#66daysofdata #66daysofhealthdata

Day Six: Graph Algorithms

Reference: Needham, M. & Hodler, A.E. (2019). *Graph Algorithms, Practical Examples in Apache Spark & Neo4j*. Sebastopol: O'Reilly.

CHAPTER 2: GRAPH THEORY & CONCEPTS

TERMS

- Label: marks a node as part of a group
- Properties: attributes; can contain a variety of data types
- Sub-graph: a graph within a larger graph, useful as filters for focused analysis
- Path: a group of nodes and their connecting relationships

GRAPHS

- Simple graph: node pairs can have one relationship between them
- Multigraph: node pairs can have multiple relationships between them
- Graph/pseudograph: node pairs can have multiple relationships and/or self-referential relationships

NETWORKS

- Random network: no hierarchies, completely avg. distribution of connections, flat with no discernable pattern, all nodes have same probability of attachment to any other node (normally distributed, most nodes have the same # of relationships)

- Small-world network: common social network; hub and spoke pattern, "Six degrees of Kevin Bacon"

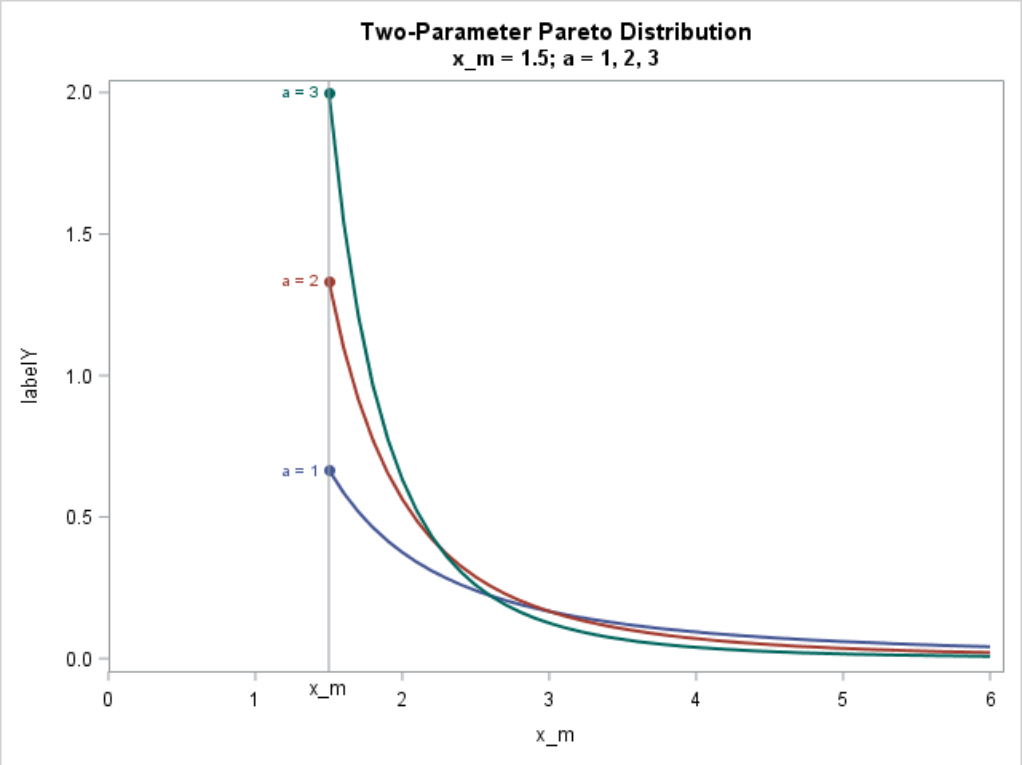
- Scale-free network: power-law distributions, hub and spoke architecture preserved regardless of scale; World Wide Web

A **POWER-LAW** (or scaling law) **DISTRIBUTION** is where one quantity varies as a power of another.

Examples:

1. Area of a cube = x^3
2. Pareto distribution (20% of the population controls 80% of the wealth)

In a graph with a power-law distribution most nodes have very few relationships but a small number of them have a lot, creating hub and spoke structures.



