Graph Mining

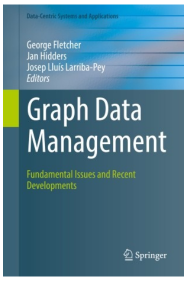
with neo4j

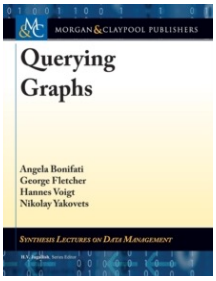
Graph Mining?

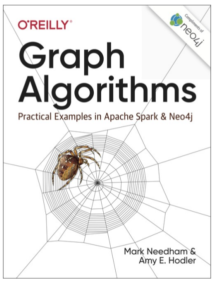
- Local Patterns
  - Queries with Cypher
  - Online, local decision, pattern matching
  - Not mining
- Global graph analysis
  - Graph algorithms
  - Learn the overall structure, prediction

Introduction

- The Graph Data Science (GDS) Library
- Graph Mining
- Course Organization







Links on images

Others
Stanford – J. Leskovec (movie, youtube)
CNAM – R. Fournier
Univ. Udine – E. Franceschet

Some representative graph processing systems

	Native graphs	Online query	Query Language	Programming Language	Data Sharding	In-Memory storage	Transaction support
Neo4j	✔	✔	Cypher	Pregel, Java			✔
Trinity (Microsoft)	✔	✔	GE API			✔	Atomicity
Horton (Microsoft)	✔	✔	regular exp		✔	✔	
TinkerPop (Apache)	✔	✔	Gremlin			✔	
InfiniteGraph	✔	✔	DO		✔		✔
Cayley (Google)	✔	✔	GraphQL, Gizmo		backend-dependent	backend-dependent	✔
Titan	✔	✔	Gremlin		backend-dependent	backend-dependent	✔
GraphX (Spark)			(Cypher)	Pregel	✔		
FlockDB (Twitter)		✔	Thrift		✔		✔
MapReduce				MapReduce	✔		
PEGASUS				MapReduce	✔		
ArangoDB (Google)			GraphQL	Pregel	✔		
Giraph (Google/Apache)				Pregel	✔		
GraphLab (Univ Bordeaux)				JavaScript	✔		
GraphChi (Nvidia/Facemovie)			Cypher	SparQL			

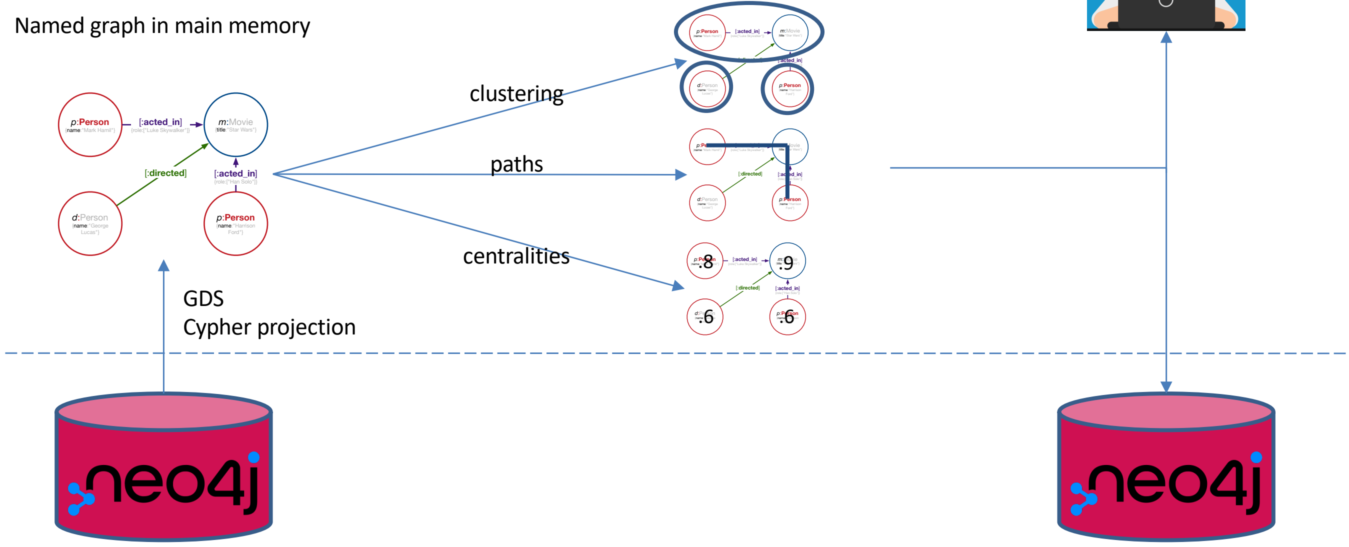
Graph Mining – Plan

1. Graph Data Science
   - Open Source Neo4j plugin
   - Cypher projection
2. Dedicated to graph analytics
   - Paths finding, Communities, Centrality, Similarity, Link prediction
3. GDS & Machine Learning
   - Node embedding, Node classification, Link prediction
4. Advanced GDS
   - Graph data management memory/storage
   - Pregel
   - Neosemantics

GDS – Evolutions

- Neo4j Contrib – Graph algorithms *2017*
- Neo4j – Graph algorithms (product engineering) *2019*
- Open-Source release *2020 Q1*
- Neo4j GDS *2020 Q2*
  - Open-Source
    - All algorithms, 4 CPU
  - Enterprise Edition
    - Unlimited CPU, RBAC, unlimited models (+persisting), optimized in-memory

