Neo4j & Graph Algorithms

Graph and Mining

A5

ESILV
nicolas.travers (at) devinci.fr

Chapitre

Table des matières

1 Import the Tourism Circulation dataset **3**
- 1.1 Create your database . . . 3
- 1.2 Install/Activate GDS & APOC plugins . . . 4
- 1.3 Import Data . . . 6
  - 1.3.1 Mono-partite graph . . . 8

2 Mining Bi-partite Graphs **9**
- 2.1 Global Instruction for Practice Works . . . 9
- 2.2 Commands . . . 9
- 2.3 Similarity Algorithms . . . 10
- 2.4 Link Prediction . . . 10

3 Mining Mono-partite Graphs **11**
- 3.1 Cypher Projection . . . 11
- 3.2 Community Detection . . . 11
- 3.3 Path finding . . . 11
- 3.4 Centrality . . . 12
- 3.5 Graph Viz . . . 12

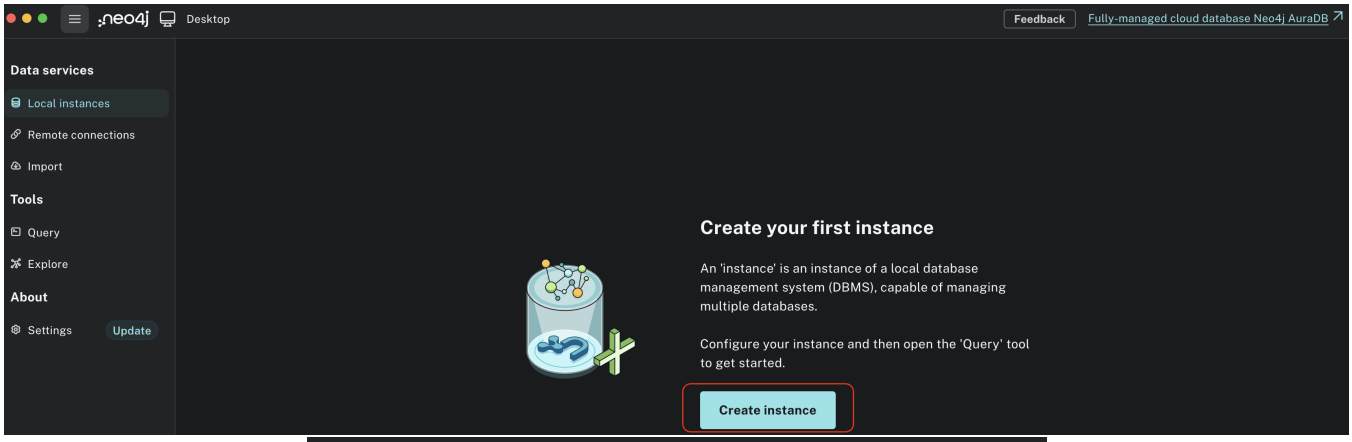
4 Node Embedding and Prediction **14**
- 4.1 Settings . . . 14
- 4.2 Node Embeddings . . . 14
- 4.3 Node Classifier (Region classifier) . . . 14

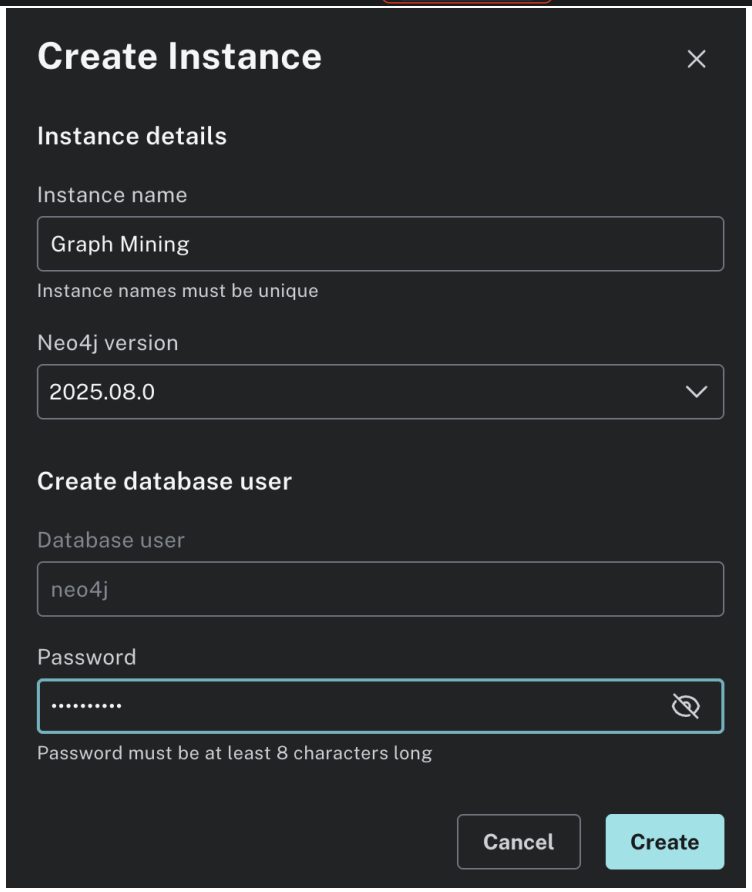
5 DIY - Do It Yourself **16**
- 5.1 Groups & Dataset . . . 16
- 5.2 Job to Do . . . 16

