Graph Mining

Practice Works on Neo4j

ESILV
nicolas.travers (at) devinci.fr

Chapter

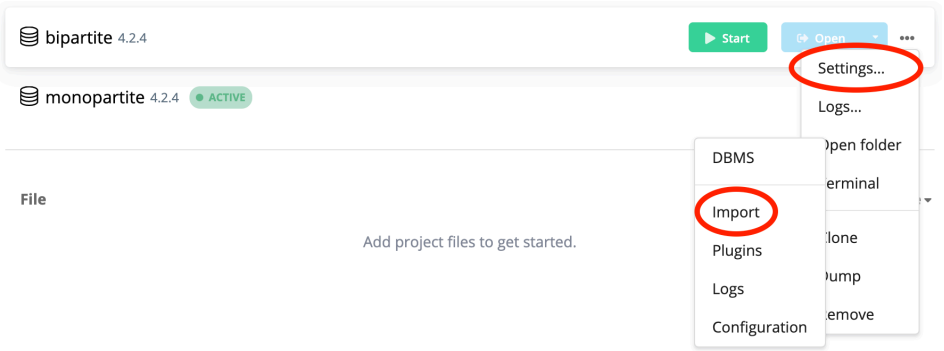
Contents

Chapter 1
Import the Tourism Circulation dataset

The dataset has been used to study the Circulation of tourists on a geographic territory. You will find the corresponding scientific publication here: `https://link.springer.com/chapter/10.1007%2F978-3-030-62005-9_29`

1.1 Create your database

- Check the *Neo4j desktop version* (if not already installed). Make an update before.
  Tested versions: Neo4j Desktop 1.4.9, with Neo4j 4.3.5;
- Check if your firewall blocks ports 7474 (Neo4j browser) & 7687 (Bolt protocol);
- Create a projet "*Graph Mining*", and create two DBMS "**bi-partite**" and "**mono-partite**".
  If needed, refer to the guide used last year: `https://chewbii.com/neo4j-travaux-pratiques/`
- In order to guarantee good performances, edit on both graphs the "**Settings...**" in order to put sufficient memory:




```
dbms.memory.heap.initial_size=1G
dbms.memory.heap.max_size=4G
```

Edit settings

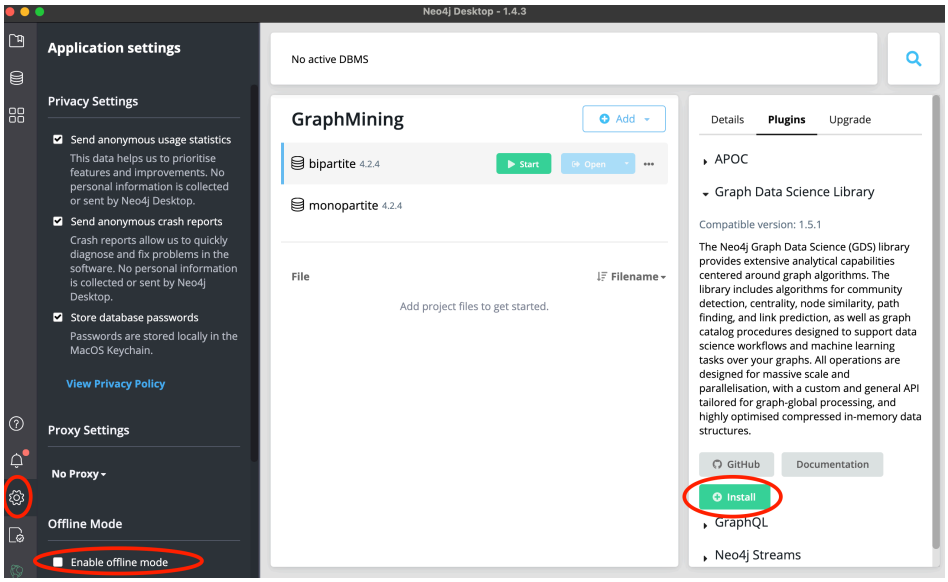
```
# Java Heap Size: by default the Java heap size is dynamically calculated based
# on available system resources. Uncomment these lines to set specific initial
# and maximum heap size.
dbms.memory.heap.initial_size=1G
dbms.memory.heap.max_size=6G
```

