# Messung_analyse

lucmiaz

13 November 2015

## Analysis of rail lubrification

### Reproducing the calculations

The following calculations are done by calling data stored in a neo4j graph database from R. To insert data in this database, first use `main_analysis.py` to generate a serie of json file containing all the useful info on your new data. Then use the code `toDB.py` to import these data in the neo4j database – before running the script, please read the head lines and make the necessary changes described there.

In R, the database is called as follows – don't forget to launch the neo4j interface and start the local server if necessary :

```
library(RNeo4j)#call RNeo4j package
graph=startGraph('http://localhost:7474/db/data', username='neo4j', password='admin')#open a
#with username 'neo4j' and password 'admin'
```

### Structure of the database

The databases in Neo4j are graphs, containing nodes and relationships between these nodes. Nodes and relationships have types (or labels). This will differentiate between Passings and Algorithms, in the case of Nodes, and between the relation of seeing (Passing saw this type of train) and the relation of taking place there (Passing took place here).

We have structured our database - as depicted on figure – to contain the following Node Types – Properties names are in *italic*:

- **Algorithms**
  - *Name*

  - *class*
  - *dt* for delta time used in discretisation
  - *overlap* used in the STFT
  - *fmin* lower bound for low band
  - *fmax* upper bound for high band
  - *fc* threshold between low and high bands
  - *noiseType* (detected by the algorithm)
  - *threshold* applied to low/high band ratio

- **Passings** (i.e. measurement characteristics of one train passage)
  - *Name*
  - *Measurement*
  - *Temperature*
  - *Humidity*
  - *direction*
  - *rain*
  - *Wind*
  - *axleProLength*
  - *Location*
  - *v1*
  - *v2*

- **MicMes** (i.e. the measurements values recorded by one microphone at one passing)
  - *Name*
  - *micN* number of this microphone
  - *dt*
  - *Tb* start time
  - *Te* end time
  - *Tp_b* start time masked
  - *Tp_e* end time masked
  - *tNoise* Period in s. where noise was detected
  - *tNoisemasked* Period in s. where noise was detected (masked between Tp_ b and Tp_e)
  - *TEL*
  - *Location*

- **TrainType**
  - *Name*

- **Location**
  - *Name* It contains the following Relationships – the example is the canonical use of the relationship :

- **IN** (a Passing took place IN a Location)
  - *Time*

  - *Date*
  - *Track*

- **SAW** (a Passing SAW a TrainType)
  - *Time*
  - *Date*
  - *TrainLength*
  - *Track*
- **ISANEVALOF** (a MicMes is an evaluation of a Passing)
  - *tEvalmasked* total length of the masked recording
  - *tEval* total length of the recording
- **WASEVALWITH** (a MicMes was evaluated with an Algorithm)

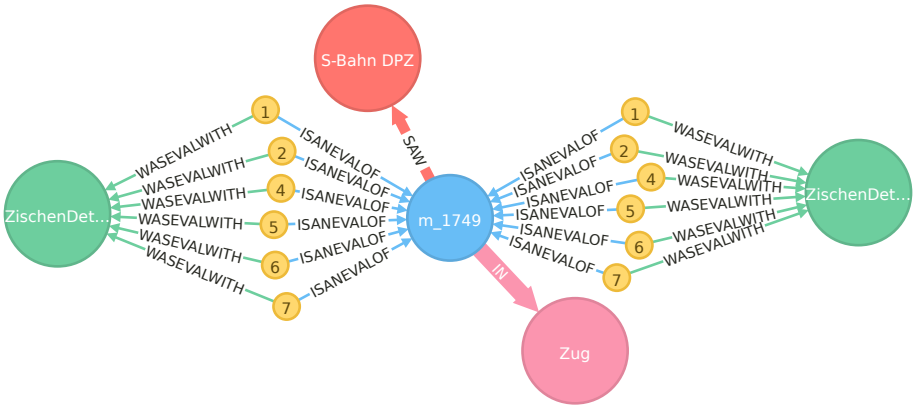


Figure 1: Neo4j database

## Getting started in R

We will import the measurements for all the algorithms. RNeo4j accepts `data.frames` as input, so we will give it that. In the following R-code, we will call all evaluations made with the algorithms in the database, all Passings attached to those evaluations and all trains types of those Passings.

```
query='
MATCH (a:Algorithm)
MATCH (b)<-[s:ISANEVALOF]-(c)-[:WASEVALWITH]->(a)//the relationship s contains the specific
MATCH (t)<-[r:SAW]-(b)//the relationship r contains info in the train of this Passing (lengh
RETURN a.Name, c.tNoisemasked, c.tNoise, s.tEval, s.tEvalmasked, t.Name, r.trainLength, r.Tr
'
q<-cypher(graph,query)#takes the info of each MicMes for one algorithm
```