

Data 603 – Big Data Platforms



UMBC

Lecture 11
Graph Analytics

Graph Analytics

Graphs

- A representation of the relationships between pairs objects.
 - A graph models “things” and relationships between “things.”
 - A representation of a set of objects (e.g. people) and the pairwise relationships between them (e.g. friendships)
- Data structures composed of nodes (vertices) and edges that define relationships between the nodes.
- Vertices are arbitrary objects (people, objects, places, concepts, etc.).
- Graph analytics is the process of analyzing the relationships between vertices.
 - A node represents a person, place, or thing
 - Edge represents a relationship

Graph Analytics Intro

Graph Terminology

Vertex (or Node)

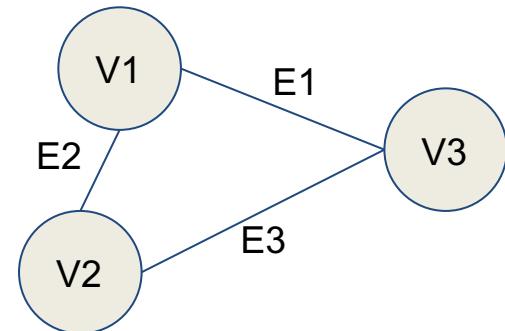
- The object being represented.
- Represented by V .

Edges

- Representation of pairwise relationships between nodes.
- Represented by E .

$$G = (V, E)$$

- Graph G with vertices V and edges E .



Note: Edges and vertices in graphs can have data associated with them.

Graph Terminology

Directed vs. Undirected Graphs

- Directed Graph
 - Each edge is an ordered pair, with the edge traveling from the first vertex to the second.
 - The relationship is from a source (or head) vertex to a destination (or tail) vertex.
 - Edges do not have specified “start” and “end” vertices.
 - The two ends of the edge play different roles (e.g. parent-child relationship)
 - Examples:
 - A link from on the web from page A to page B
 - A path on the route from Baltimore to Disney World

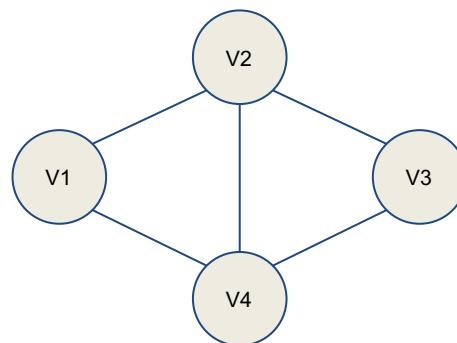
Graph Terminology

Directed vs. Undirected Graphs

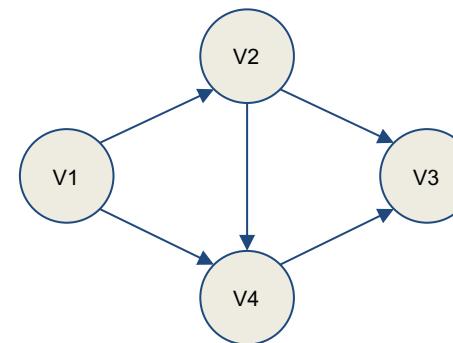
- Undirected graph, each edge corresponds to an unordered pair of vertices.
- Edges have no arrow; the relationship is symmetrical
 - A is a friend of B
 - B is a friend of A

Graph Analytics

Directed vs. Undirected Graphs



Undirected Graph

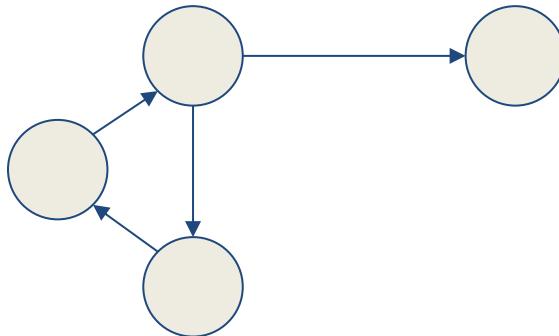


Directed Graph

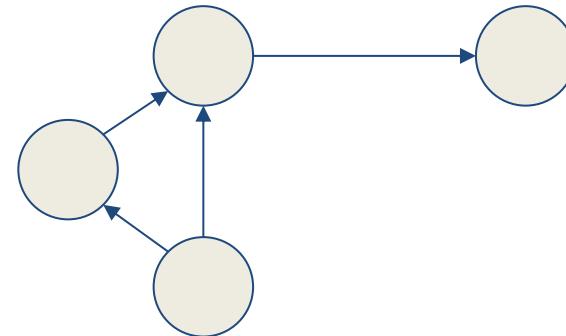
Graph Terminology

Cyclic vs. Acyclic Graphs

- A cyclic graph contains cycles
 - A series of vertices connected in a loop.
 - Poses the risk that traversing of such graph can follow edges in an infinite-loop.
- An acyclic graph contains no cycles.