Chapitre 1
Import the Tourism Circulation dataset

1.1 Create your database

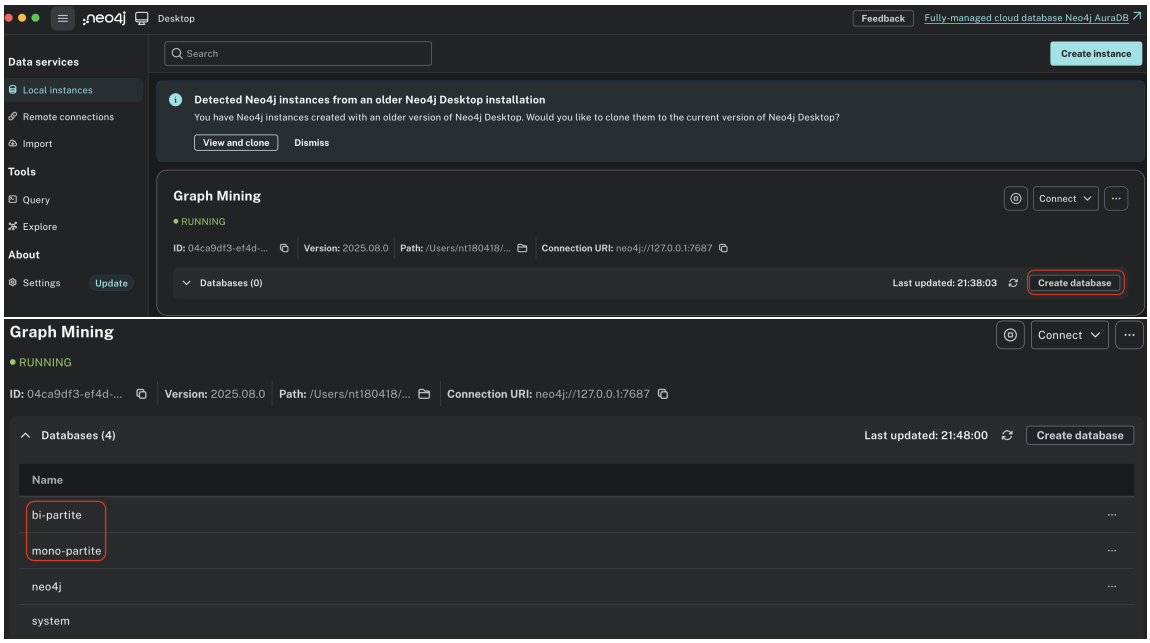
— Check the *Neo4j Desktop 2* version (if not already installed). Make an update before.
You can download it here : https://neo4j.com/deployment-center/?desktop-gdb
Tested versions : Neo4j Desktop **2.0.3**, with Neo4j **2025.08**;
— Check if your firewall blocks ports **7474** (Neo4j browser) & **7687** (Bolt protocol [1]);
— Create a projet "*Graph Mining*" with the last neo4j version and choose a password (DO NOT FORGET IT);



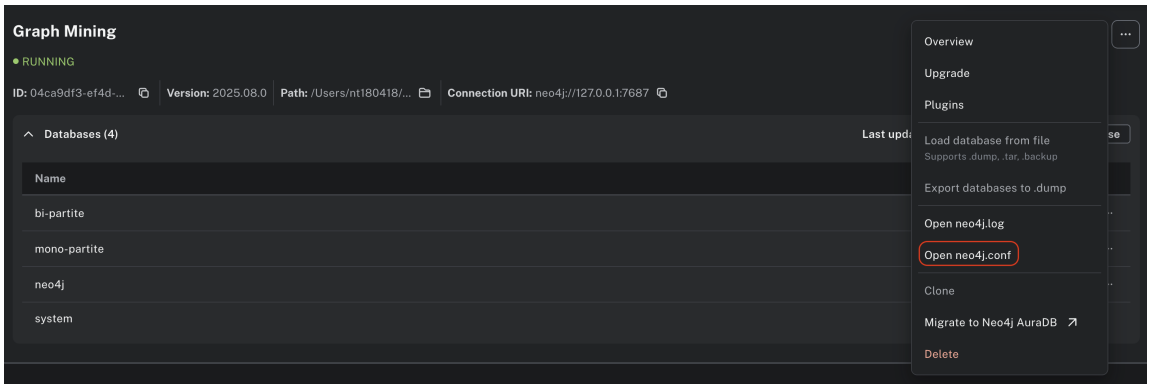


— Create two DBMS "**bi-partite**" and "**mono-partite**".

1. If you restart the database, it should change the ports. Becareful!



— To guarantee good performances, edit the database config (triple dots "**...**") by choosing "**Open neo4j.conf**" such that you get sufficient memory [2] :



— Edit the file in your favorite file app, setting the memory heap size to :

