

Design and Implementation of a Social
Networks Analysis framework
combining the Neo4j Graph Database
and the Python programming Language

Thesis presented by
Gaétan Guru
In order to obtain the
Master's degree in Computer Science Industrial Engineering.

Academic year 2018 - 2019

Cahier des charges relatif au travail de fin d'études de
Gaétan Guru, inscrit en 2^{ème} Master, orientation Informatique

- Année académique : 2018 - 2019
- Titre provisoire : Développement d'un outil d'analyse de réseau sociaux avec Neo4j et R.
- Objectifs à atteindre :
 - Outil de traduction relationnel – orienté graph à l'aide du moteur Neo4j dans le cas d'un réseau social égo-centré
 - Contribution à la mise en place d'un API sécurisé d'importation d'un graphe depuis une tablette vers un serveur sécurisé
 - Intégration d'outils d'analyse SNA depuis le langage R
- Principales étapes :
 - Compréhension de l'outil Morpheus
 - Utilisation avancée moteur Neo4j
 - Apprentissage R et package SNA
 - Travail sur les différents développements
 - Test
 - Rédaction de documentation

Fait en trois exemplaires à Woluwé-Saint-Lambert, le ²⁰~~16~~/11/2018

L'étudiant

Nom – Prénom :
Guru Gaétan

Signature :



Le tuteur

Nom – Prénom :
MARCHANT Nicolas
Département/Unité :

Signature :



Le promoteur

Nom – Prénom :
COMBÉFIS Sébastien
Société :
CEROECAM
Signature :



This work is pursued in the frame of the Egonet research project. One of the objectives of the Egonet project is to automatize the analysis process of social support networks (SSN) using social network analysis (SNA) tools. The goal of this work is to design a tool and implement a prototype that enables researchers to compute several metrics and visualise the results expressively. For the moment, the researchers use the tool almost manually which is slow and not error free. Our solution provides immediate feedback and data centralisation. The set of networks are stored in a Neo4j database, the data is then processed/used through the R programming language. At this instant there are more than 4500 nodes in the database for a total of 381 networks. This work explores alternative ways of computing a metric, comparisons are made between all the different methods. According to the results of the latter, we select the best manner suited for our application.

CERDECAM
ECAM- Brussels Engineering School

Promoter: Sébastien Combéfis,
Department of computer science,
ECAM, CERDECAM,
Brussels, Belgium

Supervisor: Sébastien d'Oreye,
Researcher,
CERDECAM,
Brussels, Belgium

Defence of the thesis: 25 June 2019, Brussels
Copyright: CERDECAM, 2019

A special thanks to Queenie and the research crew for integrating me in the team. I would also like to thank Sébastien Combéfis and Sébastien d'Oreye for the help when I needed it.

Contents

Abstract	3
Introduction	1
1 Company	2
1.1 Introduction	2
1.2 CERDECAM	2
1.2.1 Activities	3
1.2.2 ECAM	3
1.3 Institute of Health and Society (IRSS)	3
1.4 Paul Lambin institute (IPL)	3
2 Social network analysis	3
3 Technology and tools	5
3.1 R	5
3.1.1 Introduction	5
3.1.2 Packages	5
3.2 Neo4j	5
3.2.1 Introduction	5
3.2.2 Context	6
3.2.3 Features	6
3.2.4 Packages	6
3.3 Cypher	7
3.3.1 Introduction	7
3.3.2 Structure	7
3.3.3 Algorithms	8
3.4 Python	8
3.4.1 Introduction	8
3.4.2 Package	8
3.5 Postman	9
4 Project description	10
4.1 Specifications	10
4.2 Context	10
4.3 Implementation	10
4.3.1 Database elaboration using R	10
4.3.2 Metric computation using R	11
4.3.3 Python and Cypher	11
4.3.4 RESTful API	12
4.3.5 Metric research	12

5	Application architecture	13
5.1	Representation	13
5.1.1	Encoding interface (1)	13
5.1.2	Researcher analyser (2)	13
5.1.3	Doctor (3)	13
5.1.4	The API	13
5.1.5	The database	13
5.2	R interface	14
5.3	Python API	14
6	Data model	15
6.1	Generality	15
6.2	Representation	15
6.2.1	Ego	15
6.2.2	Alter	16
6.2.3	Data	16
6.3	Calculations	16
6.4	Outgoing and incoming data	16
7	Metric computation	17
7.1	Generality	17
7.2	Betweenness centrality	17
7.2.1	Formulas	17
7.2.2	Centralisation	17
7.2.3	Example	18
7.2.4	Cypher	20
7.2.5	Interpretation	20
7.3	Ego betweenness	20
7.3.1	Cypher	20
7.3.2	Interpretation	21
7.4	Closeness centrality	21
7.4.1	Formula	21
7.4.2	Example	21
7.4.3	Cypher	22
7.4.4	Interpretation	22
7.5	Fragmentation	23
7.5.1	formula	23
7.5.2	Cypher	23
7.5.3	Interpretation	23
7.6	Diameter	23
7.6.1	Cypher	23
7.6.2	Interpretation	24
7.7	Modularity	24
7.7.1	Formula	24

7.7.2	Cypher	24
7.7.3	Interpretation	24
7.8	Louvain algorithm	24
7.8.1	Cypher	25
7.8.2	Interpretation	25
7.9	Density	25
7.9.1	Formula	26
7.9.2	Example	26
7.9.3	Cypher	26
7.9.4	Interpretation	27
7.10	Discussion	27
8	Classification	27
8.1	Context	27
8.2	Bidart's topology	27
8.2.1	Pear collar	28
8.2.2	Regular dense	28
8.2.3	Centred Dense	28
8.2.4	Centred Star	28
8.2.5	Segmented	28
8.2.6	Dispersed	28
8.2.7	Result	29
9	Coding practice	29
9.1	R	29
9.2	Python	29
9.3	Github	29
10	Results and interpretation	30
10.1	Metric computation	30
10.2	Metric veracity	30
11	Issues	31
11.1	CSV files	31
11.2	Technical issues	31
	Conclusion	31
	Acronyms	32
	Glossary	32
A	JSON	33
B	R documentation	36
C	Documentation Python	39

D	Loading dyads from Neo4j into R	42
E	Loading dyads from Neo4j into R using igraph package	43
F	Timeline	43

Introduction

Background

Severely mentally ill patients require help and support in different aspects of their life. The complex combination of needs results in individual situations that require very specific solutions suited for each person. It has been observed that the care for chronic and complex conditions involves the quality of life that is the living condition, the social integration of the patient.

Most patients with severe, chronic mental illness (Severe Mentally Ill (SMI)) combine several psychic, medical and social issues. In many systems the coordination and continuity of mental-health care services aren't properly provided. Too often the patient is not involved enough in his own treatment which is a set back because patient involvement and personalised care strategies are really important for recovery oriented care. In order to successfully reintegrate a patient in society, the care given to patients has to be reevaluated by :

- improving the coordination between patients and professionals,
- giving more involvement to the patient and the caretaker on the patient's condition,
- adapting each solution to the patient, customise the solution.

In Brussels, the issues of care coordination, patient-centred and personalised care are concerning. A significant part of the population carries the burden related to the combinations of social, medical and psychic vulnerabilities. Brussels has the highest unemployment rate in Belgium and a high risk of exclusion. Symptoms like depression, anxiety, suicidal ideation and suicide risk are noticeable.

The Brussels-Capital Region provides the population with a wide range of social, medical and mental health services. However, despite the important and diversified care supply, the care given to vulnerable people are often inconsistent and poorly adapted to the needs of severe mentally ill patients. The Belgian healthcare system is not optimal in terms of collaboration across care professionals and continuity of care. Consequently, re-admissions in psychiatric hospitals within 30 days after discharge is frequent.

Each patient has a Social Support Network (SSN) which is the set of people who interact with the patient and with each other to provide a support to the patient. Usually people can count on relatives, parents, friends and basically every person with whom someone can connect and interact. However, in this particular case, patients SSN usually include professionals caretakers. The SSN of a patient with chronic and complex needs plays an important role in care continuity, care coordination, personal care and patient involvement.

However, care professionals are not aware of the SSN of a patient and don't use its potential. There is lack of tools and interventions that care professionals might use for identification, mapping and use of the resources from the patient's SSN.

Thus, it is really interesting to map the network on a support. Analysing one support gives relevant data about the configuration of the caretakers of a single patient.

The study of SSNs is still scarce, especially for SMI patients who are not well socially integrated, are vulnerable people suffering from long-term conditions. The background data is provided by the Scientific

intermediate report of January 2019 of the Egonet project.

Aim

Egonet is a research and a development project.

The first goal is to develop a tool that can be used in routine clinical practice in order to explore, map and use the SSN of SMI patients. To this end, Egonet includes the development of a computer application to assist the clinical gather of information. The application must have a user friendly interface for the clinician and the patient. The software contributes in involving the patient into the description of his own social support resources and the organisation of his own care.

The second aim of the project is the development of a database and an analytic module of the SSN data collected with the computer-assisted clinical intervention. This second goal is about analysing the SSN of SMI patients. To this end, the analytic module is foreseen to return notifications to the end-users of the interface with relevant data about the aspects of the SSN.

My work in this project is to contribute for the second aim that is the development of a software able to interact with the database by taking into account the needs of the client. A second part of my work is to search information and implement operations (indicators) that give on overview about what kind of network the researcher is dealing with.

The Morpheus sociogram tool

The Morpheus sociogram is a tool that collects data about a patient's condition and a patient's social support network. This data is used for the preliminary analysis. The tool is developed for a research partner on the Ego network project at UCL. It uses a name generator to collect the SSN data.

1 Company

1.1 Introduction

Egonet is a research and development project funded by the programme Bridge. It involves the partnership of 3 organisations : CERDECAM, IRSS, and IPL.