Graph Mining Workflow



GDS – Graph Projection

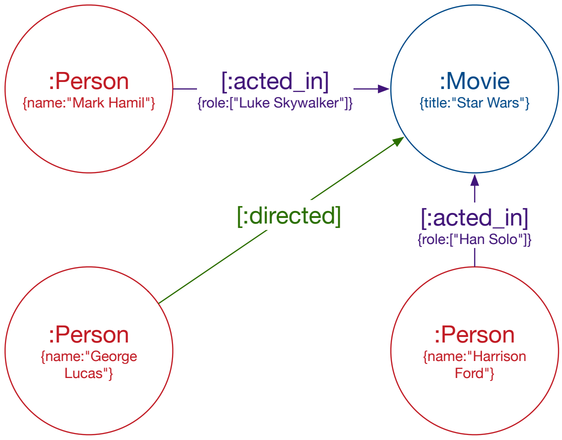
- Cypher projection = Cypher query + storage (graph catalog)
  - CALL gds.graph.create(
    graphName: String,
    nodeProjection: String or List or Map,
    relationshipProjection: String or List or Map,
    configuration: Map (readConcurrency, nodeProperties, relationshipProperties, validateRelationships)
)
    YIELD
    graphName: String,
    nodeProjection: Map,
    nodeCount: Integer,
    relationshipProjection: Map,
    relationshipCount: Integer,
    createMillis: Integer

- Filter
- Focus
- Aggregation

https://neo4j.com/docs/graph-data-science/current/graph-create/
https://neo4j.com/docs/graph-data-science/current/management-ops/graph-catalog-ops/

GDS – Graph Projection – Catalog Example

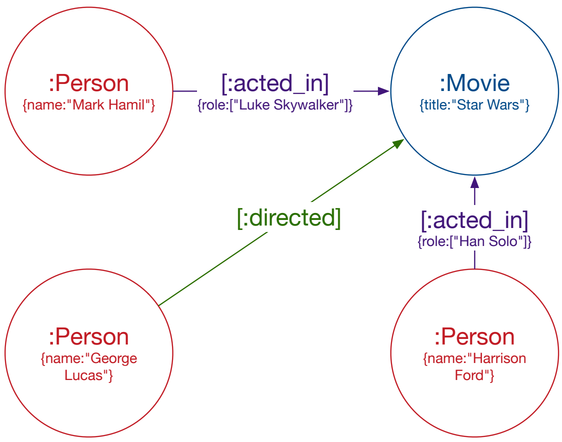
```
CALL gds.graph.create(
   'SW4',
   {
      Person:{properties:'name'},
      Movie:{properties:{title:'Star Wars'}}
   },
   ['acted_in','directed']
) YIELD
   graphName AS graph,
   nodeProjection,
   nodeCount AS nodes,
   relationshipProjection,
   relationshipCount AS rels
```



GDS – Graph Projection – Cypher Example

```
CALL gds.graph.create.cypher(
  'SW_saga',
  'MATCH (p:Person) --> (m:Movie)
    WHERE m.title contains 'Star Wars'
    RETURN id(p) AS id
   UNION
    MATCH (m:Movie)
    WHERE m.title contains 'Star Wars'
    RETURN id(m) as id',
  'MATCH (n:Person)-[r:acted_in|directed]->(m:Movie)
    RETURN id(n) AS source, id(m) AS target'
) YIELD
  graphName AS graph,
  nodeProjection, nodeCount AS nodes,
  relationshipProjection, relationshipCount AS rels
```

Possibility to add 'weights'



GDS – Syntax

```
CALL gds.<algo-name>.<mode>(
  graphName: STRING,
  configuration: MAP
)
```

mode: write/stats/stream

```
CALL gds.pagerank.stream(
  "SW_saga",
  {
    writeProperty: 'pageRank',
    maxIterations: 20,
    dampingFactor: 0.85
  }
) YIELD nodeId, score
RETURN gds.util.asNode(nodeId).name AS name, score
ORDER BY score DESC, name ASC
```

gds.alpha.<fn-name>(…)

- Various set functions (similarities, vectors, etc.)

Graphs – Recall

Graphs:

(bi|mono|k)**partite**, (un)**weigthed**, (un)**directed**, (a)**cyclic**,
(dis)**connected**, (rooted|binary|spanning) **trees**
Nodes: **(in|out) degree**
Density: ${}^{2R}/_{N(N-1)}$
Diameter: max(rel) between two nodes

Graph algorithms:

Pathfinding, Centrality, Community detection

Graph Mining – Plan

1. Graph Data Science
   - Open Source Neo4j plugin
   - Cypher projection
2. Dedicated to graph analytics
   - Paths finding, Communities, Centrality, Similarity, Link prediction

   https://neo4j.com/docs/graph-data-science/current/algorithms/
3. GDS & Machine Learning
   - Node embedding, Node classification, Link prediction
4. Advanced GDS
   - Graph data management memory/storage
   - Pregel
   - Neosemantics

GDS – Algorithms – Path finding

- Parallel Breadth First Search
- Parallel Depth First Search
- Shortest Path
- Minimum Spanning Tree
- A* Shortest Path
- Yen's K Shortest Path
- K-Spanning Tree (MST)
- Random Walk

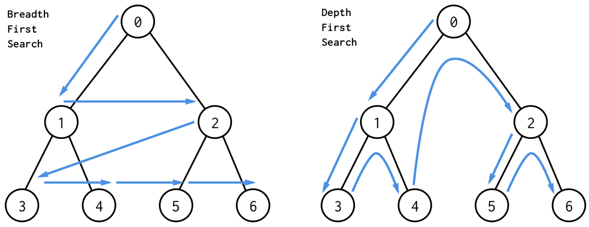
GDS – Path finding
Breadth|Depth First Search

- For given starting node
- BFS
  - Breadth before to seen how far are all nodes

➢ *Likelihood of finding a node*

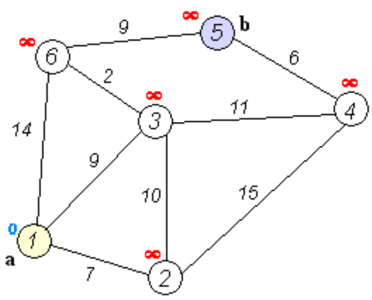
- DFS
  - Depth before backtraching

➢ *Help to find a matching, maze, traversal-salesman, Ford-Fulkerson*



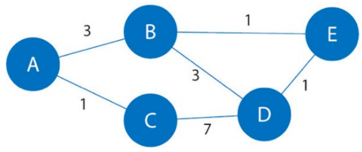
GDS – Path finding
Shortest Path

- [Dijkstra]
  - Iteration on finding the lowest-weight relationship
- [A*]
  - Heuristic function cost to expand paths
  - Approximation
- [Yen K]
  - Calculates the *k* shortest path at the same time (alternative paths)