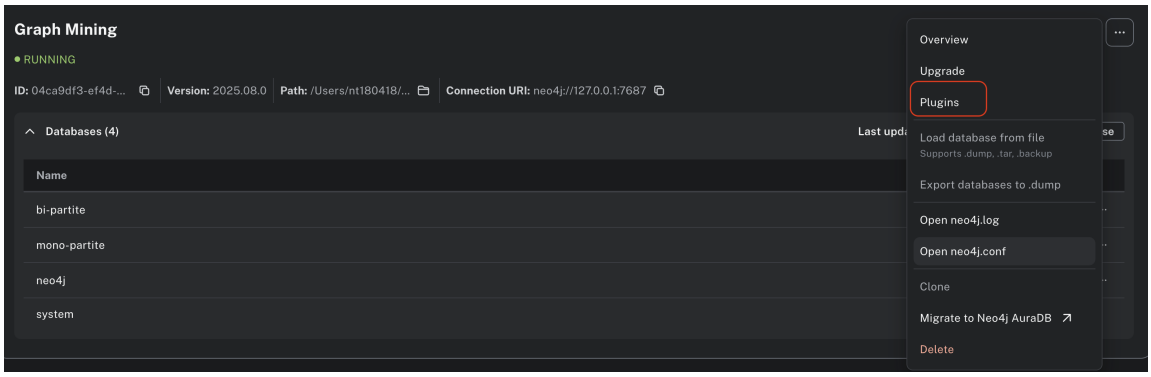
```
73
74  # Java Heap Size: by default the Java heap size is dynamically calculated based
75  # on available system resources. Uncomment these lines to set specific initial
76  # and maximum heap size.
77  server.memory.heap.initial_size=1G
78  server.memory.heap.max_size=6G
79
80  # The amount of memory to use for mapping the store files.
81  # The default page cache memory assumes the machine is dedicated to running
82  # Neo4j, and is heuristically set to 50% of RAM minus the Java heap size.
83  server.memory.pagecache.size=512m
84
```

Can be 6G or more if you want to keep more space for the graph. Be careful, do not exceed the amount of memory *left* on your laptop (OS, browser, apps, services take a lot of memory).

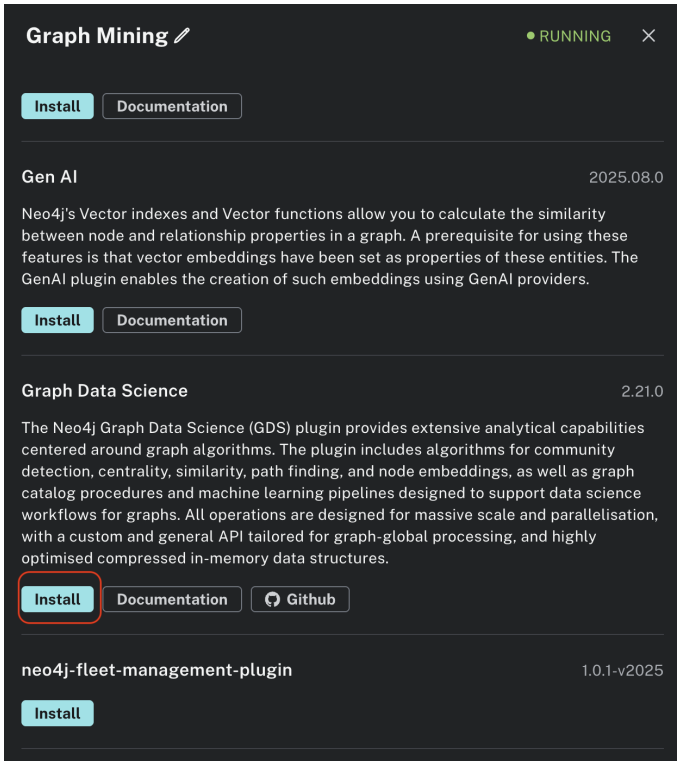
1.2 Install/Activate GDS & APOC plugins

— Again, click options "**...**" to install plugins ;

2. Do not exceed the capacity of your laptop (see "system" on Windows)



— Install **GDS** (Graph Data Science);



— Idem for **APOC**;
— If necessary, restart the database in order to take into account the plugin.

1.3 Import Data

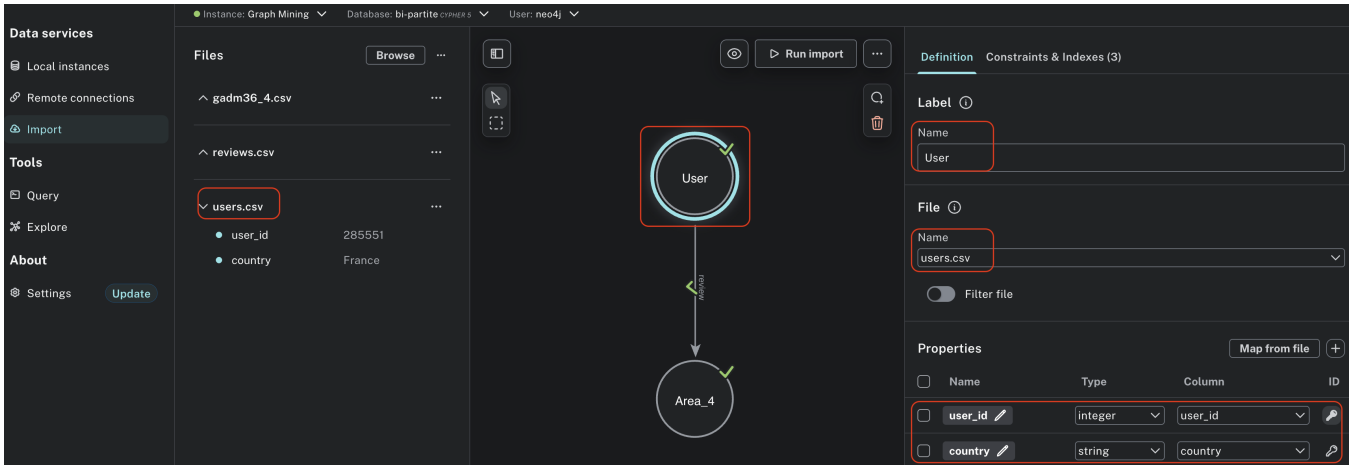
The dataset has been used to study the Circulation of tourists on a geographic territory. You will find the corresponding scientific publications here [WISE'20a, WISE'20b, SAC'22, ADBIS'25] (published by ESILV's researchers in collaboration with ESILV students).