1.2 CERDECAM

I worked on my thesis at CERDECAM. It is a non profit organisation specialised in help for R&D projects that are in partnership with external companies.

It is a team of four people :

- Queenie Halsberghe, Research Unit Manager,
- Manoel da Silva Casquilho, researcher,
- Guillaume Demaude, researcher,
- Sébastien d'Oreye, researcher.

Each researcher works on an independent project and collaborates with a teacher from ECAM on the project. Sébastien d'Oreye is the scientist in charge of the Egonet project at CERDECAM.

1.2.1 Activities

CERDECAM is involved in the training of students. Indeed, many students, like myself, come to perform an internship or a thesis.

Research is performed for the different projects.

1.2.2 ECAM

CERDECAM is in a close relationship with Brussels Engineering School ECAM.

1.3 Institute of Health and Society (IRSS)

The team leader coordinates the project Egonet. The team is specialised in sociological research.

1.4 Paul Lambin institute (IPL)

This organisation is in charge of the front end part of the application.

2 Social network analysis

Graph basics

A *graph* $G = (V, E)$ is an abstract object composed of a set V of vertices and a set E of edges that join pairs of vertices.

$V(G)$ denotes the vertex set and $E(G)$ denotes the edge set of a graph G . The cardinality of V is usually denoted by n and the cardinality E is usually denoted by m .

Graphs can either be directed or undirected. In the latter, the order of the endvertices of an edge is not important. An undirected edge joining vertices $u, v \in V$ is written $\{u, v\}$. A directed graph is composed of edges with an origin and a destination denoted (u, v) [10].

There are graphs where endvertices are bounded by multiple edges, those are called multigraphs. Graphs where each endvertices are bounded by a single edge are called simple graphs. An edge whose endvertices are the same vertex is called a loop[10].

It is possible to associate a numerical value to an edge and a vertex of a graph. The weight of an edge can be described as a function $\omega : E \rightarrow \mathbb{R}$ [10].

This value gives additional information about the relation which is different according to the use case :

- a cost,
- a capacity,
- strength of interaction,
- similarity.

An unweighted graph has unit edge weight, $\forall e \in E, w(e) = 1$.

A graph $G' = (V', E')$ is a subgraph of the graph $G = (V, E)$ if $V' \subseteq V$ and $E' \subseteq E$.

2 types of subgraphs can be highlighted :

- (vertex-)inducted subgraph,
- (edge-)inducted subgraph[10].

A subgraph is a vertex induced subgraph only if E' contains all edges $e \in E$ that join vertices in V' . The induced subgraph of $G = (V, E)$ with vertex set $V' \subseteq V$ is denoted by $G[V']$.

A subgraph is an edge induced subgraph with edge set $E' \subseteq E$, denoted by $G[E']$, is the subgraph $G' = (V', E')$ of G , where V' is the set of all vertices in V that are the endvertices of the edges in E' [10].

Below are examples of induced subgraphs :

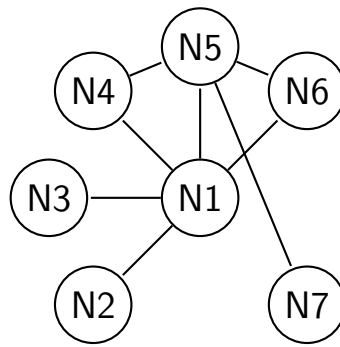
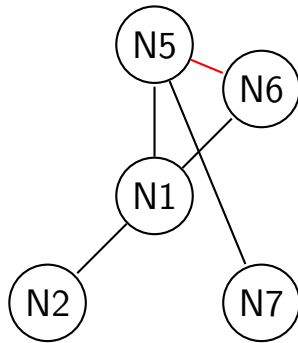
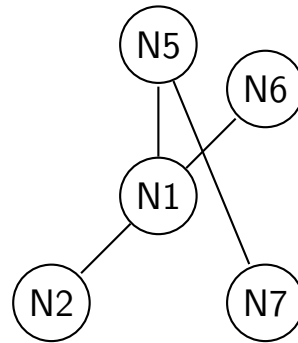


Figure 1: network



(a) Vertex induced subgraph example



(b) Edge induced subgraph example

Figure 2: Example of induced subgraph

Figure 2a describes a vertex induced subgraph from Figure 1 and Figure 2b is an edge induced subgraph from Figure 1.

3 Technology and tools

3.1 R

3.1.1 Introduction

R is a statistical programming language specialised in data analysis and statistical computation. R is available as Free Software under the terms of the Free Software Foundation's GNU General Public License in source code form and is actively maintained. R is used for data manipulation, statistical modeling, and graphics. R has multiple structures with which data can be processed. These have a specific form for storing and organising data. Below are the basic datastructure :

Vector A data structure that contains the same type of element.

Matrix A two dimensional data structure.

List Similar to a vector but can contain several type of data.

Dataframe Similar to a matrix but can store several type of data within each row and column.

3.1.2 Packages

The package *rneo4j* is used to interact with the Neo4J database. Many fundamental functions such as creating a node are default functions included in the package. It is also possible to create a custom query with Cypher, typically to extract data.

The package *igraph* is used for network analysis. This package contains SNA functions, these only require the graph built from the dyads. A graph can also be plotted using the 'plot' function which is convenient.

An example of representation below using the plot function:

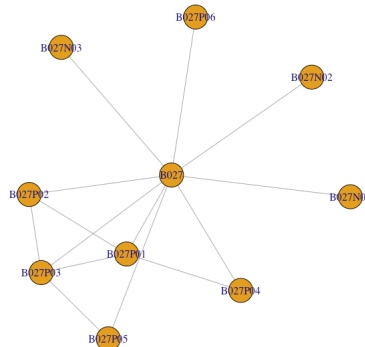


Figure 3: Graph representation using *igraph* package

3.2 Neo4j

3.2.1 Introduction

Neo4j is a NoSQL graph oriented database that stores the data using nodes and relations. A node is categorised using one or many labels. A relation is categorised using only one label.

The Neo4j Community Edition software is open source. However a number of commercial licensed versions are available.

3.2.2 Context

Relational database management systems (RDBMS) or relational databases are a common tool used for typical business environments. A user can extract and insert data into the database by using SQL (Structured Query Language) queries which is supposed to be the standard way of communication. Nowadays incoming and outgoing data from the web is almost always unstructured and is a large data set not convenient for RDBMS.

3.2.3 Features

NoSQL databases is a new approach that can process such incoming data. It has four primary independent technologies :

- Graph database,
- Key-value databases,
- Document databases,
- Column family databases.

For graph databases, data is stored in property graphs which is similar to an entity-relationship diagram. This paradigm uses powerful objects that describe a database's data elements and their connections. Here are the most important characteristics :

- Nodes are used to represent entities. A node can be a person, a service for example. Each entity can have a single or many labels attached to it. A label describes the role of an entity in the database.
- Relationships are named objects that describe how the entities relate to each other.
- Nodes and relationships can contain properties and metadata using key-value pairs.
- Constraints can be applied on nodes and/or relationships to enforce data integrity, existence and uniqueness requirements.

A relation always has a start node, an end node, a direction and a type. Relationships can have properties related to them that in most cases express a quantitative measure such as weights, costs, distances, ratings, time intervals or strengths. Nodes can share any number or type of relationships without sacrificing performance. Relations always have a direction but it is possible to consider a path as undirected while analysing.

3.2.4 Packages

Creation, deletion and updates are available by default while using Neo4j. However, for this application, more complex behaviour is required. There are a few packages available to extend the Neo4j Graph platform functionalities.

The package *Graph Algorithms Library* is a library that provides efficiently implemented graph algorithms for Neo4j using Cypher procedures. The package is directly installed in Neo4j.

The library contains implementations for the following types of algorithms:

- Path Finding : these algorithms help find the shortest or evaluate the availability of routes,
- Community Detection : these algorithms evaluate how a group is clustered or partitioned,
- Centrality : these algorithms evaluates the importance of distinct nodes in a network,
- Similarity : these algorithms help calculate the similarity of nodes.

The similarity algorithms are not currently used in this application.

3.3 Cypher

This largely inspired by *An introduction to Cypher* from Neo4j's website.

3.3.1 Introduction

Cypher is a declarative graph query language that allows for expressive and efficient querying and updating of the graph store. Complicated database queries can easily be expressed through Cypher. As mentioned, Cypher is a declarative language meaning that it focuses on the clarity of expressing what to retrieve from a graph and not how to retrieve it.

3.3.2 Structure

Cypher borrows its structure from SQL and queries are built using various clauses. These are chained together, and they feed intermediate result sets between each other.

Here are a few clauses used to read from the graph:

- MATCH : The graph pattern to match. This the most common way to get data from the graph.
- RETURN : What to return from the current context.
- WHERE : Adds constraints to a pattern.

The following clauses are used to update the graph:

- CREATE : Create nodes and relationships.
- SET : Set values to properties and add labels on nodes.

Example :

```
MATCH (n:PATIENT) WHERE n.code = "B001" RETURN n
```

The query above starts by looking for all the nodes that have the label PATIENT. A condition is summoned on the subset of nodes previously trimmed in order to only retrieve the nodes that have the value 'B001' for the attribute 'code'. The node is then returned to the user.

Example :

```
MATCH (n)-[r:KNOWS]->( ) RETURN n
```

The example above uses the label of a relation in order to restrict the number of results, only the nodes that have a relationship 'KNOWS' with other nodes are returned. This is a fundamental way to extract information from the database using Cypher.

3.3.3 Algorithms

The library *Graph Algorithms Library* discussed in the Neo4j subsection supports two approaches for loading projected graphs :

- Label and relationship-type projection,
- Cypher projection.