➢ Finding directions, distance between nodes

GDS – Path finding
All Pairs Shortest Path

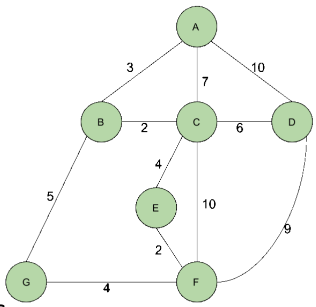


- [APSP]
  - More efficient than computing all shortest paths separately
  - Keep track of distances from the beginning

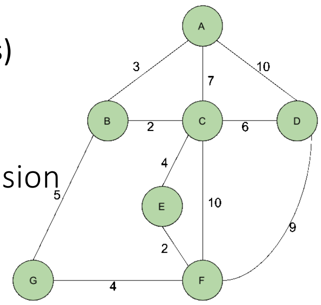
All nodes start with a ∞ distance and then the start node is set to a 0 distance			Each Step Keeps or Updates to the Lowest Value Calculated so Far Only steps for node A to all nodes shown				
			1st from A	2nd from A to C to Next	3rd from A to B to Next	4th from A to E to Next	5th from A to D to Next
A	∞	0	0	0	0	0	0
B	∞	∞	3	3	3	3	3
C	∞	∞	1	1	1	1	1
D	∞	∞	∞	8	6	5	5
E	∞	∞	∞	∞	4	4	4

➢ Understanding alternate routes

GDS – Path finding
Minimum Spanning Tree



- MST (Boruvka 1926, Kruskal 1956, Prim 1957)
  - From a given node
  - Connects **all** nodes with relationships with min weights
    - *Boruvka*: Compacting nodes and removing heavy relationships
    - *Prim*: Choosing the relationship to extend the tree
    - *Kruskal*: Choosing the min weight relationship (without cycles)



➢ Minimize cost of traversal, correlations, propagation/transmission

GDS – Path finding
Random Walk

- Random navigation in the graph
  - Can use probability distributions
  - Efficient

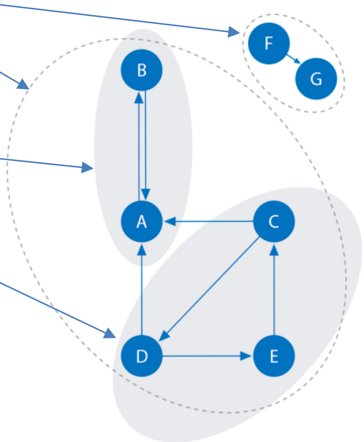
➢ Node2vec, graph2vec, infomap community detection, Monte Carlo simulations, training process for ML, Tweets recommendation

GDS – Community detection

- Connected Components
- Triangle Count
- Clustering Coefficients
- Label Propagation
- Louvain

GDS – Community detection
Connected Components

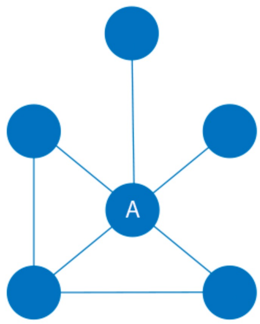
- **Connected** = all nodes reachable (no direction)
- **Strongly connected** = all nodes reachable in both directions



➢ *Fast grouping for other algo*

GDS – Community detection
Triangle Count

- Measures how many nodes wrt.
  Nb triangles that pass through a node



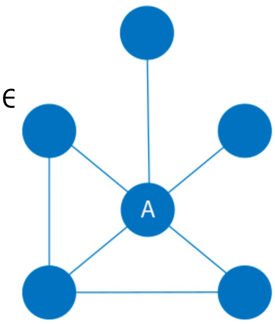
$T_A = 2$

➢ *Estimate group stability, small world behavior*

NB: gds 2.2 -> the graph must be undirected

GDS – Community detection
Clustering Coefficient

- Probability for the **neighbors** (of a node) to be connecte
- Can be normalized globally (measure)



- $CC_n = \frac{2T_n}{d_n(d_n-1)}$

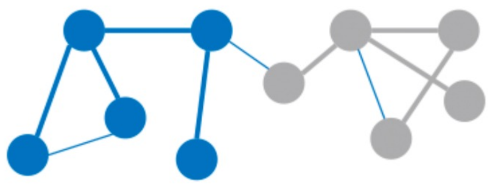
$$CC_A = \frac{2*2}{5(5-1)} = 0,2$$

➢ *Estimate group stability, small world behavior*

GDS – Community detection
Label Propagation

- Fast algorithm for graph
- Propagation of labels
  - Efficient for densely connected group of nodes
  - Overlaps resolution (several communities)
  - Weights can be used
  - A property can be used for seeds
- Adapted to less clear groups

LPA push mode



➢ ***Understand consensus, finding strong combinations***

GDS – Community detection

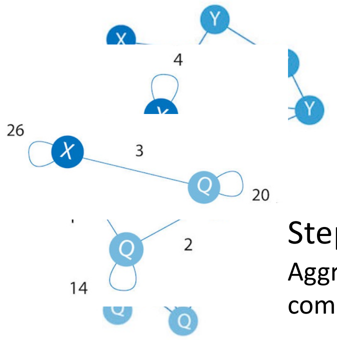
Louvain [J. of Statistical Mechanics'08]

$$M = \sum_{c=1}^{n_c} \left[\frac{L_c}{L} - \left(\frac{k_c}{2L} \right)^2 \right]$$

where:

- L is the number of relationships in the entire group.
- L_c is the number of relationships in a partition.
- k_c is the total degree of nodes in a partition.

