

Project Phase 3

<http://www.github.com/sohammehta95/identifying-genealogical-relations-using-prolog>

Soham Mehta sohmehta@cs.stonybrook.edu 20 December 2017

The paper selected for implementation of my project is "Searching Personnel Relationship from Myanmar Census Data using Graph Database and Deductive Reasoning Prolog Rules" published at the 2016 International Conference on Computer Communication and Informatics (ICCCI -2016).

1 Aim

The aim of this paper is to explore graph database structure that can support effective storage structure for peoples' connected information, to study efficient searching algorithm that can find the relationship from separated person nodes and to provide deductive reasoning for defining the indirect relationship among persons

2 Overview

The paper tries to find to find relationship among people given a census data. Most of the relationship searching researches have been developed based on the predefined relationship among every node using existing graph traversal algorithms. Therefore, this proposed system is aimed to develop searching relationship based on the personnel information of separated nodes stored in graph database. Relationships between person nodes are not predefined and separated person nodes with respective personnel information are created by the authors. The relations ships are in the form of may be direct links, links with one or more intermediate persons and disconnected persons.

The proposed framework include three parts

1. For storage structure, the personnel information is stored as graph structure with persons as nodes by using Ne04j graph database.
2. For graph searching, the user needs to provide two persons' names to search their relation using Personnel Relationship Searching Algorithm results are relationship types
3. For reasoning, Personnel Relationship Deduction Algorithm is used to define the final relation for given two persons using the deductive reasoning rules.

3 Implementing the paper using Neo4j Graphs

Neo4j graph is a database which is used for the storage of graph-oriented data structures with nodes, edges, and properties to represent and store data. It is an occurrence based and schema less.

3.1 Learning more about Neo4j Graphs

Neo4j graphs stores data and relationships as they are encountered. It can store complex and dynamic relationships of highly connected data like personnel information. Both nodes and relationships can hold any desired properties called key-value pairs. Storage is optimized for the traversal of the graph, without using an index when following edges. It has no rigid schema, node-labels and relationship-types can be defined arbitrary by users. It has fast deep traversal instead of slow SQL queries that span many tables joins. Graph databases are used in many application domains like Social Networking and Recommendations, Calculating Routes, Network and Cloud Management, Master Data Management, Geospatial, Bioinformatics, Content Management, Security and Access Control

3.2 Drawing the Graph using the Arrow Tool

Since the paper repeatedly stresses on creating person nodes that are separately created with no predefined relationships, I tried to duplicate the same concept with the example learned in the class. The graph nodes

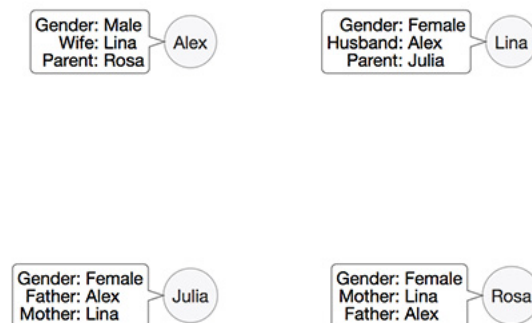


Figure 1: Each node with its properties

3.3 Including the relationships using the properties of the data nodes

After the nodes were created, I defined the relations of each of the nodes with respect to the other nodes.