Cyclic Graph

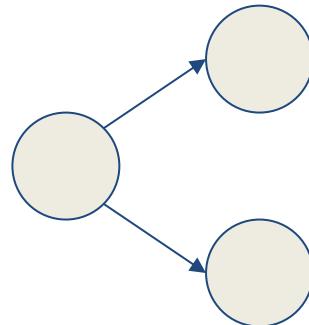


Acyclic Graph

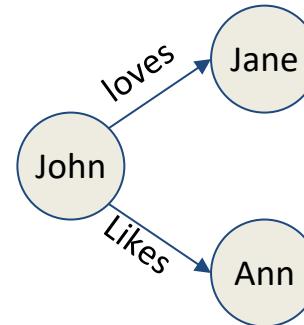
Graph Terminology

Unlabeled vs. Labeled Graphs

- In a labeled graph, vertices and/or edges have data (labels) associated with them other than their unique identifier
 - Vertex-labeled graphs: graphs with labeled vertices
 - Edge-labeled graphs: graphs with labeled edges
- A completely unlabeled graph is not useful.
 - Normally at least the vertices are labeled.



Unlabeled Graph

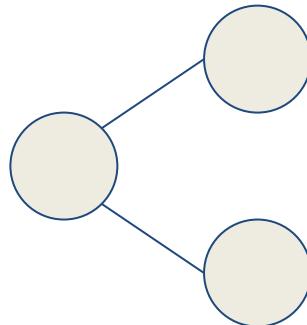


Labeled Graph

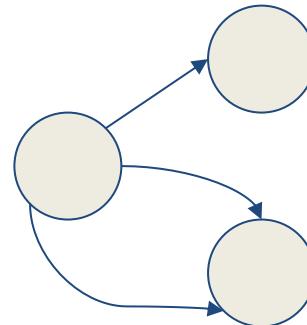
Graph Terminology

Parallel Edges and Loops

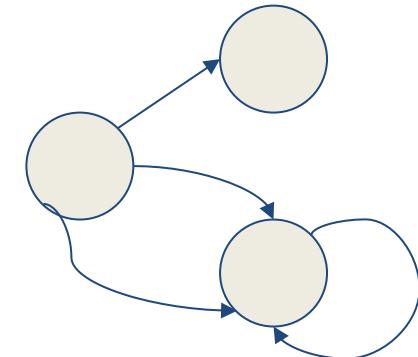
- Multigraph: Allowing multiple edges between the same pair of vertices
- Pseudograph: An edge that starts and ends with the same vertex.



Simple Graph



Multigraph

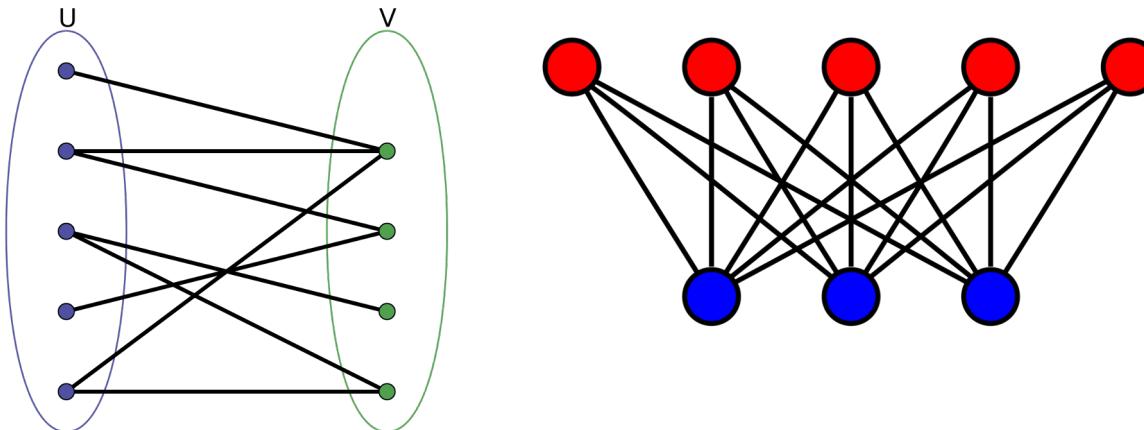


Pseudograph

Graph Terminology

Bipartite Graph

- A bipartite graph (or bigraph) is a graph whose vertices can be divided into two disjoint and independent sets U and V such that every edge connects a vertex in U to one V .
- Vertex sets U and V are usually called the parts of the graph.
- A bipartite graph is a graph that does not contain any odd-length cycles.



Source: https://en.wikipedia.org/wiki/Bipartite_graph

Graph Query Language

SPARQL

- A SQL-like language promoted by W3C for querying Resource Description Framework (RDF) graphs

Cypher

- A query language used in Neo4j, an open source database

Tinkerpop Gremlin

- An attempt to create a standard interface to graph databases and processing systems.

GraphX + GraphFrame

Refer to [https://en.wikipedia.org/wiki/GQL_\(Graph_Query_Language\)](https://en.wikipedia.org/wiki/GQL_(Graph_Query_Language))

Other Graph Processing Tools

- JanusGraph (Distributed, open source, graph database)
- Neo4j (open source, graph database)
- ArangoDB (Distributed, open source, multi-model database)
- Dgraph (Distributed, open source, graph database)
- OrientDB (Distributed, open source, multi-model database)
- TigerGraph (Proprietary graph database)
- NetworkX (Python package for complex network analytics)
- Azure CosmosDB (Cloud-native, multi-model database)
- AWS Neptune (Cloud-native, graph database)

Why Graph Processing on Spark?

