: 11008 Properties: 346004098
(SI) Nodes: 1127 Relations: 10827 Properties: 47857401
(LI) Nodes: 989 Relations: 8700 Properties: 6548079
(LII) Nodes: 989 Relations: 8700 Properties: 6548079
3) Query 3 : Search for Stadiums proposed for FIFA 2015
MATCH (matchid:Match),
RETURN matchid.stadium;
(MI) Nodes: 111 Relations: 12321 Properties: 151807
(SI) Nodes: 101 Relations: 10201 Properties: 10406
(LI) Nodes: 34 Relations: 1919 Properties: 3079
(LII) Nodes: 347 Relations: 1919 Properties: 3079

**Table 2.** Neo4j-Twitter queries processed from 2015 FIFA

1) Query 1 : Search for people name start with ans who are following2015 FIFA
MATCH (folwid,folwname:Followingfolwname: ’Ans→abc’),RETURN folwid.folwname;
(MI) Nodes: 101 Relations: 10201 Properties 104060401
(SI) Nodes: 78 Relations: 9799 Properties: 26991
(LI) Nodes: 36 Relations: 100 Properties: 432
2) Query 2 : Search for posts on Apr 18 in 2015 FIFA
MATCH (postid,postdate:Postspostdate: ’Apr 18’),RETURN postid.postdate;
(MI) Nodes: 91 Relations: 8281 Properties: 75864961
(SI) Nodes: 45 Relations: 5329 Properties: 29891
(LI) Nodes: 12 Relations: 89 Properties: 342
3)Query 3 : Search for player RAMPONE from USA in 2015 FIFA
MATCH (playerid, playername:Playersplayerid.playername: ’RAMPONE’),

MATCH (playerid, country:Playersplayerid.country: 'USA'),RETURN playerid.playername;
(MI) Nodes: 111 Relations: 12321 Properties: 151807041
(SI) Nodes: 108 Relations: 9799 Properties: 26991
(LI) Nodes: 24 Relations: 99 Properties: 432

4) Query 4 : Search for player names whose age ≤30 from USA in 2015FIFA
MATCH (playerid, playername, playerage:Players),WHERE (playerid.playerage ≤ 30),RETURN playerid.playername;
(MI) Nodes: 219 Relations: 47961 Properties: 2057521
(SI) Nodes: 127 Relations: 1827 Properties: 857401
(LI) Nodes: 89 Relations: 700 Properties: 48079
5) Query 5 : Search coaches of 2015
MATCH (coachid,coachname:Coaches),RETURN coachid.coachname;
(MI) Nodes: 78 Relations:6084 Properties: 37015056
(SI) Nodes: 47 Relations:3827 Properties: 47857401
(LI) Nodes: 66 Relations:1400 Properties: 43279

**Table 3.** Neo4j-twitter social circle ego networks database

1) Query 1 : Search of people who work as Cryptanalyst
MATCH (nodeId:WorknodeId.worktitle: 'Cryptanalyst'),RETURN nodeId.worktitle, nodeId.wname;
(MI) Nodes: 278 Relations:9984 Properties: 370150
(SI) Nodes: 168 Relations:5424 Properties: 21381
(LI) Nodes: 94 Relations:2156 Properties: 7636
2) Query 2 : Search for people with nickname 'Weekend Riders' who work in company 'Dobbler Consulting Service'
MATCH (nodeId,workid:Work), (nodeId:People),(nodeId,cid:Company),WHERE (nodeId.nickname = 'Weekend Riders' and cid.cname ='Dobbler Consulting Service'),RETURN nodeId.fname, nodeId.mdname, nodeId.lastname,nodeId.nickname, nodeId.cname;
(MI) Nodes: 219 Relations: 47961 Properties: 230025
(SI) Nodes: 209 Relations: 47861 Properties: 23002
(LI) Nodes: 74 Relations: 5476 Properties: 2998
3) Query 3 : Search people with age 30 who did education in Cambridge
MATCH (nodeId:People),WHERE (nodeId.age = 30 and nodeId.education = 'Cambridge'),RETURN nodeId.fname, nodeId.mdname, nodeId.lastname;
(MI) Nodes: 181 Relations: 18761 Properties: 151807
(SI) Nodes: 101 Relations: 10001 Properties: 12106
(LI) Nodes: 58 Relations: 2199 Properties: 4899
4) Query 4 : Search for people information who have twitter and facebook account with ego level > 80percent
MATCH (nodeId:People), (nodeId,tid:Twitter), (nodeId,fid:Facebook),WHERE (tid.status = 'Active' and fid.status = 'Active'),RETURN nodeId.fname, nodeId.mdname, nodeId.lastname;
(MI) Nodes: 110 Relations: 8281 Properties: 685731
(SI) Nodes: 98 Relations: 8181 Properties: 68564
(LI) Nodes: 54 Relations: 1466 Properties: 5436
5) Query 5 : Find number of ego circles formed and order by name
MATCH (nodeId:People),RETURN nodeId.fname, nodeId.mdname, nodeId.lastname,nodeId.ecnum,ORDER BY nodeId.fname;
(MI) Nodes: 101 Relations: 10201 Properties 104060
(SI) Nodes: 91 Relations: 8281 Properties: 68574
(LI) Nodes: 47 Relations: 2209 Properties: 4879

