Analyse de graphes avec Neo4j
La Librairie Data Science

Hamida SEBA Hamida.seba@univ-lyon1.fr

Equipe Graphes Algorithmes et Applications (GOAL)

LIRIS, CNRS UMR 5205

Graph (Network) Analysis

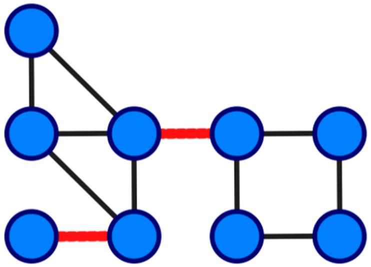
- Graph analytics: use of any graph-based approach to analyze connected data
  - Understand real networks (social networks, protein-protein interactions, etc.)
  - Predict behaviors within connected systems
  - reveal the workings of intricate systems and networks at massive scales
  - Radical novelty (features not previously observed in systems);
  - Coherence or correlation (meaning integrated wholes that maintain themselves over some period of time);
  - Identify the least costly or fastest way to route information or resources.
  - Predict missing links in your data.
  - Locate direct and indirect influence in a complex system.
  - Discover unseen hierarchies and dependencies.
  - Forecast whether groups will merge or break apart.
  - Reveal communities based on behavior for personalized recommendations.
  - Etc.

Graph (Network) Analysis

- Graph analytics:
  - Global topological metrics:
    - Diameter : the longest shortest path
    - Eccentricity: of a vertex *v* is the greatest distance between *v* and any other vertex.
    - Radius: is the minimum eccentricity of any vertex
    - Density : ratio of the number of edges and the number of possible edges (in complete graph) $d = \frac{2m}{n(n-1)}$
    - Average degree
    - Etc.
  - Traversal and paths: explore a graph either for general discovery or explicit search
    - Shortest (weighted) path between two nodes, All shortest paths
    - Minimum spanning tree: path in a connected tree structure with the smallest cost for visiting all nodes
    - Random walk: a list of nodes along a path of specified size selected by randomly choosing relationships to traverse.
    - Etc.

Graph (Network) Analysis

- Graph analytics:
  - Centrality: used to identify the most important nodes in the network.
    - ✓ Degree Centrality : the most important is the one with the greatest number of neighbors (degree)
    - ✓ Closeness Centrality: the most important is the most central, i.e., the closest to all the others
    - ✓ Betweenness Centrality: the most important is a bridge
    - ✓ PageRank : the most important is the one with the greatest number of important neighbors
    - ✓ Etc.



  - Communities: identify groups of nodes that have more relationships within the group than with nodes outside their group.
    - ✓ Triangle Count and Clustering Coefficient(local and global) which quantifies how close the neighbors of a vertex are to being a **clique** (complete graph)
    - ✓ Strongly Connected Components and Connected Components for finding connected clusters
    - ✓ Label Propagation for quickly inferring groups
    - ✓ Louvain Modularity for looking at grouping quality and hierarchies

Neo4j Data Science Library

- A set of Algorithms (procedures): https://neo4j.com/docs/graph-data-science/current/algorithms/
- To see the complete list

```
CALL gds.list()
```

- To use the results, two utility functions https://neo4j.com/docs/graph-data-science/current/management-ops/utility-functions/
  - gds.util.asNode(): Return the node object for the given node id or null if none exists.
  - gds.util.asNodes(): Return the node objects for the given node ids or an empty list if none exists.

- To call a procedure in cypher: https://neo4j.com/docs/cypher-manual/current/clauses/call/#query-call-introduction
  - CALL[…YIELD], the YIELD sub-clause is used to explicitly select which of the available result fields are returned. Filtering results is also possible with the clause WHERE

```
CALL db.labels() YIELD label
WHERE label CONTAINS 'User'
RETURN count(label) AS numLabels
```

Graph catalog

- Graph algorithms have a high complexity (NP-hard problems), Neo4j GDS uses a specific in-memory representation to run efficiently these algorithms: : the graph catalog: https://neo4j.com/docs/graph-data-science/current/management-ops/graph-catalog-ops/

- To use an algorithm, we need to load the Neo4j graph into the graph catalog: the in-memory graph is a projection of a neo4j graph

- List of projected graphs in the graph catalg:

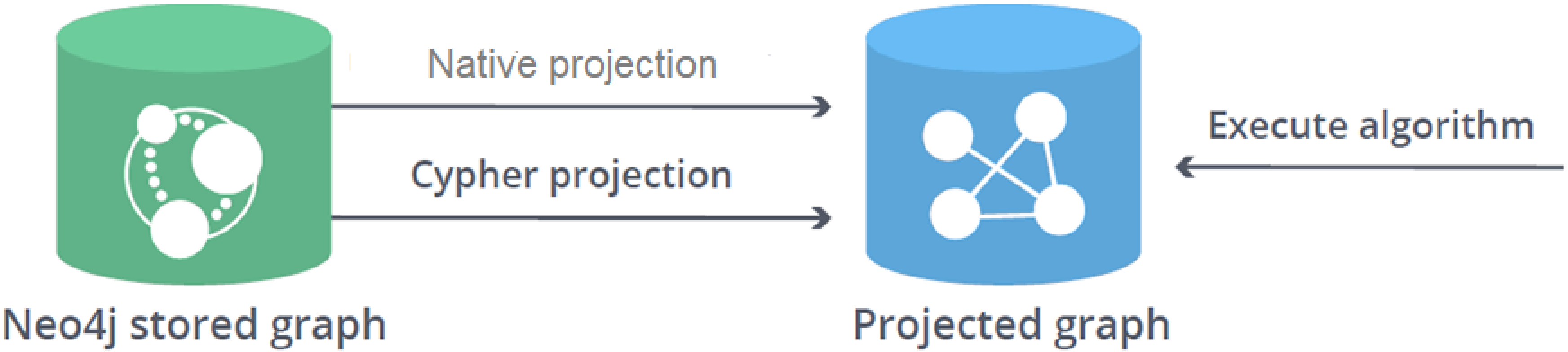
```
CALL gds.graph.list() YIELD  graphName
```

- The graph catalog allows to manage several named graphs as well as anonymous graphs

Graph catalog

- Named graph projection: create a projection with a name, to use with several algorithms
- Anonymous graph projection: the projection is aimed for a single use

- For both cases, there are 2 types of projections:
  - Native projection: https://neo4j.com/docs/graph-data-science/current/management-ops/native-projection/
    - ✓ Native projections provide the best performance
  - Cypher projection: https://neo4j.com/docs/graph-data-science/current/management-ops/cypher-projection/



Graph catalog

- Native projection: allows us to project a graph from Neo4j into an in-memory graph. The projected graph can be specified in terms of node labels, relationship types and properties.
  - Syntax for a named graph:

```
CALL gds.graph.project(
   graphName: String,
   nodeProjection: String, List or Map,
   relationshipProjection: String, List or Map,
   configuration: Map
)
```

  - Short-hand String-syntax for nodeProjection:

<neo4j-label> or [<neo4j-label>..., <neo4j-label>]

  - Short-hand String-syntax for relationshipProjection:

<neo4j-type> or [<neo4j-type>, ..., <neo4j-type>]

Graph catalog

- Native projection
  - Exemple

```
CALL gds.graph.project(
 'GraphAmitie',
 'User',
 'FRIEND_OF'
)
YIELD graphName, nodeCount, relationshipCount
```

	graphName	nodeCount	relationshipCount
1	"GraphAmitie"	200	800

Graph catalog

- Native projection:
  - Node Projection: Extended Map-syntax

```
{
   <node-label-1>: {
      label: <neo4j-label>,
      properties: <node-property-mappings>
   },
 ...
   <node-label-n>: {
      label: <neo4j-label>,
      properties: <node-property-mappings>
   }
}
```

Example:

```
CALL gds.graph.project(
 'G1',
 {
 Utilisateur: {label: 'User'}
 },
 '*'
)
YIELD graphName, nodeCount,
relationshipCount
```

graphName	nodeCount	relationshipCount
"G1"	200	800

Graph catalog

- Native projection
  - Relationship Projection: Extended Map-syntax

```
{
   <relationship-type-1>: {
      type: <neo4j-type>,
      orientation: <orientation>,
      aggregation: <aggregation-type>,
      properties: <relationship-property-
mappings>
   },
// ...
   <relationship-type-n>: {
      type: <neo4j-type>,
      orientation: <orientation>,
      aggregation: <aggregation-type>,
      properties: <relationship-property-
mappings>
   }
}
```

NATURAL
REVERSE
UNDIRECTED

```
CALL gds.graph.project(
 'G2',
 {
 Utilisateur: {label: 'User'},
 Film : {label: 'Movie'}
 },
 {
 RATE: {
       type: 'RATED',
       orientation: 'NATURAL',
       properties: 'score'}
 }

)
YIELD graphName, nodeCount
, relationshipCount
```