- Based on modularity
  - Quantifies how well a node contributes to the density of connections
  - **Modularity** = Quality measure of connections density
- Can reveal a hierarchy of communities
  - Merges smaller communities into larger ones
  - Adapted to large graphs



Pass 2

Step 2

Aggregate communities

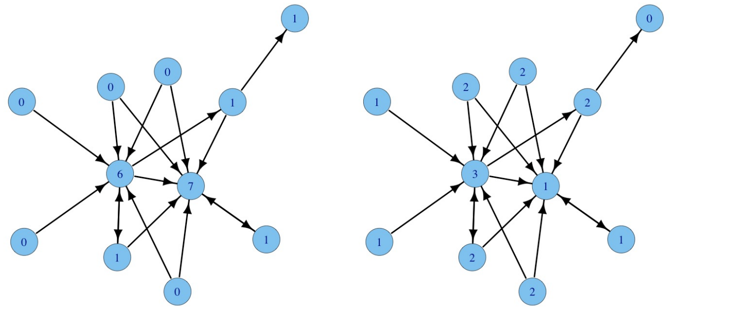
➢ *Fraud analysis, detecting discrete behavior*

GDS – Algorithms – Centrality

- Degree Centrality
- Closeness Centrality
- Betweenness Centrality
- Eigenvector Centrality
- PageRank
- Personalized PageRank
- ArticleRank

GDS – Centrality
Degree Centrality

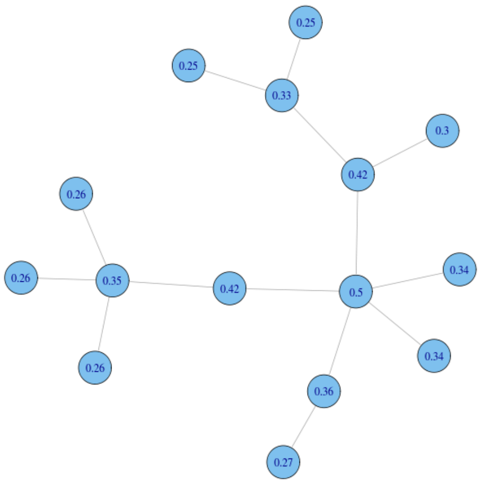
- Counts the nb of in+out relationships
  - **$C_D(n) = \frac{deg(n)}{max(deg(N))}$**
- Can use:
  - **in or out degrees**
  - **weights**



➢ *Most important nodes*

GDS – Centrality
Closeness Centrality

- Capacity of a node to spread informatio
  - Average farness
  - $C_{C-norm}(n) = \frac{N-1}{\sum_1^{N-1} d(u,v)}$
  - d = distance of the shortest path
  - Can use weights

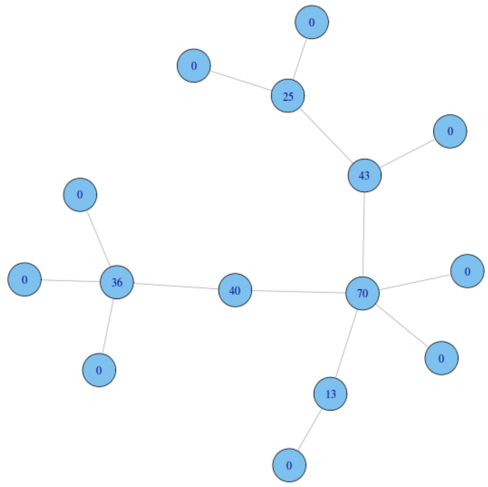


➢ Identify fastest disseminator

GDS – Centrality
Betweenness Centrality

- Detecting node influence wrt. Flow
  - Computes all shortest paths (u,v)
  - $C_B(n) = \sum_{s,t \in N} \frac{p(s,t)}{p}$
  - p: nb paths. p(s,t): nb paths through node
  - Can be weighted
- Approximation: [RA-Brandes] algo

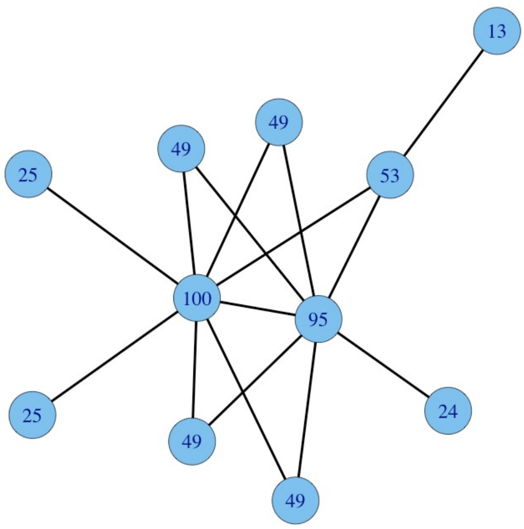
➢ Influencers, bottlenecks, control points,



GDS – Centrality
Eigenvector Centrality

- A node is important if it is linked to
  - Transitive influence of nodes
  - Power iteration approach
- $C_E(n) = \frac{1}{\lambda} \sum_k a_{k,n} C_E(k)$
  - $a_{k,n}$: matrice d'adjacence
  - λ: Eigenvalue (normalized)

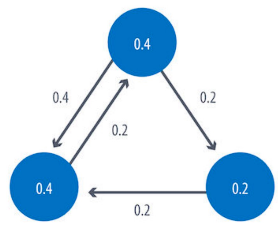
➢ *Opinion influence over a network, ranking systems, multilayer graphs*



GDS – Centrality
PageRank - Google

- Transitive influence of nodes
    - Derived from Eigenvector
- Ranking of the potential impact of nodes
- PR(u)

$$PR(u) = (1 - d) + d\left(\frac{PR(T1)}{C(T1)} + \ldots + \frac{PR(Tn)}{C(Tn)}\right)$$



    - Damping factor: probability of propagation/random (0.85)
    - C(Ti): out-degree
- Power iterations on the graph
    - Until convergence
    - In pratice: 50

➢ *Broad influence over a network, traffic tendency*

GDS – Centrality
PageRank Variants

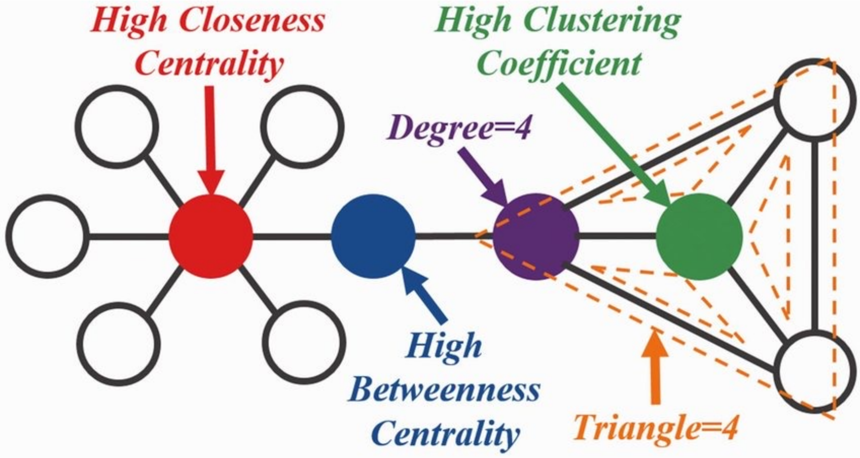
- Personalized PageRank
    - PPR is a variant of PR
    - Focus on a given node influence