1.3.1 Download the two files : `bi-partite_graph.zip` & `mono-partite_graph.zip`, and unarchive them ;

1.3.2 Come back to `Neo4j Desktop 2`, and choose "*import*" ;

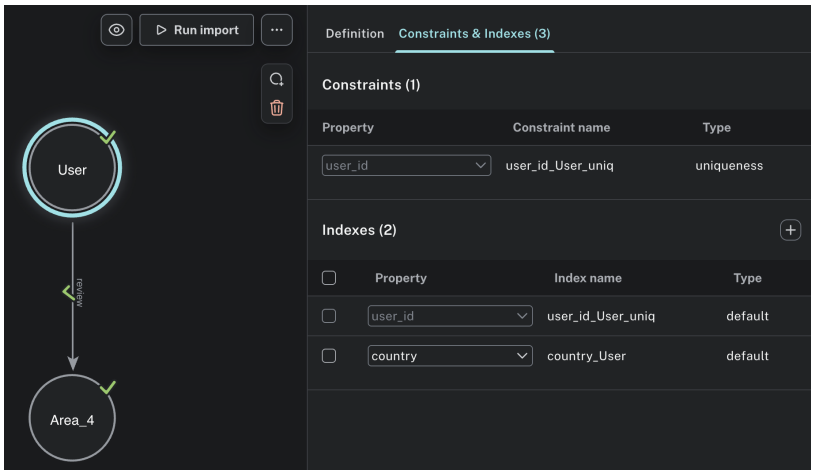
1.3.3 Drag-and-drop the first file "`users.csv`" and fill **properly** the graph data model as the following :

- Node label : **User**
- Node property 1 : **user_id**, mapped with **user_id**
- Node property 2 : **country**, mapped with **country**



NB you can selected "*Map from file*" to map properties quickly.

1.3.4 The second tab helps to design indexes. Choose one on the property "**Country**" Create indexes as in the following but *one query at a time only* :

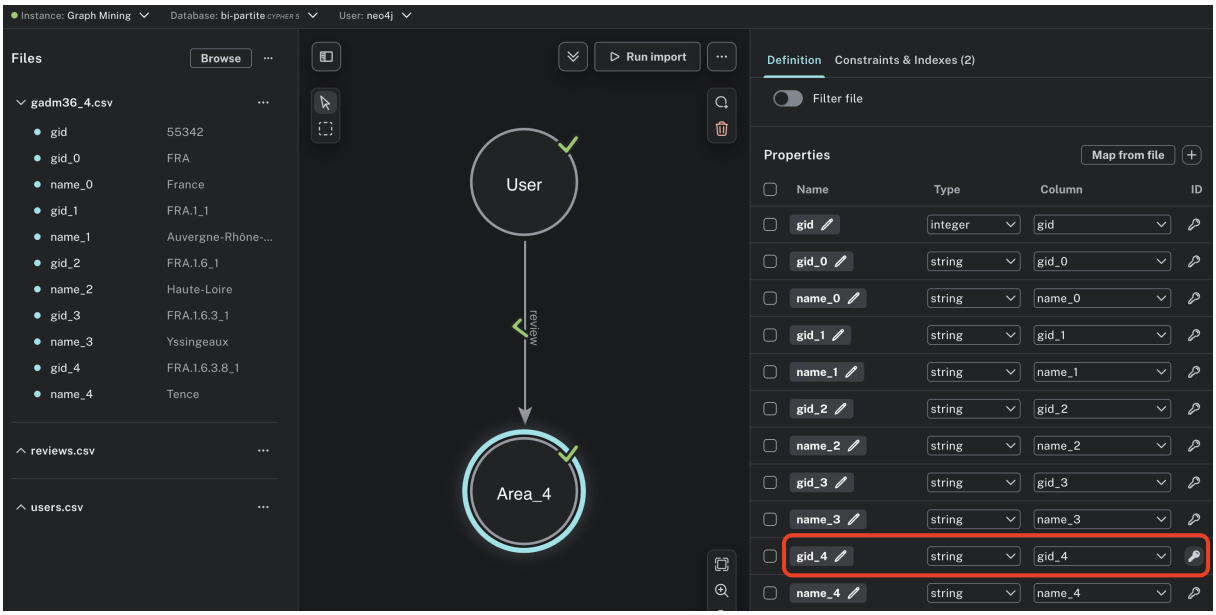


NB If you forgot it, you execute the query in the "*Query*" tool.

