# Performance Benchmarking for Link Prediction Algorithms in Social Networks

## Table of Contents


Abstract ..... 2

Keywords ..... 2

1. Introduction ..... 3

2. Link Prediction ..... 4

3. Implementation ..... 6

   3.1 Relational Database: MySQL ..... 6

   3.2 Graph Database: Neo4j ..... 7

   3.2.1 Changes to the reference model and queries ..... 7

4. Experiments ..... 8

5. Plots ..... 10

6. Conclusion ..... 13

7. References ..... 14

## Abstract

For the past one decade social network has gained a lot of popularity and more users are making their online presence to connect with each other. Hence it brings up new challenges of analyzing data generated from these users. One such analyses is the social connection between two users. A lot of work has been done in the past with regard to link analysis. From a given snapshot of a social network database, we can predict for a given person (or the entire network), the people who she can be potentially connected to, by analyzing her existing links. Although there are is a lot of effort put into developing new prediction techniques, there is no solid function for analyzing which database is suited for a particular link analysis method. Link prediction can be done either for the entire network, or for a small subset of the network graph centered on a particular user. We consider the latter in this project.

We take open datasets and import it into MySQL (for relational), and Neo4J (for Graph based) and evaluate several link metrics. Experimentally, we plan on classifying how performance varies with respect to metrics for different databases. We also plan to analyze on how link metrics vary according to the network topology/parameters. We try to improve the performance of the queries implemented in the referenced paper : “Implementing link-prediction for social networks in a database system” by Sarah Cohen et al. Our experiment is performed on eight different real social network datasets taken from SNAP and DBLP databases. Finally, the results verify that the changes brought about in the neo4J schema and query structure improve the performance indeed



## Keywords

Link prediction, MySQL, Neo4J, database, social networks

# 1. Introduction

Recent research focuses on large, complex network graphs and their properties and hence a considerable amount of attention has been devoted to the computational analysis of social networks. Such networks have structures whose nodes represent people or other entities in a social context, and whose edges represent interaction, collaboration, or influence between entities. This experiment considers the set of authors in a particular discipline, with edges joining pairs who have co-authored papers. Similar datasets can also be used for repeating this experiment for eg : the set of all employees in a large company, with edges joining pairs working on a common project.

Social networks are highly dynamic objects - they grow and change quickly over time through the addition of new edges, signifying the appearance of new interactions in the underlying social structure. For this reason, researchers have defined the *link prediction problem* where they study a snapshot of a social network at time t1, and try to accurately predict the edges that will be added to the network from time t1 to a future time t2. If we consider the co-authorship network that the project uses, link prediction might be able to hint new collaborations using the topology of the network. For eg: two scientists who are "close" in the network will have colleagues in common, and will travel in similar circles; this suggests that they themselves are more likely to collaborate in the near future.

There are several approaches to deal with the link prediction problem, the most common one is based on applying metrics to non-existent links at time t1 to determine if a link will appear at a time t2 (t2>t1). There are many existing similarity metrics like : Common Neighbors, Salton Index, Jaccard Index, Hub Depressed Index, Hub Promoted Index, Leicht–Holme–Newman Index (LHN1), Preferential Attachment Index, etc. The referenced paper - "Implementing link-prediction for social networks in a database system" implements seven such metrics on the relational (MySQL), key-value (Redis) and graph database (Neo4J).

Due to the incapability of the standard query languages to express important social network queries, implementing a database system for social network data is a huge challenge. In this experiment, we wish to specify how performance of social network queries is affected by the choice of the database.

In this paper, we are analyzing the performance improvement of Neo4J's cost based optimizer over the rule based optimizer. We also changed the structure of the queries, introduced indexing and changed the original data model to bring more optimization. These improved the performance significantly as shown in our results section. Hence this experiment is a meaningful step towards examining the performance of Neo4J under various conditions.

Organization of this paper is as follows : In Section 2 we precisely define the link prediction problem and present seven popular metrics for use in solving this problem. In Section 3, we explain the implementation of these metrics over the different data storage systems. Section 4 contains the performance results. Conclusions and analysis are present in Section 5.

## 2. Link Prediction

We have majorly referenced three papers for implementing link prediction. Apart from the base paper – "Implementing link-prediction for social networks in a database system" by Sarah Cohen et al. , we also have "A Multilayer Approach to Multiplexity and Link Prediction in Online Geo-Social Networks" by Hristova et al. It proposes that the link prediction is more efficient if we construct multiple layers of network and calculate measures of overlap like as Adar/Adamic co-efficient. The empirical study takes tweets and check-ins from two social networks (Twitter and Foursquare) using web API and find the overlap of users in three major cities in USA. The authors define node neighborhood for multiple networks and show that the users who are in both networks are more likely to interact with same values and have a significant neighborhood overlap. As a result, we can predict Twitter links from Foursquare features and vice versa. The study also compares the multiplexity of interaction (e.g., common hashtag, mentions, check-in) with single network layer links and shows how their multilayer structure is useful. Different similarity measures and supervised learning methods from past experiments have been adopted in this research. Above all, the primary contribution of the paper is to address the use of heterogeneous social networks which can be useful for our benchmarking. However, there are implications in the research such as defining the network structure, degree of node, asymmetric activity and so on. Moreover, the study can not include private data which is limited to only one network and some users might not link their accounts in both network.