- Relational databases are not suited for graph analytics.
 - SQL is not built to present graph notions such as following a trail of connections.
- Traditional methods of data processing fail to scale as the size of the data to be analyzed increase.
- Graph processing systems (including Apache Spark GraphFrames) provide the means to create graph structures from raw data sources.
- GraphX (RDD) and GraphFrames (DataFrame) are graph processing layers on top of Spark that bring the power of distributed and scalable Big Data processing to graphs.

Q:Why Graph?

Can you think of any use-cases?

Where is Graph used?

- Fraud Detection & Analysis
- Network and Database Infrastructure Monitoring for IT Operations
- Recommended Engine & Product Recommendation Systems
- Master Data Management
- Identity and Access Management
- Knowledge Graph
- Google Map

Refer to <https://neo4j.com/use-cases/>

Where is Graph used?

- Motif finding
- Determining importance of papers in bibliographic networks (which papers are most referenced)
- Ranking web pages (Google's PageRank algorithm)
- Other important things such as [Six Degrees of Kevin Bacon](#) (Social Media and Social Network Graphs)

Refer to <https://neo4j.com/use-cases/>

Spark Graph Analytics

Spark Graph Analytics

- GraphX
 - RDD-based library
 - Provides low-level interface
 - Powerful, but not easy to use or optimize
- GraphFrames
 - Extends GraphX to provide a DataFrame API and support for Spark's different language bindings so that users of Python can take advantage of the scalability of the tool.
 - Available as an external package.
 - Improved user experience with small penalty in performance due to some overhead .

Building a Graph

Users can create GraphFrames from vertex and edge DataFrames.

- Define vertices and edges using DataFrames with special named columns.
- Both DataFrames can have arbitrary other columns which can represent vertex and edge attributes.
- A GraphFrame can also be constructed from a single DataFrame containing edge information.
 - The vertices will be inferred from the sources and destinations of the edges.

Building a Graph

- Vertex DataFrame
 - Should contain a special column named “id” which specifies unique IDs for each vertex in the graph.
- Edge DataFrame
 - Should contain two special columns:
 - “src” (source vertex ID of edge)
 - “dst” (destination vertex ID of edge).

Building a Graph

```
# Vertex DataFrame
v = sqlContext.createDataFrame([
    ("a", "Alice", 34),
    ("b", "Bob", 36),
    ("c", "Charlie", 30),
    ("d", "David", 29),
    ("e", "Esther", 32),
    ("f", "Fanny", 36),
    ("g", "Gabby", 60)
], ["id", "name", "age"])

# Edge DataFrame
e = sqlContext.createDataFrame([
    ("a", "b", "friend"),
    ("b", "c", "follow"),
    ("c", "b", "follow"),
    ("f", "c", "follow"),
    ("e", "f", "follow"),
    ("e", "d", "friend"),
    ("d", "a", "friend"),
    ("a", "e", "friend")
], ["src", "dst", "relationship"])
# Create a GraphFrame
g = GraphFrame(v, e)
```

Building a Graph

- Define vertices and edges using DataFrames with special named columns.
- To define a graph, use the naming conventions for columns presented in the GraphFrames library.
 - In the vertices table, define the identifier as *id*
 - In the edges tables, label each edge's source vertex ID as *src* and destination ID as *dst* (for directed graph)

```
stationVertices =  
bikeStations.withColumnRenamed("name", "id").distinct()  
tripEdges = tripData \  
    .withColumnRenamed("Start Station", "src") \  
    .withColumnRenamed("End Station", "dst")  
from graphFrames import GraphFrame  
stationGraph = GraphFrame(stationVertices, tripEdges)  
stationGraph.cache()
```

Querying the Graph

- GraphFrames provide simple access to both vertices and edges as DataFrames.

```
From pyspark.sql.functions import desc
stationGraph.edges.groupBy("src", "dst"). \
    count().orderBy(desc("count")).show(10)
```

- Filter by any valid DataFrame expression

```
stationGraph.edges
    .where("src='Townsend at 7th' OR dst = 'Townsend at
7th'")
    .groupBy("src", "dst").count()
    .orderBy(desc("count"))
    .show(10)
```

Subgraphs

- Subgraphs are smaller graphs within the larger one

```
townAnd7thEdges = stationGraph.edges \
    .where("src = 'Townsend at 7th' OR dst = 'Townsend at
7th' ")
subgraph = GraphFrame(stationGraph.vertices, townAnd7thEdges)
```

Motif Finding

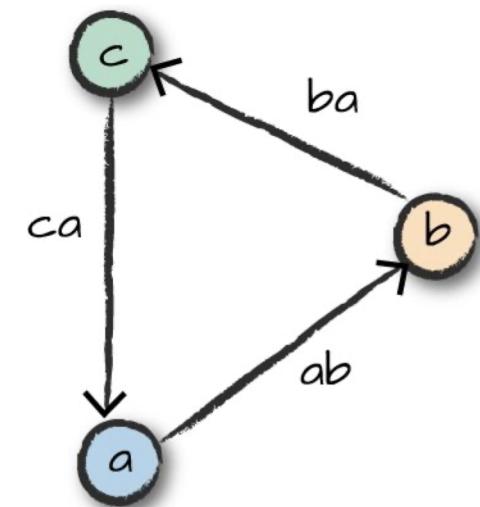
- *Motifs* are a way of expressing structural patterns in a graph.
- When a motif is specified, patterns are queried instead of actual data.
- In GraphFrames, a query is specified in a domain-specific language similar to Neo4J's Cypher language.
 - Allows specifying of combinations of vertices and edges and assign them names.
 - Specify that a given vertex connects to another vertex b through an edge ab: (a)-[ab]->(b)
 - The names inside parentheses or brackets do not signify values but instead what the columns for matching vertices and edges should be named in the resulting DataFrame.
 - The names can be omitted if the resulting values don't need to be queried. (e.g. (a)-[]->())