**Table 4.** Neo4j-twitter queries in northwind database

1) Query 1 : Retrieve Products exported to different Territories with date
MATCH (prodid:Products), (prodid, tid:Territory) (prodid, date:Date),RETURN prodid.prodname, tid.tname, date.day, date.month, date.year;
(MI) Nodes: 301 Relations: 18281 Properties: 168573
(SI) Nodes: 211 Relations: 18181 Properties: 109564
(LI) Nodes: 102 Relations: 9006 Properties: 11336
(LII) Nodes: 102 Relations: 9006 Properties: 11336
2) Query 2 : List customers who placed the order in the month of March
MATCH (prodid:Products), (prodid, cid:Customers) (prodid,oid:Order)(prodid, date:Date),WHERE (date.month = “March”),RETURN cid.cname;

(MI) Nodes: 267 Relations: 10201 Properties 104060
(SI) Nodes: 157 Relations: 9881 Properties: 68574
(LI) Nodes: 100 Relations: 6209 Properties: 9879

3) Query 3 : Search for region wise customer details
MATCH (prodid:Products), (prodid, cid:Customers),RETURN cid.cname, cid.region;
(MI) Nodes: 126 Relations: 9081 Properties: 90807
(SI) Nodes: 98 Relations: 8790 Properties: 8996
(LI) Nodes: 47 Relations: 4209 Properties: 6779
4) Query 4 : Search for categories of different product marketted
MATCH (prodid:Products),RETURN prodid.prodname, prodid.category;
(MI) Nodes: 132 Relations: 9891 Properties: 99805
(SI) Nodes: 100 Relations: 8976 Properties: 8992
(LI) Nodes: 64 Relations: 5476 Properties: 7028
5) Query 5 : Find the permanent employees in the firm order by name
MATCH (eid:Employee),WHERE (eid.status = ’Permanent’),RETURN eid.name,ORDER BY eid.name;
(MI) Nodes: 114 Relations:8904 Properties: 370150
(SI) Nodes: 88 Relations:7094 Properties: 21381
(LI) Nodes: 42 Relations:4016 Properties: 5136

**Table 5.** NEO4J=TWITTER QUERIES IN SOCIAL CIRCLE GOOGLE+ DATABASE

1) Query 1 : Search of people who follow you
MATCH (nodeId:Circle),WHERE (nodeId.fstatus = ’FOLLOW’ and nodeId.who = ’CurrentActive’),
RETURN nodeId.pname;
(MI) Nodes: 278 Relations:79984 Properties: 370150
(SI) Nodes: 168 Relations:65424 Properties: 21381
(LI) Nodes: 94 Relations:3156 Properties: 7636
2) Query 2 : Search for circle names and the size of each circle
MATCH (nodeId:Circle),RETURN nodeId.cname, nodeId.size;
(MI) Nodes: 219 Relations: 47961 Properties: 230025
(SI) Nodes: 209 Relations: 47861 Properties: 23002
(LI) Nodes: 74 Relations: 5476 Properties: 5998
3) Query 3 : Find the circle names that belong to category Web
MATCH (nodeId:Circle),WHERE (nodeId.category = ’Web’),RETURN nodeId.cname;
(MI) Nodes: 281 Relations: 98761 Properties: 2651807
(SI) Nodes: 201 Relations: 89001 Properties: 26106
(LI) Nodes: 107 Relations: 6599 Properties: 9899
4) Query 4 : Search for authors of each circle
MATCH (nodeId:Circle),RETURN nodeId.cname, nodeId.author;
(MI) Nodes: 110 Relations: 8281 Properties: 685731
(SI) Nodes: 98 Relations: 8181 Properties: 68564
(LI) Nodes: 64 Relations: 1466 Properties: 3436
5) Query 5 : Find number of circles within the south pacific region
MATCH (nodeId:Circle),WHERE (nodeId.region = ’South Pacific’),RETURN nodeId.cname;
(MI) Nodes: 110 Relations: 8285 Properties 685731
(SI) Nodes: 97 Relations: 8180 Properties: 68564
(LI) Nodes: 61 Relations: 1466 Properties: 3436