In third referenced paper : "Link Prediction and Recommendation across Heterogeneous Social Networks" Formal definitions of link prediction are presented in the following 4 homogeneous social networks - Epinions (trust relations), Slashdot (friendships), Wikivote (vote relations), Twitter (reciprocity relations). The paper also gives the problem formulation of link recommendation across heterogeneous networks. It first introduces us to the 4 Baseline Predictors – CN (common neighbors) method, AA (Adamic Adar) measure, JA (Jaccards) index and PA (Preferential Attachment) index. Secondly, it addresses the following problem: How to come up with a transfer-based predictive model which gives better results as compared to the individual baseline predictors, by bridging the general patterns across heterogeneous networks into a model for link prediction. Experimental results show that such a transfer-based RFG (Ranking Factor Graph) model has a better predictive power in most cases when compared to the baseline methods, including the non-transfer RFG model. Finally, they verify the predictive performance of the presented transfer model on 12 pairs of transfer cases

In this experiment, we decided to implement six out of the seven pre-defined metrics from the base paper by Sarah et al. We consider a and b as two existing nodes in the network with N(a) and N(b) as their respective neighbor count. Let Pl(a,b) denote the set of all paths of length l between a and b.

- **Common neighbor**

The common neighbor function cn(a,b) counts the number of neighbors who are common to a and b. That is:

$$CN(a,b) = |N(a) \cap N(b)|$$

- **Jaccard coefficient:**

The common neighbor function may not work as expected when the number of total number of neighbors is too large. Jaccard coefficient takes this into account. It measures the relative number of common neighbors between two nodes. This is defined as:

$$JA\ (a,b) = \frac{|N(a) \cap N(b)|}{|N(a)\ \cup N(b)|}$$

- **Preferential attachment:**

This function is based on the concept that the link between two nodes depend on the number of total neighbor they have. This value increases with the increase of number of neighbors. Preferential attachment is given by:

$$PA\ (a,b) = |N(a) * N(b)|$$

- **Graph distance:**

This measure is different from the above measures because it does not account for the number of neighbors of two nodes. Rather it depends on the distance of a and b in the graph. We find the shortest path from a to b to as the length of the distance between a and b.

- **Katz measure:**

Unlike selecting only the shortest path between a and b Katz measure considers all paths from a to b. The weighted path gives a good intuition about possible connection between a and b. Suppose, $P^l$ is the set of all paths between a and b having length l. Katz measure can be computed from the path length as follows:

$$katz\ (a,b) = \sum_{l=1}^{\infty} \beta^l |P^l(a,b)|\ , \quad 0 < \beta < 1$$

Here, β is weight factor which is changed based on the requirements of the users. We used β=0.1 in our implementation. And for efficiency, we also considered the paths up to length 3

- **Rooted PageRank:**

PageRank computes the importance value for a node while going through a random walk over the whole graph. We compute rooted PageRank rPR (a, b) which represents the importance of node b with relation to node a. A damping factor d is set to a threshold value (0.85 in our experiment). While going through a random walk we calculate the stationary distribution of a. If the probability is d then, we select a random neighbor of node a for next jump and if the probability is 1-dthen then we select any random node in the graph for next jump.

Except the graph distance metric, above metrics having a higher value indicate that there is more likelihood of forming a link between a and b. For graph distance metric, a lower value indicates a higher likelihood.

## 3. Implementation

This section provides a summary of the schema and data models used in MySQL and Neo4j used for experiments.

### 3.1 Relational Database: MySQL

| Table Name | Description | Schema |
|---|---|---|
| cn | Common Neighbors | (id1 int, id2 int, id3 int) |
| cnc | Common Neighbors Count | (id1 int, id2 int, count int) |
| edges | Co-authorship relation amongst authors | (id1 int id2 int) |
| neighbors | Neighbor count for every author | (id int, neighbours int ) |
| nodes | Author Details | (id int, name varchar(64)) |
| topn | Top 100 Nodes with highest neighbors count | (id int, neighbors int) |

The schemas used for implementation in MySQL was exactly the same as used by Cohen et al [1]. There was very limited scope for improving the performance of the metrics in MySQL as the schema used in the reference model by Cohen was well optimized. The experiments were run on MySQL v5.6 with a query cache of 128MB. Each metric was implemented as a stored procedure. The schema representation used in MySQL for modelling the graphs is given below.

### 3.2 Graph Database: Neo4j

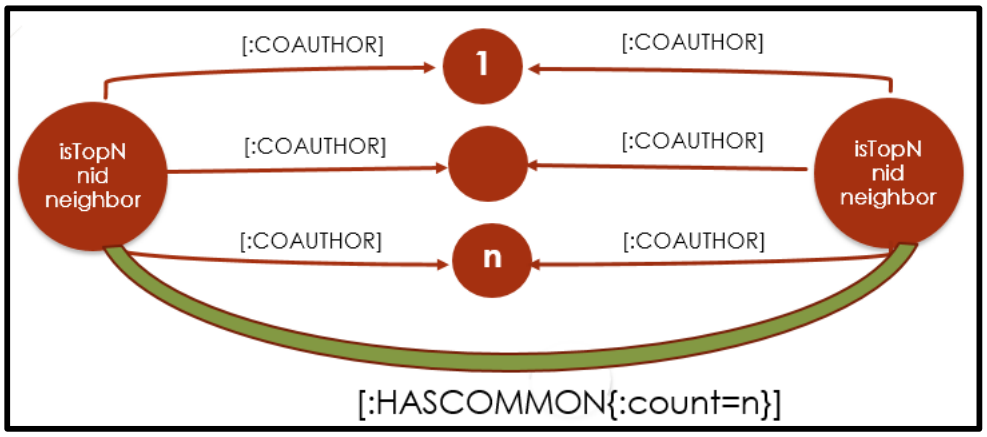
Implementation of the metrics in Neo4j was easier when compared to MySQL. We used Neo4j version 3.0.6 with a page cache of 5GB. The graphs used for evaluation consisted of author nodes and a coauthor relationship between authors and hence precisely matches a social network setup. The graph model remains the same as used by Cohen et al[1].