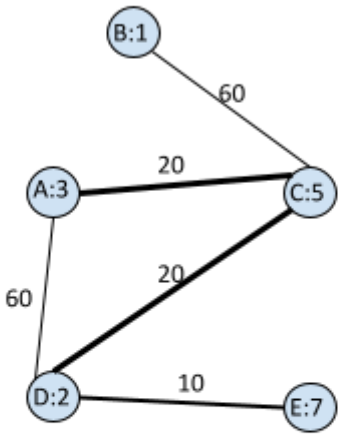
FLAVORS OF GRAPHS

- Connected vs. disconnected: is there a path between any two nodes regardless of distance? Islands can cause unexpected behavior. Graph with islands - disconnected, composed of 'components' or 'clusters'.
- Weighted vs. unweighted: are there domain-specific values on relationships or nodes? Many algorithms expect weights, and there are significant differences in performance and results when they're ignored.
- Directed vs. undirected: do relationships define a start and end node? Adds context, in some algos you can set one, two or no directions.
- Cyclic vs. acyclic: do paths start and end at the same node? Acyclic graphs or spanning trees (having a beginning and end) are the basis for many graph algorithms.
- Sparse vs. dense: what is the relationship to node ratio? Extremely dense or sparse connected graphs can cause divergent results, data modeling may help.
- Mono, bi or k-partitie: Do nodes connect to one other node type, two others or more others? Example: are we looking for users who like movies? Or users like users who like movies?

Weighted graphs - weights can represent anything such as cost, time, distance, capacity, or any domain-specific prioritization; can hold values on relationships or nodes (e.g. fastest path between two nodes, cumulative total, lowest value, optimum)

Weighted edges can denote how 'expensive' it is to travel from 1 node to another:

What's the shortest path from A to E?



A, C, D, E = 50
Going from A to E through D would 'cost' 60, and the total cost of that trip would be 70; the less 'expensive' path is A, C, D, E.

One way to think of this graph might be in terms of distance on roads. If you wanted to travel from A to E, but there are 60 miles between A and D (say, a long, winding road over some mountains), and only a total of 40 miles between A -> C -> D, going the 40 miles to D makes more sense than going 60, before you travel the final 10 miles from D to E to complete your journey.

Directed graphs - In an undirected graph, relationships are bi-directional. In a directed graph, relationships have a specific direction.
Relationships TO a node are called "*in-links*" and FROM a node, "*out-links*".

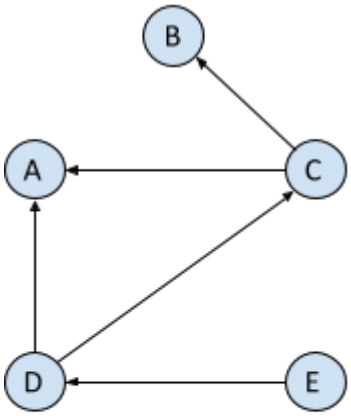
In a directed graph, the direction indicates another dimension of information.

Relationships of the same type in opposite directions carry different meanings.

Direction can be used to indicate dependency or flow.

CYCLES are paths through relationships and nodes that start and end at the same node. ACYCLIC graphs don't have these.

A **DIRECTED ACYCLIC GRAPH (DAG)** always has dead-ends, called 'leaf nodes'.



Directed Acyclic Graph (DAG)
A is a collider, C and D are mediators, and
A and B are leaf nodes.

Cycles are common; *converting cyclic graphs to acyclic ones by cutting relationships eliminates processing problems*.

TREES: undirected or directed acyclic graphs; also, a graph where any two nodes are connected by only one path.

Trees play a key role in designing networks, data structures, and search optimizations; they improve categorization (classification) or organizational hierarchies

Common Trees

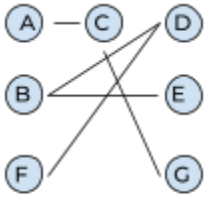
- Rooted: common root node, no cycles
- Binary: Up to 2 child nodes, no cycles
- Spanning: subgraph of all nodes but not all relationships, no cycles

SPANNING TREE:
An acyclic subgraph that includes all the nodes of a larger graph but *not all the relationships*. A minimum spanning tree connects all the nodes with either the least # of hops or the least weighted paths.

SPARSE vs. DENSE GRAPHS

The sparsity of a graph is based on the # of relationships it has compared to the maximum possible number.

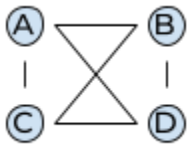
If this matrix represents all the nodes in a graph, and a 1 indicates a relationship between two nodes, then in a sparse matrix / graph, there would not be many 1s:



	A	B	C	D	E	F	G
A	0	0	1	0	0	0	0
B	0	0	0	1	1	0	0
C	1	0	0	0	0	0	1
D	0	1	0	0	0	1	0
E	0	1	0	0	0	0	0
F	0	0	0	1	0	0	0
G	0	0	1	0	0	0	0

D = 2(5) / 7(7-1) = 0.2381 (proportion of possible relationships used)
MaxD = 7(7-1)/2 = 21 (total possible relationships)

In a dense graph, all the nodes are connected to all the other nodes, and the matrix representation would be all 1s



	A	B	C	D
A	0	1	1	1
B	1	0	1	1
C	1	1	0	1
D	1	1	1	0

D = 2(6)/4(4-1) = 12/12 = 1 (proportion of relationships used)
MaxD = 4(4-1)/2 = 6 (total possible relationships)

The “dense graph” can also be called a *complete graph* or a *clique*.