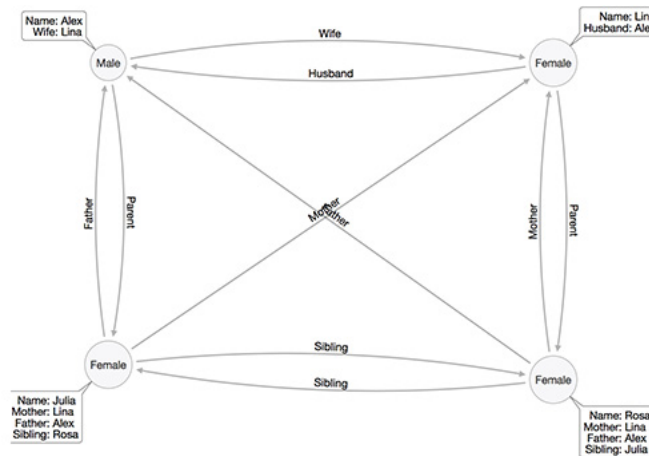


Figure 2: Relations among each of the nodes

3.4 Exporting Cypher to Neo4j Sandbox

The nodes and relations generated using the Arrow tool above can be obtained in the form of a Cypher text. This code is then used as a query in the Neo4j database to obtain the graph.

```

1 CREATE
2   (`0` :Male {Name:'Alex',Wife:'Lina'}) ,
3   (`1` :Female {Name:'Lina',Husband:'Alex'}) ,
4   (`2` :Female {Name:'Julia',Mother:'Lina',Father:'Alex',Sibling:'Rosa'}) ,
5   (`3` :Female {Name:'Rosa',Mother:'Lina',Father:'Alex',Sibling:'Julia'}) ,
6   (`0`)-[:`Wife`]->(`1`),
7   (`0`)-[:`Parent`]->(`2`),
8   (`1`)-[:`Parent`]->(`3`),
9   (`1`)-[:`Husband`]->(`0`),
10  (`3`)-[:`Mother`]->(`1`),
11  (`3`)-[:`Father`]->(`0`),
12  (`2`)-[:`Mother`]->(`1`),
13  (`2`)-[:`Father`]->(`0`),
14  (`2`)-[:`Sibling`]->(`3`),
15  (`3`)-[:`Sibling`]->(`2`)

```

Figure 3: The CypherText obtained from the Arrow tool

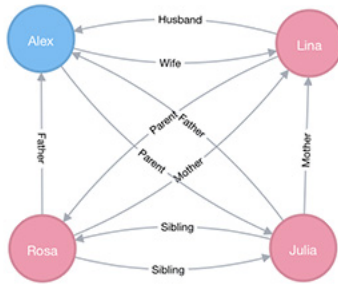


Figure 4: Relations implemented in Neo4j Sandbox

3.5 Querying the Neo4j Graph

After the Cypher text is imported, an entire graph is visible in the Neo4j database for querying. Various relations can be simply found by querying the graph. One of such example is when we try to find all the "Father" relationships. The Neo4j query and output can be seen as follows:

```
MATCH p = () -[r:Father]->() RETURN p LIMIT 25
```

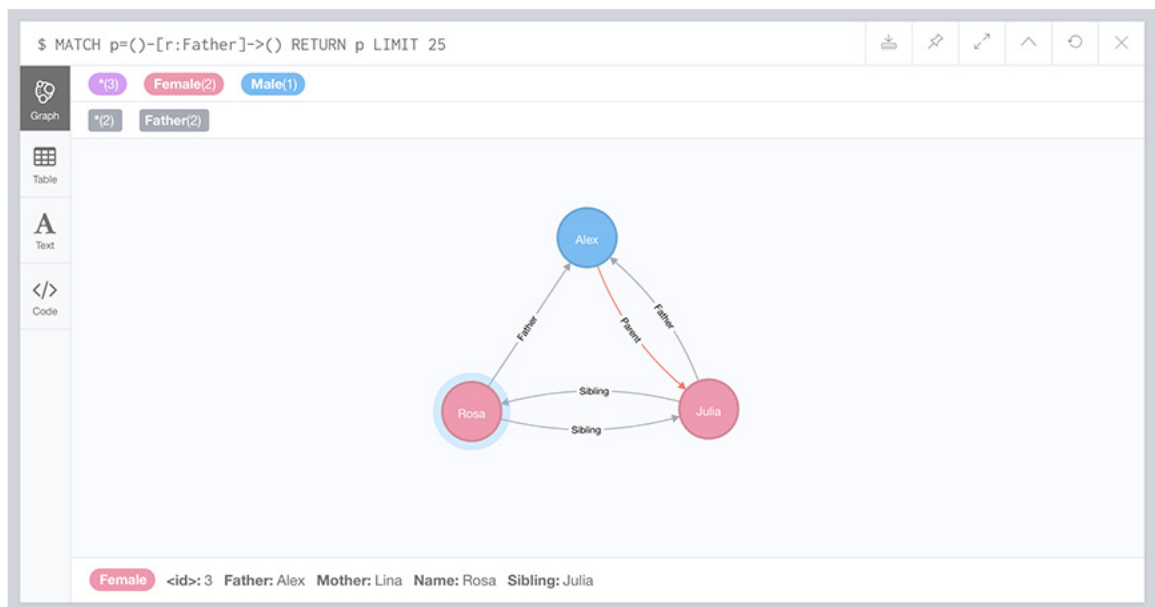


Figure 5: Querying the Neo4j Graph to find all Father relations

4 Extending the Paper

The authors have tried to implement to model the relation ships in the form of graphs so that the traversal Is fast. It justifies the reason of using graphs by stating that it is faster than SQL queries that require joins. It indeed can store complex and dynamic relationships of highly connected data like personnel information. However, the relations between each nodes have to built separately. The effort of building relations for each entity in a large dataset is not practically possible

4.1 Using Prolog for Deductions

The main advantage for using Prolog is that we do not need to create relations for each and every node. Instead, we just need to create a set of rules which will help make the system to make deductions based on rules and fact. Prolog is a general purpose logic programming language in which the program logic is expressed in terms of relations, represented as facts and rules. This language has been used for theorem proving, expert systems, as well as natural language processing. It is well-suited for specific tasks that benefit from rule-based logical queries such as databases searching for finding relations.