Both methods involve a CALL clause followed by the desired algorithm. The first method :

```
CALL algo.<name>("NodeLabel", "RelationshipType", {config})
```

This method only involves the label of a node and the label of a relationship.

The second method :

```
1 CALL algo.<name>(
2   "MATCH (n) RETURN id(n) AS id",
3   "MATCH (n)-->(m) RETURN id(n) AS source, id(m) AS target",
4   {graph: "cypher"})
```

The first MATCH returns a list of nodes ids. The second query returns all the endvertices (source - target) that are in the projected graph from the first query. According to the use case, one of the two queries is used for the computation of an algorithm, this is **developed** in Section 7 of this paper.

3.4 Python

3.4.1 Introduction

Python is an interpreted, high level, general purpose programming language.

3.4.2 Package

The package *neo4j* is used in order to communicate with the database. The package *flask* is used to create a server, it is a micro-framework for Python based on Werkzeug and Jinja 2.

3.5 Postman

Postman is a tool that simplifies the development of APIs. This software is used to test the various requests that the application's API provides. Another benefit of this tool is that the queries are saved and properly classified into the right project folder.

4 Project description

4.1 Specifications

This work has multiple objectives :

- Contribution to the elaboration of a secured tool that imports a graph from a tablet to a secured server,
- Integration of a SNA tool from R,
- Elaboration of a SNA tool using Python and Cypher queries,
- Redaction of a documentation.

4.2 Context

I started working on this project in November 2018. Two technologies were already assigned to the project before I began implementing the solution, R and Neo4j. The solution was elaborated from scratch. At first, the relevant data was only located in CSV files. The clinicians use CSV files to encode data gathered during the interviews with patients. These contain information about the patients, the alters, the patient's health, self harm tendencies, etc.

4.3 Implementation

4.3.1 Database elaboration using R

My first task was to build the database using the CSV files as a source of information. R is used as programming language. The data must be loaded from the CSV files into R for processing.

First, the CSV is transformed into a dataframe. From there onward, the data can easily be extracted by selecting a column. I started by creating the patients nodes in the database. Then I implemented a script that creates the alters in the datastore as well as the relations ego–alter and alter–alter. Along side the implementation was the elaboration of an interface. This module is introduced in subsection 5.2.

The interface is made of functions that interact with the database. An example of function

```
newNode = createNode(graph, label, attributeList)
```

In this case a new node is created with the corresponding graph object, a certain label and a list of attributes attached to the node. This script is straightforward.

However, there are more complicated operations. The line below describes such a case :

```
1 query <- paste("MATCH(n {name:'", nodeName,"'}) RETURN n", sep="")
2 n = getSingleNode(graph, query)
```

In this script, a node is returned using the *getSingleNode* function. However, there must be a Cypher query that points out what to extract, in this case it is a node with property 'name' equal to 'nodeName'.

The documentation of the R interface can be found in the appendix. For this part of my work, I had to get use to R and Neo4j because I was not familiarised yet.

4.3.2 Metric computation using R

My second task was to incorporate Bidart's classification in the R software. It involves the computation of 4 metrics. This classification is explained in Section 8.2 of this report.

The desired network is first loaded from Neo4j in R by creating dyads out of the SSN, resulting in the creation of a dataframe of dyads. The dataframe is then transformed using the *graph.data.frame* function from the *igraph* package into a graph. From there onward, the metrics can be computed using that graph. Below are these different metrics :

```
1   gBetw <- centr_betw(g)\$centralization
2   gModul <- modularity(cluster, membership(cluster))
3   gDens <- edge_density(g, loops=F)
4   gDiam <- diameter(g)
5   gNumber <- length(alterns)
```

where *g* is the graph.

4.3.3 Python and Cypher

Furthermore, I started doing the measurements using Python. Instead of loading the network, the metrics were partly computed using the Neo4j engine. This is possible using Cypher queries which is the query language for Neo4j databases. More information is available about Cypher in Section 3.

An interface was developed alongside the main implementation. The interface is presented in subsection 5.3 of this paper. This was done in order to see which method is the fastest. However in many cases Cypher provided an algorithm that would give an intermediate result. In those situations, data processing is done on those intermediate results.

Another time consuming aspect of this work is to search how does a metric operates on a network and understand what the measure returns.

The interface itself was constantly improved while the rest of the software was implemented. An example of this evolution would be the label addressed to the relations between alters of a same network. At first, these had a custom name to ease the suppression of a network. The problem with this configuration is that a relation can only have one label unlike a node. I modified the interface to suppress these relations only according to the ID of the ego. This enhances scalability since the next developer could use a label for relations to extend the application.

The complete documentation of the Python's interface can be found in appendix C.

The script implemented for the indicators is discussed in Section 7 of this paper. Every queries have a variable that is the ID of a node. An ID is unique and a default classifier for each node, using this identifier is thus the best approach to search a node in the database.

4.3.4 RESTful API

Finally, I developed an API using Python. This API is part of the global architecture of the Egonet project. Its purpose is to respond to HTTP requests and easily communicate with a standard JavaScript Object Notation (JSON) formatted data stream. I implemented a server that uses the package *flask*. The routes are found in Section 5.

A module is implemented in order to process the incoming or outgoing JSON formatted data. There aren't any function that easily inserts a JSON into Neo4j and vice versa. I did try to use the built in Neo4j package *apoc* and its *apoc.json* functions. However, these were too limited for the distinct use cases. The nodes extracted from the JSON stream are given an attribute 'path' that contains every key in which the final value is located.

```
1  {
2      "Interview":{
3          "medical":{
4              "HoNOS":{
5                  "hallucinate": "3",
6                  ...
7              }
8          }
9      }
10 }
```

The example above is from the JSON stream and describes the tendency of a patient to be subject to hallucinations[11]. A node with the labels "Interview", "medical", "HoNOS" is created in the database. The key and value, which in this case are "hallucinate" and "3", are attributes in the node created above. In this case the 'path' attribute is "Interview:medical:HoNOS". As long as the value of a key is not a dictionary or a list, the pair are attributes in a node.

The path is integrated as attribute in the node to rebuild the JSON with ease when a user extracts data.

4.3.5 Metric research

Along side the implementation, I was looking for new metrics to compute until the end of April. Most of the details concerning those metrics were found in books regarding either social network analysis or graph theory.

5 Application architecture

5.1 Representation

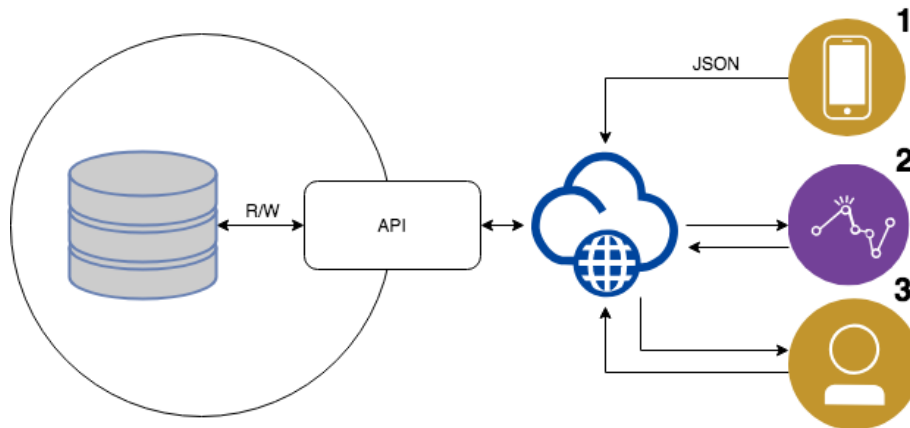


Figure 4: Global architecture

Figure 4 represents the application's global architecture. The frontend and the back end are clearly connected through the internet. Depending on the request and the user's clearance, three interactions are possible. At the moment data is only accessible for the clinician, the SMI patient.

5.1.1 Encoding interface (1)

The encoding interface is a user interface used to insert a new SSN and data gathered during the interview into the database. The encoded data is first arranged in a JSON format then is transferred through the internet towards the database.

5.1.2 Researcher analyser (2)

This is an analytical module that uses metrics to compute a result. This analytic module automatically detects the variables of interest when the indicators and outcomes present values that attracts attention. The researcher displays or inserts information to categorise a graph.

5.1.3 Doctor (3)

The doctor has access to all of his patients medical data.

5.1.4 The API

The API has a RESTful architecture and transfers/retrieves data safely from the database. It uses Python and is described in subsection 5.3.

5.1.5 The database

The database is a NoSQL graph oriented database.

My work is focused on the API and the Neo4j database for the development part of this project.

5.2 R interface

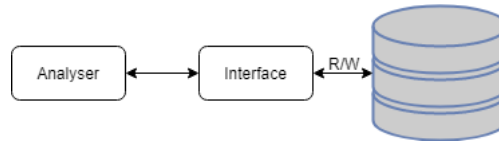


Figure 5: R interface

Communication with the database is done through an interface. The interface uses functions from the package `rneo4j`. Custom cypher commands are used in case a specific behaviour is desired. R is not yet used in the global architecture of the project, only Python is used for the moment.

5.3 Python API

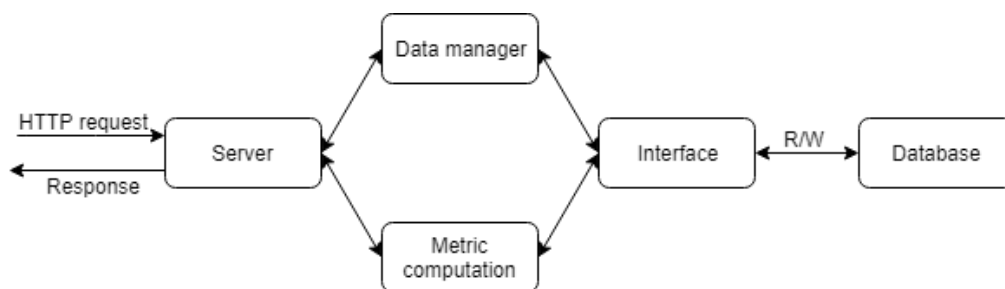


Figure 6: Architecture

Reading or writing data is done through the API. It is composed of a server, a data manager, a metric computer and an interface.

