GRAPH & MINING



WITH



Nicolas Travers

GRAPH MINING?

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



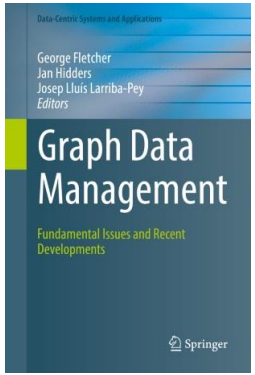
You know what you're looking for and making a decision

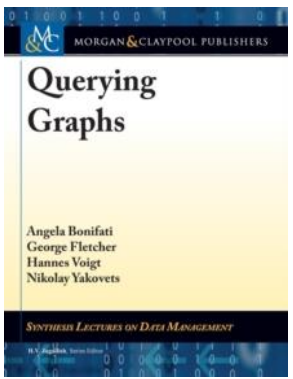
You're learning the overall structure of a network, updating data, and predicting

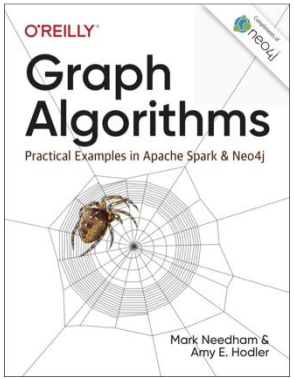
INTRODUCTION

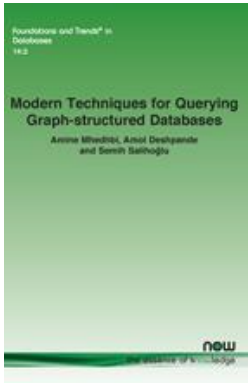
- The Graph Data Science (GDS) Library
- Graph Mining
- Course Organization

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











Links on images

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
   - Indexing
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
5. GQL

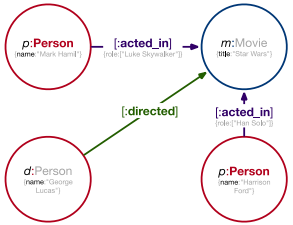


LIKE,SHARE, FOLLOW ! @poledevinci | DEVINCI.FR

GRAPH MINING WORKFLOW



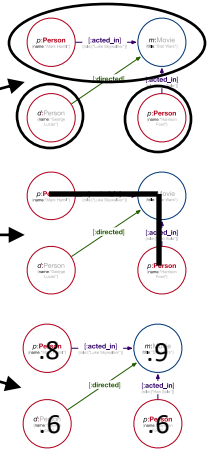
Named graph in main memory



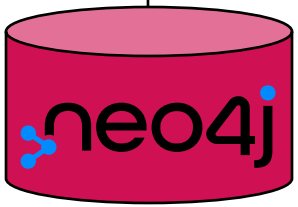
clustering

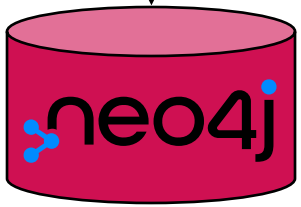
paths

centralities



GDS
Cypher projection

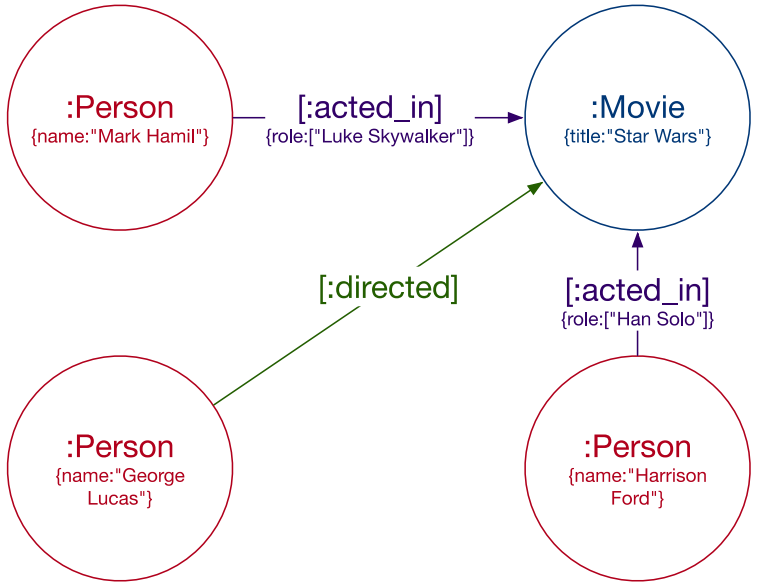




GDS – GRAPH PROJECTION

Cypher projection = Cypher query + storage (graph catalog)

```
MATCH (p:Person)-->(m:Movie)
WHERE m.title CONTAINS "Star Wars"
WITH gds.graph.project(
        "SW_saga",
        p,
        m,
        {relationshipProperties: r{NB:sum(r)}}
) AS g
RETURN
        g.graphName AS graph,
        g.nodeCount AS nodes,
        g.relationshipCount AS rels
```



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

GDS – SYNTAX FOR ALGORITHMS

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,
    relationshipTypes: ["NB_links"]
  }
) 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.)

LIKE,SHARE, FOLLOW ! @poledevinci | DEVINCI.FR

GRAPHS – RECALL

Graphs:

(bi|mono|k)**partite**, (un)**weigthed**, (un)**directed**, (a)**cyclic**, (dis)**connected**, (rooted|binary|spanning) **trees**, **clique**

Nodes: **(in|out) degree**

Density: $^{2R}/_{N(N-1)}$

Diameter: max(rel) between two nodes

Graph algorithms:

Pathfinding, Centrality, Community detection

NEO4J INDEXES

Before importing data!

Build index to optimize Cypher queries

- **CREATE** TEXT **INDEX FOR** (a:Person) **ON** (a.name);
- Can be made on a pattern (FOR <pattern> ON), Node & Relationship

Range Index : Range values. Single or composite index.

Text Index : String values

Point Index : Point values (lat/long)

Full-Text index : Lucene index. Search engine like

Lookup Index on labels

- **CREATE** LOOKUP **INDEX FOR** (n) **ON** *labels*(n);
- **CREATE** LOOKUP **INDEX FOR** ()-[r]-() **ON** *types*(r);

INDEX HINTS

- **Give hints to the query optimizer for indexes**
- **MATCH** (p)… **USING INDEX** p:Person(name)

Vectors and embeddings Indexes

- **CREATE** VECTOR **INDEX** personMovies
  FOR (p:Person) **ON** p.embedding
  OPTIONS { indexConfig:
  { 'vector.dimensions': 1536,
  'vector.similarity_function': 'cosine'}}
- Quantization can be enabled

Optimal Neo4j memory configuration or Vector Indexes

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/

LIKE,SHARE, FOLLOW ! @poledevinci
DEVINCI.FR

GDS – OPERATIONS REFERENCE

GDS functions reference

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

Cypher RefCard

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

LIKE,SHARE, FOLLOW ! @poledevinci
DEVINCI.FR

GRAPH MINING – PLAN

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

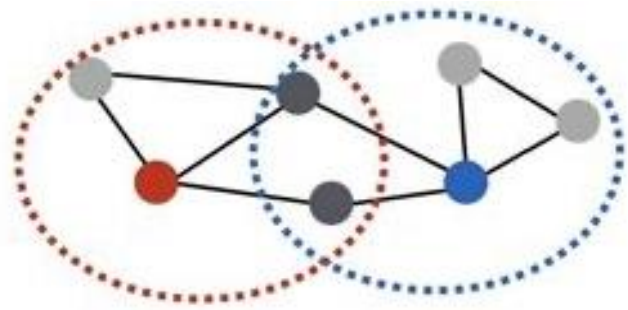




SIMILARITY

Based on nodes similarity

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



JACCARD SIMILARITY

Comparison of shared neighbors

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

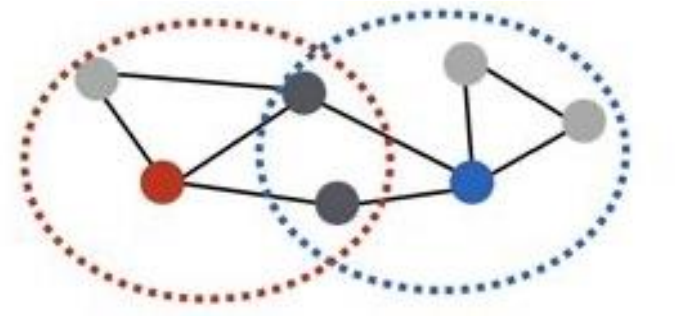
➢ Bi-partite graph comparison, recommendation

```
CALL gds.nodeSimilarity.filtered.stream(
                'SW_saga',
                {sourceNodeFilter: 'Movie',
                 targetNodeFilter: 'Person',
                 similarityMetric: "JACCARD" } )

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
```



OVERLAP SIMILARITY

Measures the overlap between 2 nodes

- $O(A,B) = \frac{|A \cap B|}{min(|A|,|B|)}$

➢ **Dedicated to small numbers of relationships**

- Adapted to big graphs

```
WITH X AS pA, Y AS pB

// NB of area (pA) for Node X
MATCH (:Movie {title: pA}) <-[r1:acted_in]- (itA)
WITH pA, pB, collect({k: elementId(itA), v: toFloat(r1.NB)}) AS aList
// NB of area (pA) for Node Y
MATCH (:Movie {title: pB}) <-[r2:acted_in]- (itB)
WITH aList, collect({k: elementId(itB), v: toFloat(r2.NB)}) AS bList

// Mapping of "NB" per Person
WITH apoc.map.fromPairs([x IN aList | [x.k, x.v]]) AS A,
     apoc.map.fromPairs([y IN bList | [y.k, y.v]]) AS B

// Merge keys and align vectors (missing value = 0)
WITH A, B, apoc.coll.toSet(keys(A) + keys(B)) AS allItems
WITH [k IN allItems | coalesce(A[k], 0.0)] AS vA,
     [k IN allItems | coalesce(B[k], 0.0)] AS vB,
     size(allItems) AS dim

//Compute the similarity
RETURN gds.similarity.euclidean(vA, vB) AS euclidean,
       gds.similarity.cosine(vA, vB) AS cosine,
       dim AS dimensions
```

EUCLIDEAN | COSINE SIMILARITY

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

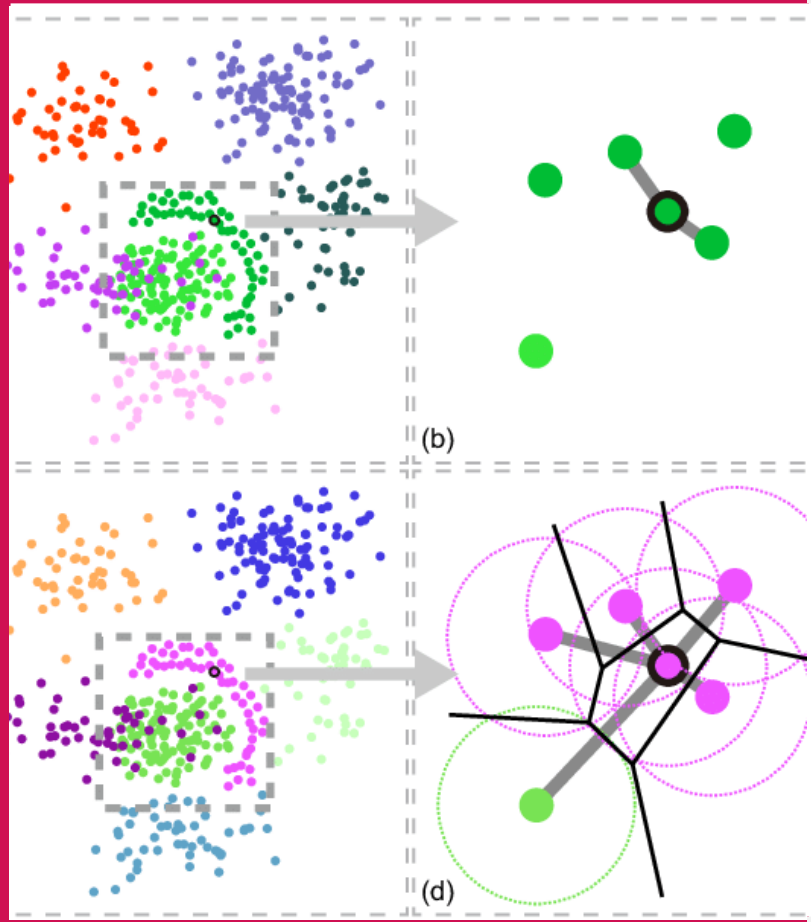
➢ **Compare similar behaviors / links**

PEARSON SIMILARITY

$$Pearson(A,B) = \frac{cov(A,B)}{\sigma_A \sigma_B} = \frac{\sum_{i=1}^{n}(A_i - \bar{A})(B_i - \bar{B})}{\sqrt{\sum_{i=1}^{n}(A_i - \bar{A})^2 (B_i - \bar{B})^2}}$$

- **Closest properties have higher scores**
  - Covariance divided by product of standard deviations
  - Between -1 and 1
  - Good for performances