Motif Finding

- *Find all the rides that form a "triangle" pattern between three stations.*

```
motifs = stationGraph.find("(a)-[ab]->(b);(b)-[bc]->(c);(c)-[ca]->(a)")
```

- (a) signifies the starting station
- [ab] represents an edge from (a) to the next station (b)
- This is repeated for stations (b) to (c), and then from (c) to (a)
- The resulting DataFrame from the query contains nested fields for vertices a, b, and c, as well as respective edges.
- The resulting query can be queried just like any other DataFrames.



Graph Algorithms

- A graph is a logical representation of data
- Graph theory provides numerous algorithms for analyzing graph data.
- GraphFrames provides a number of algorithms out of the box.
 - http://graphframes.github.io/graphframes/docs/_site/user-guide.html#graph-algorithms

Types of Graph Algorithms

Pathfinding

- Finding shortest path from vertex A to vertex B.
- e.g. Google Map.

Centrality

- Understanding which nodes are important within the network.
- e.g. Who is the central figure in the class? Who is most popular important person? Web search ranking.

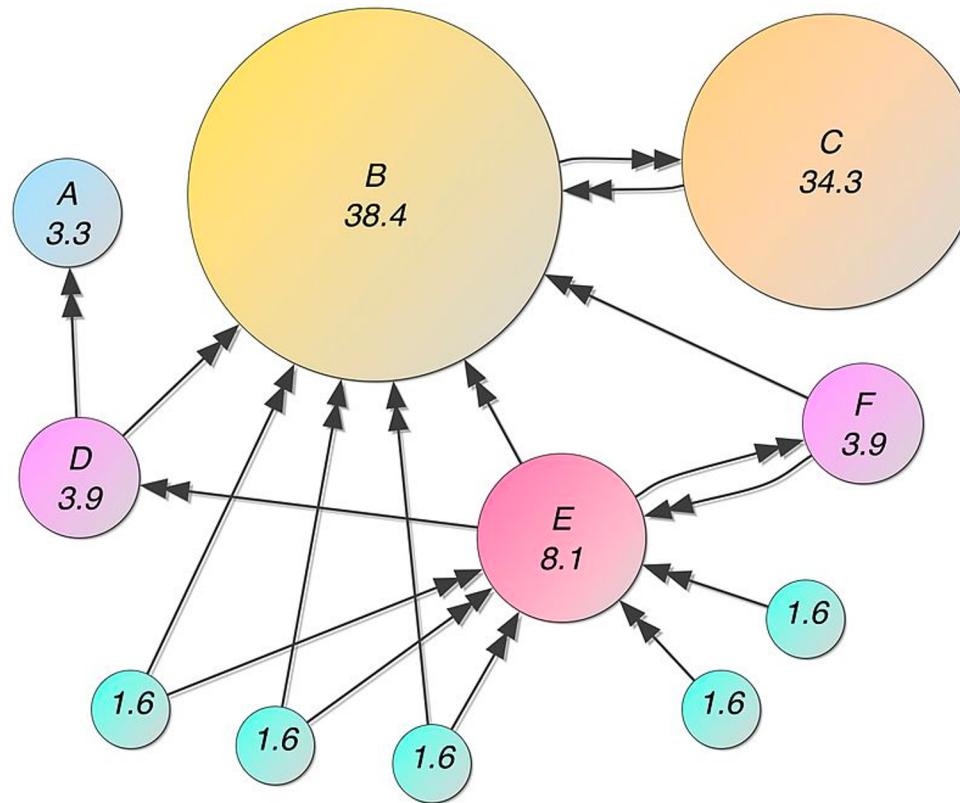
Community Detection

- Used to find communities (sub-structures/graphs) and quantify the quality of groupings.

Graph Algorithms - PageRank

- Created by Larry Page (cofounder of Google) as a research project for how to rank web pages.
- A way to measure the “authority” or centrality of vertices in a graph
- It measures the transitive (directional) influence of nodes.
- PageRank considers the influence of a node’s neighbors, and their neighbors.
- It is not based on the number of vertices that have edges pointing to the vertex in question, but on the PageRanks of those vertices.
- Measures the number and quality of incoming relationships to a vertex to determine an estimation of how important the vertex is.