    ➢ *Make recommendations for a node*

- ArticleRank
    - Low-degree nodes more influence than high-degree
    - Special cases on "*spider traps*" and "*infinite cycles*"

    ➢ *Recommendation based on networks of citations*

Summary



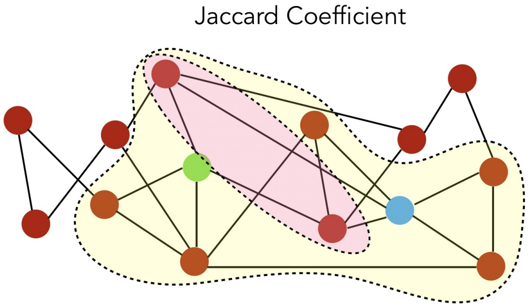
GDS – Algorithms – Similarity

Based on nodes similarity

- Jaccard Similarity
- Euclidean Distance
- Cosine Similarity
- Overlap Similarity
- Pearson Similarity
- KNN

GDS – Similarity
jaccard Similarity

- Comparison of shared neighbors
  - Based on Jaccard similarity
  - $J(A,B) = \frac{|A \cap B|}{|A|+|B|-|A \cap B|}$




```
MATCH (p1:Person {name: 'Karin'})-[:LIKES]->(movie1)
WITH p1, collect(id(movie1)) AS p1Movie
MATCH (p2:Person {name: "Arya"})-[:LIKES]->(movie2)
WITH p1, p1Movie, p2, collect(id(movie2)) AS p2movie
RETURN p1.name AS from, p2.name AS to,
       gds.alpha.similarity.jaccard(p1movie, p2movie) AS similarity
```

➢ Bi-partite graph comparison, recommendation

GDS – Similarity
Overlap similarity

- Measures the overlap between 2 nodes
- $\boldsymbol{O(A,B)} = \frac{|\boldsymbol{A} \cap \boldsymbol{B}|}{\boldsymbol{min}(|\boldsymbol{A}|,|\boldsymbol{B}|)}$

```
MATCH (movie:Movie)-[:HAS_GENRE]->(genre)
WITH {item:id(genre), categories: collect(id(movie))} AS movieData
WITH collect(movieData) AS data
CALL gds.alpha.similarity.overlap.stream({data: data})
YIELD item1, item2, count1, count2, intersection, similarity
RETURN gds.util.asNode(item1).name AS from, gds.util.asNode(item2).name AS to,
       count1, count2, intersection, similarity
ORDER BY similarity DESC
```

➢ Dedicated to small numbers of relationships
  - Adapted to big graphs

GDS – Similarity
Euclidean|Cosine Similarity

- Comparison of shared nodes+weights
  - Applied on bi-partite graphs
  - $E(p,q) = \sqrt{\sum_{i=1}^{n}(p_i - q_i)^2}$
  - $cos(p,q) = \frac{\sum_{i=1}^{n} p_i \times q_i}{\sqrt{\sum_{i=1}^{n}(p_i)^2} \times \sqrt{\sum_{i=1}^{n}(q_i)^2}}$

```
MATCH (p1:Person {name: "Nicolas"})-[likes1:LIKES]->(movie)
MATCH (p2:Person {name: "Cedric"})-[likes2:LIKES]->(movie)
RETURN p1.name AS from, p2.name AS to,
    gds.alpha.similarity.cosine( collect(likes1.score), collect(likes2.score) ) AS similarity
```

➢ Compare similar behaviors / links

GDS – Similarity
Pearson Similarity

- Closest properties have higher scores
  - Covariance divided by product of standard deviations
    - $Pearson(A,B) = \frac{cov(A,B)}{\sigma_A \sigma_B} = \frac{\sum_{i=1}^{n}(A_i - \bar{A})(B_i - \bar{B})}{\sqrt{\sum_{i=1}^{n}(A_i - \bar{A})^2 (B_i - \bar{B})^2}}$
  - Between -1 and 1
  - Good for performances

```
MATCH (p1:Person {name: "Nicolas"})-[rated:RATED]->(movie)
WITH p1, gds.alpha.similarity.asVector(movie, rated.score) AS p1Vector
MATCH (p2:Person {name: "Cedric"})-[rated:RATED]->(movie)
WITH p1, p2, p1Vector, gds.alpha.similarity.asVector(movie, rated.score) AS p2Vector
RETURN p1.name AS from, p2.name AS to,
        gds.alpha.similarity.pearson(p1Vector, p2Vector,
                    {vectorType: "maps"}) AS similarity
```

➢ Degree of high correlation between nodes

GDS – Similarity
KNN Similarity

- Computes on all node pairs – [Wei Dong et al.]
  - Produces new relationships between each nodes
  - Param: nb iterations, threshold

```
CALL gds.beta.knn.stream('myGraph',
                { topK: 3,
                nodeWeightProperty: 'rating',
                randomSeed: 42, concurrency: 1, sampleRate: 1.0, deltaThreshold: 0.0 })
YIELD node1, node2, similarity
RETURN          gds.util.asNode(node1).name AS Person1,
                gds.util.asNode(node2).name AS Person2, similarity
ORDER BY similarity DESCENDING, Person1, Person2
```


GDS – Algorithms – Link prediction

Based on the topology of the graph

- Common Neighbors
- Total Neighbors
- Preferential Attachment
- Resource Allocations
- Adamic Adar
- Same Community

GDS – Link prediction
Common Neighbors

- Based on shared neighbors
  - $\mathbf{CN}(\boldsymbol{x}, \boldsymbol{y}) = |\boldsymbol{N}(\boldsymbol{x}) \cap \boldsymbol{N}(\boldsymbol{y})|$