➢ **Degree of high correlation between nodes**



KNN SIMILARITY

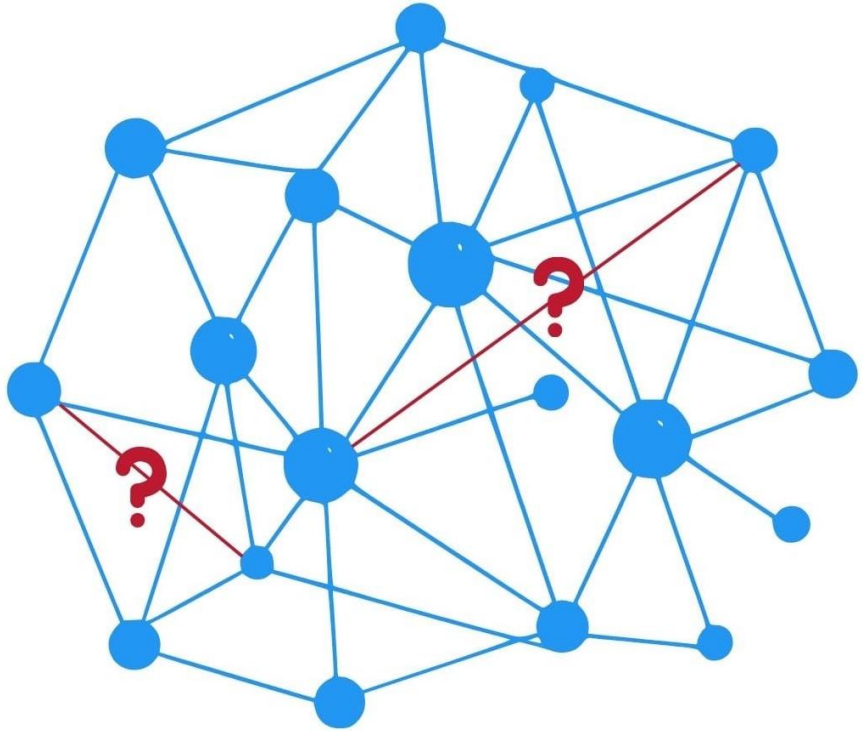
- Computes on all node pairs – Wei Dong et al.
  - Produces new relationships between each nodes
  - Param: nb iterations, threshold

```
CALL gds.knn.write('myGraph',
      {topK: 3, nodeProperties: ['rating'],
       writeRelationshipType: 'SIMILAR', writeProperty: 'score',
       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
```

TOPOLOGICAL LINK PREDICTION

Based on the topology of the graph

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

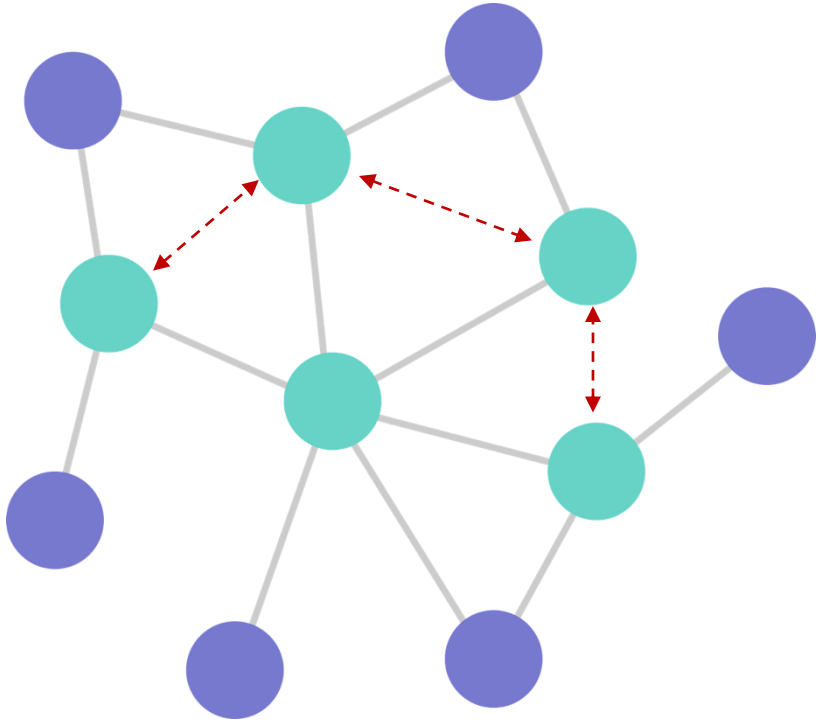


COMMON NEIGHBORS

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

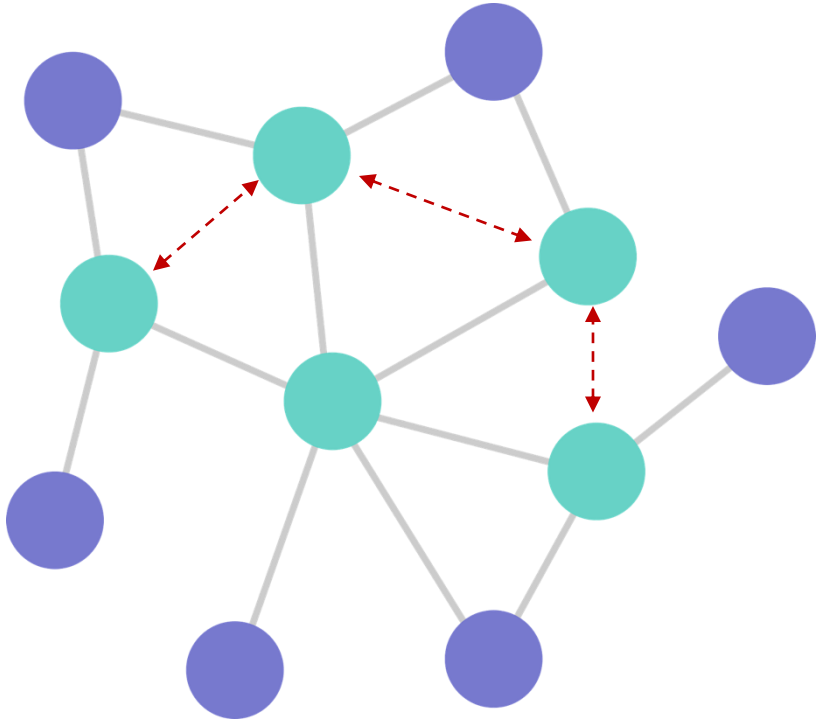
Adjacent set of nodes of *y*

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



TOTAL NEIGHBORS

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



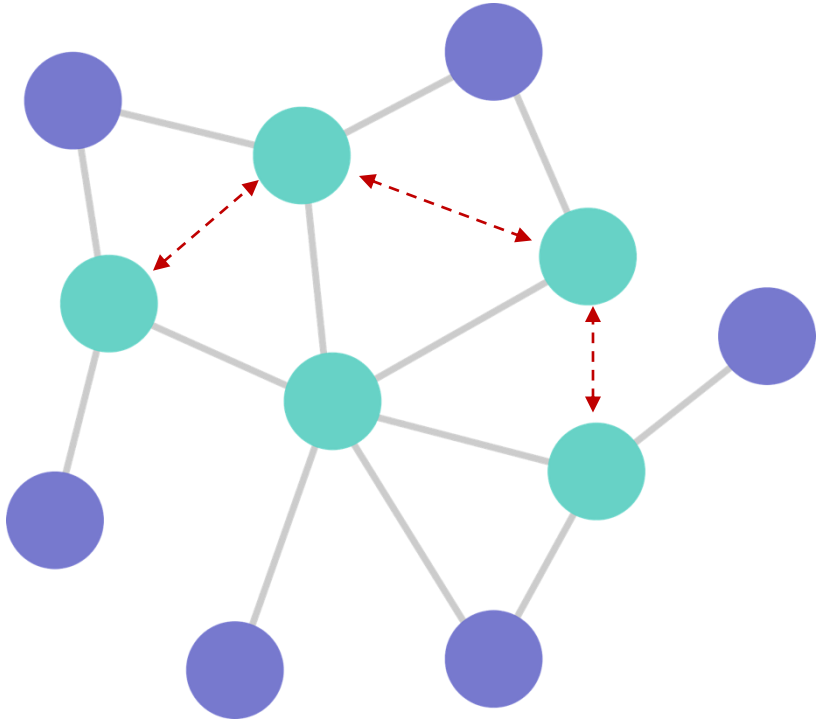
```
MATCH (p1:Person {name: 'Mark'})
MATCH (p2:Person {name: 'harrison'})
RETURN gds.alpha.linkprediction.totalNeighbors(
        p1, p2, {relationshipQuery: 'acted_in'}) AS score
```


LIKE,SHARE, FOLLOW ! @poledevinci | DEVINCI.FR

PREFERENTIAL ATTACHMENT

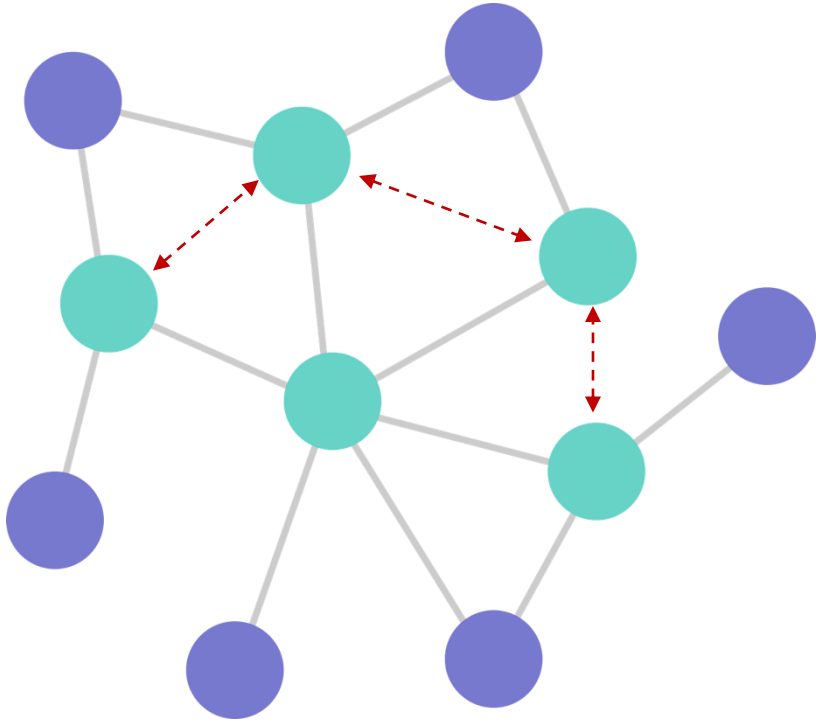
- The more connected nodes are, the best they are attached
  - $\mathbf{PA}(x, y) = \boldsymbol{deg}(x) \times \boldsymbol{deg}(y)$
  - Two nodes with high degrees are more likely to be connected

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



RESOURCE ALLOCATIONS

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



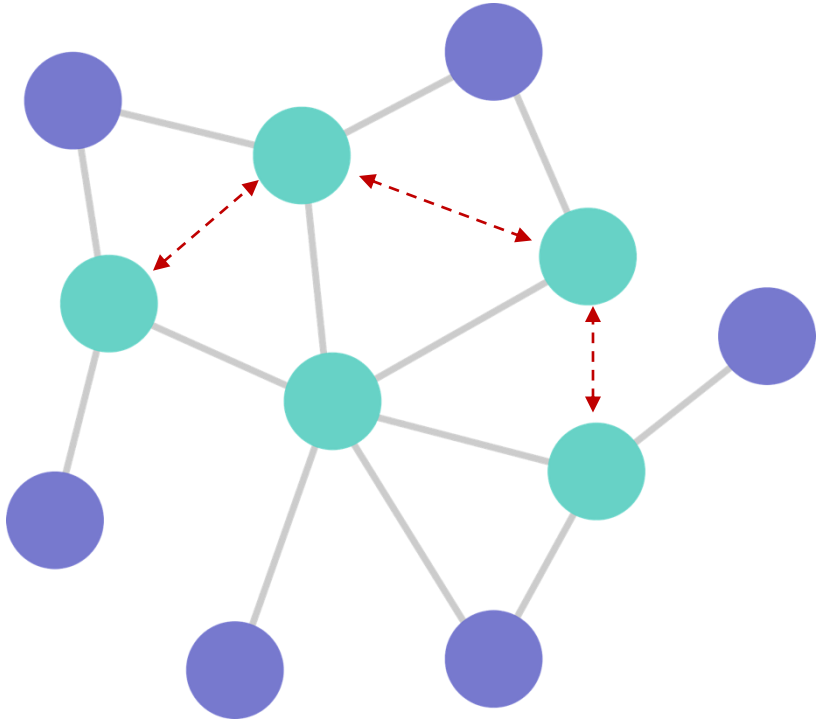
```
MATCH (p1:Person {name: 'Mark'})
MATCH (p2:Person {name: 'Harrison'})
RETURN gds.alpha.linkprediction.resourceAllocation(
        p1, p2, {relationshipQuery: 'acted_in'}) AS score
```