Graph Algorithms - PageRank



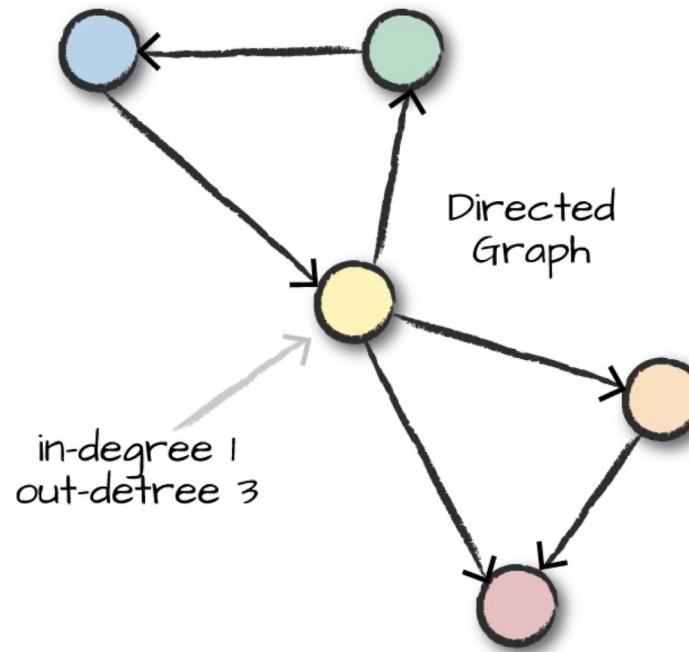
<https://en.wikipedia.org/wiki/PageRank>

Graph Algorithms - PageRank

- The results of the algorithm are stored as one or more columns in the GraphFrame's vertices and/or edges or the DataFrame
- For PageRank, the algorithm returns a GraphFrame. Estimated PageRank values for each vertex can be extracted from the new *pagerank* column.

```
from pyspark.sql.functions import desc
ranks = stationGraph.pageRank(resetProbability=0.15, maxIter=10)
ranks.vertices.orderBy(desc("pagerank")).select("id", "pagerank")
.show(10)
```

In-Degree and Out-Degree Metrics



- One of the common tasks is to count the number of edges that are inbound and outbound in relation to a vertex.
- Applicable in the context of social networking.

In-Degree and Out-Degree Metrics

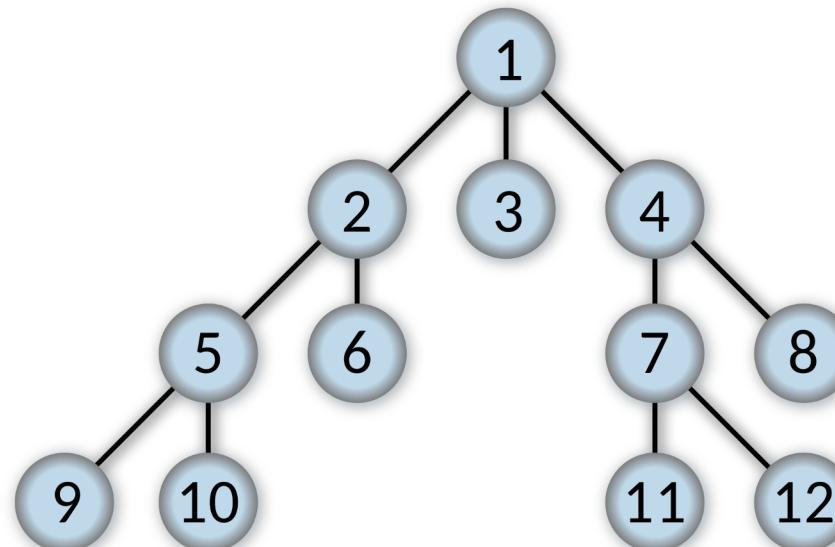
- The results of the algorithm are stored as one or more columns in the GraphFrame's vertices and/or edges or the DataFrame
- High ratio: where large number of trips end. Lower ratio: where the trips start

```
outDeg = stationGraph.outDegrees
outDeg.orderBy(desc("outdegree")).show(5, False)

degreeRatio = inDeg.join(outDeg, "id") \
    .selectExpr("id", "double(indegree)/double(outDegree) as degreeRatio")
degreeRatio.orderBy(desc("degreeRatio")).show(10, False)
degreeRatio.orderBy("degreeRatio").show(10, False)
```

Breadth-First Search

- Breadth-first search searches a graph for how to connect two sets of nodes based on the edges in the graph. E.g. Find shortest paths to different bike stations.
- Explores the vertices of a graph in layers in order of increasing distance from the starting vertex.



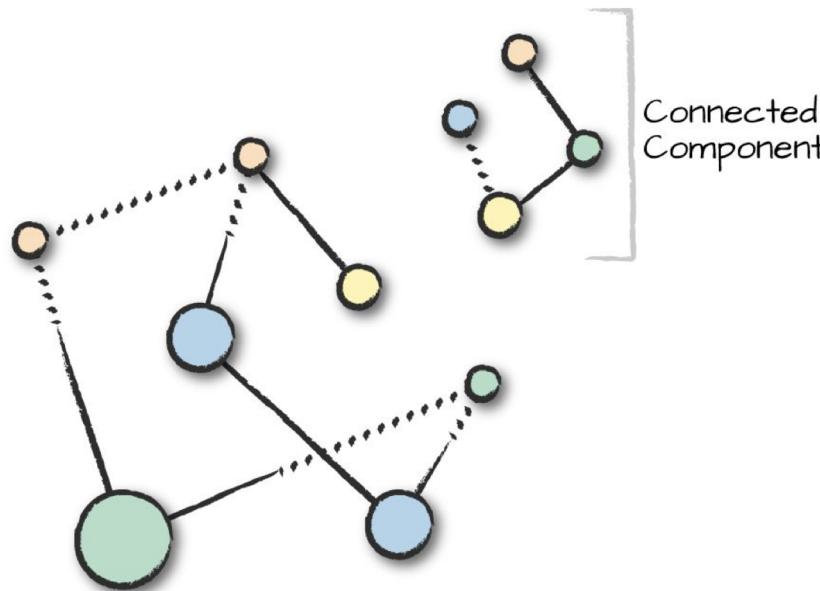
Breadth-First Search

```
stationGraph.bfs(fromExpr="id='Townsend at 7th'",  
                  toExpr = "id='Spear at Folsom'"), maxPathLength = 2).  
show(10)
```

- `maxLengthPath`: maximum edges to follow
- Possible to specify an `edgeFilter` to filter out edges that do not meet a requirement

Connected Components

- A connected component defines an (undirected) subgraph that has connections to itself but does not connect to the greater graph.
- Connected component assumes an undirected graph.
- One of the **most expensive algorithms** in GraphFrames.