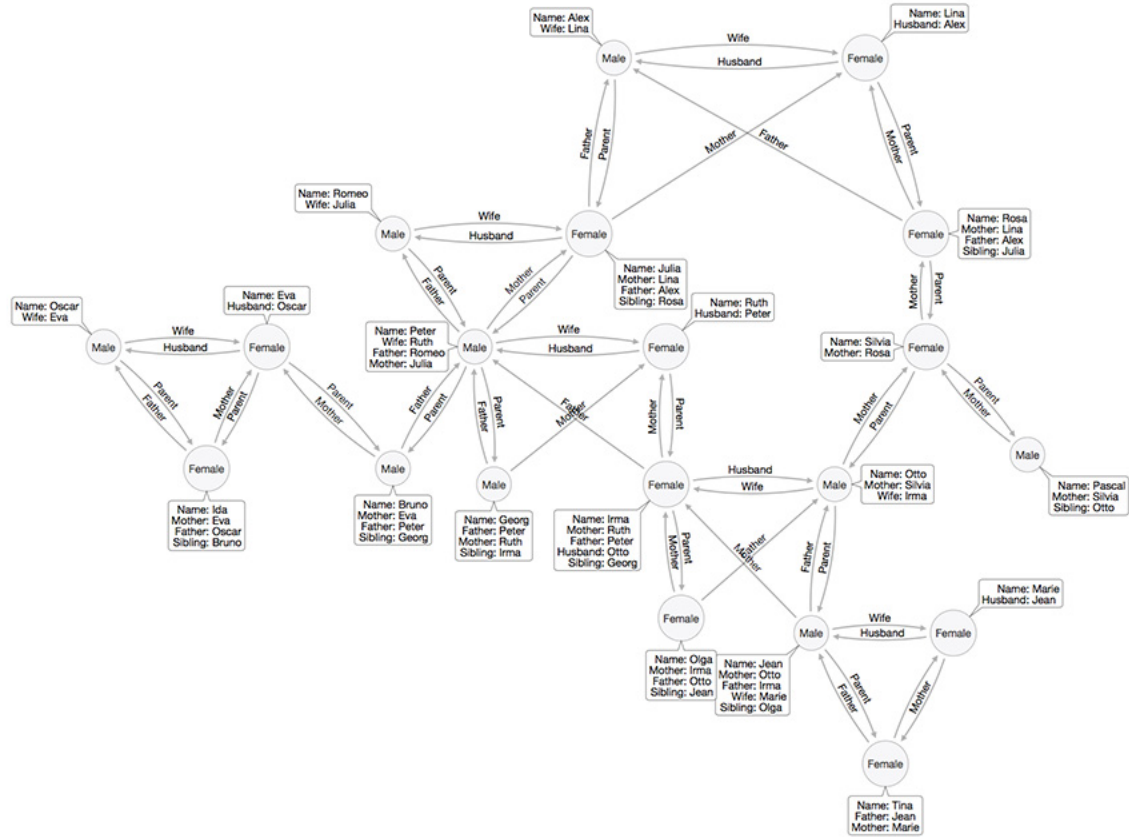


Figure 6: The Entire Family Relations Graph

4.2 Exporting as CypherText

The given graph is made using the Arrow tool. It can be exported as a Cypher Text used in Neo4j graphs. However this time, we will not import this cypher text in Neo4j sandbox but instead run a Python script on it.

```
1 CREATE
2   (`0` :Male {Name:'Alex',Wife:'Lina'}) ,
3   (`1` :Female {Name:'Lina',Husband:'Alex'}) ,
4   (`2` :Female {Name:'Julia',Mother:'Lina',Father:'Alex',Sibling:'Rosa'}) ,
5   (`3` :Female {Name:'Rosa',Mother:'Lina',Father:'Alex',Sibling:'Julia'}) ,
6   (`4` :Male {Name:'Romeo',Wife:'Julia'}) ,
7   (`6` :Female {Name:'Silvia',Mother:'Rosa'}) ,
8   (`7` :Male {Name:'Otto',Mother:'Silvia',Wife:'Irma'}) ,
9   (`9` :Male {Name:'Pascal',Mother:'Silvia',Sibling:'Otto'}) ,
10  (`10` :Male {Name:'Jean',Mother:'Otto',Father:'Irma',Wife:'Marie',Sibling:'Olga'}) ,
11  (`11` :Female {Name:'Marie',Husband:'Jean'}) ,
12  (`12` :Female {Name:'Tina',Father:'Jean',Mother:'Marie'}) ,
13  (`13` :Male {Name:'Peter',Wife:'Ruth',Father:'Romeo',Mother:'Julia'}) ,
14  (`14` :Female {Name:'Ruth',Husband:'Peter'}) ,
15  (`15` :Female {Name:'Irma',Mother:'Ruth',Father:'Peter',Husband:'Otto',Sibling:'Georg'}) ,
16  (`16` :Female {Name:'Olga',Mother:'Irma',Father:'Otto',Sibling:'Jean'}) ,
17  (`17` :Male {Name:'Georg',Father:'Peter',Mother:'Ruth',Sibling:'Irma'}) ,
18  (`18` :Male {Name:'Bruno',Mother:'Eva',Father:'Peter',Sibling:'Georg'}) ,
19  (`19` :Female {Name:'Eva',Husband:'Oscar'}) ,
20  (`20` :Female {Name:'Ida',Mother:'Eva',Father:'Oscar',Sibling:'Bruno'}) ,
21  (`21` :Male {Name:'Oscar',Wife:'Eva'}) ,
```

Figure 7: Cypher Text of the Family Graph

4.3 Converting the Cyphertext to Prolog facts using Python

The obtained cyphertext can be converted to a set of Prolog facts using a Python script. This script performs a regular expression matching and extracts all the functors and constants in a line. We can obtain all the Prolog facts by running the following command:

```
python convert.py -input read.txt -output final.pl
```



```
final.pl
1 male(alex).
2 wife(alex,lina).
3 female(lina).
4 husband(lina,alex).
5 female(julia).
6 mother(julia,lina).
7 father(julia,alex).
8 sibling(julia,rosa).
9 female(rosa).
10 mother(rosa,lina).
11 father(rosa,alex).
12 sibling(rosa,julia).
13 male(romeo).
14 wife(romeo,julia).
15 female(silvia).
```

Figure 8: Prolog Facts obtained after running Python Script

```

1 convert.py
2 import argparse
3 import re
4
5 parser = argparse.ArgumentParser()
6 parser.add_argument('-input', help='training file', required=True)
7 parser.add_argument('-output', help='test file', required=True)
8
9
10 args = vars(parser.parse_args())
11
12
13 inputfile = args['input']
14 outputfile = args['output']
15
16 with open(inputfile) as f:
17     lines = f.readlines()
18
19 #Removing extra spces in the end of each line
20 lines = [x.strip() for x in lines]
21
22
23 file = open(outputfile, "w")
24
25
26 #Clean Each Line and get it in the form of a List
27 for x in lines:
28     eachline = re.split(' |:|{|}|\\[\\]|\\]|\\.|\\.|>', x)
29     eachline = [x for x in eachline if x != '']
30     eachline = [x.lower() for x in eachline]
31     # file.write(''.join(eachline))
32
33     if len(eachline)>3:
34         #print eachline
35         file.write(eachline[1] + "(" + eachline[3] + ")" + "," + "\n")
36         file.write(eachline[4] + "(" + eachline[3] + "," + eachline[5] + ")" + "," + "\n")
37         if len(eachline)>6:
38             file.write(eachline[6] + "(" + eachline[3] + "," + eachline[7] + ")" + "," + "\n")
39             if len(eachline)>8:
40                 file.write(eachline[8] + "(" + eachline[3] + "," + eachline[9] + ")" + "," + "\n")
41                 if len(eachline)>10:
42                     file.write(eachline[10] + "(" + eachline[3] + "," + eachline[11] + ")" + "," + "\n")
43 file.close()
44