LIKE,SHARE, FOLLOW ! @poledevinci | DEVINCI.FR

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
```

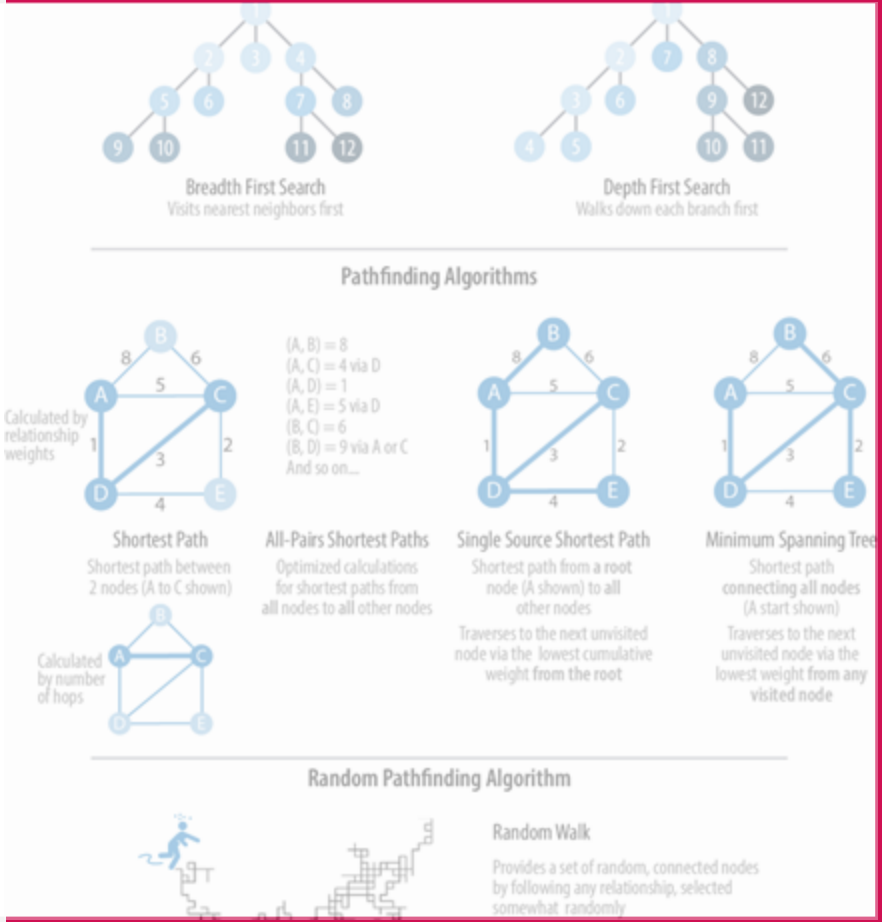


SAME COMMUNITY

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

PATH FINDING ALGORITHMS

- 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



BREADTH|DEPTH FIRST SEARCH

For given starting node

BFS

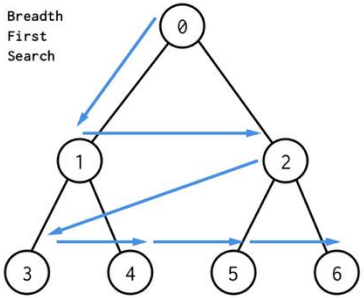
- Breadth before to seen how far are all nodes

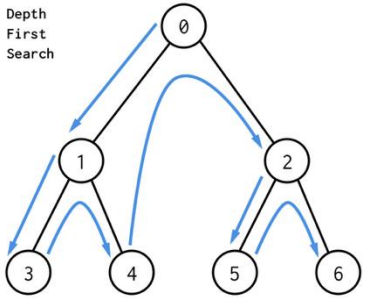
➢ *Likelihood of finding a node*

DFS

- Depth before backtraching

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



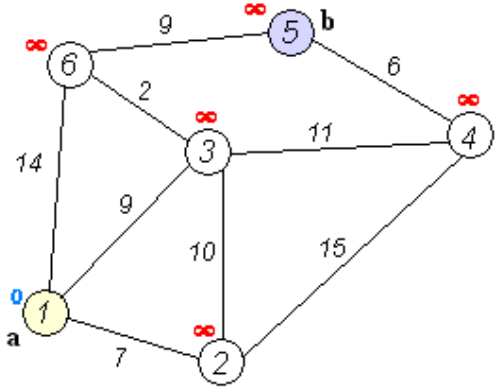



```
MATCH (source:Person{name:'Mark Hamil'})
CALL gds.bfs.stream('SW_saga', {
                sourceNode: source })
YIELD path
RETURN path
```

SHORTEST PATH

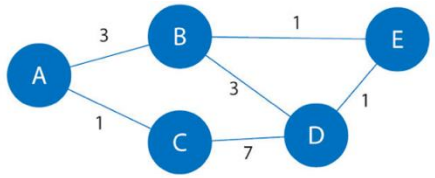
- **Dijkstra**
  - Iteration on finding the lowest-weight relationship
- **A***
  - Heuristic function cost to expand paths
  - Approximation
- **Yen K**
  - Calculates the *k* shortest path at the same time (alternative paths)

➢ **Finding directions, distance between nodes**



Others:
Delta-Stepping Single-Source Shortest Path
Bellman-Ford Single-Source Shortest Path

ALL PAIRS SHORTEST PATH



APSP

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

➢ Understanding alternate routes

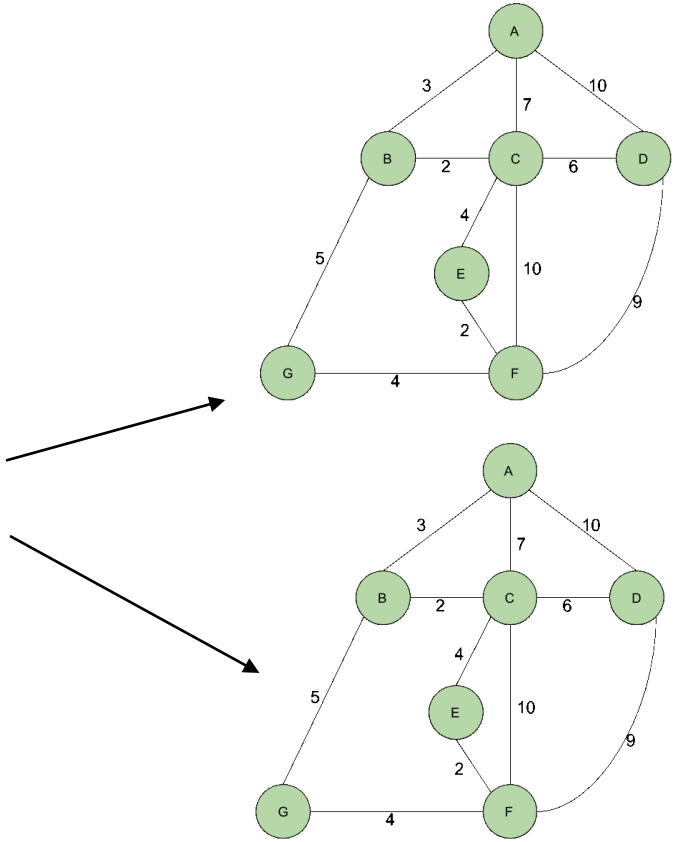
All nodes start with a ∞ distance and then the start node is set to a 0 distance			Each Step Keeps or Updates to the Lowest Value Calculated so Far Only steps for node A to all nodes shown				
			1st from A	2nd from A to C to Next	3rd from A to B to Next	4th from A to E to Next	5th from A to D to Next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

LIKE,SHARE, FOLLOW ! @poledevinci | DEVINCI.FR

MINIMUM SPANNING TREE

MST (Boruvka'26, Kruskal'56, Prim'57)

- 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

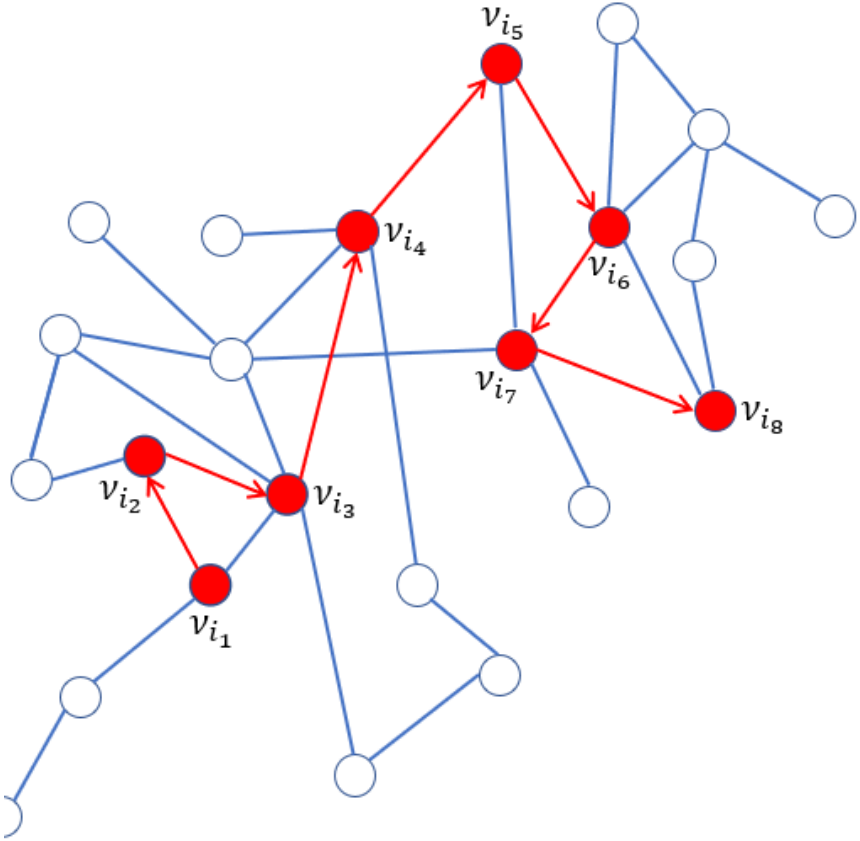
Variation: Prize-collecting Steiner Tree

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

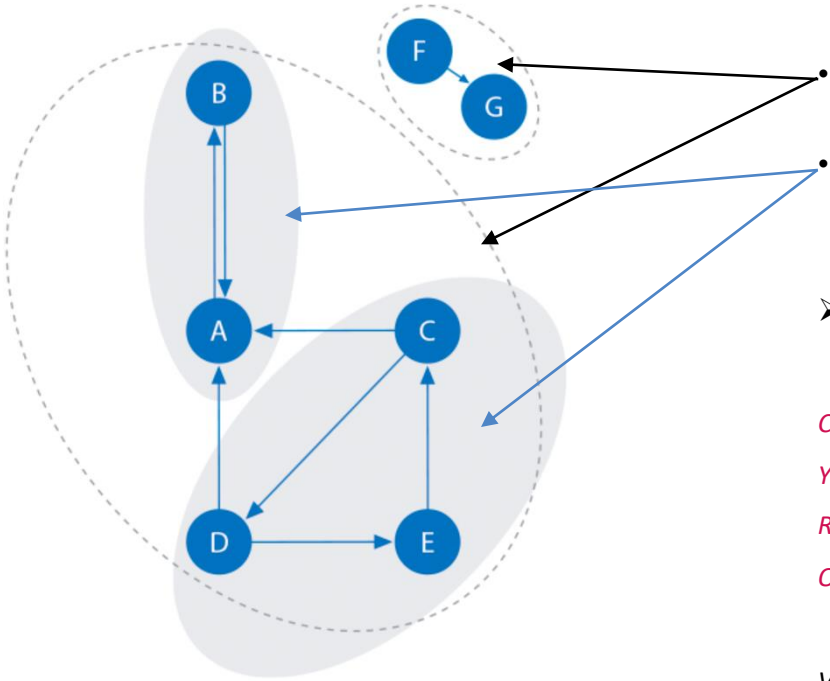




COMMUNITY DETECTION

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

CONNECTED COMPONENTS



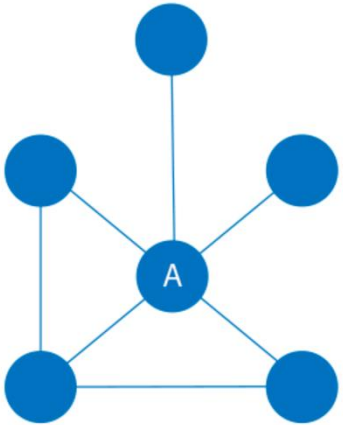
- **Connected** =
  all nodes reachable (no direction)
- **Strongly connected** (scc) =
  all nodes reachable in both directions

➢ *Fast grouping for other algo*