#### 3.2.1 Changes to the reference model and queries

In addition to the reference model, the following section summarizes the model changes that were implemented as part of improving the performance of the metrics.

1. After careful analysis of the queries used for generating the metrics, it was observed that most of the queries took a significant amount time computing the number of common neighbors between two nodes. Hence, a new edge with label 'common' was introduced which had a count of the number of common neighbors shared between two nodes. This was done as part of the experimental setup of the graph.
2. A new attribute called 'isTopN' was introduced to mark whether a node belongs to the set of top 100 nodes with maximum neighbors. An index was created on the attribute.
3. Most of the queries were rewritten to avoid using the 'start' keyword by replacing it with the 'match' version.
4. An index on the 'nid' attribute was created which acts as a schema index for the author nodes.
5. All queries were rewritten so as to use the newly introduced 'common' edge that significantly improves performance.

A pictorial representation of the graph model is shown below :



# 4. Experiments

All experiments were run on a system with an Intel Core-i7 2.5Ghz processor, and a memory of 12GB running Ubuntu 14.04. Each metric was repeated 100 times with randomly picked 100 nodes and the total and average times were recorded. The experiments were run on the same datasets as in the original paper by Cohen et al[1].

The datasets considered were open social network data from DBLP Computer Science Bibliography XML (http://dblp.uni-trier.de/xml/) and Stanford Large Network Dataset Collection (http://snap.stanford.edu/data/). The datasets are of three types:

- *Coauthorship:* This dataset is extracted from DBLP network. It has nodes as authors and edges when the authors have common publications. It is a large dataset having more than 350 thousand nodes and 4 million edges. The dataset is filtered so that it contains nodes with three or more publications. The whole dataset is named as *dblp-all-core3* and is divided into subsets named as *2002_2009_core3, 2010_2012_core3, ca-AstroPh, ca-CondMat, ca-GrQc, ca-HepPh* and *ca-HepTh.* The subsets are made according to either range of year of a field of interest.

- *Facebook:* This dataset contains friend lists of Facebook users which is obtained from anonymized online survey. The survey was conducted by a Facebook app and the name of the dataset is *facebook.*

A summary of the graph size is shown below :

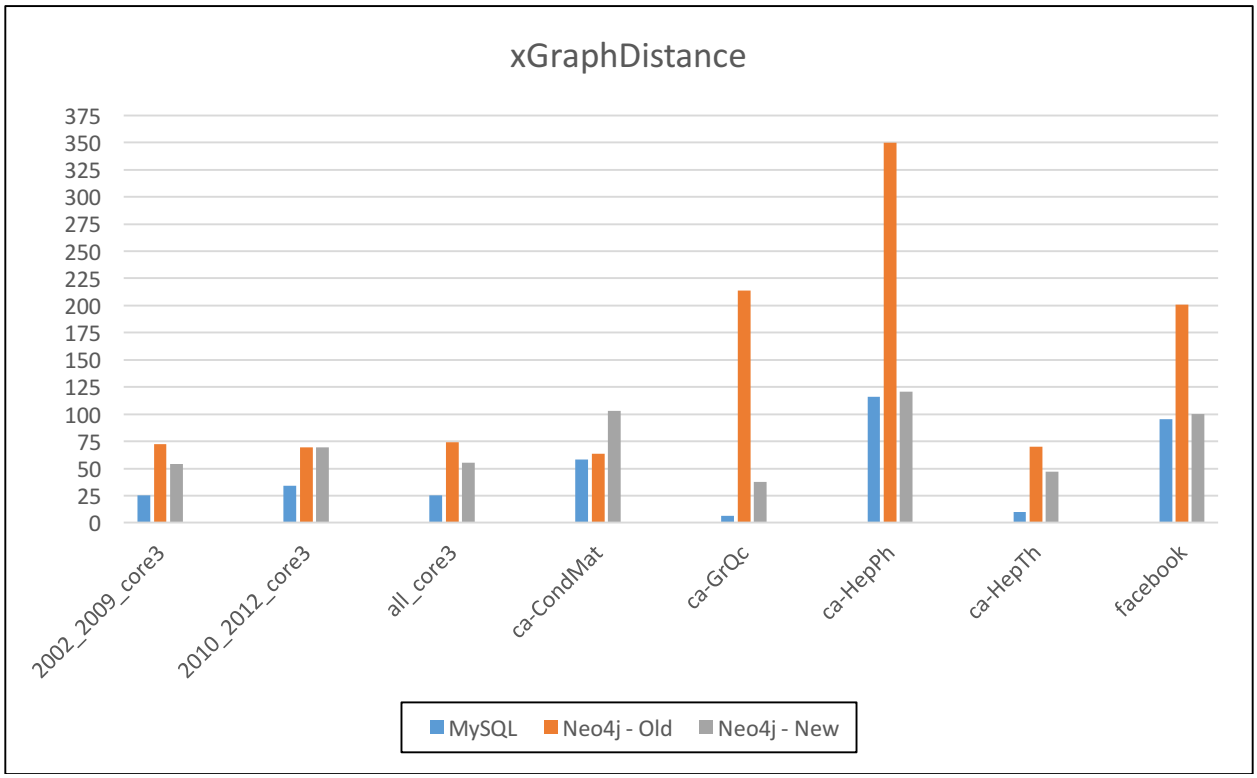
| **Type** | **Name** | **# Nodes** | **# Edges** |
|---|---|---|---|
| Coauthorship | dblp-all_core3 | 366,600 | 4,349,796 |
| | dblp-2010-2012 | 248,695 | 2,589,320 |
| | dblp-2002-2009 | 182,493 | 1,621,846 |
| | ca-CondMat | 23,133 | 186,936 |
| | ca-HepPH | 12,006 | 237,010 |
| | ca-HepTh | 9,875 | 51,971 |
| | ca-GrQc | 5,241 | 28,980 |
| Online Social Network | facebook | 4039 | 170,174 |