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

- Now download the two datasets from DVO, unzip the archives, and put the files in the each import folder ("...", "Open Folder", "Import") for "bipartite" and "monopartite" graphs;

1.2 Activate GDS

- If necessary, remove the "*Enable offline mode*" in **Settings** (bottom left of the Desktop - see picture above);
- Click on your database in order to show the *details* bar (right side);
- Click on the *Plugins* tab, then on "Graph Data Science Library";



- Install the plugin;
- Do it on both graphs;
- If necessary, restart the database in order to take into account the plugin.

1.2.1 Bi-partite Graph

This dataset is an extraction from Tripadvisor reviews where you can find correlations between users (anonymized) and French locations.

- Start the "*bipartite*" database;
- Open the "**bi-partite**" DBMS browser.
  If the button is not clickable, use in your own Web browser: `http://localhost:7474`;
- Create indexes as in the following but *one query at a time only*:

```
CREATE INDEX ON :User(id);
CREATE INDEX ON :User(country);
CREATE INDEX ON :Area_4(gid);
CREATE INDEX ON :Area_4(gid_4);
```

- Import `User` nodes:

```
:auto LOAD CSV WITH HEADERS FROM "file:/users.csv" as l FIELDTERMINATOR "\t"
MERGE (user:User{id:toInteger(l.user_id), country:l.country});
```

- Import `Area_4` nodes:

```
:auto LOAD CSV WITH HEADERS FROM "file:/gadm36_4.csv" as l FIELDTERMINATOR "\t"
MERGE (loc:Area_4{gid:toInteger(l.gid),name:l.nom,
       gid_0:l.gid_0,name_0:l.name_0, gid_1:l.gid_1,name_1:l.name_1, gid_2:l.gid_2,name_2:l.name_2,
       gid_3:l.gid_3,name_3:l.name_3, gid_4:l.gid_4,name_4:l.name_4});
```

  Gadm3.6 is a database which stores all information according to administrative areas all around the world. Here are extracted information from France.

  – Area_0: Country

  - Area_1: Region
  - Area_2: Department
  - Area_3: District (*Canton* in French)
  - Area_4: Cities (*Communauté de communes* in French)
  - Area_5: Towns (*villes/villages* in French) - not shown in this file

- Import `Reviews` relationships.
  It can take a while - sometimes several minutes - do not forget to index nodes and change the heap size!

```
:auto LOAD CSV WITH HEADERS FROM "file:/reviews.csv" as l FIELDTERMINATOR "\t"
MERGE (area:Area_4{gid_4:l.gid_to} )
MERGE (user:User{id:toInteger(l.user_id)} )
MERGE (user) -[:review{year:toInteger(l.year),rating:toFloat(l.rating),NB:toInteger(l.NB)}]-> (area);
```

  If more than 2 minutes are required, it means that your laptop computes to much on writing in memory and the disk (swaping). Instead, you can use the archive "*reviews_split*" which contains the same data but split in 10 distinct files you have to import separately. Change the file name by "*reviews_0.csv*".

1.2.2 Mono-partite graph

This dataset is a transformation of the bi-partite graph imported previously. Here are the steps already applied:

- Locations are grouped by GADM3.6 at level 4 : **Cities**
- For each couple of reviews from a same user, create a link between the two corresponding cities.
- Group all the links from a given country of origin and year of review to create a weighted relationship.

A Java program has been developped to extract this circulation graph.