Connected Components

```
spark.sparkContext.setCheckpointDir("/tmp/checkpoints")
minGraph = GraphFrame(stationVertices,
                      tripEdges.sample(False, 0.1))
Cc = minGraph.connectedComponents()
cc.where("components != 0").show()
```

Strongly Connected Components

- Strongly connected components takes directionality into account.
- Strongly connected component is a subgraph that has paths between all pairs of vertices inside it.

```
scc = minGraph.stronglyConnectedComponents(maxIter = 3)
scc.groupBy("component").count().show()
```

Saving and Loading

Saving and Loading

- Since GraphFrames are built around DataFrames, they automatically support saving and loading to and from the same set of datasources.

```
from graphframes.examples import Graphs
g = Graphs(sqlContext).friends() # Get example graph

# Save vertices and edges as Parquet to some location.
g.vertices.write.parquet("hdfs://myLocation/vertices")
g.edges.write.parquet("hdfs://myLocation/edges")

# Load the vertices and edges back.
sameV = sqlContext.read.parquet("hdfs://myLocation/vertices")
sameE = sqlContext.read.parquet("hdfs://myLocation/edges")

# Create an identical GraphFrame.
sameG = GraphFrame(sameV, sameE)
```

Useful Links

- GraphFrames user guide:
 - https://graphframes.github.io/graphframes/docs/_site/user-guide.html
 - <https://docs.databricks.com/spark/latest/graph-analysis/graphframes/user-guide-python.html>
- Graph Algorithm e-Book (free):
 - <https://neo4j.com/lp/book-graph-algorithms/>
 - <https://resources.oreilly.com/examples/0636920233145>
 - https://github.com/neo4j-graph-analytics/book/blob/master/data/188591317_T_ONTIME.csv.gz
- PageRank
 - PageRank Beyond the Web: <https://arxiv.org/abs/1407.5107>
 - https://graphframes.github.io/graphframes/docs/_site/user-guide.html#pagerank
 - PageRank Convergence: <https://stackoverflow.com/a/29321153>

Installing GraphFrames

The screenshot shows a user interface for managing clusters, likely in a cloud environment. On the left is a vertical sidebar with icons for Home, Workspace, Recents, Data, Clusters (which is selected), Jobs, and Search. The main area is titled 'Clusters' and shows two tabs: 'All-Purpose Clusters' and 'Job Clusters'. Below is a table with columns: Name, State, Nodes, Runtime, Driver, Worker, Creator, and Actions. One row is visible, labeled 'My Cluster' with status 'Running', 1 node, runtime 8.1, and creator 'donghwa....'. A large callout box with the text 'Click on the cluster' points to the 'My Cluster' row. The top right of the interface has filters for 'All', 'Created by me', 'Accessible by me', and a search bar.

Name	State	Nodes	Runtime	Driver	Worker	Creator	Actions
My Cluster	Running	1 (0 spot)	8.1 (includes Apache Spark 3.1.1, S...)	Commu...	Commu...	donghwa....	2

Click on the cluster

Installing GraphFrames

The screenshot shows the Databricks Cluster configuration interface. On the left is a dark sidebar with icons for Home, Workspace, Recents, Data, Clusters (which is selected and highlighted in red), and Jobs. The main area is titled "Clusters / My Cluster" and shows "My Cluster" status with a green circle icon. Below the cluster status are several tabs: Configuration (selected), Notebooks (2), Libraries (highlighted with a blue border), Event Log, Spark UI, Driver Logs, Metrics, Apps, and Spark Cluster UI - Master. Under the Configuration tab, there are sections for Databricks Runtime Version (8.1 (includes Apache Spark 3.1.1, Scala 2.13)), Driver Type (Community Optimized), and Instance (Free 15GB). A callout box with a blue border and white background points to the "Libraries" tab with the text "Click on ‘Libraries’". Below the instance section, a note states: "Cluster will automatically terminate after an idle period of two hours. This is a feature of your Databricks subscription." At the bottom, there is an Availability Zone dropdown set to "us-west-2c".

Installing GraphFrames

The screenshot shows the Clusters / My Cluster interface. On the left is a sidebar with icons for Home, Workspace, Recents, Data, Clusters (which is selected and highlighted in red), Jobs, and Search. The main area has a title bar with 'Clusters / My Cluster' and a cluster status 'My Cluster' (green). Below the title bar are buttons for Edit, Clone, Restart, Terminate, and Delete. A navigation bar includes Configuration, Notebooks (2), Libraries (selected), Event Log, Spark UI, Driver Logs, Metrics, Apps, and Spark Cluster UI - Master. In the Libraries section, there are two buttons: 'Uninstall' and 'Install New'. A callout box with a blue arrow points from the text 'Click on "Install New"' to the 'Install New' button. The table below the buttons has columns for Name, Type, Status, and Source.