```
CALL gds.scc.stream('SW_saga')
YIELD nodeId, counts
RETURN gds.util.asNode(nodeId).name AS name, counts
ORDER BY name ASC
```

Variation: weakly connected (not strongly)

TRIANGLE COUNT



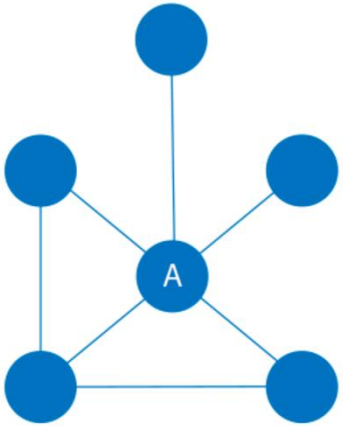
$T_A = 2$

Measures how many nodes wrt.

Nb triangles that pass through a node

- *Estimate group stability, small world behavior*

NB: The graph must be <u>undirected</u>



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

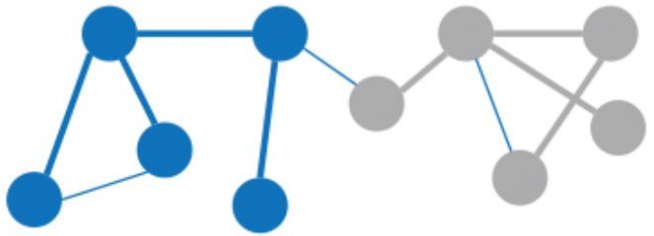
LOCAL CLUSTERING COEFFICIENT

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

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

➢ *Estimate group stability, small world behavior*

LPA push mode

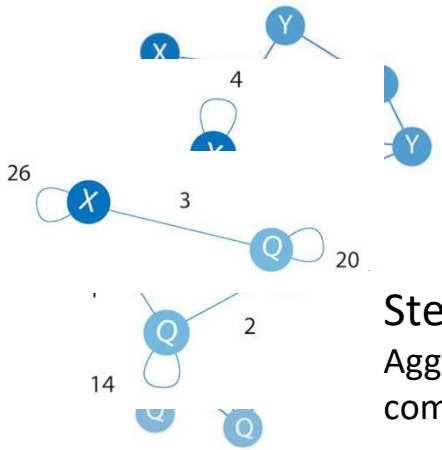


LABEL PROPAGATION

- 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

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

Variation: Speaker-Listener (SPLLPA) – multiple communities per node



Pass 2

Step 2

Aggregate communities

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

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.

LOUVAIN

J. OF STATISTICAL MECHANICS'08

- **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

➢ ***Fraud analysis, detecting discrete behavior***

Variation: ***Leiden*** *(more connected components)*

SOME MORE ALGORITHMS

- ***Clique counting***
  - *Nb of cliques within the community*
- ***Conductance metric***
  - *Ratio of in/out relationship between communities*
- ***HDBSCAN***
  - *Hierarchy of clusters based by varying the density of the community*
- ***K-CORE Decomposition***
  - *Based on maximal subgraphs with a degree of K*
- ***K-1 coloring***
  - *1 color per node, no same colors for neighbors, minimum nb of colors*
- ***K-Means Clustering***
  - *Node classification based on node properties (float array)*
- ***Approximate Maximum k-cut***
  - *Maximum cut (relationship removal) which maximizes the sum of weights within communities*

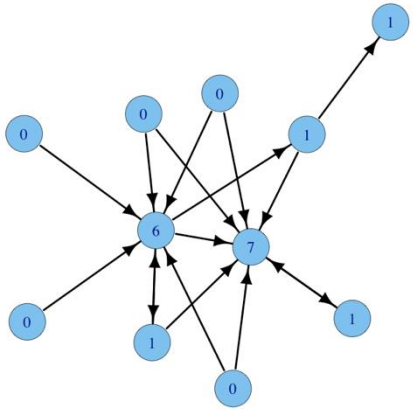
CENTRALITY

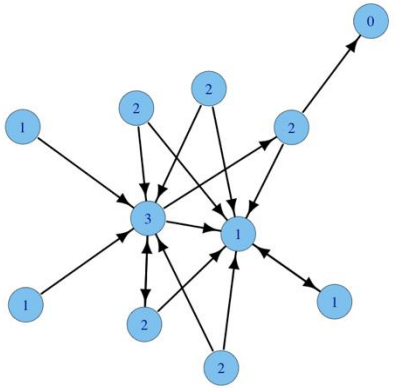
- Degree Centrality
- Closeness Centrality
- Betweenness Centrality
- Eigenvector Centrality
- PageRank
- Personalized PageRank
- ArticleRank
- Assortativity



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**






```
CALL gds.degree.stream('SW_saga')
YIELD nodeId, score
RETURN gds.util.asNode(nodeId).name AS name,
       score AS best_SW_movie
ORDER BY best_SW_movie DESC, name DESC
```

➢ ***Most important nodes***

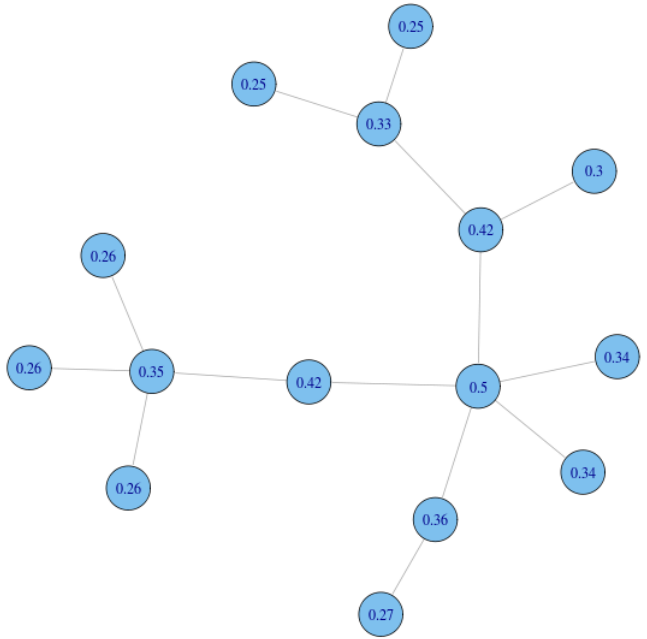
LIKE,SHARE, FOLLOW ! @poledevinci | DEVINCI.FR

CLOSENESS CENTRALITY

Capacity of a node to spread information

- 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



LIKE,SHARE, FOLLOW ! @poledevinci | DEVINCI.FR

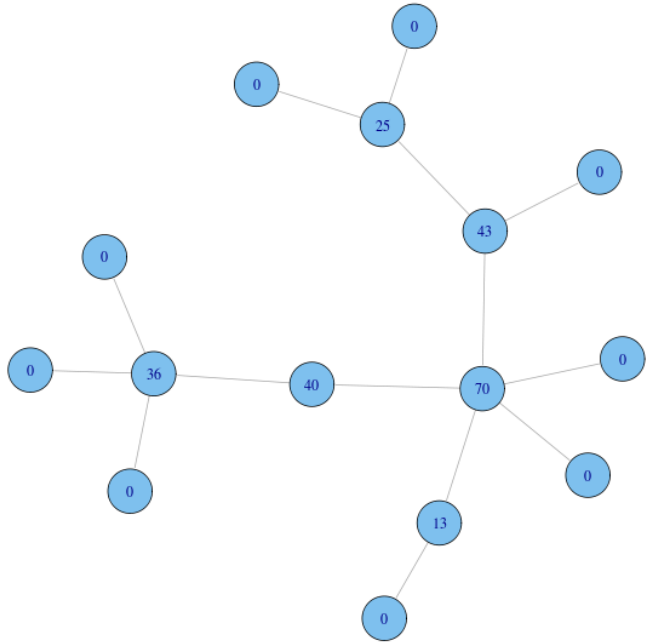
BETWEENNESS CENTRALITY

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 *n*
- Can be weighted

Approximation: RA-Brandes algo

➢ Influencers, bottlenecks, control points,



EIGENVECTOR CENTRALITY

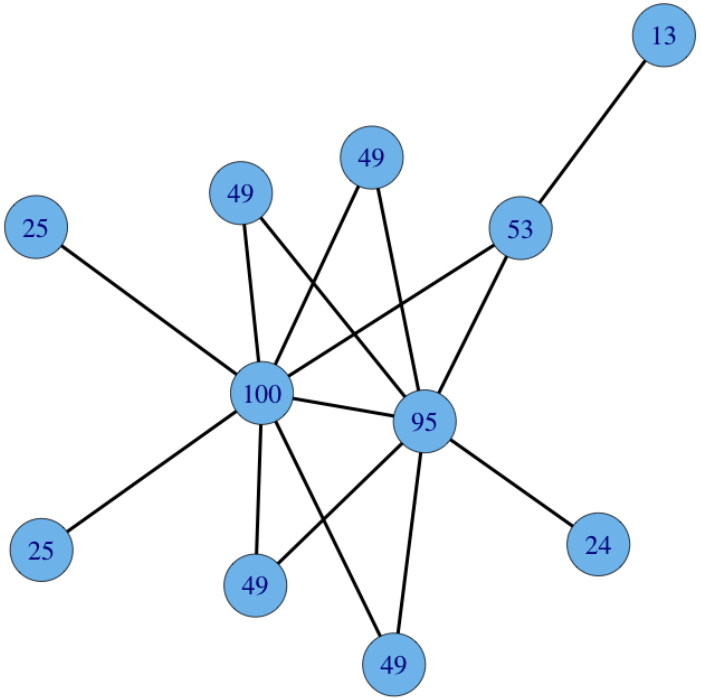
A node is important if it is linked to by other important nodes

- Transitive influence of nodes
- Power iteration approach

$$C_E(n) = \frac{1}{\lambda}\sum_k a_{k,n} C_E(k)$$

- $a_{k,n}$: adjacency matrix
- λ: Eigenvalue (normalized)

➢ ***Opinion influence over a network, ranking systems, firing rate for neural networks, comparison of multilayer graphs***



PAGERANK - GOOGLE

Transitive influence of nodes

- Derived from Eigenvector

Ranking of the potential impact of nodes: PR(u)

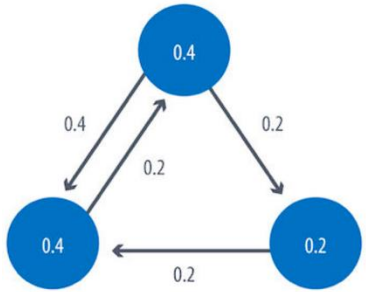
$$PR(u) = (1-d) + d\left(\frac{PR(T1)}{C(T1)} + \ldots + \frac{PR(Tn)}{C(Tn)}\right)$$

- Damping factor (d): 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***



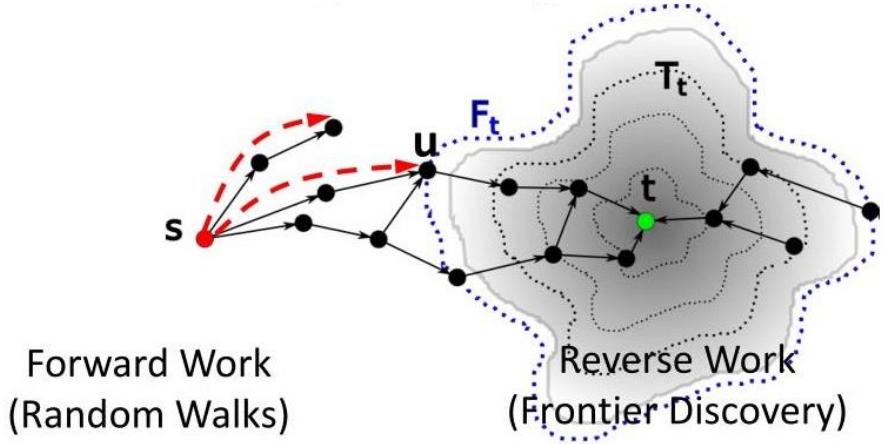
PAGERANK VARIANTS

- **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*



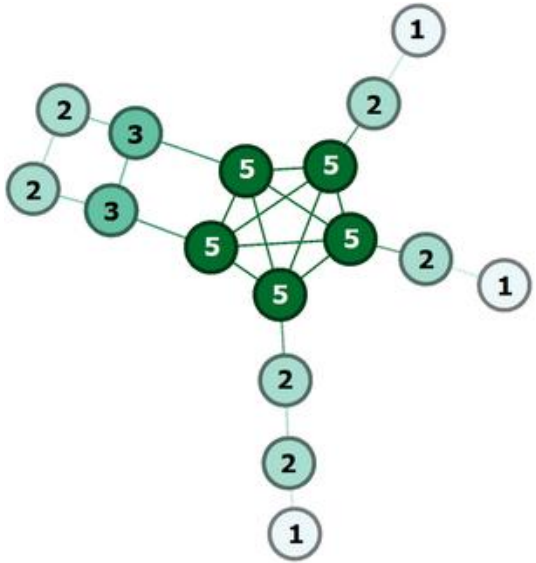
LIKE,SHARE, FOLLOW ! @poledevinci | DEVINCI.FR

ASSORTATIVITY [NEWMAN'22]

Connectivity measure

- **Average degree centrality of direct neighbors**

➢ **Homophily in Social Science**

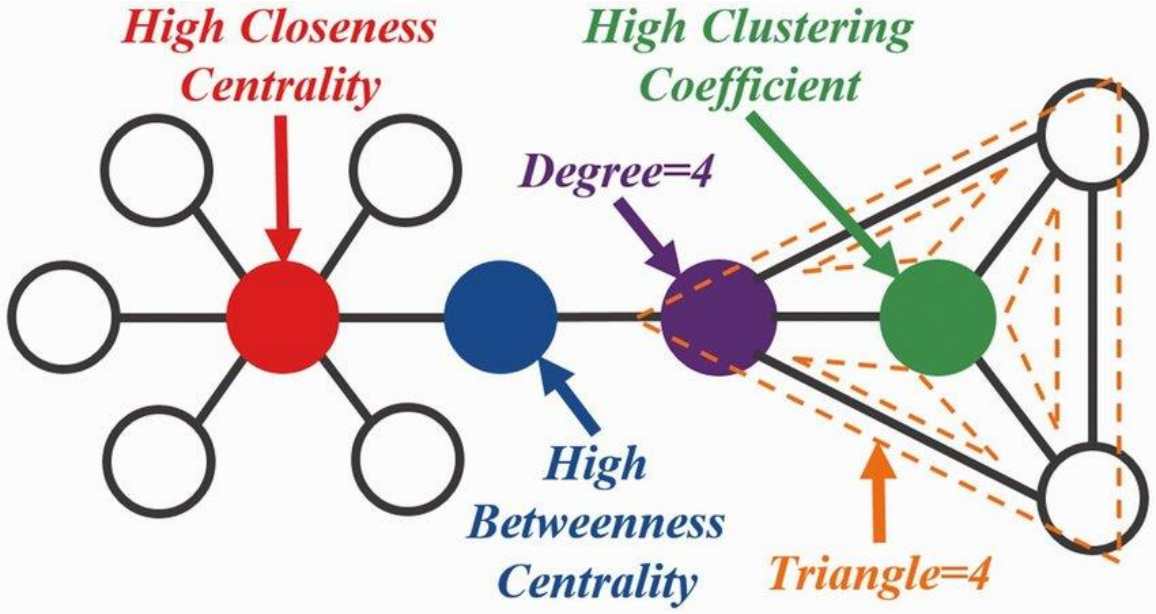


SOME MORE

- **Articulation Points**
  - Removing a node increases the number of connected components
- **Bridges**
  - Removing a relationship increases the number of connected components
- **CELF**
  - Influence maximisation (Independent Cascade propagation model)
- **Harmonic Centrality**
  - ~closeness centrality (sums the inverse of distances)
- **HITS (alpha version)**
  - Hyperlink-Induced Topic Search: *hub* (value of relationships) vs *authority* (importance of a node)

LIKE,SHARE, FOLLOW ! @poledevinci DEVINCI.FR

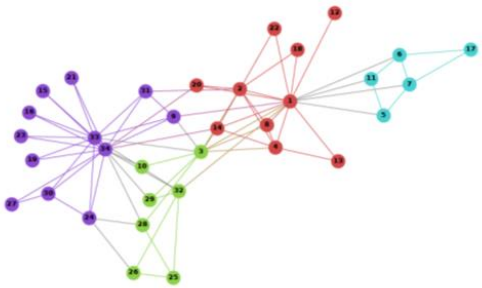
SUMMARY


High Closeness Centrality
High Clustering Coefficient
Degree=4
High Betweenness Centrality
Triangle=4

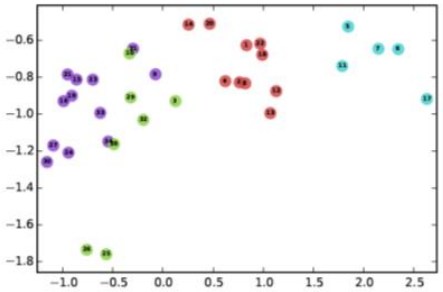


GRAPH MINING – PLAN

1. Graph Data Science
   - Open Source Neo4j plugin
   - Cypher projection
2. Dedicated to graph analytics
   - Similarity , Link prediction, Paths finding, Communities, Centrality
3. GDS & Machine Learning
   - Node embedding, Node classification, Link prediction, GraphRAG
   - https://neo4j.com/docs/graph-data-science/current/algorithms/node-embeddings/
4. Advanced GDS
   - Graph data management memory/storage
   - Pregel
   - Neosemantics
5. GQL



Input

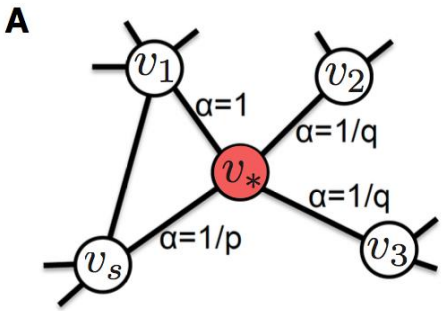


Output

GRAPHS & MACHINE LEARNING

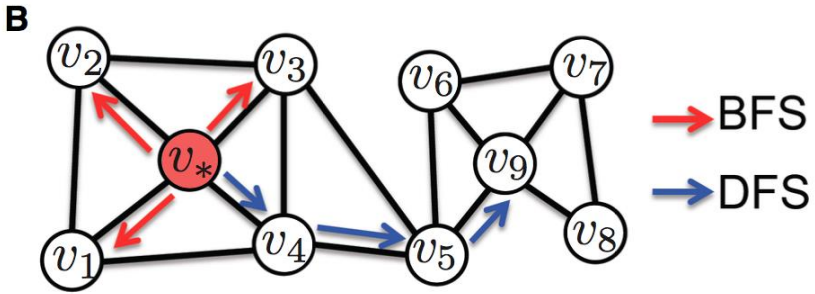
- Node Embedding
  - Nodes' vectorization in a low-dimensional representation.
  - Node2Vec
  - GraphSAGE
  - FastRP
  - #GNN
- Models
  - Node Classification
  - Link Prediction

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



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



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

NODE2VEC

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

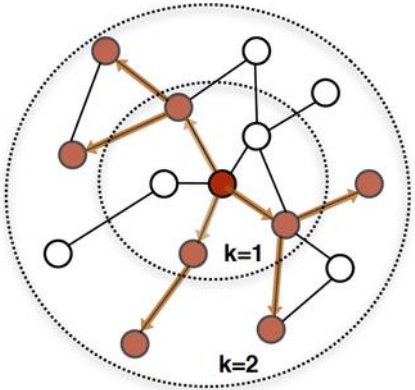
```
CALL gds.node2vec.stream(
        'myGraph',
        { embeddingDimension: 128,
          walkLength: 80, walksPerNode: 10,
          inOutFactor: 1.0,
          relationshipWeightProperty: "rating" }
) YIELD nodeId, embedding
```

Nb of random walks

Local (1.0) vs global (0.0) walk

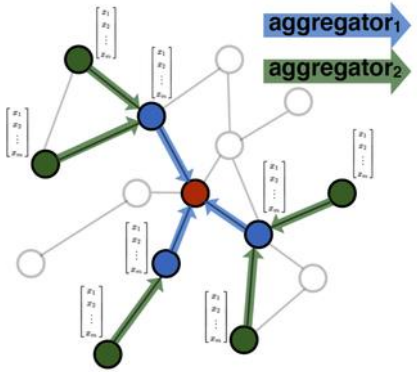
LIKE,SHARE, FOLLOW ! @poledevinci | DEVINCI.FR

GRAPHSAGE [NIPS'17]

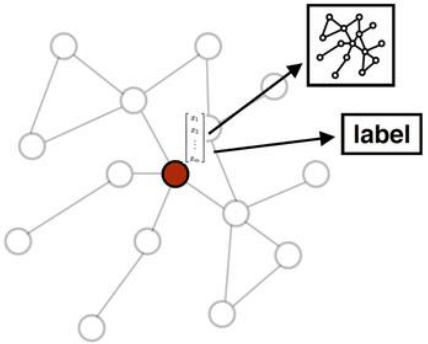


1. Sample neighborhood

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



2. Aggregate feature information from neighbors



3. Predict graph context and label using aggregated information

GRAPHSAGE

Train and name the generated model

- Store in the **model catalog**

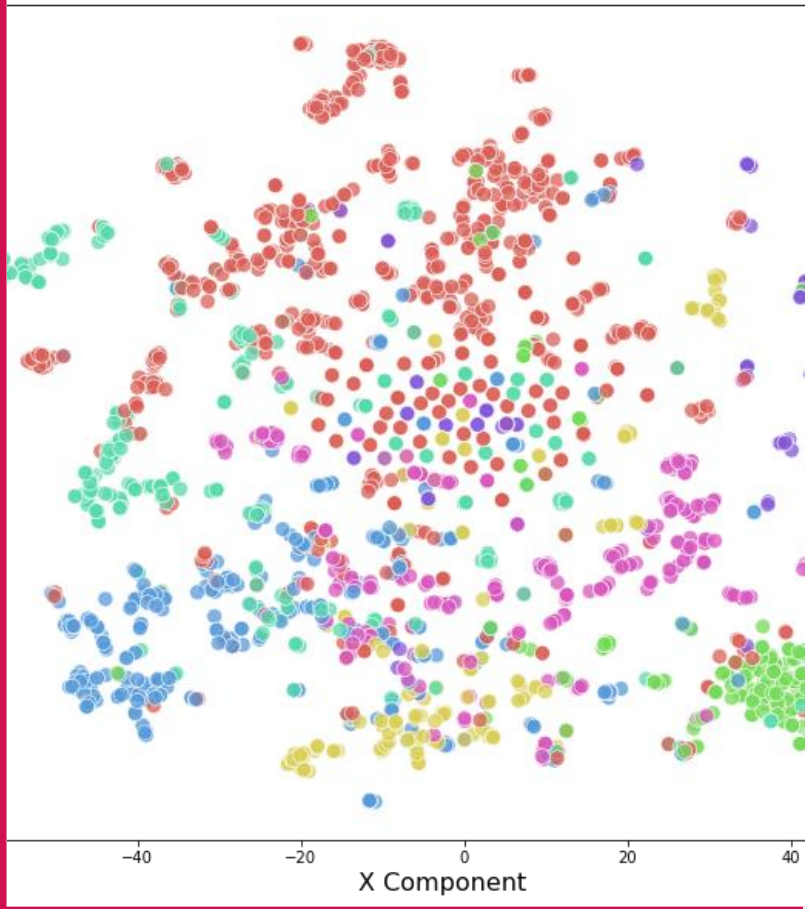
```
CALL gds.beta.graphSage.train(
        'myGraph',
        { modelName: 'GraphSAGE1',
          featureProperties: ['age', 'nationality', 'hobbies'],
          aggregator: 'mean',
          activationFunction: 'sigmoid',
          sampleSizes: [25, 10] }
) YIELD modelInfo as info RETURN info.name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