Adjacent set of nodes of *y*

```
MATCH (p1:Person {name: 'Mark'})
MATCH (p2:Person {name: 'harrison'})
RETURN gds.alpha.linkprediction.commonNeighbors(p1, p2, {relationshipQuery: 'acted_in'}) AS
score
```


GDS – Link prediction
Total Neighbors

- Based on unique neighbors
  - $\mathbf{CN}(\boldsymbol{x}, \boldsymbol{y}) = |\boldsymbol{N}(\boldsymbol{x}) \cup \boldsymbol{N}(\boldsymbol{y})|$

```
MATCH (p1:Person {name: 'Mark'})
MATCH (p2:Person {name: 'harrison'})
RETURN gds.alpha.linkprediction.totalNeighbors(p1, p2, {relationshipQuery: 'acted_in'}) AS score
```

GDS – Link prediction
Preferential Attachment

- The more connected nodes are, the best they are attached
  - $\mathbf{PA}(x, y) = deg(x) \times deg(y)$
  - Two nodes with high degrees are more likely to be connected

```
MATCH (p1:Person {name: 'Mark'})
MATCH (p2:Person {name: 'Harrison'})
RETURN gds.alpha.linkprediction.preferentialAttachment(p1, p2, {relationshipQuery: 'acted_in'})
AS score
```


GDS – Link prediction
Resource Allocations

- Closeness of nodes on shared neighbors – [T. Zhou et al. 2009]
  - $RA(x, y) = \sum_{u \in N(x) \cap N(y)} \frac{1}{deg(u)}$

```
MATCH (p1:Person {name: 'Mark'})
MATCH (p2:Person {name: 'Harrison'})
RETURN gds.alpha.linkprediction.resourceAllocation(p1, p2, {relationshipQuery: 'acted_in'}) AS score
```

GDS – Link prediction
Adamic Adar

- Closeness of nodes on shared neighbors – Adamic & Adar (SocNet'03)
  - $A(x,y) = \sum_{u \in N(x) \cap N(y)} \frac{1}{log(deg(u))}$

```
MATCH (p1:Person {name: 'Mark'})
MATCH (p2:Person {name: 'Harrison'})
RETURN gds.alpha.linkprediction.adamicAdar(p1, p2, {relationshipQuery: 'acted_in'}) AS score
```


GDS – Link prediction
Same Community

- 1 if two nodes belongs to the same community
- 0 otherwise

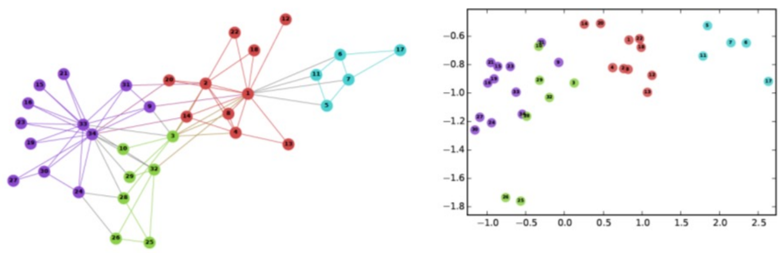
Graph Mining – Plan

1. Graph Data Science
   - Open Source Neo4j plugin
   - Cypher projection
2. Dedicated to graph analytics
   - Paths finding, Communities, Centrality, Similarity, Link prediction
3. GDS & Machine Learning
   - Node embedding, Node classification, Link prediction

   https://neo4j.com/docs/graph-data-science/current/algorithms/node-embeddings/
4. Advanced GDS
   - Graph data management memory/storage
   - Pregel
   - Neosemantics

GDS – Machine Learning

- Node Embedding
  - Nodes' vectorization in a low-dimensional representation.
  - FastRP
  - GraphSAGE
  - Node2Vec
- Models
  - Node Classification
  - Link Prediction



Input Output

https://neo4j.com/docs/graph-data-science/current/algorithms/node-embeddings/

GDS – Machine Learning
Fast Random Projection - CIKM'19

Random Projection Family

- From n to log(n) dimensions (*Johnsson-Lindenstauss* Lemma)
- Preserve similarity between nodes (neighborhood)
- Algorithm:
  - Random vectors for all nodes (Very sparse random projection – KDD'06)
  - Intermediate embedding by averaging neighbors (Euclidean norm*)
    - Several iterations*
    - Weights* on radius of neighbors
    - Can use relationship weights* and direction*
  - Result: Weighted sum of intermediate embeddings

* Hyperparameter

GDS – Machine Learning
Fast Random Projection

```
CALL gds.fastRP.stream(
        'myGraph',
        { embeddingDimension: 100,
          randomSeed: 42,
          iterationWeights: [0.5, 1.0, 1.0],
          normalizationStrengh: 0,
          relationshipWeightProperty: 'rating' }
) YIELD nodeId, embedding
```

Nodes' vector dimension

Weights for each iteration
(Contribution to the final embedding)

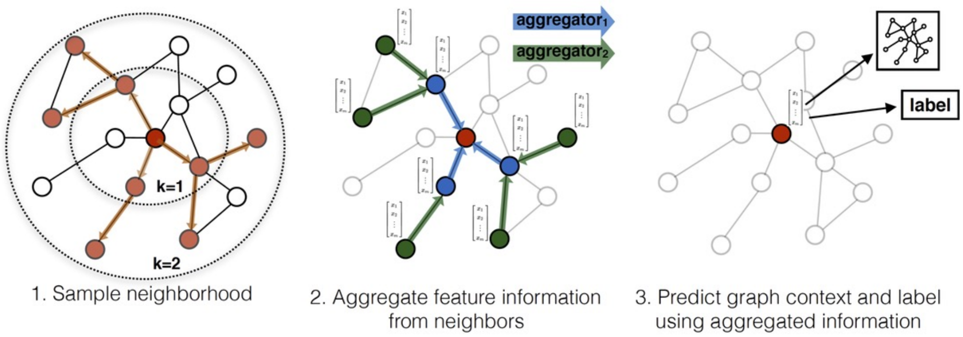
Degree normalization of initial random vectors
(power of this value)

.stream() can be replaced by .mutate() – add the embedding to the node

GDS – Machine Learning
GraphSAGE – NIPS'17

Inductive Algorithm

- Use node Features: sampling & aggregating neighbors' features
- Node embeddings L2-normalization