The server uses a RESTful configuration to receive or send data. The content sent or received is mostly JSON formatted. The package Flask is used to create the server.

The data manager processes incoming data from the web or from the database. This is done according to the type of request. It uses the API to insert data or retrieve data into/from the database. One typical task of this module is to insert a JSON formatted data stream into the database and vice versa.

The interface has a set of functions that interacts with the database. It contains metric computations that allow a researcher to obtain relevant information regarding a selected network. Functions for creating nodes and relationships are also available.

The metric computer computes the indicators for the desired network.

Modules are added with ease between the interface and the server.

API requests

- `POST /api/input` : Receives a JSON data stream containing all the data gathered during an interview with a patient. This query either creates or overwrites the data in the database.

- GET /api/egonet/<id> : This query receives a patient code as parameter and returns the complete interview of a patient.
- GET /api/services : Returns a JSON with all the services available in the database. Every services have an id and a name.
- GET /api/service/users/<id> : This query receives a service id in parameter and returns all the doctors attached to that service in a JSON.
- GET /api/patients : Receives a doctor code in parameter and returns all the patients treated by that doctor in a JSON.
- GET /api/login : Enter the correct credentials to login to the database.
- GET /api/metrics/all : This request returns all the metrics for all the patients in a JSON.
- GET /api/metrics/<id> : Receives a patient code as parameter and returns the metrics of that patient.

6 Data model

6.1 Generality

The patient is called Ego and his relatives are called Alter in the database.

6.2 Representation

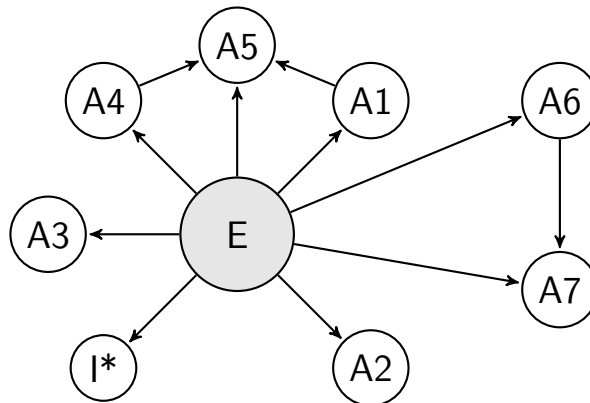


Figure 7: Network configuration.

The network in Figure 7 illustrates a patient's network. This graph is a general case, all networks don't have the same topology.

6.2.1 Ego

The central node in Figure 7 is *E*. It represents the Ego in this network. Those nodes have the label 'PATIENT' in the database.

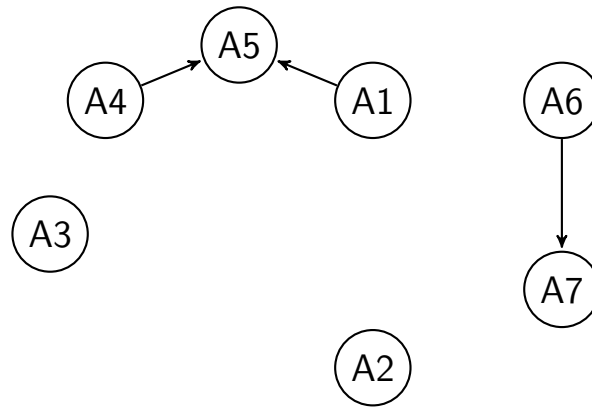


Figure 8: Network configuration without ego

6.2.2 Alter

An Alter represent a person who takes care of the Ego. They all have a main label which is 'ALTER', followed by a second label corresponding to the function in the network. For example, an alter could have a label 'psychiatrist'. These actors are either professionals, i.e. caretakers, or non professional, i.e. relatives and other. Alters are represented in the Figure 7 with an *A*.

6.2.3 Data

There might be interviews involving the patient and a clinician in order to summarise what the patient experiences in his current network. Data is collected to conclude the interview and then saved in the database. The 'Interview' vertices are labelled I^* where the star represents one or many. These nodes are removed while analysing the SSN because these are not alters.

6.3 Calculations

In this work the Ego node is usually removed from the graph in order to analyse the patient's SSN.

The SSN of the network in Figure 7 is represented below: The indicators used in this project are discussed in Section 7.

6.4 Outgoing and incoming data

The interview is sent trough the internet using a JSON formatted data stream. It is then processed in order to be inserted in Neo4j. If there is a GET request from the user, the data is extracted and processed into a JSON formatted data stream. It is then sent through the internet to the GUI of the application.

The JSON built from an interview contains all the topological elements of the network as well as information about the patient's mental state.

7 Metric computation

7.1 Generality

The graphs are treated as undirected that is the user can decide which is the end node and which is the begin node while analysing a path.

V and E are unweighted that is all the edges have the same value which is 1, $\forall e \in E$ the weight $w(e) = 1$.

The SSNs are vertex-induced subgraphs.

7.2 Betweenness centrality

Betweenness centrality measures the intent to which a node tends to be on a geodesic between any pair of nodes in the network. A high betweenness value for a node indicates that the node is central in the network. A betweenness of 0 indicates a node is on no geodesic in the graph. The minimum value is zero in case where the vertex n_i doesn't fall on any geodesic. The maximum value is $(n-1)(n-2)2^{-1}$ which is the number of pairs of actors not including n_i .

7.2.1 Formulas

In order to compute the betweenness[10] of each node, the following formula is used :

$$C_B(i) = \sum_{\substack{j < k \\ i \neq k, j}} \frac{g_{jk}(i)}{g_{jk}} \quad (1)$$

where

- $g_{jk}(i)$ is the number of geodesics connecting nodes j and k through i ,
- g_{jk} is the total number of geodesics between nodes j and k ,
- $\forall g \in V'$.

7.2.2 Centralisation

Equation (1) computes the betweenness of a single node. It is possible to obtain a measurement of betweenness for the whole graph. This operation is known as *graph centralisation*.

Graph centralisation allows a researcher to compare different networks according to the heterogeneity of the betweenness of the members of the networks.

The measure of centrality at the graph level using the theoretical maximum H :

$$H = (g-1)C_{Bmax} \quad (2)$$

where

$$C_{Bmax} = \frac{(g-1)(g-2)}{2} \quad (3)$$

where C_{Bmax} is equivalent to the betweenness in the reconfigured graph G^* using the same number of

nodes as in $G'[9]$.

The centrality of graph G' is computed using the following formula :

$$C_B = \sum_{i=1}^{|v|} \frac{C_B(n^*) - C_B(n_i)}{H} \quad (4)$$

where

- $C_B(n^*)$ is the betweenness of node n^* which is the highest in graph G' ,
- $C_B(n_i)$ is the betweenness of node n_i in the graph.

7.2.3 Example

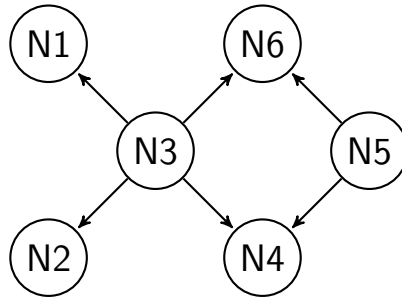


Figure 9: Network example

Below are examples of computation of the betweenness of two nodes using equation (1) from the graph above starting with node N5.

N5

Path	g_{jk}	$g_{jk}(i)$
(N_1, N_2)	1	0
(N_1, N_6)	1	0
(N_1, N_4)	1	0
(N_2, N_6)	1	0
(N_2, N_4)	1	0
(N_4, N_6)	2	1

The betweenness of node N5 using equation (1) can be seen below.

$$C_B(i) = \frac{0}{1} + \frac{0}{1} + \frac{0}{1} + \frac{0}{1} + \frac{0}{1} + \frac{1}{2} = 0.5$$

N3

Path	g_{jk}	$g_{jk}(i)$
(N_1, N_2)	1	1
(N_1, N_6)	1	1
(N_1, N_4)	1	1
(N_1, N_5)	2	2
(N_2, N_6)	1	1
(N_2, N_4)	1	1
(N_2, N_5)	2	2
(N_4, N_6)	2	1

The betweenness of node N3 using equation (1) can be seen below.

$$C_B(i) = \frac{1}{1} + \frac{1}{1} + \frac{1}{1} + \frac{2}{2} + \frac{1}{1} + \frac{1}{1} + \frac{2}{2} + \frac{1}{2} = 7.5$$

Following are the values of betweenness for all the other nodes in the graph of Figure 9. The computation is done through Neo4j desktop.

name	centrality
"N3"	7.5
"N4"	1.5
"N6"	1.5
"N5"	0.5
"N1"	0.0
"N2"	0.0

Figure 10: Betweenness centrality of all nodes

The arrangement with the highest betweenness is when the graph is a star as shown in Figure 11. Using formula (2) :

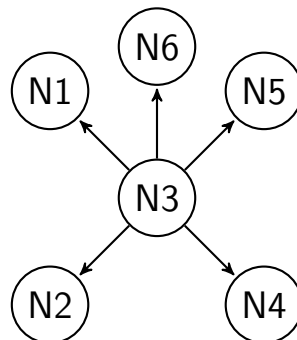


Figure 11: Star configuration

$$H = \frac{(6-1)^2 * (6-2)}{2} = 50$$

In Figure 10 the highest betweenness is equal to 7.5.