- Stop the "*bipartite*" database;
- Start the "*monopartite*" database;
- Open the "**mono-partite_circulation**" DBMS. After downloading the dataset on DVO, unzip the archive, and put the files in the import folder ("...", "Open Folder", "Import"). Then, open the Neo4j browser.
- In order to guarantee good performances, edit the settings in order to put sufficient memory
- Create indexes as in the following but *one query at a time only*:

```
CREATE INDEX ON :Area_4(gid);
CREATE INDEX ON :Area_4(gid_4);
```

- Import Area nodes:

```
:auto LOAD CSV WITH HEADERS FROM "file:/gadm36_4.csv" as l FIELDTERMINATOR "\t"
MERGE (loc:Area_4{gid:toInteger(l.gid),name:l.nom,
        gid_0:l.gid_0,name_0:l.name_0, gid_1:l.gid_1,name_1:l.name_1, gid_2:l.gid_2,name_2:l.name_2,
        gid_3:l.gid_3,name_3:l.name_3, gid_4:l.gid_4,name_4:l.name_4});
```

- Import circulation relationships:

```
:auto LOAD CSV WITH HEADERS FROM "file:/circulationGraph_4.csv" as l FIELDTERMINATOR "\t"
MERGE (from:Area_4{gid:toInteger(l.gid_from)} )
MERGE (to:Area_4{gid:toInteger(l.gid_to)} )
MERGE (from) -[:trip{year:toInteger(l.year),NB:toInteger(l.NB),country:l.country}]-> (to);
```

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

Chapter 2

Mining Bi-partite Graphs

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

2.1 Similarity

2.1.1 Take the two French users who reviewed the most (sum of NB);
Correction :

```
MATCH (u:User{country:"France"}) -[r]-> ()
RETURN u, sum(r.NB) as NB
ORDER BY NB DESC limit 2
```

2.1.2 Give their *Jaccard* Similarity (use `WITH` clause to exploit previous result - user 1 and then user 2);
Correction :

```
MATCH (u1:User{id:70})-[:review]->(a:Area_4)
        WITH u1, collect(id(a)) AS u1Areas
MATCH (u2:User{id:76})-[:review]->(a:Area_4)
        WITH u1, u1Areas, u2, collect(id(a)) AS u2Areas
RETURN u1.id AS u1, u2.id AS u2,
        gds.alpha.similarity.jaccard(u1Areas, u2Areas) AS similarity
```

2.1.3 Take the two French users who reviewed the most areas. Give their similarity;
Correction :

```
MATCH (u:User{country:"France"}) --> ()
RETURN u, count(*) as NB
ORDER BY NB DESC limit 2

MATCH (u1:User{id:387312})-[:review]->(a:Area_4)
WITH u1, collect(id(a)) AS u1Areas
MATCH (u2:User{id:2639})-[:review]->(a:Area_4)
WITH u1, u1Areas, u2, collect(id(a)) AS u2Areas
RETURN u1.id AS u1, u2.id AS u2,
        gds.alpha.similarity.jaccard(u1Areas, u2Areas) AS similarity
```

2.1.4 Explain the difference;
Correction :

user 1	user 2	Jaccard similarity
70	76	0.0512396694214876
387312	2639	0.028642590286425903

Here are the two corresponding subgraphs (Figure 2.1 & 2.2). The **intersection** is high for Sum of NBs and the **union** of areas is almost the same. So the Jaccard similarity between 70 & 76 is higher than for the other ones.

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

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

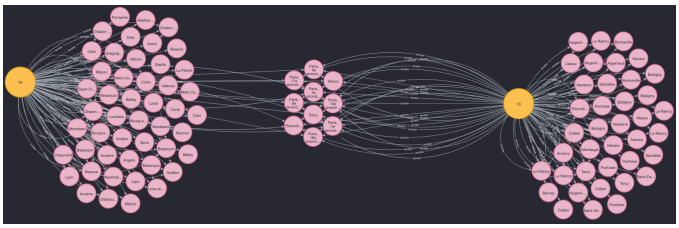


Figure 2.1: Subgraph for users 70 & 76 - top SUM(NB)