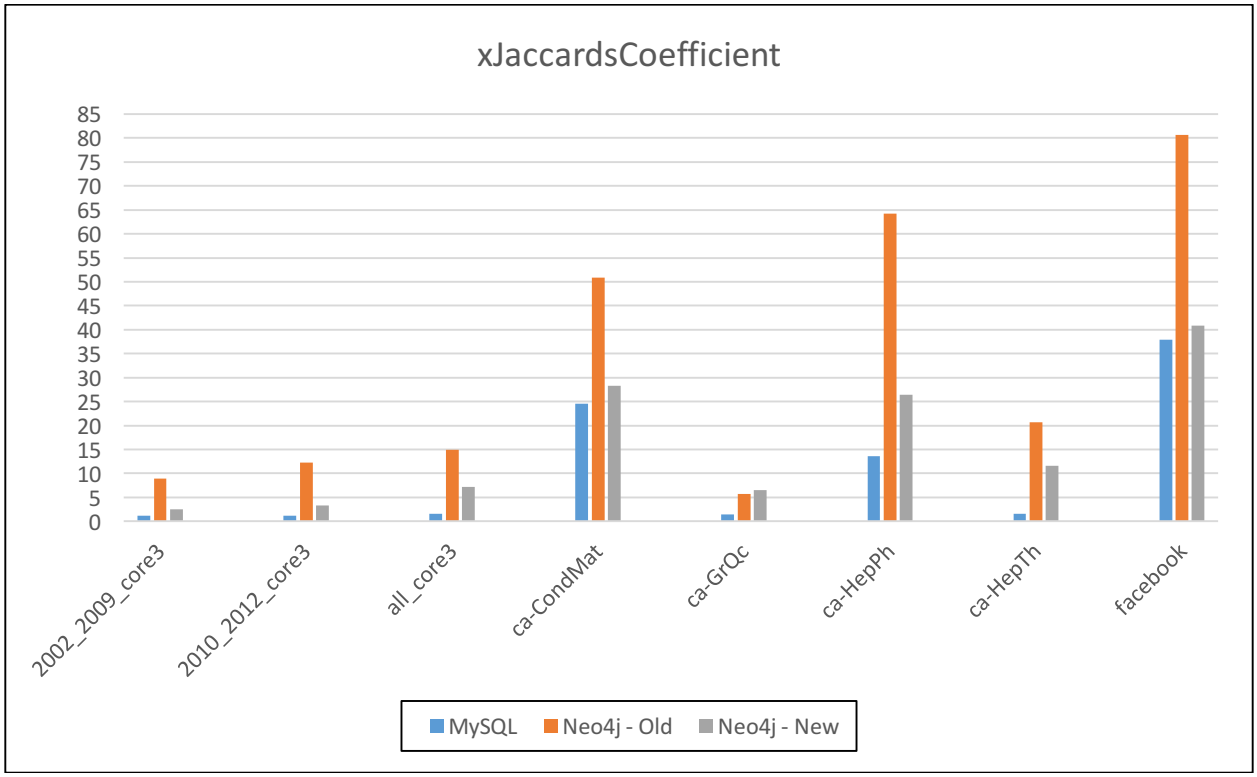
We compare the size of the Neo4j database before and after the changes made as described in section 3.2.1. The following table shows the increase in size of the database for few of the datasets.