Using formula (4) :

$$\frac{(7.5 - 7.5) + (7.5 - 1.5) + (7.5 - 1.5) + (7.5 - 0.5) + (7.5 - 0) * 2}{50} = 0.68$$

The betweenness centralisation is thus equal to 0.68.

7.2.4 Cypher

There is a betweenness algorithm available in Neo4j's Graph algorithm package. This algorithm displays the betweenness of all nodes from the SSN of one patient's network.

```

1 CALL algo.betweenness.stream("
2 MATCH p=(n:PATIENT {name:'PatientName'})-[r]-(n2)
3 RETURN id(n2) AS id","
4 MATCH (n)-[r]-(n2) RETURN id(n) AS source, id(n2) AS target",
5 {graph:"cypher", direction:"both"})
6 YIELD nodeId, centrality
7 MATCH (user) WHERE id(user) = nodeId
8 RETURN LABELS(user) AS function, user.name AS name, centrality
9 ORDER BY centrality DESC;
```

The query above gives a list of values. However, in this case the graph centralisation is needed, thus a Python code manages the data returned from the query. It follows the same steps as the metric computation above.

7.2.5 Interpretation

The betweenness indicator measures the importance of a node in the network. If the betweenness is high for a node that means the vertex is significant because a lot of information passes through him.

Graph centralisation gives an overview of the typology of the network. If the indicator is large, it means the network tends to have a few nodes that have a high betweenness.

7.3 Ego betweenness

The previous metric analyses the betweenness centrality and the graph centralisation of a patient's SSN. The ego betweenness metric only evaluates the centrality of the ego node.

7.3.1 Cypher

The algorithm used for this application is the same as in subsection 7.2. The difference is that there is only one result returned which is the betweenness centrality of the patient's node.

7.3.2 Interpretation

The purpose of this metric is to measure the patient's involvement and autonomy in his own network. As the result increases, the patient involvement increases as well which means the individual communicates with the different care takers. There are situations where the patient is his own care coordinator, for example : ensuring consistency of intervention, contacting new people to meet new needs,etc[11].

Despite those benefits, this indicator is really meaningful when the patient's state is known. Indeed, for a person able to care for himself this approach could have a very positive outcome on his own treatment. However, if the patient is psychotic it would be appropriate to have a low ego betweenness as the patient wouldn't probably be able to fend for himself in this matter[7].

7.4 Closeness centrality

The closeness metric is an effective way to detect nodes that spread information through the graph efficiently. It does so by measuring the average farness which is the inverse of the distance. Nodes with high closeness have the shortest distance to all other nodes[6].

The measure focuses on how close an actor is to all other actor in the set of actors V' of graph G' . In this case, a node has importance if it can quickly interact with a lot of nodes of graph G' . Centrality is a concept that is inversely related to distance. As a node grows farther apart in distance from other nodes, its centrality decreases because of the number of relation between the pairs of nodes[9].

7.4.1 Formula

The closeness[10] of a node :

$$C_c(n_i) = \left[\sum_{j=1}^g d(n_i, n_j) \right]^{-1} \quad (5)$$

where $d(n_i, n_j)$ is the distance between n_i and n_j .

The maximum value is attained when the node is adjacent to all the other nodes in the graph, that value is equal to $(g-1)^{-1}$. The minimum value is reached when a node is disconnected from the graph, the closeness value for that node is 0. In order to compare the closeness value of a node with another value in another network the normalised closeness formula is used as it has a maximum value equal to one [9]:

$$C'_c(n_i) = (g-1)C_c(n_i) \quad (6)$$

where g is the number of nodes.

7.4.2 Example

In Figure 12, $N1$ has the highest closeness value because that node is adjacent to all the other nodes making it more 'central' in the graph.

$N1$ and $N2$ are directly connected but they are both not adjacent to $N4$. Finally, $N4$ is the further apart from the most vertices and thus has the smallest closeness value.

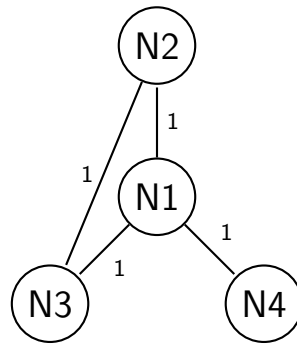


Figure 12: Closeness centrality example

N1 value :

$$C_c(n_i) = (1 + 1 + 1)^{-1} = \frac{1}{3}$$

N1, N2 value :

$$C_c(n_i) = (1 + 1 + 2)^{-1} = \frac{1}{4}$$

N4 value :

$$C_c(n_i) = (1 + 2 + 2)^{-1} = \frac{1}{5}$$

7.4.3 Cypher

There is a closeness algorithm available in Neo4j's Graph algorithm package. This algorithm displays the closeness of all nodes from the SSN of one patient's network.

```

1  CALL algo.closeness.stream("
2  MATCH p=(n:PATIENT {{name:'NameOfNode'}})-[r]-(n2)
3  RETURN id(n2) AS id', 'MATCH (n)-[r]-(n2)
4  RETURN id(n) AS source, id(n2) AS target",
5  {graph:"cypher", direction:"both"})
6  YIELD nodeId, centrality
7  MATCH (user) WHERE id(user) = nodeId
8  RETURN LABELS(user) AS function, user.name AS name, centrality
9  ORDER BY centrality DESC;

```

The query above returns a list of values that doesn't need any processing since only the closeness of each node is required.

7.4.4 Interpretation

Care takers who have a high closeness centrality in the network are very productive in communicating data since they are directly in relation with other actors.

7.5 Fragmentation

The fragmentation of a graph is the proportion of pair of nodes that can not reach each other.

7.5.1 formula

$$F = \frac{n_{sp=inf} * 2}{n * (n - 1)} \quad (7)$$

where $n_{sp=inf}$ is the number of relations separated with an infinite distance (disconnected).

7.5.2 Cypher

There aren't any default algorithms available in the package that can compute the fragmentation. This measure can partly be computed using the number of shortest paths in the graph as showed in formula (7).

```

1  CALL algo.allShortestPaths.stream("cost", {{
2  nodeQuery:"
3  MATCH (n {name:'NameOfNode'})--(n2) RETURN id(n2) AS id",
4  relationshipQuery:"
5  MATCH (n)-[r]-(p) RETURN id(n) AS source, id(p) AS target, r.cost AS weight",
6  graph:"cypher", defaultValue:1.0}})
7  YIELD sourceNodeId, targetNodeId, distance
8  MATCH (source) WHERE id(source) = sourceNodeId
9  MATCH (target) WHERE id(target) = targetNodeId
10 WITH source, target, distance
11 WHERE source <> target
12 RETURN source.name, target.name, distance
13 ORDER BY distance DESC

```

The result is then processed by a Python script to finally get the fragmentation of a patient's SSN.

7.5.3 Interpretation

The fragmentation of a SSN returns a value according to the cohesion between alters.

7.6 Diameter

The diameter metric of a graph returns the longest shortest path of a graph. If the graph is unconnected the diameter is infinite.

7.6.1 Cypher

The result is obtained using the same shortest path algorithm computed for the fragmentation metric. The value returned from that algorithm is a list of shortest paths with the first being the greatest. Only the first value is used from the list.

7.6.2 Interpretation

The diameter of a network gives an idea of how the network is spread out or how compact a graph is. The distance between two actors will never be greater than that of the diameter.

7.7 Modularity

The modularity metric gives an overview of the division into communities of the graph. The accuracy of the communities division is improved while the value of this metric increases.

7.7.1 Formula

In order to obtain this metric for the graph, the formula below[3] has to be computed :

$$Q = \frac{1}{2m} * \sum \left[\left(A_{ij} - \frac{k_i * k_j}{2m} \right) * \Delta(c_i, c_j), i, j \right] \quad (8)$$

where

- m is the number of relations in the graph,
- A_{ij} item in the matrix,
- k_i, k_j degrees of row i and j,

Δ is computed like so :

$$\Delta(c_i, c_j) = \begin{cases} 1 & \text{if } c_i = c_j \\ 0 & \text{if } c_i \neq c_j \end{cases} \quad (9)$$

where c stands for community.

7.7.2 Cypher

There aren't any default algorithms available in the package that can compute the modularity. The Louvain algorithm discussed in subsection 7.8 of this section gives the number of communities present in the patient's SSN. This result is used in a Python script that computes formula (8).

7.7.3 Interpretation

This metric returns a value that describes the 'facility' to distinguish communities.

7.8 Louvain algorithm

The Louvain algorithm returns the number of communities present in a graph and to which community an alter belongs to.

The analysis is done by iterating two consecutive steps many times.

The first step is completed on each node of the network. A node is removed from its current community and replaced in the community of one of its neighbours. Modularity changes are performed on each node's neighbours. If none of these modularity changes are positive, the node stays in its current community.

On the other hand, the node is moved in a community where changes in modularity are the best. This algorithm is applied until there is no need to move a node in another community.

The second step is completed on communities. The newly formed communities from the previous step are represented with nodes. The relations between the lower level nodes of two communities is represented with a single weighted relationship. This step creates a coarse-grained network. These are applied until step one generates no change.

7.8.1 Cypher

The Louvain algorithm is fully available in the graph package. Two versions have been implemented in the interface.

The first streams a node and its community number in the SSN.

```

1     CALL algo.louvain.stream("
2     MATCH p=(n:PATIENT {name:'NameOfNode'})-[r]-(n2)
3     RETURN id(n2) AS id", "MATCH (n)-[r]-(n2)
4     RETURN id(n) AS source, id(n2) AS target",
5     {graph:"cypher", direction:"both"})
6     YIELD nodeId, community
7     MATCH (user) WHERE id(user) = nodeId
8     RETURN LABELS(user) AS function, user.name AS name, community
9     ORDER BY community DESC;
```

The second returns the number of communities present the SSN.

```

1     CALL algo.louvain("
2     MATCH p=(n:PATIENT {name:'NameOfNode'})-[r]-(n2)
3     RETURN id(n2) AS id", "
4     MATCH (n)-[r]-(n2)
5     RETURN id(n) AS source, id(n2) AS target",
6     {graph:"cypher", direction:"both"});
```

7.8.2 Interpretation

This algorithm returns of how the alters regrouped in the SSN and how many communities are present. A patient with many communities is probably in a better state than a patient who has only one community in his SSN which is probably composed exclusively of care takers.