LIKE,SHARE, FOLLOW ! @poledevinci
DEVINCI.FR



FAST RANDOM PROJECTION [CIKM'19]

Random Projection Family

- Quantization: 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

FAST RANDOM PROJECTION

```
CALL gds.fastRP.stream(
        'myGraph',
        { embeddingDimension: 100,
          randomSeed: 42,
          iterationWeights: [1.0, 0.75, 0.5],
          normalizationStrengh: 0,
          relationshipWeightProperty: 'rating' }
) YIELD nodeId, embedding
```

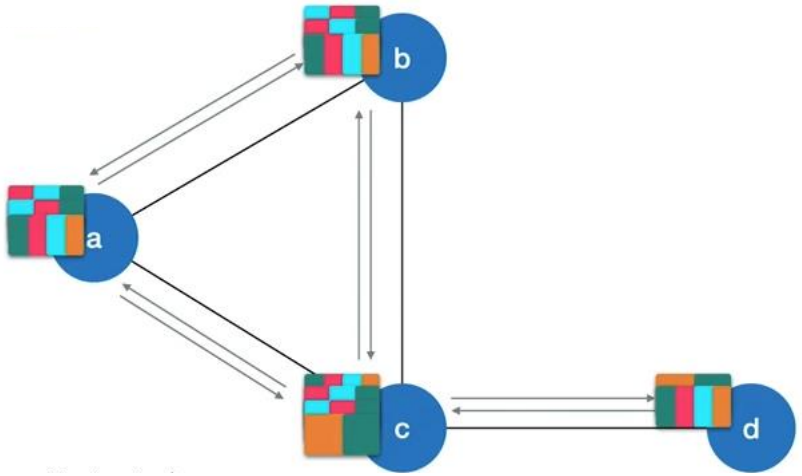
Nodes' vector dimension

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

#GNN [WWW'21]



$$\mathbf{x}_v^{(t)} = \sigma\Big(\big(\mathbf{U}_1^{(t)}\mathbf{x}_v^{(t-1)}\big) \circ \big(\mathbf{U}_2^{(t)} \sum_{u \in N(v)} \sigma(\mathbf{U}_3^{(t)}\mathbf{x}_u^{(t-1)})\big)\Big)$$

- Flow iteration using Pregel (See later on)
  - Vector passing to neighbors
  - Aggregation through graph propagation & random feature selection
    - Use ***MinHash*** for pair-wise comparison
- Neural network with activation function: σ

https://www.youtube.com/watch?v=fccFuyjNEcM

GraphRAG

Creates embeddings from node context

- Needs
  - A vector index on an *embedding* property (see later)
  - An OpenAI API Key

```
MATCH (p:Person) --> (m:Movie)
    WHERE m.title IS NOT NULL

WITH p, collect(m.title) as movies

WITH p, genai.vector.encode(movies, "OpenAI",
    {token: $openAIAPIKey}) AS vector

CALL db.create.setNodeVectorProperty(
    p,  "embedding", vector)
```

$Query: *Which actor can you recommend me with « The Phantom Menance », « a New Hope », and « Return of the Jedi »?*

```
WITH genai.vector.encode(
    $Query,
    "OpenAI",
    {token: $openAIAPIKey,}) AS question_embedding


CALL db.index.vector.queryNodes(
     "embedding ", 10, question_embedding
) YIELD node AS person, score
RETURN p.name, score
```


LIKE,SHARE, FOLLOW ! @poledevinci
DEVINCI.FR

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/

LIKE,SHARE, FOLLOW ! @poledevinci
DEVINCI.FR

NODE CLASSIFICATION

```
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
```

```
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
```

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()