```

Figure 9: Python Script to convert Cyphertext to Prolog Facts

4.4 Defining Prolog rules and importing the facts

Finally, after the facts have been extracted from the Neo4j graph, they can be used for making deductions in Prolog. They act as the facts that can be used when the different rules are applied to find relations among people.

```

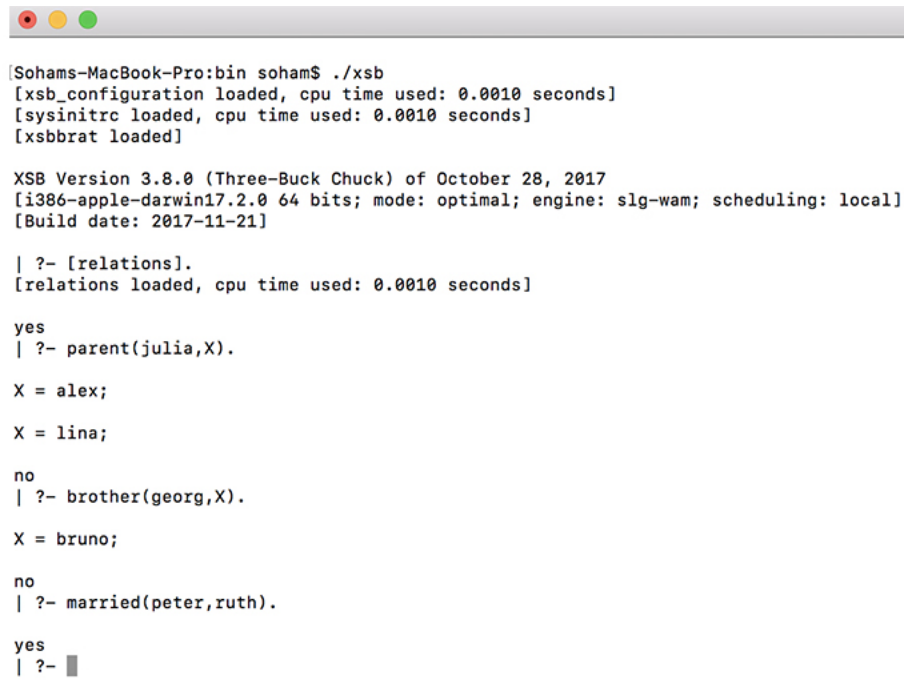
1 relations.pl
2
3 :- include(fact).
4
5 parent(X,Y):-
6     X\=Y,
7     father(Y,X).
8
9 parent(X,Y):-
10    X\=Y,
11    mother(Y,X).
12
13 grandfather(X,Y):-
14    X\=Y,
15    father(X,Z),
16    father(Z,Y).
17
18 brother(X,Y):-
19    parent(Z,X),
20    parent(Z,Y),
21    male(X),
22    X\=Y.
23
24 sister(X,Y):-
25    parent(Z,X),
26    parent(Z,Y),
27    female(X),
28    X\=Y.
29

```

Figure 10: Prolog Rules defined for finding relations

4.5 Testing the Results using Prolog

Finally, after the facts have been extracted from the Neo4j graph, they can be used for making deductions in Prolog. They act as the facts that can be used when the different rules are applied to find relations among people.

A screenshot of a macOS terminal window with a title bar showing red, yellow, and green window control buttons. The terminal displays the execution of the 'xsb' Prolog system. It shows the loading of configuration files, the version of XSB (3.8.0), and the results of several Prolog queries. The queries include loading relations, checking if 'alex' is a parent of 'julia', checking if 'lina' is a brother of 'georg', checking if 'bruno' is married to 'peter', and checking if 'ruth' is married to 'peter'. The terminal output shows 'yes' for the first query and 'no' for the others.

```
[Sohams-MacBook-Pro:bin soham$ ./xsb
[xsb_configuration loaded, cpu time used: 0.0010 seconds]
[sysinitrc loaded, cpu time used: 0.0010 seconds]
[xsbbrat loaded]

XSB Version 3.8.0 (Three-Buck Chuck) of October 28, 2017
[i386-apple-darwin17.2.0 64 bits; mode: optimal; engine: slg-wam; scheduling: local]
[Build date: 2017-11-21]

| ?- [relations].
[relations loaded, cpu time used: 0.0010 seconds]

yes
| ?- parent(julia,X).

X = alex;

X = lina;

no
| ?- brother(georg,X).

X = bruno;

no
| ?- married(peter,ruth).

yes
| ?- 
```

Figure 11: Finding relations using deductive power of Prolog

5 Summary

Hence, the genealogical relationships among large group of people can be efficiently found using Prolog. The approach for using Neo4j graphs adopted by the authors is commendable as it provides very fast graph traversal for searching a query. However, the main advantage for using Prolog is that we do not need to create relations for each and every node. Instead, we just need to create a set of rules which will help make the system to make deductions based on rules and fact.