GDS – Machine Learning
GraphSAGE

- Train and name the generated model
  - Store in the **model catalog**

CALL gds.**beta**.graphSage.train(
'myGraph',
{ modelName: 'GraphSAGE1',
featureProperties: ['age', 'nationality', 'hobbies'],
aggregator: 'mean',
activationFunction: 'sigmoid',
sampleSizes: [25, 10] }
) YIELD modelInfo as info RETURN info.name as modelName, info.metrics.didConverge as didConverge, info.metrics.ranEpochs as ranEpochs, info.metrics.epochLosses as epochLosses

Mean (~GCN)
Pool (fully connected NN)

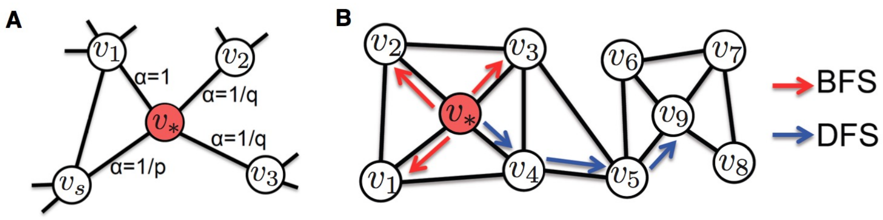
Sigmoid, ReLU

Nb of sample nodes per layer

https://snap.stanford.edu/node2vec/

GDS – Machine Learning
Node2Vec – SIGKDD'16

Second order Random Walk algorithm
- Based on structural equivalence
- Node embedding probabilities depending on the random step:
  - visited node v_*, previous node v_s, target node v_1, inOutDegrees v_2&v_3…
- Can use relationships' weights



GDS – Machine Learning
Node2Vec

- Train and name the generated model
  - Store in the **model catalog**

```
CALL gds.beta.node2vec.stream(
        'myGraph',
        { embeddingDimension: 128,
          walkLength: 80, walksPerNode: 10,     <- Nb of random walks
          inOutFactor: 1.0,                     <- Local (1.0) vs global (0.0) walk
          relationshipWeightProperty: "rating" }
) YIELD nodeId, embedding
```

GDS – Machine Learning Models
Node Classification

- Based on Node embeddings
  - Stored in the **model catalog**
  - Train vs test graphs
  - Evaluation metrics using *logistic regression*
    - F1_weighted, F1_macro, accuracy
    - Per class: F1, precision, recall, accuracy

https://neo4j.com/docs/graph-data-science/current/algorithms/ml-models/

GDS – Machine Learning Models
Node Classification

```
CALL gds.alpha.ml.nodeClassification.train(
        'myGraph',
        { nodeLabels: ['Person'],
          modelName: 'GraphSAGE1',
          featureProperties: ['age', 'nationality'],
          targetProperty: 'class',
          randomSeed: 2, holdoutFraction: 0.2, validationFolds: 5,
          metrics: [ 'F1_WEIGHTED' ],
          params: [ {penalty: 0.0625}, {penalty: 0.5}, {penalty: 1.0},
                  {penalty: 4.0} ] })
YIELD modelInfo
RETURN {penalty: modelInfo.bestParameters.penalty} AS
                winningModel,
        modelInfo.metrics.F1_WEIGHTED.outerTrain AS
                trainGraphScore,
        modelInfo.metrics.F1_WEIGHTED.test AS
                testGraphScore
```

```
CALL gds.alpha.ml.nodeClassification.predict.stream(
        'myGraph',
        { nodeLabels: ['Person', 'NewPerson'],
          modelName: 'GRAPHSAGE1',
          includePredictedProbabilities: true })
YIELD nodeId, predictedClass, predictedProbabilities
WITH gds.util.asNode(nodeId) AS houseNode, predictedClass,
        predictedProbabilities
WHERE houseNode:UnknownHouse
RETURN houseNode.color AS classifiedHouse, predictedClass,
        floor(predictedProbabilities[predictedClass] * 100) AS confidence
ORDER BY classifiedHouse
```

GDS – Machine Learning Models
Link Prediction

- Predicting relationships
  - Undirected
  - Node features combination: L2, Hadamard, Cosine
  - Evaluation *ACUPR metric* using logistic regression
  - Stored in the **model catalog**
  - *topN* most probable predictions
- Generate train relationships: gds.alpha.ml.splitRelationships()

Graph Mining – Plan

1. Graph Data Science
   - Open Source Neo4j plugin
   - Cypher projection
2. Dedicated to graph analytics
   - Paths finding, Communities, Centrality, Similarity, Link prediction
3. GDS & Machine Learning
   - Node embedding, Node classification, Link prediction
4. Advanced GDS
   - Graph data management memory/storage
   - Pregel
   - Neosemantics

GDS – Memory Estimation

- Graph algorithms applied in main memory
  - Need to be configured

```
CALL gds[.<tier>].<algorithm>.<execution-mode>.estimate(
          graphNameOrConfig: String or Map,
          configuration: Map )
YIELD     nodeCount: Integer, relationshipCount: Integer,
          requiredMemory: String,
          treeView: String, mapView: Map,
          bytesMin: Integer, bytesMax: Integer,
          heapPercentageMin: Float, heapPercentageMax: Float
```

https://neo4j.com/docs/graph-data-science/current/common-usage/memory-estimation/

GDS – Operations reference

GDS functions reference
https://neo4j.com/docs/graph-data-science/current/appendix-a/

Cypher RefCard
https://neo4j.com/docs/cypher-refcard/current/

Performance Tuning

- Performance tuning : https://neo4j.com/docs/operations-manual/current/performance/
  - Look at : locks&deadlocks -> updating nodes/relationships…
- FileSystem issue : https://community.neo4j.com/t5/neo4j-graph-platform/neo4j-import-tools-slow-ingestion/m-p/42566
- Import for small datasets : https://neo4j.com/docs/operations-manual/current/tutorial/neo4j-admin-import/#_import_a_small_data_set
  - Neo4j-admin import (shell command)
- Import for large datasets :
  - https://neo4j.com/blog/bulk-data-import-neo4j-3-0/
    - Look after “LOAD CSV tips and Tricks”
  - https://community.neo4j.com/t5/neo4j-graph-platform/extremely-slow-import-for-large-graph-database-using-neo4j-admin/m-p/32238/highlight/true#M16934
- Cache size issue : https://neo4j.com/developer/guide-performance-tuning/#_page_cache_sizing