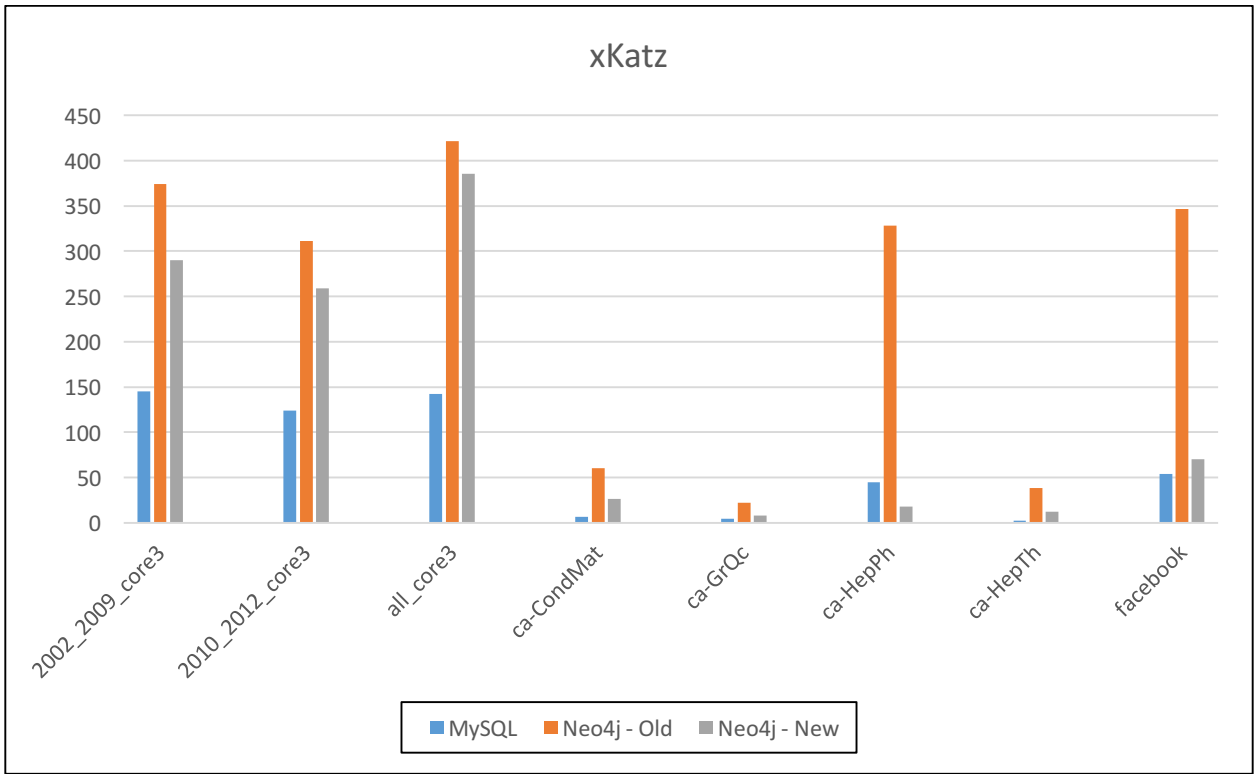
| **Model** | **Facebook** | | | **ca-HepPh** | | | **CondMat** | | |
|---|---|---|---|---|---|---|---|---|---|
| | Reference (MB) | New (MB) | % change | Reference (MB) | New (MB) | % change | Reference (MB) | New (MB) | % change |
| Node Store | 0.0639 | 0.063 | 0 | 0.175 | 0.175 | 0 | 0.343 | 0.343 | 0 |
| Property Store | 0.486 | 19.82 | 4000 | 1.41 | 65.39 | 6400 | 2.72 | 48.31 | 1676 |
| Relationship store | 5.53 | 21.86 | 295.29 | 7.69 | 60.75 | 689.9 | 6.06 | 43.87 | 623.92 |
| String store size | 0.008 | 0.008 | 0 | 0.008 | 0.008 | 0 | 0.008 | 0.008 | 0 |
| Total store size | 902.07 | 1920 | 117.9 | 2440 | 8630 | 240.16 | 4010 | 10850 | 170.57 |