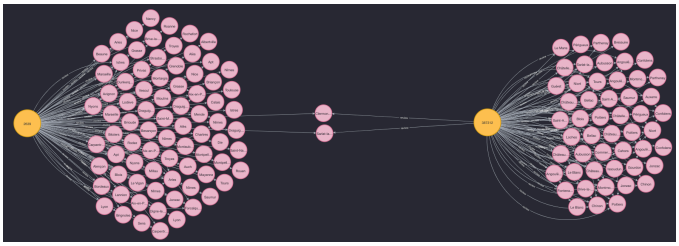


Figure 2.2: Subgraph for users 2639 & 387312 - top COUNT(*)

```
MATCH (u1:User{id:70})-[r1:review]->(a1:Area_4)
MATCH (u2:User{id:76})-[r2:review]->(a2:Area_4)
RETURN u1.id AS u1, u2.id AS u2,
       gds.alpha.similarity.euclidean(collect(r1.NB), collect(r2.NB)) AS similarity
```

2.1.7 Idem with ratings (and explanation);

2.1.8 Give the average *jaccard* and *overlap* similarities[1] for **Spanish** where they visited at least 5 places per area (NB >= 5);
Correction :

```
MATCH (u1:User{country:"Spain"})-[r1:review]->(a1:Area_4)
WHERE r1.NB >= 5
WITH u1, collect(id(a1)) as u1Areas
MATCH (u2:User{country:"Spain"})-[r2:review]->(a2:Area_4)
WHERE r2.NB >= 5
WITH u1, u1Areas, u2, collect(id(a2)) as u2Areas
RETURN AVG(gds.alpha.similarity.overlap(u1Areas, u2Areas)) AS overlap,
       AVG(gds.alpha.similarity.jaccard(u1Areas, u2Areas)) AS jaccard
```

2.1.9 Give the one for British, American and Italians. Explain the differences.
Correction :

Population	Overlap	Jaccard
Spain	0.05285516468244356	0.04018805116004531
British	0.03793307156158973	0.03350092032822436
Italian	0.045266014355186734	0.03543412769821106
American	0.05052951026275412	0.03534720205006689

2.2 Link Prediction

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

[1] Euclidean and Cosine must have same vector size which is not always the case.

```
MATCH (u1:User{id:70}), (u2:User{id:76})
RETURN gds.alpha.linkprediction.commonNeighbors(u1, u2,
                {relationshipQuery:"review"}) AS score
```

27 common neighbors

2.2.2 Get each list of neighbors and check the result;
Correction :

```
MATCH (u:User) --> (l)
WHERE u.id in [70,76]
RETURN
u.id, collect (id(l)) as visits
```

Only 27 seven locations in common while the number of total reviews are high for both them

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

```
MATCH (u1:User{id:70}), (u2:User{id:76})
RETURN gds.alpha.linkprediction.totalNeighbors(u1, u2,
{relationshipQuery:"review"}) AS totalNeighbors,
 gds.alpha.linkprediction.preferentialAttachment(u1, u2,
{relationshipQuery:"review"}) AS preferentialAttachment,
 gds.alpha.linkprediction.resourceAllocation(u1, u2,
{relationshipQuery:"review"}) AS resourceAllocation,
 gds.alpha.linkprediction.adamicAdar(u1, u2,
{relationshipQuery:"review"}) AS adamicAdar
```

Union : 323.0
Preferential Attachement : 101120.0 (degree(70) x degree(76)) ⇒ How much they have the capacity to spread information together
resourceAllocation : 0.01021758294855741 (sum of 1/degree while in the intersection nodes) ⇒ How much the common neighbors are “famous”
AdamicAdar : 3.080595487761758 (same with logs) ⇒ level down the effect of very connected common neighbors