  - D’autres examples : https://neo4j.com/docs/graph-data-science/current/management-ops/projections/graph-project/

Algorithms

- 4 main execution modes.
  - stream: returns the result of the algorithm as a stream of records.
  - stats: returns a single record of summary statistics, but does not write to the Neo4j database.
  - mutate: writes the results of the algorithm to the in-memory graph and returns a single record of summary statistics. This mode is designed for the named graph variant, as its effects will be invisible on an anonymous graph.
  - write: writes the results of the algorithm to the Neo4j database and returns a single record of summary statistics.
- An execution mode may be estimated by appending the command with estimate. This allows to estimate the required memory of a graph and an algorithm before running it in order to make sure that the workload can run on the available hardware https://neo4j.com/docs/graph-data-science/current/common-usage/memory-estimation/.

Algorithms

- Centrality algorithms: used to determine the importance of distinct nodes in a network
  - Page Rank: measures the importance of each node within the graph, based on the number of incoming relationships and the importance of the corresponding source nodes

```
CALL gds.pageRank.stream('G1')
YIELD nodeId, score
RETURN nodeId AS node, score
Order by score DESC
```

  - Betweenness Centrality: detects the amount of influence a node has over the flow of information in a graph. It is often used to find nodes that serve as a bridge from one part of a graph to another. Each node receives a score, based on the number of shortest paths that pass through the node

```
CALL gds.betweenness.stream('G3')
YIELD nodeId, score
RETURN nodeId AS node, score
ORDER BY node ASC
```

```
CALL gds.betweenness.stats('G3')
YIELD minimumScore, maximumScore, scoreSum
```

```
CALL gds.betweenness.write('G3', { writeProperty: 'betweenness' })
YIELD minimumScore, maximumScore, scoreSum, nodePropertiesWritten
```

Algorithms

```
neo4j$ match (n:User) return n.id, n.betweenness
```

```
neo4j$ match (n:User) return n.id, n.betweenness
```

	n.id	n.betweenness
1	1	360.43352203352197
2	2	554.0979125449711
3	3	391.077114506407

Algorithms

- Other definitions of centrality are also available:
  - Closeness Centrality: measures the average farness of a node (inverse distance) to all other nodes. Nodes with a high closeness score have the shortest distances to all other nodes.

```
CALL gds.beta.closeness
```

  - Harmonic Centrality: is a variant of closeness centrality that deals with unconnected graphs

```
CALL gds.alpha.closeness.harmonic
```

  - Degree Centrality: measures the number of incoming and outgoing relationships from a node

```
CALL gds.degree
```

  - Eigenvector Centrality: among the first algorithms that consider transitive importance of a node in a graph, rather than only considering its direct importance.

```
CALL gds.eigenvector
```

Algorithms

- Community detection algorithms: used to evaluate how groups of nodes are clustered or partitioned, as well as their tendency to strengthen or break apart https://neo4j.com/docs/graph-data-science/current/algorithms/community/.

  - Louvain: detect communities in large networks. It maximizes a modularity score for each community, where the modularity quantifies the quality of an assignment of nodes to communities. This means evaluating how much more densely connected the nodes within a community are, compared to how connected they would be in a random network.
  - Label Propagation: a fast algorithm for finding communities in a graph. It detects these communities by propagating labels throughout the network.
  - Weakly Connected Components: finds sets of connected nodes in an undirected graph, where all nodes in the same set form a connected component
  - Triangle Count: counts the number of triangles for each node in the graph
  - Local Clustering Coefficient: describes the likelihood that the neighbours of node v are also connected (T_v is the number of triangles of vertex v and d_v its degree)

$$CC(v) = \frac{2\,Tv}{d_v(dv-1)}$$

Algorithms

- Path finding algorithms: find the shortest path between two or more nodes or evaluate the availability and quality of paths https://neo4j.com/docs/graph-data-science/current/algorithms/pathfinding/.
    - Shortest Path: algorithm of Dijkstra
    - All Pairs Shortest Path
    - A*
    - Etc.

Algorithms

- Similarity algorithms: compute the similarity of pairs of nodes using different vector-based metrics https://neo4j.com/docs/graph-data-science/current/algorithms/similarity/

- Link Prediction algorithms: help determine the closeness of a pair of nodes. The computed scores can then be used to predict new relationships between them

- Node embeddings: compute low-dimensional vector representations of nodes in a graph. These vectors, also called embeddings, can be used for machine learning