**Table 6.** Comparison of time taken in seconds by query after index construction in Neo4j graph database

| Database | Query | NSR | SkipList | Lucene | LIndex |
|---|---|---|---|---|---|
| FIFA 2015 (Facebook) | Q1 | 6.2 | 5.1 | 1.7 | 0.2 |
| FIFA 2015 (Facebook) | Q2 | 14 | 12 | 6 | 1 |
| FIFA 2015 (Facebook) | Q3 | 3.3 | 2.2 | 1 | 0.002 |
| FIFA 2015 (Neo4j-Twitter) | Q4 | 6.2 | 5 | 1.7 | 0.2 |
| FIFA 2015 (Neo4j-Twitter) | Q5 | 2.8 | 1.6 | 0.2 | 0.002 |
| FIFA 2015 (Neo4j-Twitter) | Q6 | 3.3 | 2.2 | 1 | 0.002 |
| FIFA 2015 (Neo4j-Twitter) | Q7 | 14 | 12 | 6 | 1 |
| FIFA 2015 (Neo4j-Twitter) | Q8 | 1.3 | 1.2 | 0.1 | 0.003 |

| | | | | | |
|---|---|---|---|---|---|
| Social Circle Ego Networks (Neo4j-Twitter) | Q9 | 8.2 | 5 | 2.7 | 1.2 |
| Social Circle Ego Networks (Neo4j-Twitter) | Q10 | 4.8 | 2.6 | 1.1 | 0.05 |
| Social Circle Ego Networks (Neo4j-Twitter) | Q11 | 6.3 | 3.1 | 2 | 0.1 |
| Social Circle Ego Networks (Neo4j-Twitter) | Q12 | 18 | 14.4 | 7 | 2 |
| Social Circle Ego Networks (Neo4j-Twitter) | Q13 | 2.3 | 1.2 | 1 | 0.007 |
| Galaxy (TPC-H Neo4j-Twitter) | Q14 | 8.2 | 6 | 2.5 | 1 |
| Galaxy (TPC-H Neo4j-Twitter) | Q15 | 11 | 10.5 | 8 | 2 |
| Galaxy (TPC-H Neo4j-Twitter) | Q16 | 6.5 | 3 | 2 | 0.1 |
| Galaxy (TPC-H Neo4j-Twitter) | Q17 | 3 | 1.002 | 0.9 | 0.007 |
| Galaxy (TPC-H Neo4j-Twitter) | Q18 | 4.9 | 2.6 | 1.1 | 0.05 |
| Northwind (Neo4j-Twitter) | Q19 | 4.09 | 2.9 | 1.6 | 0.008 |
| Northwind (Neo4j-Twitter) | Q20 | 3.4 | 2 | 1.1 | 0.004 |
| Northwind (Neo4j-Twitter) | Q21 | 3.3 | 1.8 | 0.7 | 0.001 |
| Northwind (Neo4j-Twitter) | Q22 | 1.8 | 0.6 | 0.09 | 0.0005 |
| Northwind (Neo4j-Twitter) | Q23 | 1 | 0.09 | 0.01 | 0.0003 |
| Neo4j-Social Circle Google+ | Q24 | 6.3 | 3.2 | 2 | 0.1 |
| Neo4j-Social Circle Google+ | Q25 | 4.8 | 2.6 | 1.1 | 0.05 |
| Neo4j-Social Circle Google+ | Q26 | 8.2 | 6 | 2.7 | 1.1 |
| Neo4j-Social Circle Google+ | Q27 | 2.3 | 1.2 | 1 | 0.007 |
| Neo4j-Social Circle Google+ | Q28 | 2.3 | 1.2 | 1 | 0.007 |
| Neo4j-Twitter-Movie dataset | Q29 | 3.97 | 2 | 1.7 | 0.1 |
| Neo4j-Twitter-Movie dataset | Q30 | 4.3 | 4 | 2.7 | 1 |
| Neo4j-Twitter-Movie dataset | Q31 | 1.4 | 1 | 0.9 | 0.02 |
| Neo4j-Twitter-Movie dataset | Q32 | 3.3 | 3 | 2.7 | 0.1 |
| Neo4j-Twitter-Movie dataset | Q33 | 1.1 | 0.9 | 0.6 | 0.008 |
| Library Neo4j-Twitter dataset | Q34 | 1.9 | 0.9 | 0.6 | 0.01 |
| Library Neo4j-Twitter dataset | Q35 | 1.4 | 1.1 | 0.9 | 0.02 |
| Library Neo4j-Twitter dataset | Q36 | 2.3 | 2.1 | 1.7 | 0.2 |
| Library Neo4j-Twitter dataset | Q37 | 7.8 | 7.6 | 4.5 | 1 |
| Library Neo4j-Twitter dataset | Q38 | 1 | 0.8 | 0.5 | 0.002 |

**Anita Brigit Mathew** received her B.E.and M.E. degrees in Computer Science and Engineering from Karunya University, Coimbatore, India in 2004and 2006, respectively. She is currently working as a Research Scholar at National Institute of technology, Calicut in the Department of Computer Science. Her research interest includesbig data Analytics, Query Optimization and Retrieval from Social Networks.