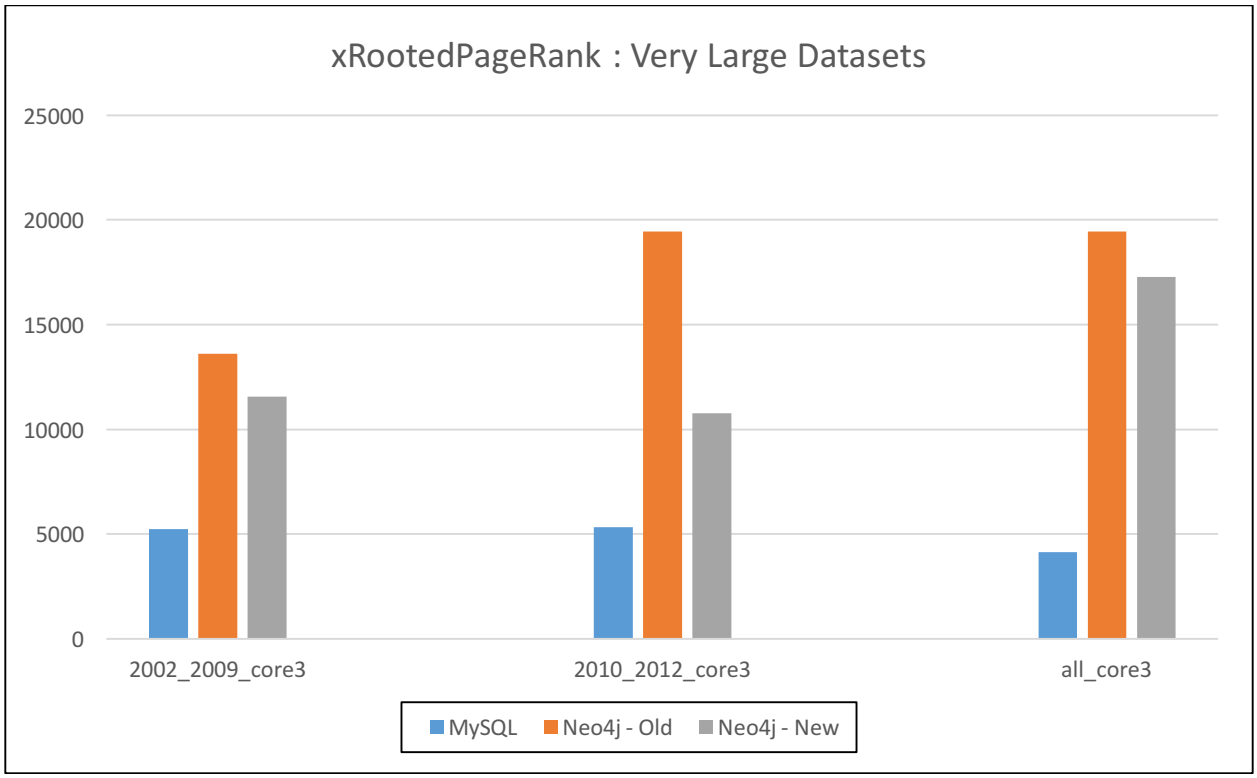
| **Model** | **caHepTh** | | | **dblp-2002-2009** | | | **Ca-GrQc** | | |
|---|---|---|---|---|---|---|---|---|---|
| | Reference (MB) | New (MB) | % change | Reference (MB) | New (MB) | % change | Reference (MB) | New | % change |
| Node Store | 0.151 | 0.151 | 0 | 1.92 | 1.92 | 0 | 0.079 | 0.079 | 0 |
| Property Store | 1.16 | 60.05 | 5076 | 15.68 | 784.31 | 4901.9 | 0.637 | 24.88 | 3896 |
| Relationship store | 1.69 | 62.57 | 3602 | 32.84 | 127.66 | 288.7 | 0.964 | 54.30 | 5666.6 |
| String store size | 0.008 | 0.008 | 0 | 0.008 | 0.008 | 0 | 0.008 | 0.008 | 0 |
| Total store size | 1830 | 7960 | 334.97 | 24910 | 61140 | 145.4 | 1740 | 6750 | 287.9 |