7.9 Density

Graph density is the ratio of actual edges to the maximum possible edges. A fully connected graph, a dense graph, will have every node connected to every other node.

A graph that has a low number of edges compared to the theoretical maximum number of edges is a sparse graph[7].

7.9.1 Formula

Since the density indicator relies on the number of nodes of a network, the maximum value is equal to $g(g-1)/2$, i.e. the maximum pair of nodes possible in the graph.

The density Δ [10] of a network with the number of edges E and the number of nodes g in the graph is equal to :

$$\Delta = \frac{2E}{g(g-1)} \quad (10)$$

7.9.2 Example

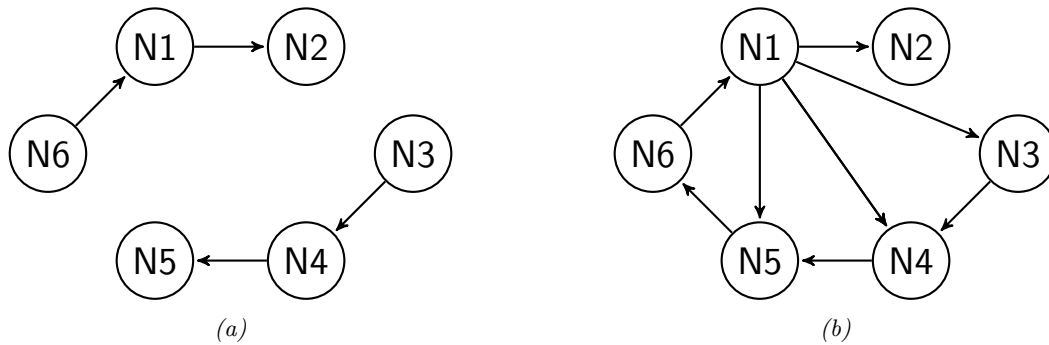


Figure 13: Density example

The density of Figure 13a using formula (10) is

$$\Delta = \frac{2*4}{6*5} = \frac{4}{15} \approx 0.27$$

The density of Figure 13b is

$$\Delta = \frac{2*8}{6*5} = \frac{8}{15} \approx 0.53$$

Without calculations it is obvious that Figure 13b has a higher density than Figure 13a. However, this is just a simple example, real SSNs are much more complex to visualise.

7.9.3 Cypher

There aren't any default algorithms available in the package that can compute the density of a network. However, the density metric is directly dependent of the number of relations in the SSN. This is fairly easy to compute, no algorithms are involved to get information from the database.

```

1 MATCH (n {name:"NameOfNode"})--(alters)
2 MATCH (alters)-[r:KNOWS_ALTER_OF_NameOfNode]->()
3 RETURN count(r)

```

This only involves a plain Cypher command, no 'CALL algo.<metric-name>'.

7.9.4 Interpretation

The density indicator describes how well the alters are connected to each other. If the density is large, the coordination in the patient's SSN is enhanced since many alters are aware of what is been done in other parts of the network.

7.10 Discussion

The indicators used for this work are all dependent of the network's topology. Despite that fact, a user can not come to a conclusion about the topology of a network according to the value of a single indicator. For example, if all the alters are related the density is one and the betweenness centrality is zero. However, if there aren't any dyads in the SSN, the density is zero and the betweenness would still be zero. In this case, the betweenness centrality does not give relevant data about the structure of the network. A network can not be categorise according to one indicator alone because it returns a value to a very specific aspect of the network.

The definitions of these metrics were found mostly in *Network Analysis: Methodological Foundations*[10].

8 Classification

8.1 Context

The classification of a network requires the comparison between the topology of that network with a reference topology. Many indicators (metrics) are available to describe a network, they help categorising a network systematically. The indicators provide very specific sociological information that contribute to the understanding of the multiple ways in which a person's social circle gets structured[1].

The classification of networks was not part of my work for this thesis. I did mention a type of classification below and a result of this classification. It gives an overview of the purpose of my work. The research team of Egonet project did the classification and came up with a result, I am exposing what they found.

8.2 Bidart's topology

The following analysis is based on the *Personal networks typologies: A structural approach* article[1]. The latter focuses on personal networks which is interesting for this application.

Bidart's classification was the first used in this project. The result is obtained by computing the betweenness, the modularity, the density and the diameter of a graph. A description of the network's topology is provided according to those 4 metrics.

This classification focuses exclusively on the structure characteristics of the network putting aside people or relationships characteristics. Below is the decision tree enabling the classification of a network.

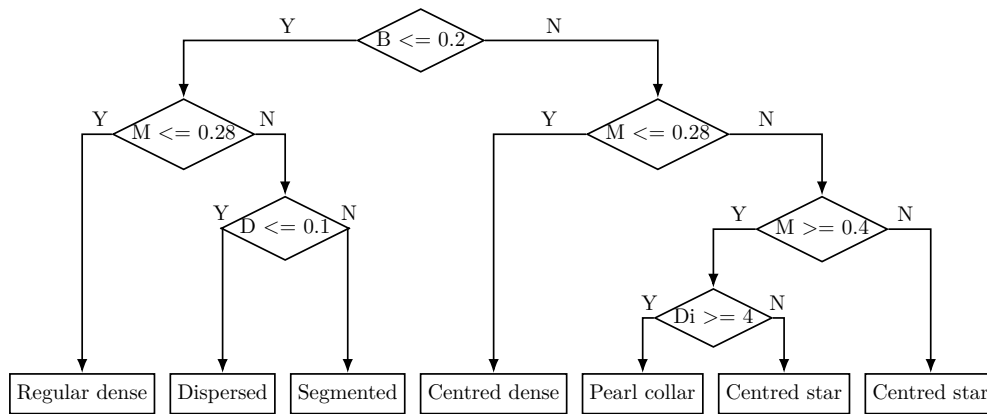


Figure 14: Bidart's classification

8.2.1 Pear collar

This is an elongated graph where the diameter is relatively high and the values on the other indicators are not significant. In this case, the topology is supposedly made of long chains of ties.

8.2.2 Regular dense

The network is highly and quite regularly connected, sometimes it can be depicted as a dense core and a few isolated points. Its density index is high but the index of betweenness centralisation is low.

8.2.3 Centred Dense

The general shape of this network looks alike to the regular dense topology. However, the difference is that it has a high betweenness centralisation index. That means there is a central alter connected to most of the other nodes.

8.2.4 Centred Star

Network that is centred but has several small parts and a lower density than the centred dense topology. It looks like a star that is it has a heart and branches.

8.2.5 Segmented

This network is distributed within several relatively large and dense components and some isolated points.

8.2.6 Dispersed

This network appears very fragmented into small groups and isolated individuals.

8.2.7 Result

The results obtained with this practice aren't reliable because of many incorrect classifications. My supervisor, Sébastien d'Oreye, is the one who found these results.

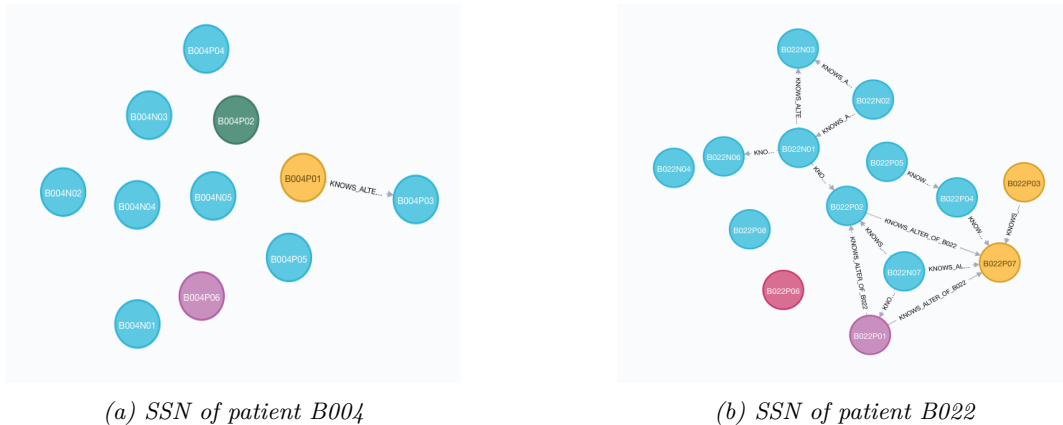


Figure 15: Example misclassification

For Figure 15a, the algorithm classified this network as Regular Dense which is not correct, the graph is sparsely connected. This network is clearly dispersed since most nodes are isolated.

For Figure 15b, the algorithm believes this network is segmented. However, since there is only one connected subgraph in this network and a few isolated components this graph is more alike a Pearl Collar graph than segmented. Another observation is that this network has a diameter equal to five which quite high compared to all the other networks and fits the theoretical description of a Pearl Collar network.

An unpublished article is written by a part of the Egonet team about this result[11].

9 Coding practice

9.1 R

For R, I used the Google style guide.

9.2 Python

For Python I used the PEP 8 coding style. I also used the Python package Pylint in order to validate my code. Each run provides a score that increases or decreases according to the previous run thus informing about the evolution of quality.

9.3 Github

Every time a significant version was completed, I saved it on Github. I made as many updates of the interfaces as possible since it is the basic support of my application.