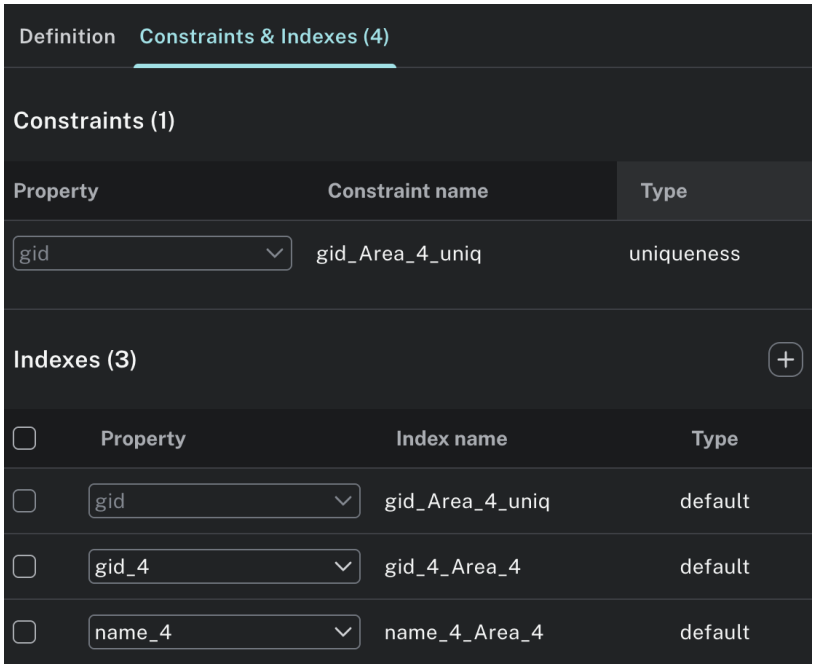
```
CREATE INDEX IF NOT EXISTS FOR (u:User) on u.user_id;
```

1.3.5 Import the file "**gadm36_4.csv**" with the following graph data model :

— Node label : **Area_4**
— gid as the primary key (*integer*)
— all others are *strings*

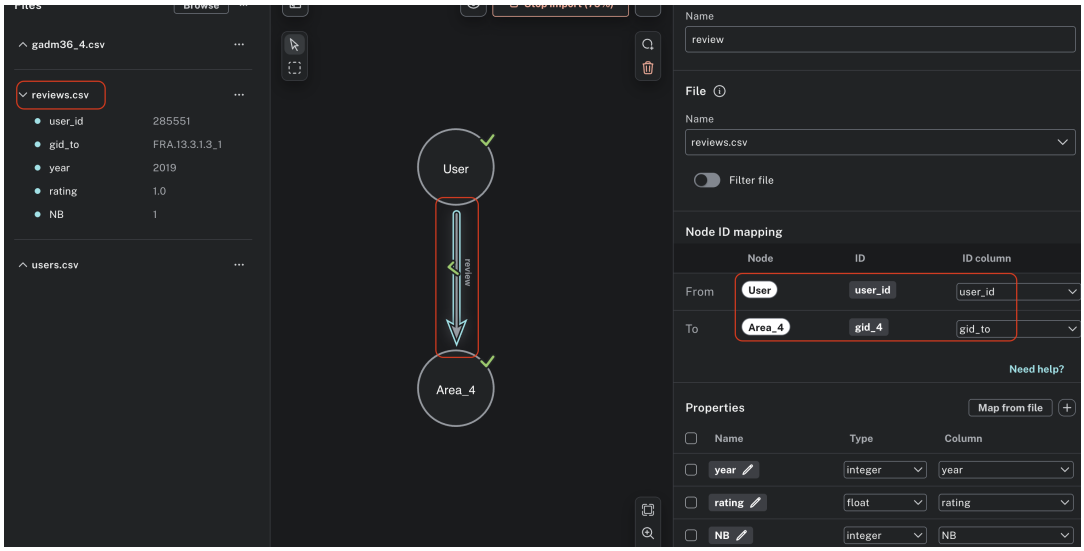


1.3.6 And the corresponding indexes on `gid_4` and `name_4`;



1.3.7 To finish with, you can import the "*reviews.csv*" file (In case of memory space issues, use split files reviews_X.csv)

1.3.8 To create the "*review*" relationship data model, click on the "*User*" border a "+" appears, drag it to "*Area_4*".

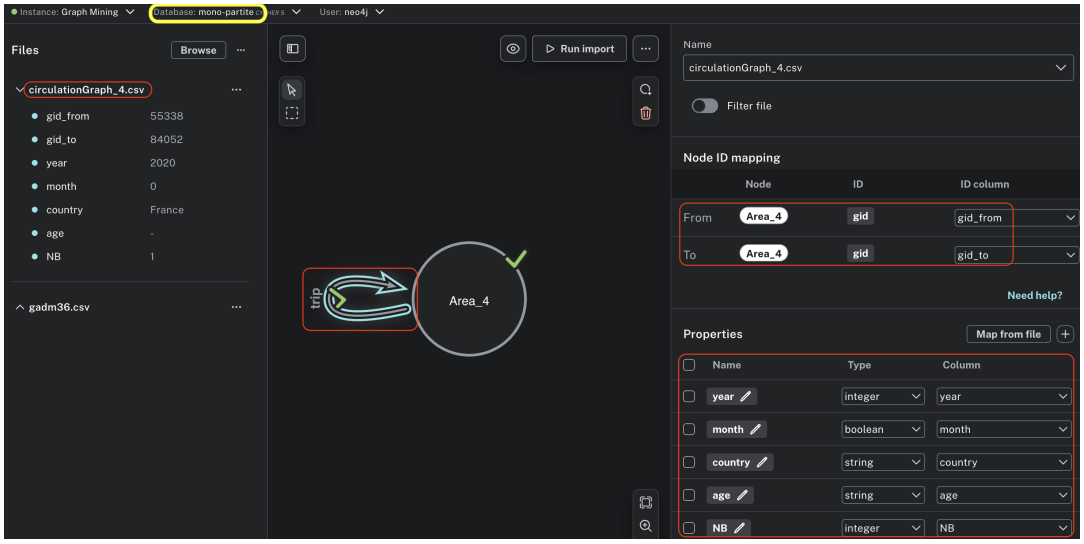


1.3.9 Click on "*Run import*". It can takes a while.

1.3.1 Mono-partite graph

This dataset is a transformation of the bi-partite graph imported previously. Here are the steps already applied : 1) Locations are grouped by GADM3.6 at level 4 (**Cities**), 2) for each couple of reviews from a same user, create a link between the two corresponding cities, 3) group all the links from a given country of origin and year of review to create a weighted relationship. A Java program has been developed to extract this circulation graph.