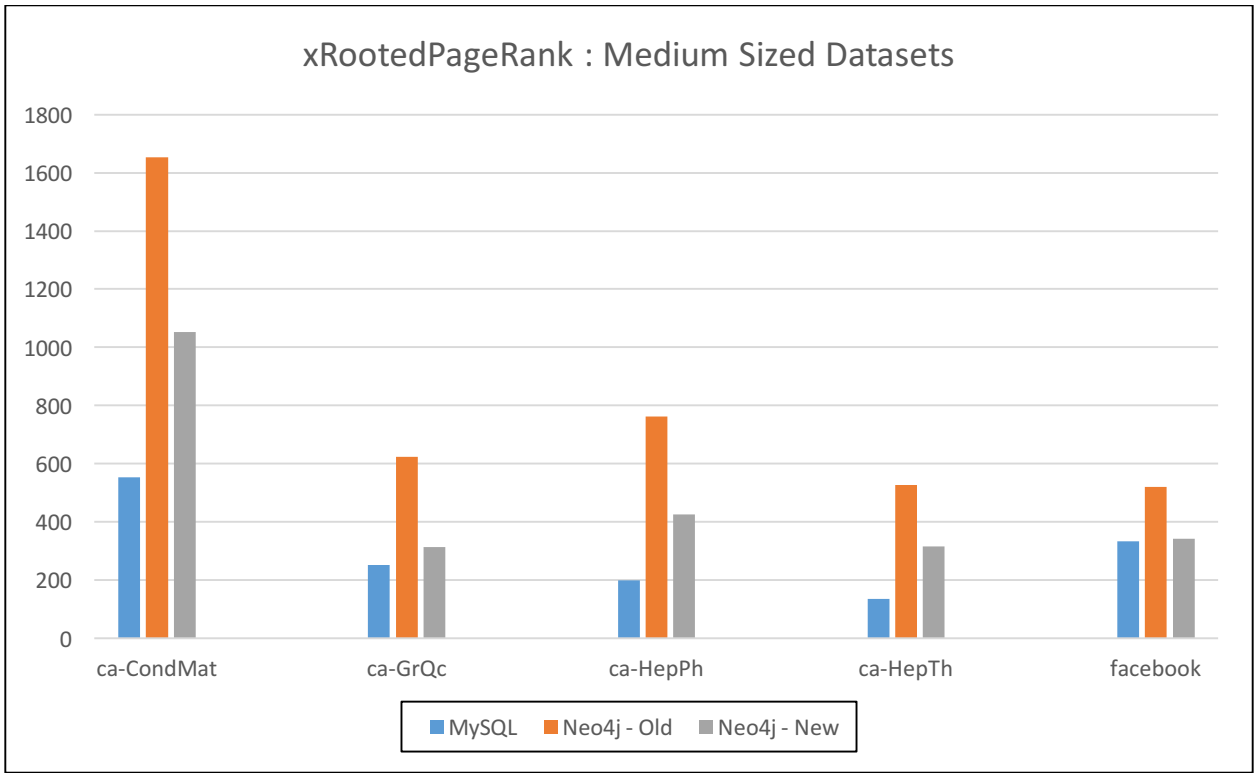
# 5. Plots

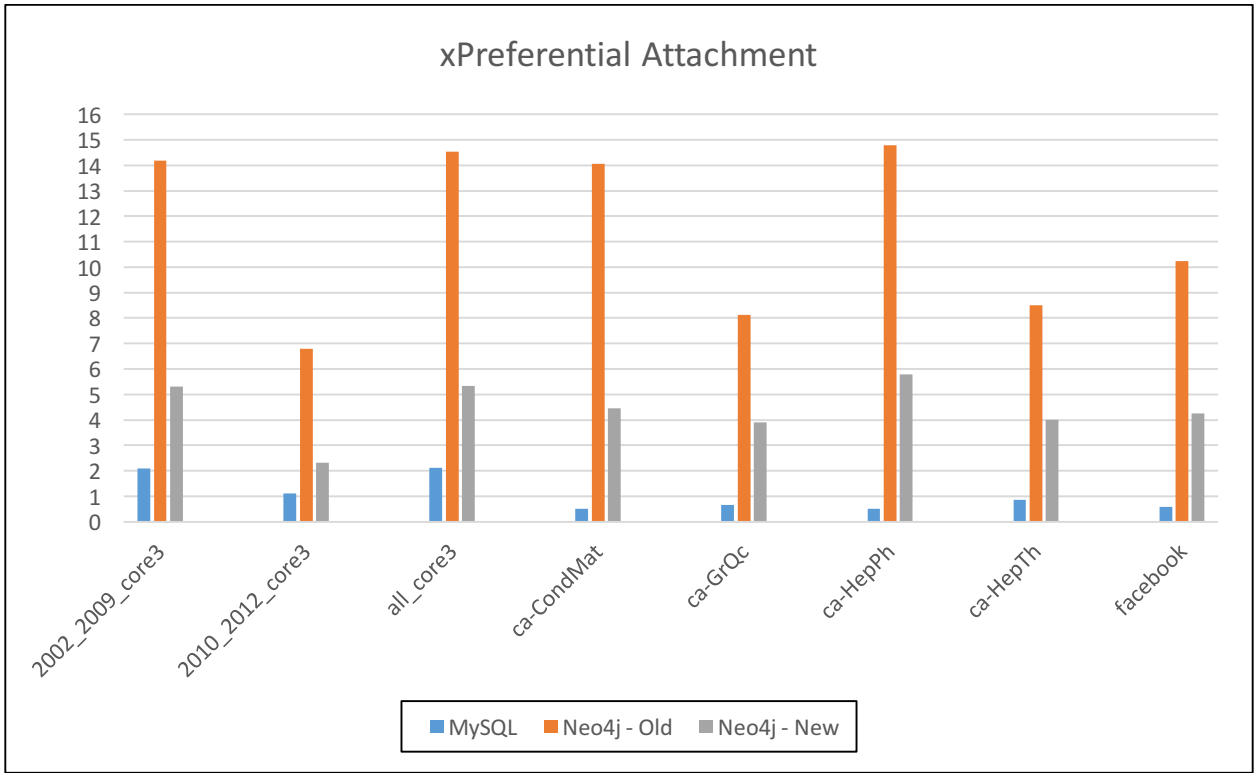

xGraphDistance
375
350
325
300
275
250
225
200
175
150
125
100
75
50
25
0
2002_2009_core3
2010_2012_core3
all_core3
ca-CondMat
ca-GrQc
ca-HepPh
ca-HepTh
facebook
MySQL
Neo4j - Old
Neo4j - New