Name	Type	Status	Source

Click on “Install New”

Installing GraphFrames

Clusters / My Cluster

My Cluster

Edit Clone Restart Terminate Delete

Configuration Notebooks (2) Libraries

Uninstall Install New

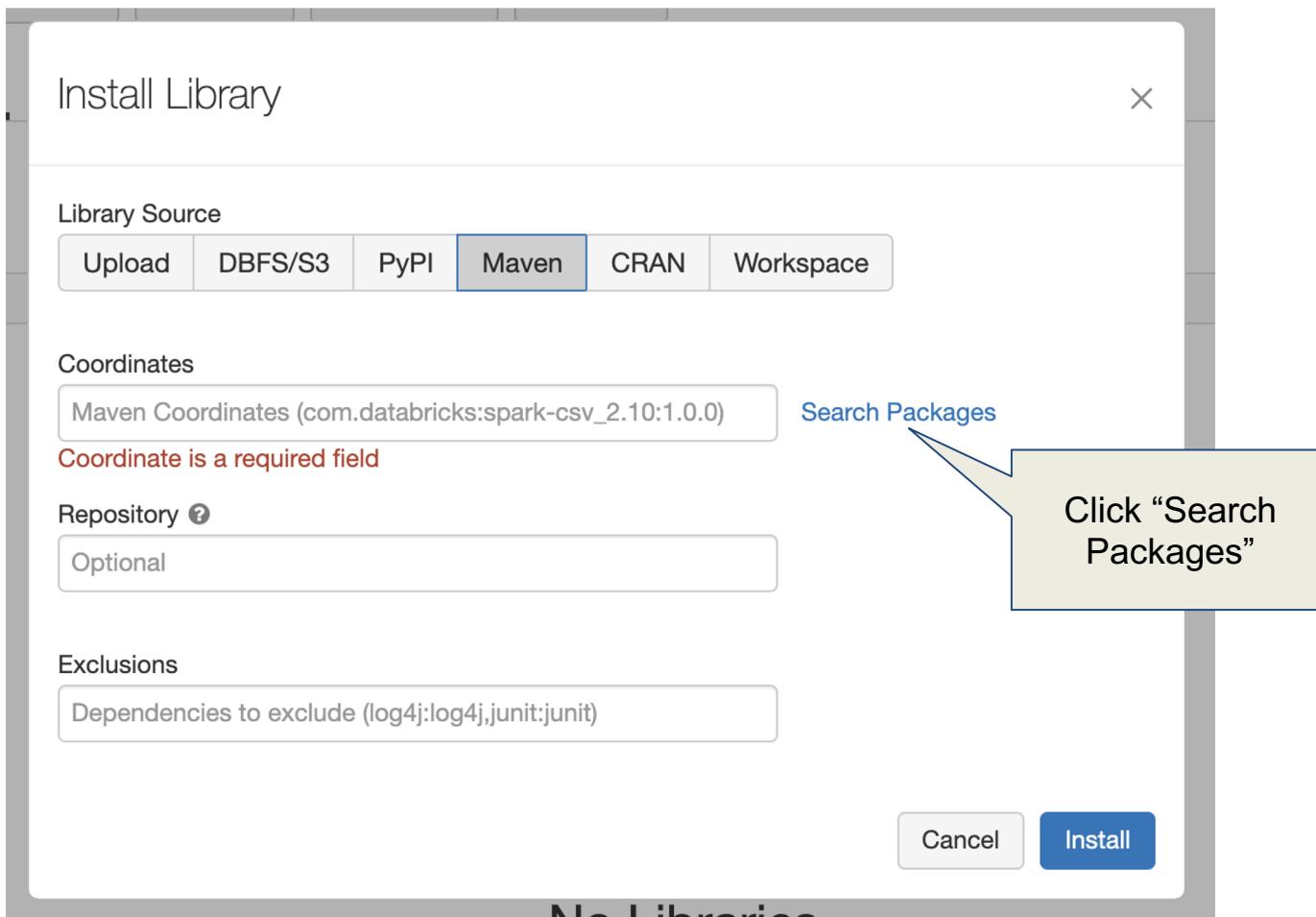
Name

Select “Maven”