2.2.4 Explain the differencies;

2.2.5 Give the top 10 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.
Correction : Before, get the top 10 spanish reviewers (WITH), combine them with the top 10 (combination of 100 possibilities - limit 100)

```
MATCH (u1:User{country:"Spain"})-[r1:review]->(a1:Area_4)
        WITH u1, SUM(r1.NB) as nb_reviews
        ORDER BY nb_reviews DESC LIMIT 10
MATCH (u2:User{country:"Spain"})-[r2:review]->(a2:Area_4)
WHERE u1.id < u2.id
        WITH u1, u2, SUM(r2.NB) as nb_reviews
        ORDER BY nb_reviews DESC LIMIT 10
RETURN u1.id, u2.id,
        gds.alpha.linkprediction.adamicAdar(u1, u2,
                {relationshipQuery:"review"}) AS aa,
        gds.alpha.linkprediction.commonNeighbors(u1, u2,
                {relationshipQuery:"review"}) AS cn,
        gds.alpha.linkprediction.totalNeighbors(u1, u2,
                {relationshipQuery:"review"}) AS tn,
        gds.alpha.linkprediction.preferentialAttachment(u1, u2,
                {relationshipQuery:"review"}) AS pa,
        gds.alpha.linkprediction.resourceAllocation(u1, u2,
                {relationshipQuery:"review"}) AS ra
ORDER BY aa DESC LIMIT 10
```

2.2.6 Discuss the result by looking at common neighbors.

Correction :

line	u1	u2	aa	cn	tn	pa	ra
1	67654	164748	0.9949528076656134	8.0	79.0	2755.0	0.0031171040416941792
2	164748	211248	0.4757948735437447	4.0	138.0	7695.0	0.0011131796151207492
3	134651	211248	0.23320052921128373	2.0	112.0	3726.0	0.0006391303070042378
4	79	164748	0.15180150401129003	1.0	72.0	665.0	0.0013774104683195593
5	67654	211248	0.11085104177163535	1.0	96.0	2349.0	0.0001208313194780087
6	79	211248	0.0	0.0	83.0	567.0	0.0
7	1504	211248	0.0	0.0	93.0	1620.0	0.0
8	19623	211248	0.0	0.0	100.0	2349.0	0.0
9	1504	164748	0.0	0.0	83.0	1900.0	0.0
10	19623	164748	0.0	0.0	90.0	2755.0	0.0

Line 1 has 8 neighbors in common whose degree is very low so the Adamic Adar value growse up accordingly.

Line 3 & 4 can be interchanged if we look at Adamic Adar vs. Resource Allocation. In fact, nodes involved in the intersection has huge impact while the log decrease it.

Chapter 3

Mining Mono-partite Graphs

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 named "French2019" where you extract the graph for the French in 2019 population with NB;
Correction :

```
CALL gds.graph.create.cypher(
        "French2019",
        "MATCH (a1:Area_4)          RETURN DISTINCT id(a1) AS id",
        "MATCH (n:Area_4) -[r:trip{country:'France', year:2019}]-> (m)
         RETURN id(n) as source, id(m) as target, sum(r.NB) as NB"
) YIELD graphName AS graph, nodeCount AS nodes, relationshipCount AS rels

//to drop it
call gds.graph.drop("French2019")
```

3.1.2 Idem with "French2020", "British2019", "British2020", "US2019", "US2020";

3.2 Community Detection

3.2.1 Give the number of triangles per node for French2019 and French2020, in decreasing order;
Correction :

```
CALL gds.triangleCount.stream("French2019")
YIELD nodeId, triangleCount
RETURN gds.util.asNode(nodeId).name_4, triangleCount
ORDER BY triangleCount DESC
```

3.2.2 Idem but grouped by department (Area_2). Discuss the result;
Correction :

```
CALL gds.triangleCount.stream("French2019")
YIELD nodeId, triangleCount
RETURN gds.util.asNode(nodeId).name_2, SUM(triangleCount) as triangleCount
ORDER BY triangleCount DESC
```

3.2.3 Idem with the *clustering coefficient*. Discuss the result (Infinity and different results);
Correction :

```
CALL gds.localClusteringCoefficient.stream("French2019")
YIELD nodeId, localClusteringCoefficient
RETURN gds.util.asNode(nodeId).name_2,
        SUM(localClusteringCoefficient) as localClusteringCoefficient
ORDER BY localClusteringCoefficient DESC
```