10 Results and interpretation

10.1 Metric computation

Two implementations of metric computation have been done, one in Python and one in R. Both algorithms operate on the same database. The purpose of this redundancy is to understand which of these two methods is the fastest.

For the moment, the implementation in Python seems to be the fastest with an execution time of 76.38 seconds for all the networks in the database involving the computation of nine metrics. The R script execution time is equal to 178.8 seconds and it involves the computation of four metrics for all the networks. The execution time using R is larger than the Python algorithm execution time for less than half the metrics which is astonishing.

However, the R script might not be implemented optimally. When loading the graph from the database, I created dyads using four 'for' loops with complexity $\mathcal{O}(n^2)$ which is time consuming. A much simpler solution is to load the dyads using Cypher commands. The solution involving the loop is found in appendix D and the optimised solution is found in appendix E. Both of these methods use function `graph.data.frame` from package `igraph` to return a graph object from the data frame containing the dyads. The graph is then used to compute the metrics. I loaded the dyads with the algorithm of complexity $\mathcal{O}(n^2)$ because I was still used to working with a relational data structure.

10.2 Metric veracity

In order to ensure that the metrics computed using Python are correct, a comparison is done between the results from the formal packages used in R and the algorithm used in Python. These comparisons are only done for the metrics involved in Bidart's classification. There are few distinctions between the modularity metric as shown in Figure 16

	Patient	Modul Python	Modul R
1	44	0.4612500	0.47125000
2	81	0.1244444	0.11777778
3	164	0.1130401	0.10146605
4	178	0.1224490	0.07142857
5	223	0.3730159	0.33900227
6	282	0.1179982	0.11524334
7	362	0.3571429	0.34693878

Figure 16: Distinctions modularity computation

The table above depicts the modularity metric using the algorithm in Python and the algorithm in R. The two values are given for a patient ID. For the moment, there aren't any explanations for these inequalities.

11 Issues

11.1 CSV files

In the very beginning of this work, the integrity of the data was compromised. Indeed, there were a lot of errors in the name of the different actors especially the alters (they are more numerous). These mistakes are human related, the clinician entered the data.

At first I wanted to modify the software to make a stop when there was an error. This solution involves marking the current position of the iterator somewhere in the CSV file. That was not appropriate because if the user changes the CSV file after an error occurred, the networks saved in the database might end up corrupted. The best approach was to check the integrity of the data in the CSV files before running the main code and make the appropriate corrections.

11.2 Technical issues

I started using the server at cerdecam in order to save some time while executing the program. It wasn't easy to install the latest version of R. Indeed, I needed the version 3.5 to use the package *rneo4j*, only the version 3.4.4 was available for Linux Mint through command line.

I also have an issue with the Neo4j engine on my computer. When I want to compute all the indicators for all the networks in the database using Python, Neo4j crashes. I resolved that issue by working on the server at Cerdecam.

Conclusion

This work contributed in the development of a framework to interact with and compute metrics from a NoSQL graph oriented database.

Several improvements could be done for the interface in R. Indeed, most of the request match rely on attributes which is not a good practice. For the interface to be as less specific as possible the search queries should rely exclusively on the node's ID. Every node in the database has an ID and it is unique thus this is the best way to stay generic.

A missing aspect is encryption. This project uses and saves medical data about patients and professionals. Some of these information are delicate. This can be done in Python using packages like *pyAesCrypt*.

The analysis of networks is only based on the typology of these networks, there are other ways of measuring the care's efficiency of a network. For example a weight could be added for relations that could depict the status of relationship, i.e. a good on or a bad one.

Another idea would be to consider some of the node's attributes to build up a stronger case about the patient's state. Indeed, the attributes are extracted from the JSON data stream when saving the interview in the database. This data contains information according to the patient mental state and for the moment it is not considered for the support network analysis of patients.

Acronyms

JSON JavaScript Object Notation. 12

SMI Severe Mentally Ill. 1

SSN Social Support Network. 1

Glossary

alter A person in the SSN of the patient. 10

attribute Synonym of property. 8

cardinality Dimension of a set (number of elements). 3

degree Number of relations a node shares. 24

dictionary Implementation of a data structure that is more generally known as an associative array. A dictionary consists of a collection of key-value pairs. Each key-value pair maps the key to its associated value. 12

dyad Two people tied together in the network. 5

edge A relation in a graph. 3

endvertex Two vertices joined by an edge. They are adjacent and neighbours. 3

geodesic Path representing the shortest path between two nodes. 17

indicator Synonym of metric. 2

metric Measurement done on a graph in order to evaluate a network according. 3

Neo4j NoSQL oriented graph database. 3

NoSQL Stands for "not only SQL" is an alternative to traditional relational databases in which data is placed in tables and data schema is carefully designed before the database is built. 5

property Metadata associated with a node. For example a node that has a property 'surname' equal to 'John' and a property 'last-name' equal to 'Doe'. 6

relation Two linked nodes are people who know each other. Synonym of tie. 5

RESTful Good practice that uses HTTP requests to GET, PUT, POST and DELETE data for interacting with an API. 5, 12, 13

vertex A node in a graph. 3

References

- [1] CLAIRE BIDART, ALAIN DEGENNE, M. G. Personal networks typologies: A structural approach. *Social Networks* 54, 1 (2018), 1–11.
- [2] DEVELOPPER, N. Properties: The heart of graph databases. <https://neo4j.com/business-edge/properties-the-heart-of-graph-databases/>. Accessed: 2019-05-19.
- [3] DEVELOPPERS, R. Modularity of a community structure of a graph. <https://igraph.org/r/doc/modularity.igraph.html>. Accessed: 2019-03-15.
- [4] DRAKOPOULOS, G., BAROUTIADI, A., AND MEGALOOIKONOMOU, V. Higher order graph centrality measures for neo4j.
- [5] FRANÇOIS WYNGAERDEN, PABLO NICAISE, H. G. S. D. D. L. S. C., AND LORAN, V. Does the structure of the personal networks of patients with severe mental illness differ from the common types found in the general population? Unpublished Manuscript, 2019.
- [6] HODLER, M. N. . A. E. Graph algorithms in neo4j: Closeness centrality. <https://neo4j.com/blog/graph-algorithms-neo4j-closeness-centrality/>. Accessed: 2019-02-15.
- [7] RICHARD BRATH, D. J. *Graph Analysis and Visualization: Discovering Business Opportunity in Linked Data*, 1 ed. Wiley, January 30 2015.
- [8] SLUZKI, C. E. Personal social networks and health: Conceptual and clinical implications of their reciprocal impact. *Families, Systems & Health* 28, 1 (2010), 1–18.
- [9] STANLEY WASSERMAN, K. F. *Social Network Analysis: Methods and Applications (Structural Analysis in the Social Sciences)*. Cambridge University Press; 1 edition, November 25, 1994.
- [10] ULRIC BRANDES, T. E. *Network Analysis: Methodological Foundations*. Springer, March 24, 2005.
- [11] WYNGAERDEN, F., NICAISE, P., DUBOIS, V., AND LORANT, V. Social support network and continuity of care: an ego-network study of psychiatric service users. *Social Psychiatry and Psychiatric Epidemiology* (02 2019).

A JSON

```

1   {
2   "Interview":{
3     "ego":{
4       "hospitalisation": [
5
6       ]
7     },
8     "alters": [
9       {
10        "code": "P1",
11        "category": "pro"

```

```
12     },
13     {
14         "code": "P2",
15         "category": "pro"
16     },
17     {
18         "code": "P3",
19         "category": "pro"
20     },
21     {
22         "code": "N1",
23         "category": "nonpro"
24     },
25     {
26         "code": "P4",
27         "category": "pro"
28     },
29     {
30         "code": "N2",
31         "category": "nonpro"
32     },
33     {
34         "code": "N3",
35         "category": "nonpro"
36     }
37 ],
38 "dyads": [
39
40 ],
41 "satisfaction": {
42     "netSupportSatis": "1",
43     "netExchangeSatis": "1",
44     "mentalHealthSatis": "1",
45     "physicalHealthSatis": "1",
46     "jobSatis": "1",
47     "housingSatis": "1",
48     "funSatis": "1",
49     "familySatis": "1",
50     "securitySatis": "1",
51     "careSatis": "1",
52     "practicalHelpSatis": "1",
53     "professionnalSatis": "1",
54     "friendsSatis": "1"
```

```
55     },
56     "additional":{
57         "moreMentalHealth":true,
58         "morePhysicalHealth":true,
59         "moreJob":true,
60         "moreHousing":true,
61         "moreFun":true,
62         "moreFamily":true
63     },
64     "data":{
65         "admission":"2019-04-19T22:00:00.000Z",
66         "gender":"ApacheHelicopter",
67         "DoB":"2019-04-10T22:00:00.000Z",
68         "work":"1",
69         "wage":"2",
70         "housing":"0"
71     },
72     "medical":{
73         "honos":{
74             "hyperactive":"0",
75             "selfharm":"3",
76             "drugs":"2",
77             "cognitive":"4",
78             "physical":"3",
79             "hallucinate":"3",
80             "depression":"3",
81             "other":"4",
82             "activities":"4",
83             "lifestyle":"4",
84             "daily":"4",
85             "social":"4"
86         },
87         "patient":{
88             "code":"B003"
89         },
90         "diagnose":"F0"
91     },
92     "firstContact":"2019-04-26T22:00:00.000Z",
93     "serviceContact":{
94         "psychiatrist":1,
95         "otherDoc":2,
96         "socialWorker":1,
97         "homeCare":1,
```

```

98         "socialServ":1,
99         "somHosp":1
100     }
101 }
102 }