Pregel - SIGMOD'10 (Google)

Vertex-centric computation model

- Build your algorithms with functions (Java API)
- Supersteps : multiple iterations
  - Computation at node level
    - Interactions with the graph – *message passing*
    - Combination with local values (or state value after several iterations)
  - Iterations' end: no more messages or fixed number
  - Parallelized (one node = one thread)

https://neo4j.com/docs/graph-data-science/current/algorithms/pregel-api/
GitHub Pregel Examples

Pregel
Example: Label Propagation

```
public class LabelPropagationPregel implements PregelComputation<LabelPropagationPregelConfig> {

public static final String LABEL_KEY = "label";

public PregelSchema schema(LabelPropagationPregelConfig config) {
        return new PregelSchema.Builder().add(LABEL_KEY, ValueType.LONG).build(); }

public void init(InitContext<LabelPropagationPregelConfig> context) {
        context.setNodeValue(LABEL_KEY, context.nodeId()); }

...
```

Define messages schema between nodes

Initialize with node's Id


```
public void compute(ComputeContext<LabelPropagationPregelConfig> context, Messages messages) {
        if (context.isInitialSuperstep()) { context.sendToNeighbors(context.nodeId()); }
        else { if (messages != null) {
                long oldValue = context.longNodeValue(LABEL_KEY); long newValue = oldValue;
                long[] buffer = new long[context.degree()];
                int messageCount = 0;
                for (var message : messages) { buffer[messageCount++] = message.longValue(); }
                int maxOccurences = 1;

                if (messageCount > 1) {
                        Arrays.sort(buffer, 0, messageCount);
                        int currentOccurences = 1;
                        for (int i = 1; i < messageCount; i++) {
                                if (buffer[i] == buffer[i - 1]) {
                                        currentOccurences++;
                                        if (currentOccurences > maxOccurences) {
                                                maxOccurences = currentOccurences; newValue = buffer[i]; }
                                } else { currentOccurences = 1; } } }

                if (maxOccurences == 1) { newValue = Math.min(oldValue, buffer[0]); }
                if (newValue != oldValue) { context.setNodeValue(LABEL_KEY, newValue);
                        context.sendToNeighbors(newValue); }}}
        context.voteToHalt(); }
```

If the chosen node has not been labeled send his nodeId to neighbors

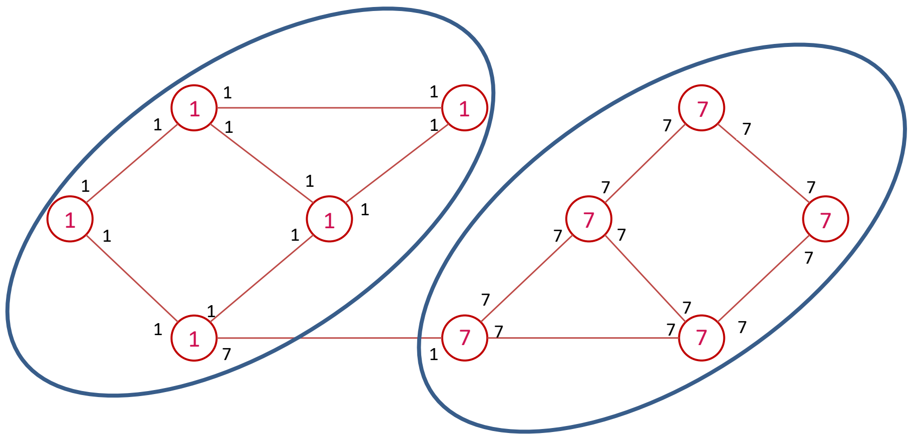
Receive all neighbors' messages (label)

Get top labels' occurrences

Change the label if neighbors' occurrences are higher and send to neighbors

Ask to end Supersteps iterations

Pregel
Example: Label Propagation



Pregel
Example: PageRank

```
static final String PAGE_RANK = "pagerank";
private static boolean weighted;

public PregelSchema schema(PageRankPregelConfig config) {
    return new PregelSchema.Builder().add(PAGE_RANK, ValueType.DOUBLE).build();
}

public void init(InitContext<PageRankPregelConfig> context) {
    var initialValue = context.config().seedProperty() != null
        ? context.nodeProperties(context.config().seedProperty()).doubleValue(context.nodeId())
        : 1.0 / context.nodeCount();
    context.setNodeValue(PAGE_RANK, initialValue);

    weighted = context.config().hasRelationshipWeightProperty();
}
```

Initial probability state

```
public void compute(ComputeContext<PageRankPregelConfig> context, Messages messages) {
      double newRank = context.doubleNodeValue(PAGE_RANK);

      if (!context.isInitialSuperstep()) {
         double sum = 0;
         for (var message : messages) { sum += message; }

         var dampingFactor = context.config().dampingFactor();
         var jumpProbability = 1 - dampingFactor;

         newRank = (jumpProbability / context.nodeCount()) + dampingFactor * sum;
         context.setNodeValue(PAGE_RANK, newRank);
      }

      if (weighted)
         context.sendToNeighbors(newRank);
      else
         context.sendToNeighbors(newRank / context.degree());
}

public double applyRelationshipWeight(double nodeValue, double relationshipWeight) {
      return nodeValue * relationshipWeight;}
```

Summing neighbors' messages (state of PageRank)

Random walk probability (damping factor)

Current PageRank score (neighbors + random walk)

Weighted: call "*applyRelationshipWeight*"

Unweighted

Weight implied by the out-relationship

Neosemantics

n10s neosemantics

RDF support

- Importing triples as property graphs (rdf4j):
  - 2 types of triples
    - Node, directed relation+type, node
    - Node, property, value
  - Require ontology: OWL, Turtle
- SPARQL queries handling vs Cypher queries
- Graph App (UI): *n10s*
- Inference with neosemantics: simple rules
  - Hierarchies of categories

https://neo4j.com/labs/neosemantics/tutorial/