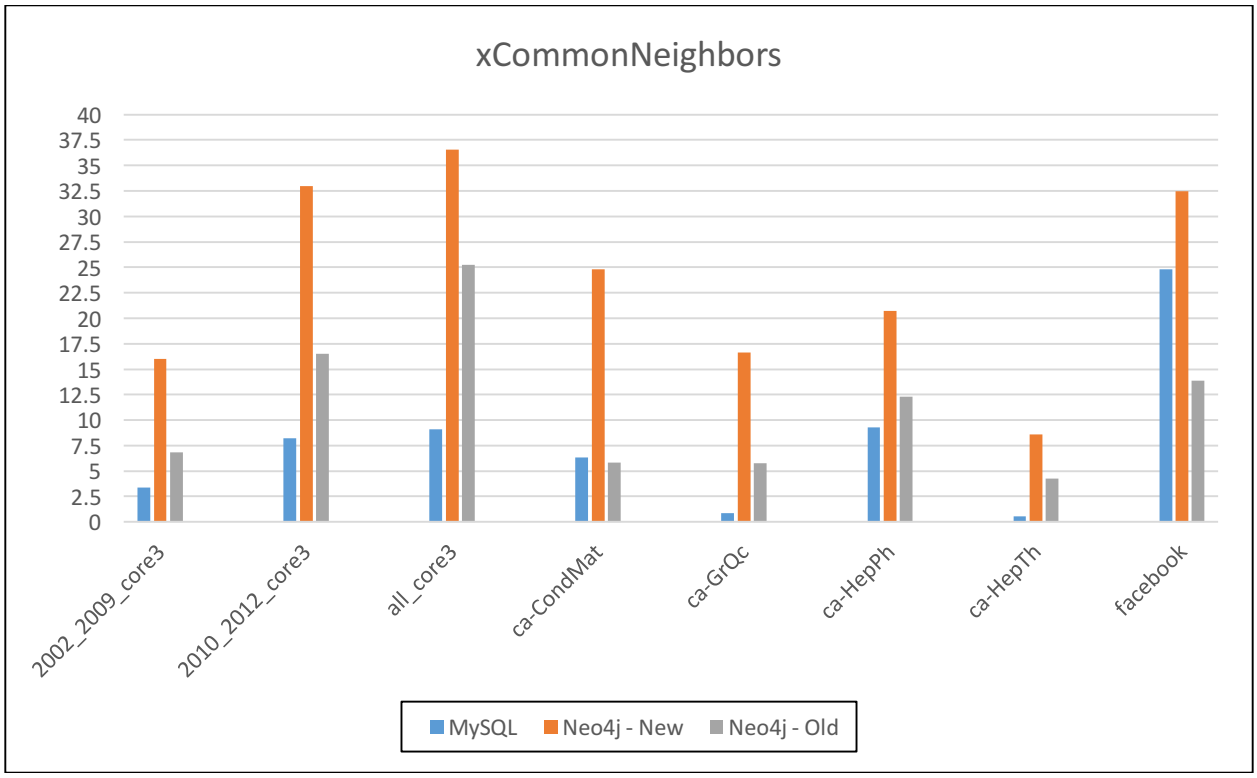

xJaccardsCoefficient
85
80
75
70
65
60
55
50
45
40
35
30
25
20
15
10
5
0
2002_2009_core3
2010_2012_core3
all_core3
ca-CondMat
ca-GrQc
ca-HepPh
ca-HepTh
facebook
MySQL
Neo4j - Old
Neo4j - New


xKatz
450
400
350
300
250
200
150
100
50
0
2002_2009_core3
2010_2012_core3
all_core3
ca-CondMat
ca-GrQc
ca-HepPh
ca-HepTh
facebook
MySQL
Neo4j - Old
Neo4j - New


xRootedPageRank : Very Large Datasets
25000
20000
15000
10000
5000
0
2002_2009_core3
2010_2012_core3
all_core3
MySQL
Neo4j - Old
Neo4j - New


xRootedPageRank : Medium Sized Datasets
1800
1600
1400
1200
1000
800
600
400
200
0
ca-CondMat
ca-GrQc
ca-HepPh
ca-HepTh
facebook
MySQL
Neo4j - Old
Neo4j - New


xPreferential Attachment
16
15
14
13
12
11
10
9
8
7
6
5
4
3
2
1
0
2002_2009_core3
2010_2012_core3
all_core3
ca-CondMat
ca-GrQc
ca-HepPh
ca-HepTh
facebook
MySQL
Neo4j - Old
Neo4j - New




## 6. Conclusion

In most of the metrics MySQL performs better than Neo4j. One reason is the optimizations in MySQL that helps joining faster. But we can also see that the modified model and the cost based optimizer in Neo4j makes performance comparable to MySQL in certain metrics. We can also observe that query performance in Neo4j is highly dependent on the model used to design the database. Careful model design and use of attribute indexes has a significant improvement in query performance.

The study made based on MySQL and Neo4j does not show insight of the type of the graph on which link prediction is done. The results are likely to vary based on the type of the graph (like density, transitivity co-efficient etc) which the study doesn't consider. Evaluating suitable database types for implementation can lead to development of hybrid systems that can lead to optimal performance of systems.

## 7. References

[1] Cohen, Sara, and Netanel Cohen-Tzemach. "Implementing link-prediction for social networks in a database system." In Proceedings of the ACM SIGMOD Workshop on Databases and Social Networks, pp. 37-42. ACM, 2013.\

[2] Liben-Nowell, David, and Jon Kleinberg. "The link-prediction problem for social networks." *Journal of the American society for information science and technology* 58, no. 7 (2007): 1019-1031.

[3] Cohen, Sara, Lior Ebel, and Benny Kimelfeld. "A Social Network Database that Learns How to Answer Queries." *CIDR*. 2013.

[4] Katz, Leo. "A new status index derived from sociometric analysis." *Psychometrika* 18, no. 1 (1953): 39-43.

[5] Liben-Nowell, David, and Jon Kleinberg. "The link-prediction problem for social networks." *Journal of the American society for information science and technology* 58, no. 7 (2007): 1019-1031.

[6] San Martın, Mauro, Claudio Gutierrez, and Peter T. Wood. "SNQL: A social networks query and transformation language." *cities* 5 (2011): r5.

[7] Ronen, Royi, and Oded Shmueli. "SoQL: A language for querying and creating data in social networks." In *2009 IEEE 25th International Conference on Data Engineering*, pp. 1595-1602. IEEE, 2009.

[9] https://www.cs.cornell.edu/home/kleinber/link-pred.pdf

[10] http://www.sciencedirect.com/science/article/pii/S0378437116301005