```

B R documentation

ConnectToDatabase

```

# Creates the connection with the database and returns the graph.
With Neo4j Desktop # there is only one server running and the last only makes available one database.
#
# Args:
# url:the url to reach the server, default port :7474,
# username:username to access the database,
# password:password to access the database,
# with no missing values: #
# Return:
# An object containing a representation of the entire database.
The database is one huge # graph: #
# Errors: # If the server is not on, a 'Connection refused' error is thrown.
User has to make sure # that a server is running at the specified url in order to connect to the database:

```

ModifyQuery

```

# Reformat string query to avoid error: #
# Args:
# queryString:the query that has to be executed: # with no missing values: #
# Return:
# The query string that has been modified:

```

AddNewIndividual

```

# Add a new node in the database: #
# Args:
# graph:represents the graph in the database that we wish to modify: # nameIndividual:the name of
the new node,
# typeOfNode:specify the label,
# attributes:all other information except the name of the node,
# attributes can be a missing value: #
# Return:
# The new node:

```

SetValueAttribute

```
# Set the value of one node attribute: #
# Args:
# graph:represents all the graph in the database: # attributeName:the name of the attribute to change,
# name:the name of the node,
# attribute:the value to insert,
# with no missing values: #
# Return:
#
```

SetValueAttributes

```
# Set the value of many attributes: #
# Args:
# graph:represents all the graph in the database,
# attributeName:the name of the attribute to change,
# listOfName:a list of node names,
# listAttributeValue:a list with different attribute values: # with no missing values: #
# Return:
#
```

DeleteIndividual

```
# Suppress a node from the database: #
# Args:
# graph:represents all the graph in the database:+ # typeOfNode:the type of node to suppress,
# name:name of the node to suppress,
# typeOfNode can be a missing value: #
# Return:
#
```

RetrieveIndividual

```
# Retrieve a node from the database.
#
# Args:
# graph:represents all the graph in the database: # typeOfNode:the type of node to retrieve,
# name:name of the node to retrieve,
# typeOfNode can be a missing value: #
# Return:
# The node that is required:+(error,
)
```

RetrieveNodeName

```
# Returns the name of each node having the same label: #
# Args:
# graph:represents all the graph in the database: # typeOfNode:the type of node,
# with no missing values: #
# Return:
# A list of nodes that have the same label:
```

AlterOfEgo

```
# Returns a list filled with the name of each alter attached to one patient: #
# Args:
# graph:represents all the graph in the database: # patient:the patient for whom we desire to know the
alter: # with no missing values: #
# Return:
# A list of nodes name:
```

CreateRelationship

```
# Create a relationship between 2 nodes: #
# Args:
# individual1:first node of the relationship,
# relation:the name of the relationship,
# individual2:second node of the relationship,
# with no missing values: #
# Return:
# The newly created relation:
```

CheckIfRelationExists

```
# Check if a relation exists between 2 nodes: #
# Args:
# individual1:first node of the relationship,
# individual2:second node of the relationship,
# with no missing values: #
# Return:
# If the evaluation exists:
```

DeleteRelationship

```
# Delete a relation: #
# Args:
# name1:first name of the relationship,
```

```
# name2:second name of the relationship: # with no missing values: #
# Return:
```

GetRelOfAlter

```
# Get all the nodes name to whom the alter is connected: #
# Args:
# graph:represents all the graph in the database,
# alter:a node,
# relation:the relation of 2 nodes: # with no missing values: #
# Return:
# A list to whom that alter is connected:
```

C Documentation Python

This module enables us to communicate with the database in order to extract information. The latter could be a metric, the number of alter(s) connected to one node. Cypher queries are used to interact with the database.

```
betweenness_node(tx_object, id_patient)
```

This function returns the betweenness of each alter connected to one node.

```
:param tx_object:
:param id_patient: int
:return: list [[[...], ..., ...]]
```

```
betweenness_one_node(tx_object, n_id)
```

This function returns betweenness of one node.

```
:param tx_object: run object
:param n_id: string, the name of the node
:return: float
```

```
close_driver(driver)
```

Close the connection to the database.

```
:param driver: object of type GraphDatabase
:return: Nothing
```

```
closeness_node(tx_object, id_patient)
```

This function returns the closeness of each alter connected to one node.

```
:param tx_object:
:param id_patient: int
:return: list
```

```
create_node(tx_object, label)
```

This function creates a node. If the vertex has a label PATIENT then the node **is** unique.

:param tx_object:
:param label: string, label of the new node
:return: int, id of the new node.

create_relation(tx_object, n_id1, n_id2, label)

This function creates a relation between 2 nodes.

:param tx_object:
:param n_id1: string, id of node 1
:param n_id2: string, id of node 2
:param label: string, label of relation
:return:

database_driver(uri, user, password)

Needed information to connect to the database.

Documentation : <https://neo4j.com/docs/api/python-driver/current/driver.html>

:param uri: string
:param user: string
:param password: string
:return: object Driver. The instances of the latter are used **as** primary access point to Neo4j.

degree_of_alter(tx_object, ego_id)

This function returns the number of relations every alters has.

:param tx_object:
:param ego_id: int
:return:

delete_network(tx_object, n_id)

This function deletes the whole network of a patient.

:param tx_object: a session
:param n_id: string, the ID of the patient to suppress
:return:

get_neighbours(tx_object, id_node)

This function gives us the alters of a node.

:param tx_object:
:param id_node: int
:return: list[[{ }, []],...]

louvain_algorithm_graph(tx_object, id_patient)

This function returns how many communities are **in** a graph.

:param tx_object:
:param id_patient: int

`:return: list`

`louvain_algorithm_node(tx_object, id_patient)`

This function returns to which community number belongs a node.

`:param tx_object:`

`:param id_patient: int`

`:return: list`

`node_exists(tx_object, label, attribute_name, attribute_value)`

This function returns `if` the node exist `in` the database.

`:param tx_object:`

`:param label:`

`:param attribute_name:`

`:param attribute_value:`

`:return:`

`node_name_type(tx_object, type_of_node)`

This function returns the names of the nodes `with` a same label

`:param tx_object: object`

`:param type_of_node: string`

`:param value either 0 or 1. If 0 list of node names is returned,`

`else a list of nodes is returned.`

`:return: list[[...],[...]], containing all the PROPERTIES of the selected node.`

`number_of_relations(tx_object, ego_id)`

This function returns the number of relations between alters of an ego.

`:param tx_object:`

`:param ego_id: string`

`:return: list, number of relations between alter`

`pair_of_nodes(tx_object, ego_id)`

This function returns `all` the pair of alters.

`:param tx_object:`

`:param ego_id: int`

`:return: list`

`set_attribute(tx_object, n_id, attribute_name, attribute_value)`

This function sets an attribute of a node.

`:param tx_object:`

`:param n_id: node id`

`:param attribute_name: string, the name of the existing/new attribute`

`:param attribute_value: string, the value of the attribute`

`:return:`

```
shortest_path_count(tx_object, id_patient)
```

This function returns the number of relations **for** the shortest path between 2 nodes.

:param tx_object: **object** related to an opened session

:param id_patient: **int**

:return: **list**

```
union_find(tx_object, patient_name)
```

This function returns the number of **set** of connected nodes **or** unitary node **in** a network.

:param tx_object: **object** related to an opened session

:param patient_name: **string**

:return: **list**

D Loading dyads from Neo4j into R

```
1 Neo4jToDyad <- function(graph, patient){
2   alter = AlterOfEgo(graph, patient)
3   if(length(alter)==0){
4     # print(paste("No alter found for ego", patient))
5     return(NULL)
6   }
7   x <- alter
8   egoNetworks <- data.frame(matrix(ncol=length(x), nrow=length(x)))
9   colnames(egoNetworks) <- unlist(x, use.names=FALSE)
10  rownames(egoNetworks) <- unlist(x, use.names=FALSE)
11  egoNetworks[is.na(egoNetworks)] <- 0
12  ## Build alter matrix
13  for(item in alter){
14    l <- GetRelOfAlter(graph, item)
15    for(sub in l){
16      if(egoNetworks[sub[[1]], item[[1]] != 1){
17        egoNetworks[item[[1]], sub[[1]] <- 1
18      }
19    }
20  }
21  dyadFrame <- data.frame(matrix(ncol=2, nrow=0))
22  x <- c("from", "to")
23  colnames(dyadFrame) <- x
24  ## Association of Ego and Alter
25  for(i in 1:length(alter)){
26    dyadFrame[nrow(dyadFrame)+1,] <- list(patient, alter[[i]])
27    numberOfAlter <- c(which(egoNetworks[i]=="1"))
28    ## Association between alter
```

```
29   for(j in numberOfAlter){
30     dyadFrame[nrow(dyadFrame)+1,] <- list(alter[[i]], rownames(egoNetworks)[j])
31   }
32 }
33 return(dyadFrame)
34 }
```

E Loading dyads from Neo4j into R using igraph package

```
1   query = "MATCH (n)<--(n2:ALTER)
2   WHERE ID(n) = 4914
3   RETURN n.name as From, n2.name as To"
4
5   data = cypher(graph, query)
6
7   query = "MATCH (n)<--(n2:ALTER)
8   WHERE ID(n) = 4914
9   MATCH (n2)-->(n3:ALTER)
10  RETURN n2.name as From, n3.name as To"
11
12  data1 = cypher(graph, query)
13  data2 = merge(data, data1, all=TRUE)
14
15  g1 <- graph.data.frame(data2, directed = F)
16  g <- delete.vertices(g1, as.character(patient))
```

F Timeline

November	• Egonet project overview
	• Basic training in Neo4j and R completed
December	• Insert all data from CSV to Neo4j database using R
	• Integration of SNA tools in R
January	• Cypher algorithm overview
February	• Python API V1
	• R documentation
	• Research of metric
	• Implementation of metric
	• Cypher queries
March	• Research of metric
	• Cypher queries
April	• Import and export JSON
	• Python API

Table 1: Timeline