Drop JAR here

Cancel Install

Installing GraphFrames



Installing GraphFrames

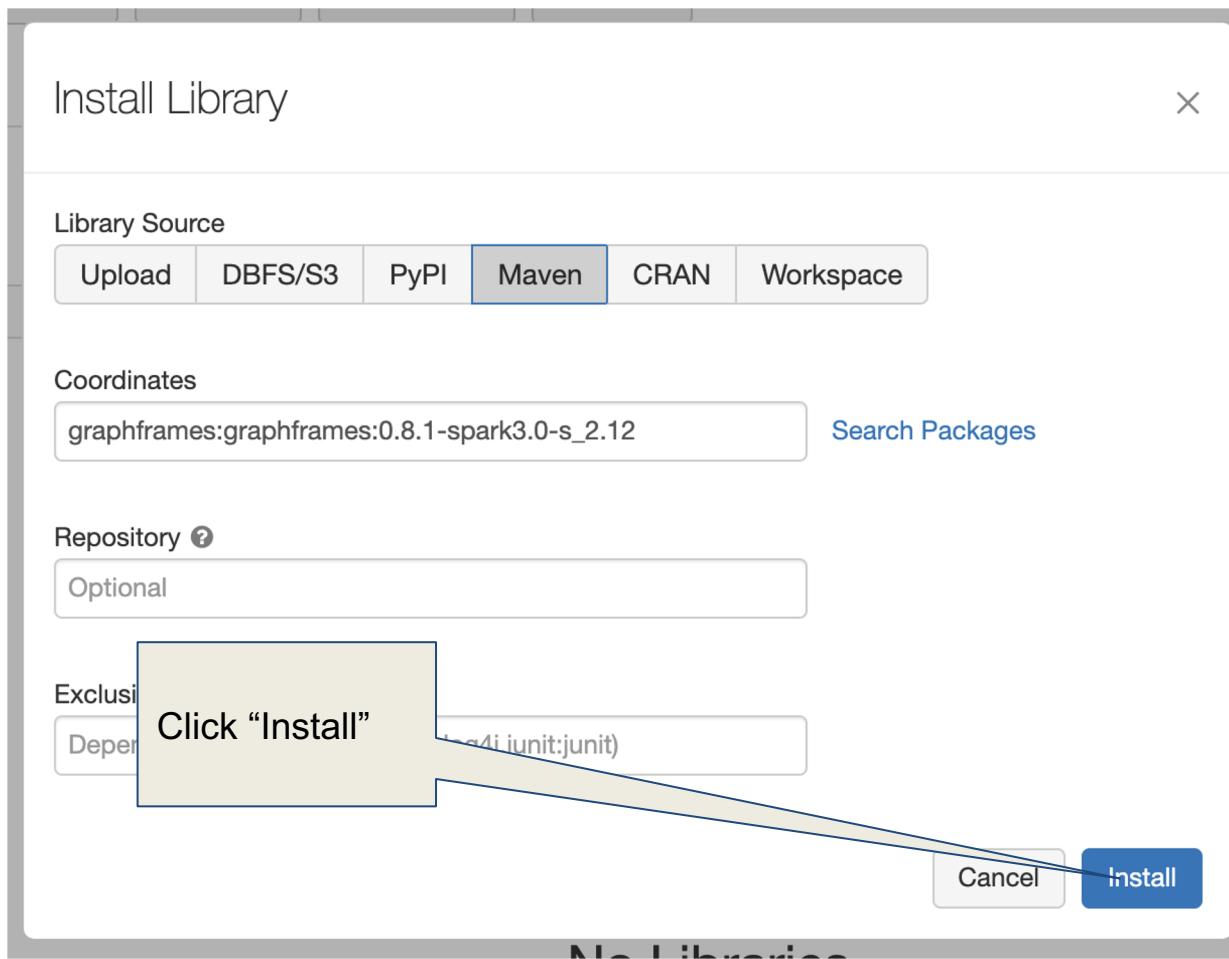
The screenshot shows a web-based interface for managing Spark packages. At the top, there's a navigation bar with a back arrow, the text "Search Packages", and a close button (X). Below this is a search bar with the placeholder "Search Packages" and a dropdown menu set to "Spark Packages". A search input field contains the query "graph". The main area is a table listing packages:

Name	Organization	Description	Releases	Options
spark-knn-graphs	tdebatty	Spark algorithms for building and processing k...	0.13	Select
graphx-diameter	Cecca	A spark package to approximate the diameter ...	0.2.0-s_2.11	Select
sparkling-graph	sparkling-gr...	Large scale, distributed graph processing mad...	0.0.7	Select
graphframes	graphframes	GraphFrames: DataFrame-based Graphs	0.8.1-spark3.0-... s_2.12	Select
graphx-overlapping-community	bhardwajank	Graphx Overlapping Community Detection	0.8.1-spark2.4- s_2.11 0.8.1-spark2.4- s_2.12 0.8.1-spark3.0- s_2.12 0.8.0-spark3.0- s_2.12 0.8.0-spark2.4- s_2.11 0.7.0-spark2.4- s_2.11	Select

A callout box with a blue border and arrow points from the text "Locate graphframes package, and click "Select"" to the "Select" button next to the "graphframes" package entry. The "Select" button for the "graphframes" entry is highlighted with a blue border.

Locate graphframes package, and click "Select"

Installing GraphFrames



Installing GraphFrames

Clusters / My Cluster

My Cluster

Edit

Clone

Restart

Terminate

Delete

Configuration

Notebooks (2)

Libraries

Event Log

Spark UI

Driver Logs

Metrics

Apps

Spark Cluster UI - Master ▾

Uninstall

Install New

<input type="checkbox"/>	Name	Type	Status	Source
<input type="checkbox"/>	graphframes:graphframes:0.8.1-spark3.0-s_2.12	Maven	 Resolving	

Installing GraphFrames

Clusters / My Cluster

● My Cluster

Edit

Clone

Restart

Terminate

Delete

Configuration

Notebooks (2)

Libraries

Event Log

Spark UI

Driver Logs

Metrics

Apps

Spark Cluster UI - Master ▾

Uninstall

Install New

<input type="checkbox"/>	Name	Type	Status	Source
<input type="checkbox"/>	graphframes:graphframes:0.8.1-spark3.0-s_2.12	Maven	Installed	

Questions