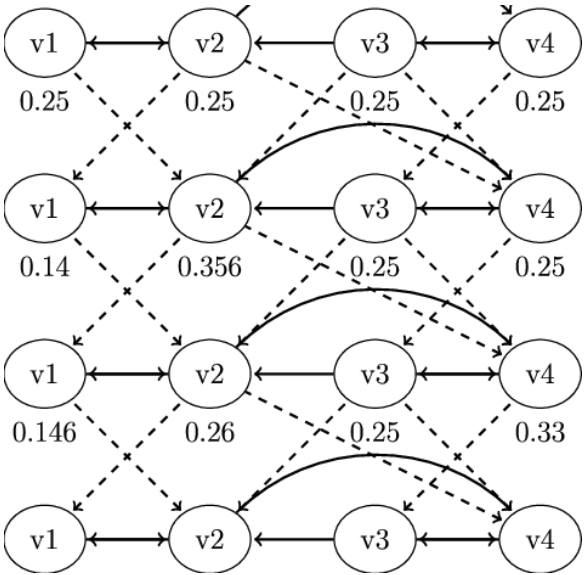
LIKE,SHARE, FOLLOW ! @poledevinci | DEVINCI.FR

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
   - Pregel
   - Neosemantics
   - Fabrics
5. GQL



LIKE,SHARE, FOLLOW ! @poledevinci | DEVINCI.FR



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

LIKE,SHARE, FOLLOW ! @poledevinci | DEVINCI.FR

```
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)

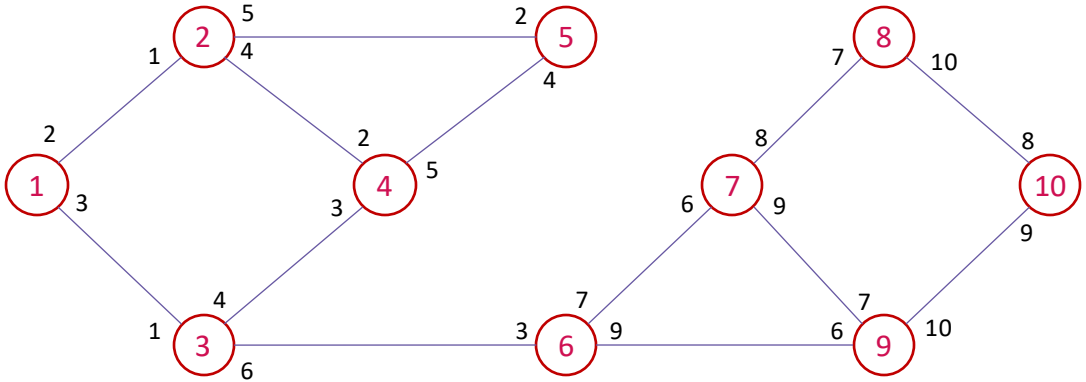
Get top labels' occurrences

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

Ask to end Supersteps iterations

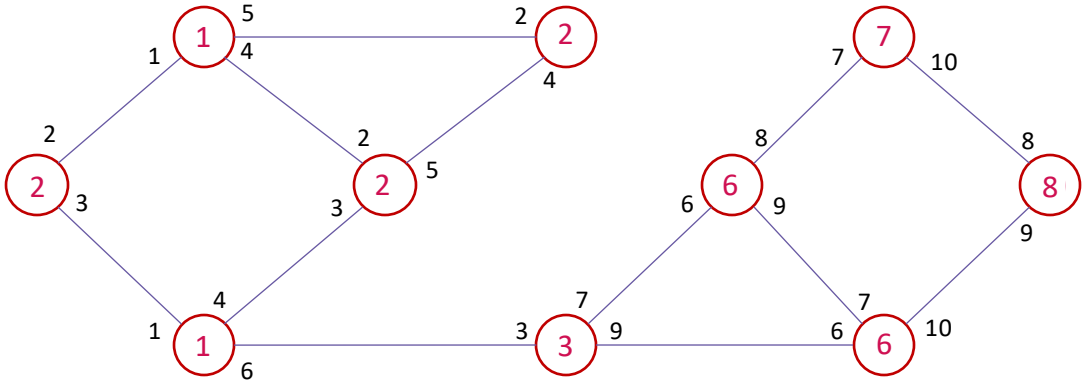
PREGEL EXAMPLE: LABEL PROPAGATION

Superstep 1



PREGEL EXAMPLE: LABEL PROPAGATION

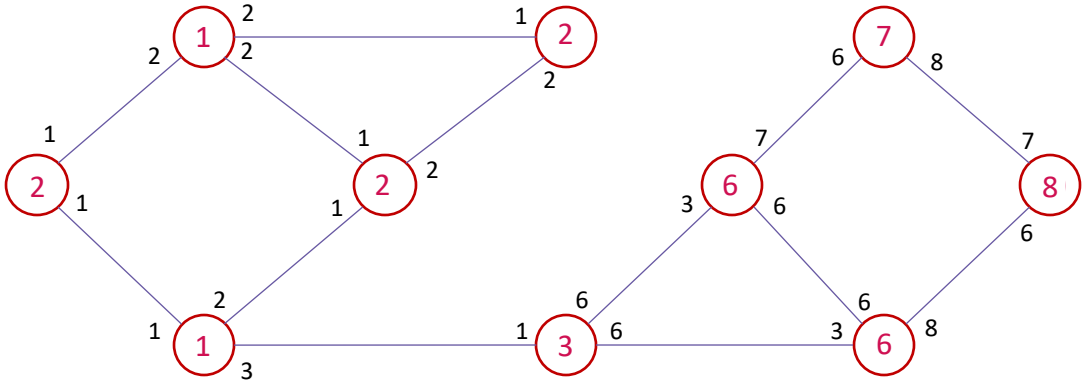
Superstep 2 – phase 1 (local top labels)



LIKE,SHARE, FOLLOW ! @poledevinci
DEVINCI.FR

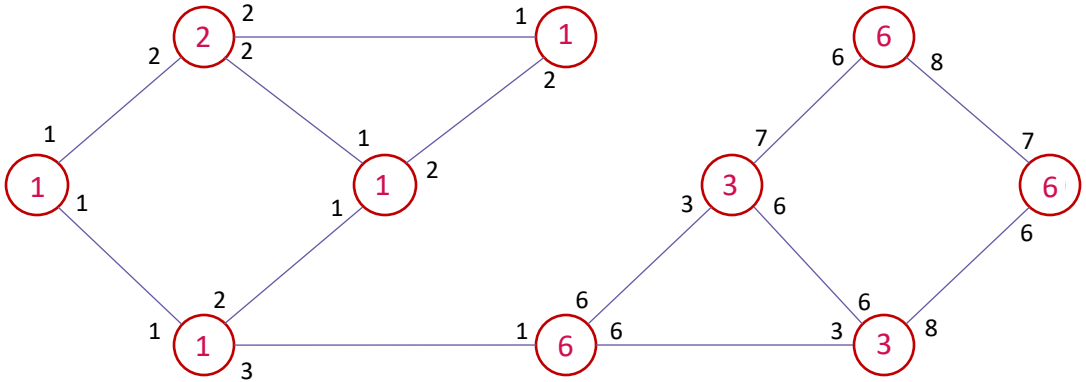
PREGEL EXAMPLE: LABEL PROPAGATION

Superstep 2 – phase 2 (send local label to neighbours)



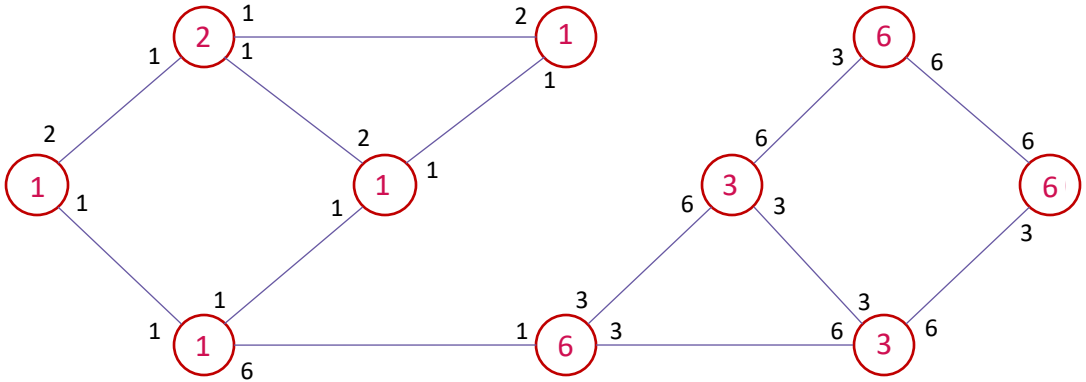
PREGEL EXAMPLE: LABEL PROPAGATION

Superstep 3 – phase 1 (local top labels)



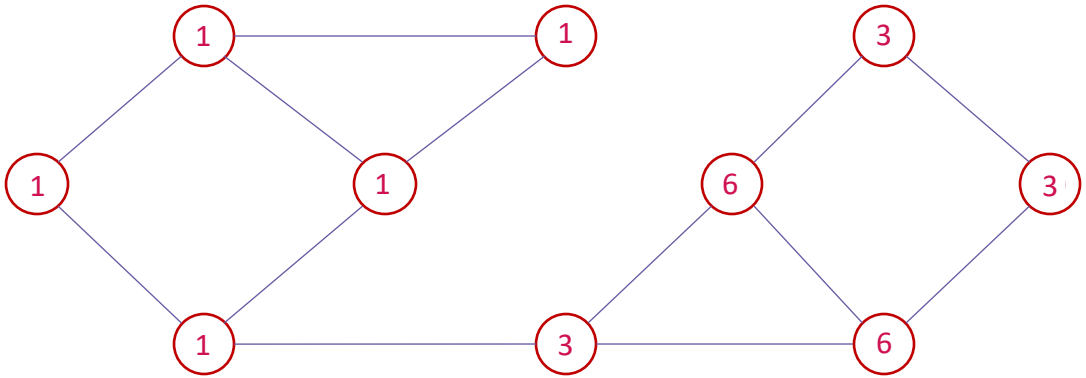
PREGEL EXAMPLE: LABEL PROPAGATION

Superstep 3 – phase 2 (send local label to neighbours)



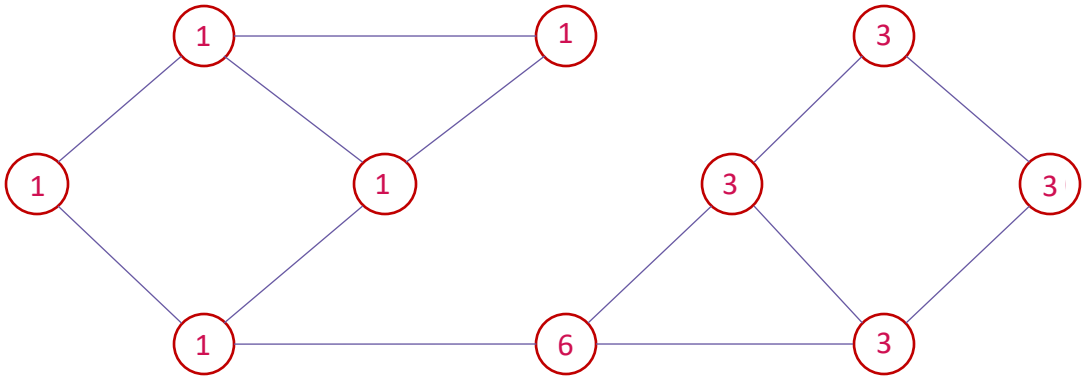
PREGEL EXAMPLE: LABEL PROPAGATION

Superstep 4



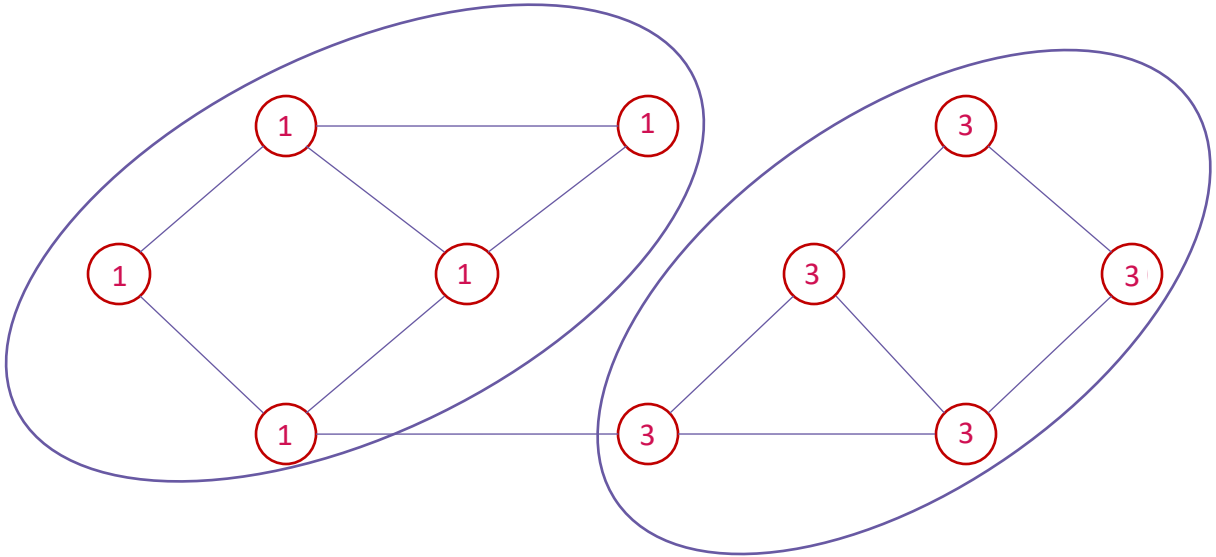
PREGEL EXAMPLE: LABEL PROPAGATION

Superstep 5



PREGEL EXAMPLE: LABEL PROPAGATION

Superstep 6



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

LIKE,SHARE, FOLLOW ! @poledevinci | DEVINCI.FR

PREGEL EXAMPLE: PAGERANK

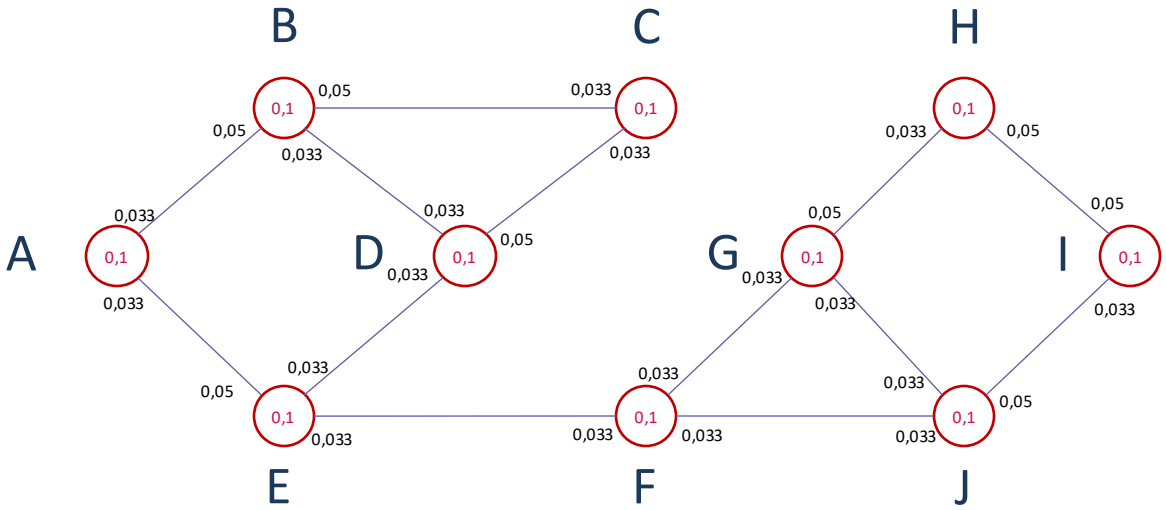
Damping Factor: df = 0.85

Init: $\frac{1}{\text{nodeCount}} = \frac{1}{10}$

PR: $\frac{1-\text{df}}{\text{nodeCount}} + \sum_i \left(\frac{\text{pr}_i}{\text{r}_i}\right) \times df$

$= \frac{0{,}15}{10} + 0{,}85 \times \sum_i \left(\frac{\text{pr}_i}{\text{r}_i}\right)$

Superstep 1



LIKE,SHARE, FOLLOW ! @poledevinci | DEVINCI.FR

PREGEL EXAMPLE: PAGERANK

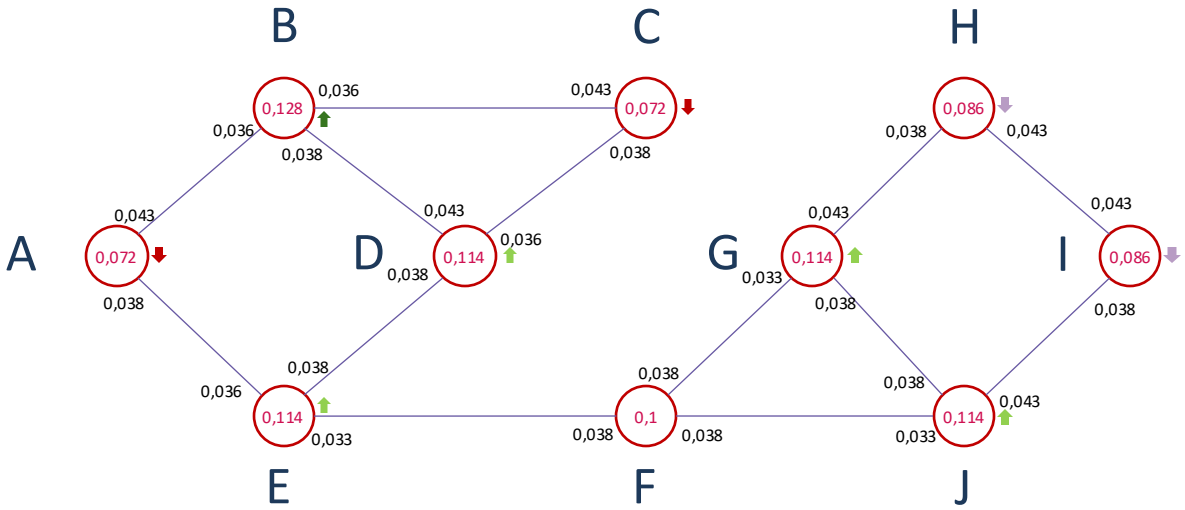
Damping Factor: df = 0.85

Init: $\frac{1}{\text{nodeCount}} = \frac{1}{10}$

PR: $\frac{1-\text{df}}{\text{nodeCount}} + \sum_i \left(\frac{\text{pr}_i}{\text{r}_i}\right) \times df$

$= \frac{0,15}{10} + 0,85 \times \sum_i \left(\frac{\text{pr}_i}{\text{r}_i}\right)$

Superstep 2



LIKE,SHARE, FOLLOW ! @poledevinci | DEVINCI.FR

PREGEL EXAMPLE: PAGERANK

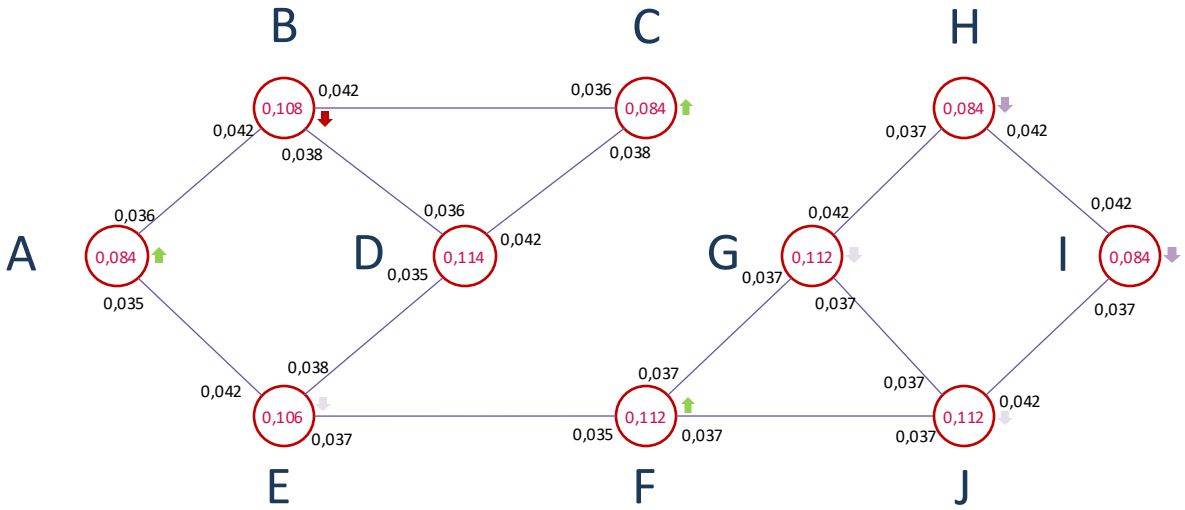
Damping Factor: df = 0.85

Init: $\frac{1}{\text{nodeCount}} = \frac{1}{10}$

PR: $\frac{1-\text{df}}{\text{nodeCount}} + \sum_i \left(\frac{\text{pr}_i}{\text{r}_i}\right) \times df$

$= \frac{0,15}{10} + 0,85 \times \sum_i \left(\frac{\text{pr}_i}{\text{r}_i}\right)$

Superstep 3



LIKE,SHARE, FOLLOW ! @poledevinci | DEVINCI.FR

PREGEL EXAMPLE: PAGERANK

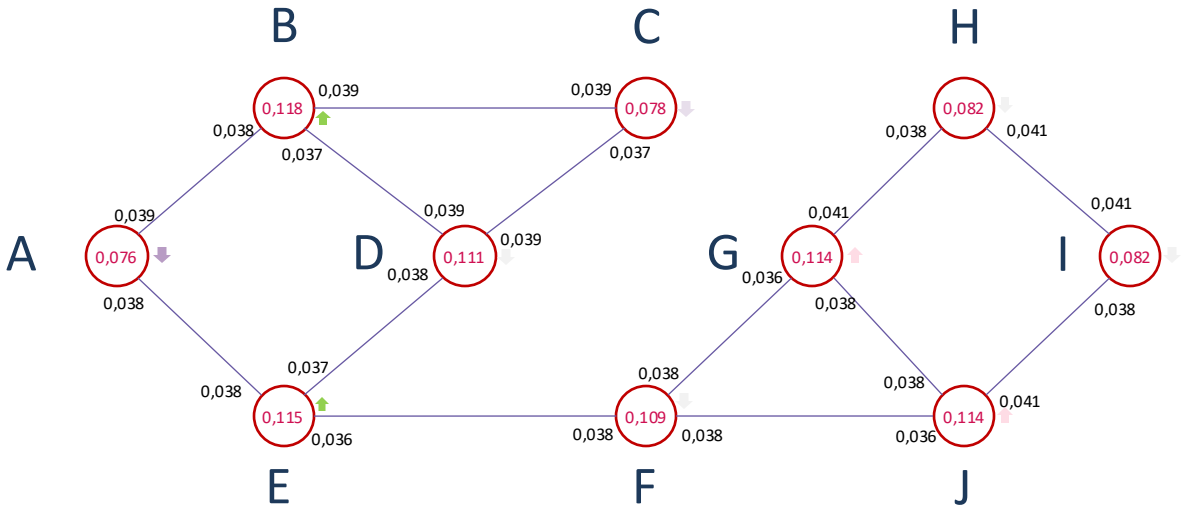
Damping Factor: df = 0.85

Init: $\frac{1}{\text{nodeCount}} = \frac{1}{10}$

PR: $\frac{1-\text{df}}{\text{nodeCount}} + \sum_i \left(\frac{\text{pr}_i}{\text{r}_i}\right) \times df$

$= \frac{0,15}{10} + 0,85 \times \sum_i \left(\frac{\text{pr}_i}{\text{r}_i}\right)$