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

```
CALL gds.labelPropagation.stream("French2019")
YIELD nodeId, communityId
RETURN gds.util.asNode(nodeId).name_4, communityId
ORDER BY communityId
```

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

```
CALL gds.labelPropagation.stream("French2019")
YIELD nodeId, communityId
RETURN gds.util.asNode(nodeId).name_2, collect(distinct communityId)
```

3.2.6 Idem with "*Louvain*";
Correction :

```
CALL gds.louvain.stream("French2019")
YIELD nodeId, communityId
RETURN gds.util.asNode(nodeId).name_2, collect(distinct communityId)
```

3.2.7 Group previous result per communityId. Discuss the result;
Correction :

```
CALL gds.louvain.stream("French2019")
YIELD nodeId, communityId
RETURN communityId, collect(distinct gds.util.asNode(nodeId).name_2)
```

3.3 Path finding

3.3.1 Give all pairs of shortest paths in Spain2019 based on NB properties. Need to use a Map configuration instead on CypherProjection (with "nodeQuery" and "relationshipQuery");
Correction :

```
CALL gds.alpha.allShortestPaths.stream(
{
        nodeQuery:"MATCH (a1:Area_4)         RETURN DISTINCT id(a1) AS id",
        relationshipQuery:"MATCH (n:Area_4) -[r:trip{country:'France', year:2019}]-> (m)
        RETURN id(n) as source, id(m) as target, sum(r.NB) as NB",
        relationshipWeightProperty: "NB"
})
YIELD sourceNodeId, targetNodeId, distance
```

3.3.2 Extract the Minimum Spanning Tree starting from "Paris 1° arrondissement";
Correction :

```
MATCH (a:Area_4) where a.name_4 contains "Paris, 1er arrondissement" return a.id

CALL gds.alpha.spanningTree.minimum.write(
{
        startNodeId:887,
        nodeQuery:"MATCH (a1:Area_4)         RETURN DISTINCT id(a1) AS id",
        relationshipQuery:"MATCH (n:Area_4) -[r:trip{country:'France', year:2019}]-> (m)
        RETURN id(n) as source, id(m) as target, sum(r.NB) as NB",
        relationshipWeightProperty: "NB",
        writeProperty: "MinFrench2019",
        weightWriteProperty: "writeCost"
})
YIELD createMillis, computeMillis, writeMillis, effectiveNodeCount

MATCH p=(:Area_4{id:887})-[r:MinFrench2019]->() RETURN p
```

3.3.3 Extract the **Maximum** Spanning Tree;
Correction :

```
CALL gds.alpha.spanningTree.maximum.write(
{
        startNodeId:887,
        nodeQuery:"MATCH (a1:Area_4)         RETURN DISTINCT id(a1) AS id",
        relationshipQuery:"MATCH (n:Area_4) -[r:trip{country:'France', year:2019}]-> (m)
        RETURN id(n) as source, id(m) as target, sum(r.NB) as NB",
        relationshipWeightProperty: "NB",
        writeProperty: "MaxFrench2019",
        weightWriteProperty: "writeCost"
})
YIELD createMillis, computeMillis, writeMillis, effectiveNodeCount
```

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);
Correction :

```
CALL gds.pageRank.stream("French2019", {
        relationshipWeightProperty: "NB",
        maxIterations: 20,
        dampingFactor: 0.85
})
YIELD nodeId, score
RETURN gds.util.asNode(nodeId).name_4 AS name_4, score
ORDER BY score DESC, name_4 ASC
```

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

```
CALL gds.pageRank.stream("French2019", {
        relationshipWeightProperty: "NB",
        maxIterations: 20,
        dampingFactor: 0.85
})
YIELD nodeId, score
RETURN gds.util.asNode(nodeId).name_2 AS name_2, AVG(score), MIN(score), MAX(score)
ORDER BY score DESC, name_2 ASC
```

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