— In the *Import* window **CHANGE THE DATABASE TO *mono-partite*!!!!**
— Import the "*gadm36.csv*" file. The graph data model is exactly the same as step 1.3.5
— Import the "*circulationGraph_4.csv*" file ;
— Create the "*trip*" relationship model by dragging the Area_4 node to itself. The properties are as follows :



Done ! You can work on the practice work on both graphs.

Chapitre 2

Mining Bi-partite Graphs

1) Verify that your local instance "*Graph Mining*" is running.
NB If you have installed GDS & APOC without restart, please stop/launch the instance again.

2) Open the `Query` tool.

3) Choose the "*bi-partite*" database.

2.1 Global Instruction for Practice Works

The usage of LLMs is not forbidden. However, we ask you to understand perfectly generated queries. Your grade will depend more on the questions we will ask you than the query itself. **We will be uncompromising with evidence of blind trust in the source code produced by the LLM, be criticism** [1].

Even though, the practice work is dedicated to graph mining algorithms. Consequently, the queries themselves are not really important. The results need to be explained and understood which will make you an asset for companies.

Your grade will be based on :

2.1.1 Your active participation during practice works ;

2.1.2 The quality of your interventions for understanding the queries and algorithms ;

2.1.3 The project on your own dataset (see on DVL) - more based on questions asked than the report itself.

2.2 Commands

Create a projection "A4" :

```
MATCH (u:User) -[:review]-> (a:Area_4)
WHERE u.user_id IN [X,Y]
WITH gds.graph.project("U_X_Y", u, a) AS g
RETURN  g.graphName AS graph, g.nodeCount AS nodes, g.relationshipCount AS rels
```

List of generated projections :

```
CALL gds.graph.list()
YIELD graphName, database, nodeCount, relationshipCount, creationTime, modificationTime
RETURN graphName, database, nodeCount, relationshipCount, creationTime, modificationTime
ORDER BY graphName;
```

Drop a generated projection :

```
CALL gds.graph.drop("U_X_Y") YIELD graphName;
```

Apply a similarity function on the projection :

```
CALL gds.nodeSimilarity.stream("U_X_Y", {similarityMetric: "JACCARD"})
YIELD node1, node2, similarity
RETURN gds.util.asNode(node1).user_id AS U1, gds.util.asNode(node2).user_id AS U2, similarity
ORDER BY similarity DESCENDING
```

1. Notice that no complain will be accepted. You have to understand and explain what you produced.

2.3 Similarity Algorithms

2.3.1 Take the two French users who reviewed the most (sum of NB);

2.3.2 Give their *Jaccard* Similarity;

2.3.3 Take the two French users who reviewed the most distinct areas. Give their similarity;

2.3.4 Explain the difference;

2.3.5 For those couples, give the *overlap* and explain the difference with *Jaccard*;

2.3.6 For those couples, give the *Euclidean* and *cosine* similarities, using the NB. Explain the difference (between couples and other similarities);

For this, you need first to create for each :User a mapping of each :Area_4 and corresponding property "NB" that need to be computed (use : collect, apoc.map.fromPairs). Once the map is set for both nodes, you can apply euclidean distance (use : apoc.coll.toSet, coalesce, gds.similarity.euclidean)

2.3.7 Idem with ratings (and explanation);

2.3.8 Do the same for areas in common **only** (sharing same Area_4 nodes);

2.3.9 Give the average *euclidean* and *cosine* similarities where user countries' is **Spain**. For efficiency issue, users must have visited at least 5 times per area
(r.NB >= 5);

2.3.10 Give the one for British, American and Italians. Explain the differences.

2.4 Link Prediction

2.4.1 Give the number of common neighbors between the two French who reviewed the most (seen before);

2.4.2 Give the link prediction on *total neighbors*, *preferential attachment*, *resource allocations* and *Adamic Adar*;

2.4.3 Explain the differencies;

2.4.4 Give the top 20 shared neighbors between the top 10 spanish reviewers (sum of NB). Give for all similarities (*total neighbors*, *preferential attachment*, *resource allocations* and *Adamic Adar*) ordered by adamic adar.

2.4.5 Discuss the result by looking at common neighbors.

2.4.6 To you, which users are more likely to be connected?

Chapitre 3

Mining Mono-partite Graphs

Change the instance/database to the "**mono-partite**" graph. You should have only *Area_4* nodes and *trip* relationships.

First, close all graph databases except the "**mono-partite**" graph. Open the browser.

3.1 Cypher Projection

In the following, we need to create several sub-graphs in order to understand various behavior from the users.

3.1.1 Create a Cypher Projection [1] for :

- country=France
- year=2019

```
MATCH (a1:Area_4) -[t:trip]-> (a2:Area_4)
WHERE t.country = "France" and t.year = 2019
WITH gds.graph.project("FR19", a1, a2, t) AS g
RETURN  g.graphName AS graph, g.nodeCount AS nodes, g.relationshipCount AS rels


//To drop it
call gds.graph.drop("FR19")
```

3.1.2 Idem with (France, United Kingdom, United States) in 2019 and 2020.
"FR20", "UK19", "UK20", "US19", "US20";

3.2 Community Detection

3.2.1 Give the number of triangles per node for FR19 and FR20, in decreasing order.
Warning! The graph must be undirected for this algorithm (use "relationships.toUndirected").

```
CALL gds.graph.relationships.toUndirected("FR19",
   {relationshipType: "*", mutateRelationshipType: "tripBetween"}
)
YIELD
inputRelationships, relationshipsWritten
```

Use the new "tripBetween" relationship in following algorithms.