Superstep 4



LIKE,SHARE, FOLLOW ! @poledevinci | DEVINCI.FR

PREGEL EXAMPLE: PAGERANK

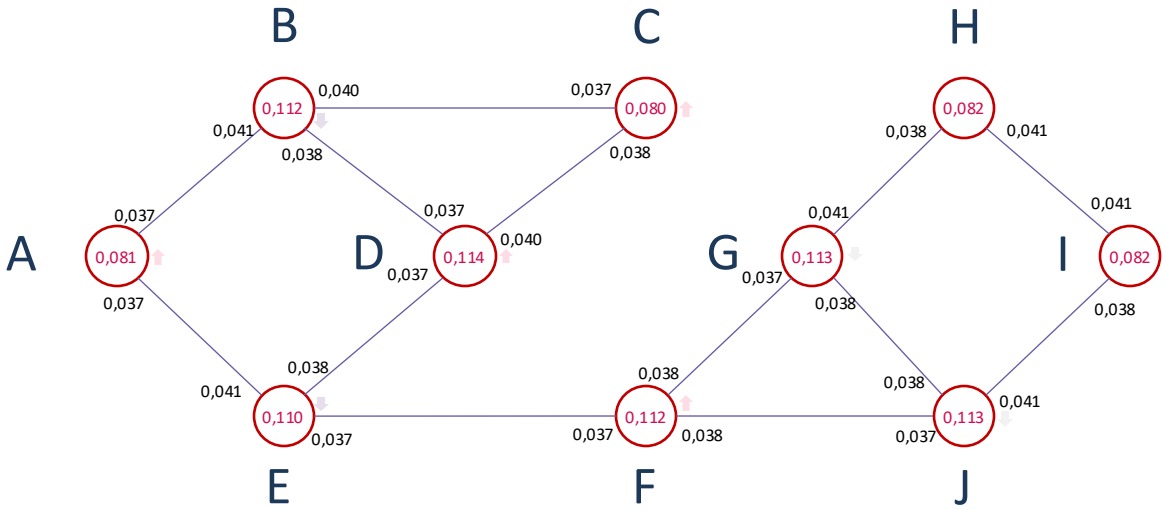
Damping Factor: df = 0.85

Init: $\frac{1}{\text{nodeCount}} = \frac{1}{10}$

PR: $\frac{1-\text{df}}{\text{nodeCount}} + \sum_i \left(\frac{\text{pr}_i}{\text{r}_i}\right) \times df$

$= \frac{0{,}15}{10} + 0{,}85 \times \sum_i \left(\frac{\text{pr}_i}{\text{r}_i}\right)$

Superstep 5



LIKE,SHARE, FOLLOW ! @poledevinci | DEVINCI.FR

PREGEL EXAMPLE: PAGERANK

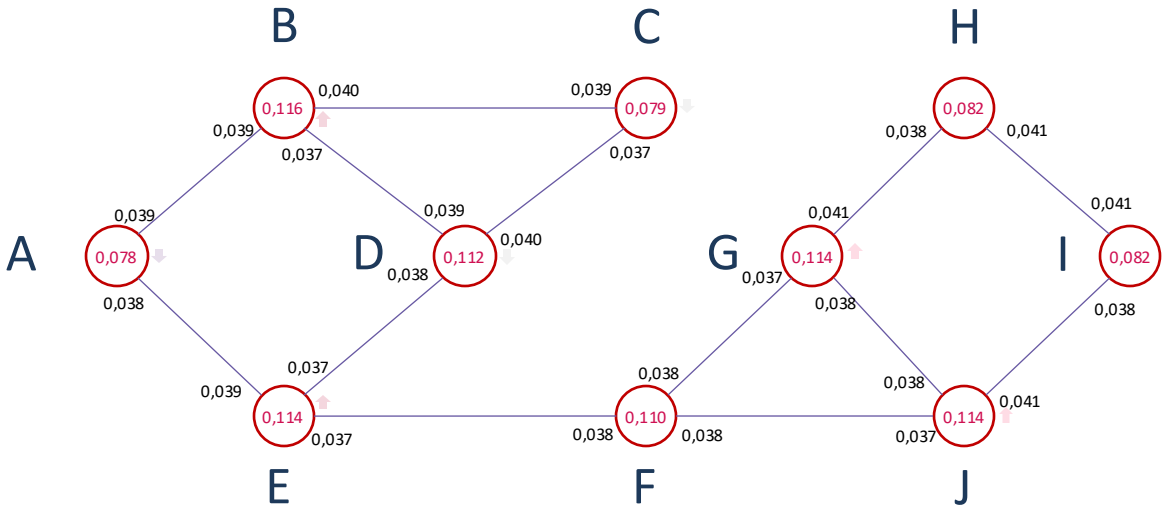
Damping Factor: df = 0.85

Init: $\frac{1}{\text{nodeCount}} = \frac{1}{10}$

PR: $\frac{1-\text{df}}{\text{nodeCount}} + \sum_i \left(\frac{\text{pr}_i}{\text{r}_i}\right) \times df$

$= \frac{0{,}15}{10} + 0{,}85 \times \sum_i \left(\frac{\text{pr}_i}{\text{r}_i}\right)$

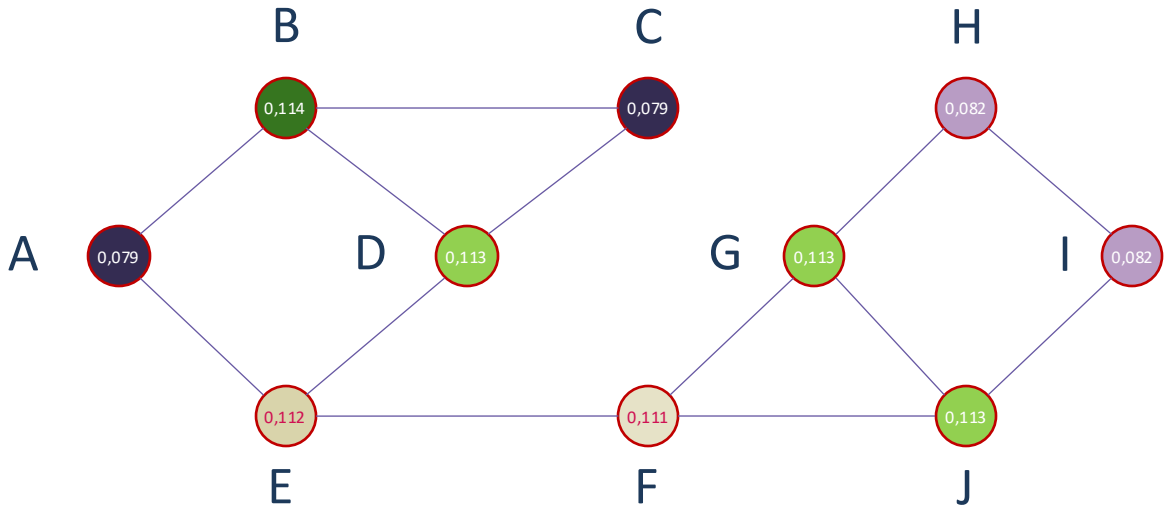
Superstep 6



LIKE,SHARE, FOLLOW ! @poledevinci | DEVINCI.FR

PREGEL EXAMPLE: PAGERANK

*… Superstep **32***



Damping Factor: df = 0.85

Init: $\frac{1}{\text{nodeCount}} = \frac{1}{10}$

PR: $\frac{1-\text{df}}{\text{nodeCount}} + \sum_i \left(\frac{\text{pr}_i}{\text{r}_i}\right) \times df$

$= \frac{0{,}15}{10} + 0{,}85 \times \sum_i \left(\frac{\text{pr}_i}{\text{r}_i}\right)$

LIKE,SHARE, FOLLOW ! @poledevinci | DEVINCI.FR

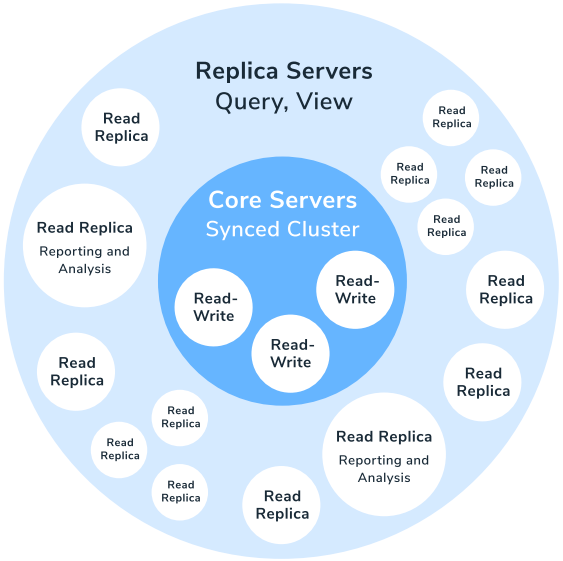


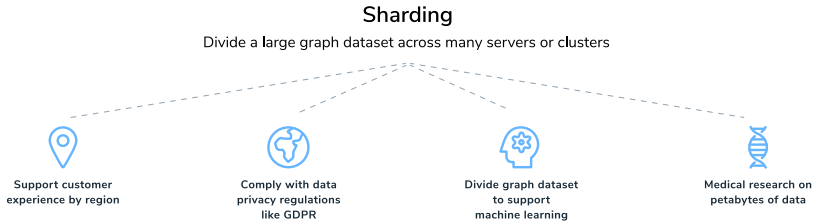
NEOSEMANTICS

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/



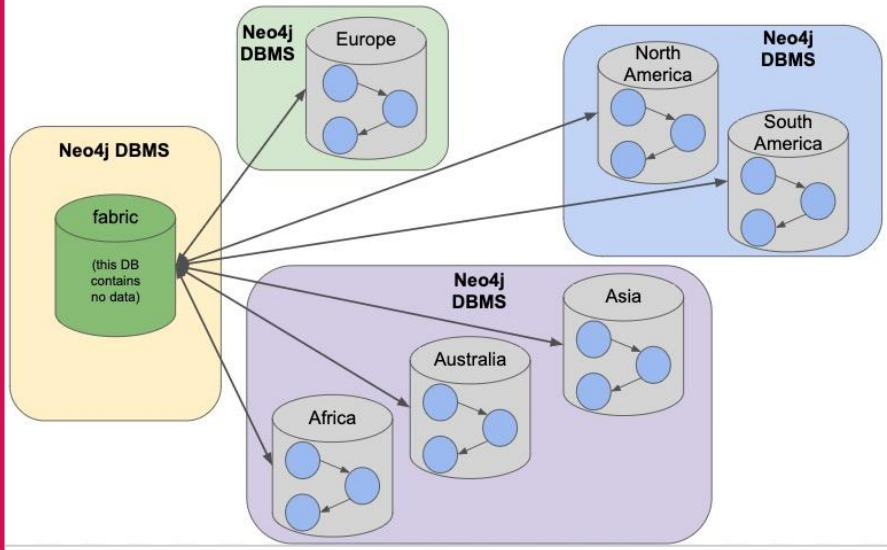


DISTRIBUTION & REPLICATION

- **Causal Clustering**
  - Causal cluster Architecture
    - Core servers
      - Causal Consistency
    - Replicates
- **Multi-clustering**
  - Multitenant database
    - Multiple dbs
    - Shared discovery service
  - Consensus consistency
    - Locally: causal clustering

https://neo4j.com/docs/operations-manual/current/clustering-advanced/multi-clustering/introduction/

DISTRIBUTION & REPLICATION: FABRIC



- Sharding graphs
- Behaves like a proxy server
- "Manual" cut of the graph
  - Few or no relationships between subgraphs

https://neo4j.com/docs/operations-manual/current/fabric/

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
   - Pregel
   - Neosemantics
   - Fabrics
5. GQL



LIKE,SHARE, FOLLOW ! @poledevinci | DEVINCI.FR

GRAPH QUERY LANGUAGE

GQL is a standardized query language for **property graphs** first described in **ISO/IEC 39075**, released in April 2024.

RDF is the first standard for Graph Data models (2014) but for semantic graphs. Needs for Label Property Graphs (LPG).

Inspired from *Cypher, G-CORE, PGQL (Oracle), SQL/PGQ, GSQL, NGSI-LD*.

Very similar to Neo4j's Cypher query language



GQL – MAIN DIFFERENCES

- Formal Graph Data Model specification
  - Nodes/Relationships/Labels, data types
  - Closed vs Open Graph types: "CREATE CLOSED GRAPH" (new unspecified types are not allowed)
  - Allow catalogs of types
  - Allow GQL-conformance for system features
- Regular Path Queries (RPQs)
- ***WITH*** => ***LET + FILTER***
- Complete semantic for paths in return clause

GQL – GRAPH DATA MODEL