3.2.2 Idem but grouped by department (name_2). Discuss the result;

3.2.3 Idem with the *clustering coefficient* (on average per name_2). Discuss the result (Infinity and different results);

3.2.4 Extract communities with "*Label Propagation*" on different Cypher projections;

3.2.5 From the previous result, give the list of communities per department. Discuss the result;

3.2.6 Idem with "*Louvain*";

3.2.7 Group previous result per communityId. Discuss the result;

3.2.8 Do the analysis with other years and nationalities to see if you can find a common behavior.

3.3 Path finding

Now we need to take into account "NB" properties on relationships from the mono-partite graph. However, since it represents the number of persons that linked those two nodes, the weight is the opposite : $\frac{1}{N}$. Which corresponds

1. Cypher query + gds.graph.project(...)

to the fact that those nodes are more similar (lower distance) if the NB is higher (notice that $N \geq 1$) Provide the new Cypher projection to integrate them :

```
MATCH (a1:Area4) -[t:trip]-> (a2:Area4)
WHERE t.country = "France" and t.year = 2019
WITH gds.graph.project("FR19", a1, a2,
   { relationshipProperties: t { weight: 1.0 / t.NB } }) AS g
RETURN  g.graphName AS graph, g.nodeCount AS nodes, g.relationshipCount AS rels
```

Use the new property "*weight*" on *trip* relationships/

3.3.1 Give all pairs of shortest paths in FR19 based on NB properties. Order by distance decreasingly. We obtain the pairs of longest distance in the graph.
NB Need to use a Map configuration instead on CypherProjection (with "relationshipWeightProperty");

3.3.2 Consequently, give the city (area_3) which is the more far from all other nodes (average distance of area_4).

3.3.3 Extract (and store) the **Minimum** Spanning Tree starting from "*Paris 1er arrondissement*";
NB The cypher projection must be undirected

3.3.4 Extract the **Maximum** Spanning Tree;

3.4 Centrality

3.4.1 Extract *PageRank* centralities from nodes in different various cypher projection. Discuss the order of results (weights are dependant on the graph);

3.4.2 Give the average, min and max PageRank score for corresponding departments. Explain the differences;

3.4.3 Give Degree, Closeness, Betweenness centralies for those graphs and explain differences.

3.5 Graph Viz

3.5.1 Store in the graph the results from :

- Louvain community;
- MinSpanningTrees;
- PageRank Centrality;

Do not forget to change the property name in order that you can manipulate the proper input.

3.5.2 Visualize the graphs with the tool "**Explore**".

- Place nodes with the coordinate layout (long/lat);
- Change node colors with the corresponding community;
- Change the size/colors of relationships according to the MinSpanningTree;
- Change node sizes according to the PageRank.



Figure 3.1 – France tourism circulation for French in 2019, using Louvain, Min Spanning Tree, Pagerank (using NB)

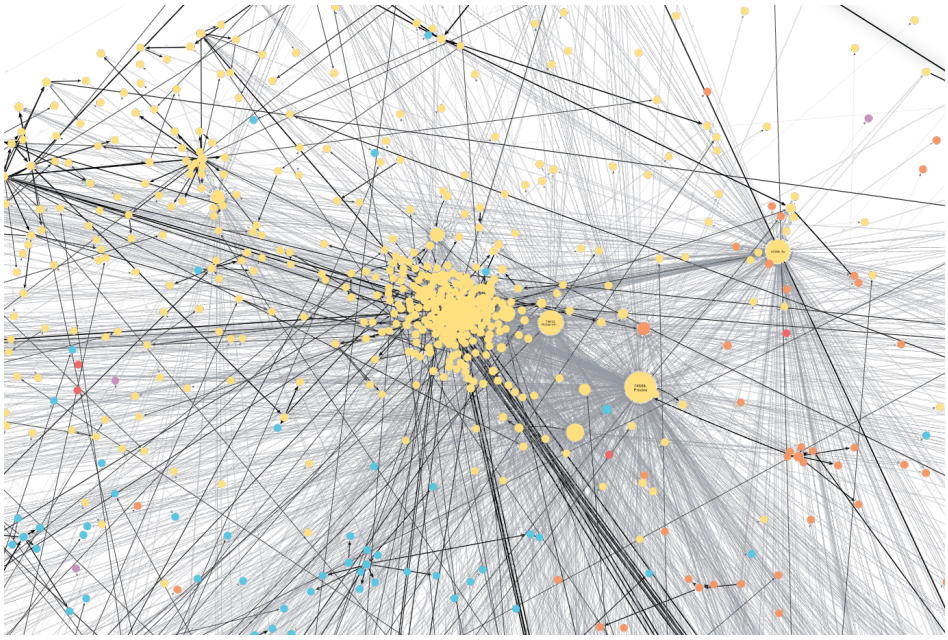


Figure 3.2 – Idem, focused on Paris' region

Chapitre 4
Node Embedding and Prediction

GenAI plugin

In this section, we will create node embeddings using the Node2Vec algorithm. Then, we will train a classifier to try to predict the region of a node (name_1).

4.1 Settings

4.1.1 First, create a Cypher Projection of the name of your choice including (:Area_4) nodes and [:trip] edges. Don't forget to include the 'NB' attribute which will be used as a weight for the graph traversal by Node2Vec.

4.2 Node Embeddings

4.2.1 Then, use the Node2Vec algorithm to create node's embeddings. Make sure to specify the dimensions of embeddings (embeddingDimension), the length of the walks and the number of iterations per node (walkLength and iterations), the probabilities (or weights) of returning to the previous node and exploring a new node (returnFactor and inOutFactor) and, above all, the attribute that will be used as a weight for relationships (relationshipWeightProperty).

4.2.2 You should obtain the following table, which you can import in CSV format. This table contains the names of the region's attributes (name_1) with their corresponding embeddings for each node in the graph.

	city	region	embedding
1	"Yssingeaux"	"Haute-Loire"	[0.32235315442085266, 0.36172258853912354, -0.24609294533729553, 0.36301079392433167, -0.053950235247612, 0.16393014788627625, 0.03348840773105621, 0.1918618232
2	"Saint-Lô"	"Manche"	[0.5904310345649719, -0.06822948902845383, 0.038585860282182693, 0.06505212932825089, -0.27604103088378906, -0.1503499448299408, 0.3430739939212799, -0.172385409
3	"Alençon"	"Orne"	[0.6049037575721741, -0.0782601460814476, -0.27620941400527954, -0.06567700952291489, -0.041889749467372894, -0.2928217649459839, 0.4771932065486908, 0.103934347(
4	"Béthune"	"Pas-de-Calais"	[0.8334031105041504, 0.2682449519634247, 0.06596867740154266, 0.1863444596529007, -0.08619867265224457, 0.2873293459415436, 0.07791266590356827, 0.1095872819423

FIGURE 4.1 – Nodes Embeddings

4.3 Node Classifier (Region classifier)

4.3.1 You now need to switch to a python editor (the one of your choice for the rest of the tutorial). First, make the following imports :

```
import pandas as pd
import ast
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import top_k_accuracy_score
```

4.3.2 Then, load the CSV using pandas and create a dataframe. Make sure to convert columns types if needed (e.g. embeddings as numpy arrays).

4.3.3 Create inputs (embeddings) and labels (regions) and make a split between a training set and a test set.

4.3.4 Create a ML model and train it ! (make sure to choose your hyper-parameters wisely)

4.3.5 Finally, evaluate your model on the test set. Select an appropriate metric (I personally chose Top-K Accuracy and varied the value of K).

If you completed the steps correctly, you should get results comparable to mine (73% Top-1 Accuracy and 95% Top-5 Accuracy). You can try to improve these results by modifying the embedding method (e.g. GraphSage or FastRP). You can also try to change the values of the hyper-parameters (for both embedding step or for the model) or chose another prediction model (e.g. a neural network instead of machine learning). Good Luck !

Chapitre 5

DIY - Do It Yourself

The project of this course relies on the knowledge and techniques acquired during previous practice works.

5.1 Groups & Dataset

First choose a dataset by joining the corresponding group on DVL (groups CDOFx). Maximum 4 members (NO MORE). Dataset selection : FIFO

Datasets are :

— On DVL (Section Deliverables)

— If not, a link below is available, or on Kaggle.

If the dataset is to huge, you can reduce it by sampling (to guarantee the distribution).

List of datasets :

5.1.1 **DBLP** : scientific publications (20k), links : authors-publis

5.1.2 **Animals** : Links between species, 2.7M animals, 2.9M relationships (and attributes)

5.1.3 **Reddit** : Links between reddit channels with sentiment analysis, 286k links

5.1.4 **Epinions** : Links between nodes in the Epinions social network, polarity between nodes, 841k links

5.1.5 **Wikispedia** : 4,615 articles, 5,216 categories, 119k links + tests of paths

5.1.6 **School donations** : 72,993 schools, 2M donors, 4.6M donations + projects, etc. [link on Kaggle]

5.1.7 **Echo nest** : Listened songs, 1M users, 384k songs, 48M play counts [Million song webpage] [Dataset]

5.1.8 **Adventure works** : Sales & customers + spatio/temporal, 754k rows [link on Kaggle]

5.1.9 **BasketBall** : Teams, players, ranks & time. 43k rows. [link on Kaggle]

5.1.10 **Credit - Home Credit Risk** : [Link on Kaggle]

5.1.11 **IMdb** : movies, crew, principals, genres. [link here]

5.1.12 **Credits** : [link here]

5.1.13 **Game Reco Steam** : [link here]

5.2 Job to Do

5.2.1 Create 2 graph data models based on your dataset, using Neo4j Desktop 2 *import*. It can be :

— A bi-partite graph

— A mono-partite graph

— A k-partite graph

— A multi-layer graph

— A temporal graph

— A hierarchical graph

— ...

NB : It's not mandatory to use all input files. The goal is to extract knowledge from data.

5.2.2 Use diverse GDS queries on your graphs. You need to practice a lot in order to extract relevant knowledge. Use : Path finding, Communities extraction, Centralities, Link prediction, Graph embeddings.
⚠A query is considered valid if comments and discussions witness your understanding of both the dataset and algorithm.
⚠Comments like "We can see that...", "The values are..." gives only *observations*, we require **conclusions/deductions/interpretations** !

5.2.3 A final presentation of your conclusions will be done during the last practice work.
Submit your presentation on DVL BEFORE. We will grade only the knowledge you bring and how you have extracted it. Be convincing !