```
-- Define the schema (graph type)
CREATE GRAPH TYPE MoviesGraph AS {
 (:Movie => {
  IDM   :: UINT NOT NULL,
  title   :: STRING NOT NULL,
  year  :: UINT
 }),

 (:Person => {
  IDP         :: UINT NOT NULL,
  name      :: STRING NOT NULL,
  birthdate :: ZONED DATETIME
 }),

 (:Person)-[:acted_in {
          roles :: LIST<STRING> NOT NULL }]->(:Movie),

 (:Person)-[:directed]->(:Movie),

  -- Key constraints (unique identifiers)
  CONSTRAINT movie_pk  FOR (m:Movie)
   REQUIRE m.IDM IS KEY,
  CONSTRAINT person_pk FOR (p:Person)
   REQUIRE p.IDP IS KEY
};

-- Instantiate a graph from this type
CREATE GRAPH MyMovies OF TYPE MoviesGraph;
```

REGULAR PATH QUERIES (RPQ) [ICDT'25]

- Regular expression for paths
  Example: "all nodes x and y such that there exists a path from x to y whose sequence of edge labels is in (parent)+"

RPQ	Cypher	GQL
(A)+	MATCH (x) -[:A*]-> (y)	MATCH (x) -/ (A*) /-> (y)
(A | B)* C	MATCH (x) -[:A|:B*]-> (z) -[:C]-> (y)	MATCH (x) -/ ((A | B)* C) /-> (y)
(A B)* C	…	MATCH (x) -/ ((A B)* C) /-> (y)
(A*B*)	MATCH (x)-[:A*0..]->()-[:B*0..]->(y)	MATCH (x) -/ ((A*) (B*)) /-> (y)
(A*B*)*	MATCH (x)-[:A*0..]->()-[:B*0..]->(y) …	MATCH (x) -/ ((A*) (B*))* /-> (y)
(A | ^B) * (reverse direction)	…	MATCH (x) -/ (A | ^B) * /-> (y)

RPQ

Aspect	Cypher	GQL (ISO/IEC 39075)
Expression of regular path patterns	**Limited support**: Cypher allows simple quantifiers like *, +, ?	Full support
Path modes (simple, trail, acyclic, etc.)	Cypher **does not always allow explicit control** over path semantics. Whether paths can contain repeated nodes or edges depends on the specific implementation (e.g., Neo4j).	GQL **standardizes multiple "path modes"**, enabling users to specify whether the query should find simple paths, trails (no edge repetition), or acyclic paths.
Query results (complete paths vs. endpoints)	Cypher can **return a path** (e.g., p = (a)-[*]->(b)), but the exact behavior varies by engine. Some systems only return node endpoints.	GQL **explicitly supports returning complete paths**, and allows fine-grained control over the level of detail and path type returned.
Compatibility and portability across implementations	Cypher is widely used but **not standardized**. RPQ behavior may differ between databases (Neo4j, Memgraph, RedisGraph, etc.).	GQL is an **international standard**, ensuring consistent and interoperable RPQ behavior across conforming implementations.
Complexity and expressiveness	Cypher is **less expressive** for highly complex path patterns (e.g., nested regular expressions or constrained repetitions).	GQL aims to **fully capture the theoretical expressiveness of RPQs**, including arbitrary regular expressions and structural constraints on paths.
Practical use and maturity	Cypher is **very mature**, efficient, and proven in production systems (Neo4j, Memgraph, etc.). It covers most real-world use cases.	GQL formal specification is complete Adoption and implementation are still underway.