The maximum density of a graph is the number of possible relationships:

MaxD = N(N-1) / 2 where N = the # of nodes

The actual density of a graph is the **proportion of relationships used**:

D = 2(R) / N(N-1) where R is the # of relationships, and N is the # of nodes

If D is 1, all the possible relationships are present

Any graph where D approaches MaxD is considered dense

Most graphs based on real networks tend toward sparseness, with an approximately linear correlation of total nodes to total relationships (e.g. wires, pipes, roads, friendships)

Relationship-types and Graph Algorithms

Most networks contain data with many node & relationship types; however, algorithms frequently only consider one node & relationship type.

A graph with one node & relationship type: **monopartite**
A graph whose nodes can be divided into 2 sets, such that relationships only connect a node from one set to a node from a different set: **bipartite**; you can usually divide a bipartite graph into 2 monopartite graphs
K-partite graphs have k node types; many knowledge graphs have a large # for k, combining many different concepts and information types

TYPES OF GRAPH ALGORITHMS

Pathfinding

Most frequent task: finding the shortest path - path has:

- Fewest hops (unweighted graph)
- Lowest weight (weighted graph)

Average shortest path: used to consider the efficiency and resiliency of networks; e.g.) distances between subway stations, optimized route of network traffic. *Use the **DIAMETER** of a graph to find the **longest shortest path** between all node points*.

Centrality

Which nodes are most important in a network; can be different definitions of importance such as ability to quickly spread info, vs. bridge groups

Community Detection

Most networks have more-or-less independent subgraphs within the overall structure; connectivity is used to find communities and quantify the quality of groupings. Can uncover hubs and hierarchies, and tendencies of groups such as the ability to attract or repel others. These factors can be used to study emergent phenomena such those that lead to echo chambers and filter bubble effects.

GRAPH DEFINITIONS

Resource: Gross, J.L. & Yellen, J. (2013). Section 1.1, Fundamentals of Graph Theory in *Handbook of Graph Theory, 2nd Edition*, Gross, J.L., Yellen, J. & Zhang, P. Eds. Chapman Hall/CRC.

D1: A ***graph*** $G = (V, E)$ consists of two sets V and E.

- The elements of V are called ***vertices*** (or ***nodes***).
- The elements of E are called ***edges***.
- Each edge has a set of one or two vertices associated to it, which are called its ***endpoints***. An edge is said to ***join*** its endpoints.

NOTATION: The subscripted notations V_G and E_G (or $V(G)$ and $E(G)$) are used for the vertex- and edge-sets when G is not the only graph under consideration.

D2: If vertex v is an endpoint of edge e, then v is said to be ***incident*** on e, and e is incident on v.

D3: A vertex *u* is ***adjacent*** to vertex *v* if they are joined by an edge.

D4: Two adjacent vertices may be called ***neighbors***.

D5: ***Adjacent edges*** are two edges that have an endpoint in common.

D6: A ***proper edge*** is an edge that joins two distinct vertices.

D7: A ***multi-edge*** is a collection of two or more edges having identical endpoints.

D8: A ***simple adjacency*** between vertices occurs when there is exactly one edge between them.

D9: The ***edge-multiplicity*** between a pair of vertices *u* and *v* is the number of edges between them.

D10: A ***self-loop*** is an edge that joins a single endpoint to itself.

GRAPH PLATFORM PROCESSING CONSIDERATIONS

Graph analytical processing has unique qualities such as:

- Structure-driven computation
- Global focus
- Parsing difficulty

Processing considerations for graph paradigms
Node-centric: uses nodes as processing units, having them accumulate and compute state and communicate state changes via messages to their neighbors; uses the provided transformation functions for more straightforward algorithm implementation

Relationship-centric: may perform better for subgraph or sequential analysis

Graph-centric: process nodes within a subgraph independently of other subgraphs while minimal communication with other subgraphs happens via messaging

Traversal-centric: traversers accumulate data while navigating the graph as a means of computation

Algorithm-centric: optimizes implementations per algorithm; hybrid approach

Example: Pregel (Google) http://bit.ly/2Twj9sY

- Node-centric, fault-tolerant approach for analysis of large graphs
- Based on BSP (bulk synchronous parallel) model
- Separates computation and communication phases

SPARK & NEO4J

Spark - a distributed and general-purpose cluster computing framework; scale-out and node-centric graph compute engine

- Algos are parallelizable or partitionable
- Algo workflow needs multilingual ops in multiple tools and languages
- Analysis can be run offline in batch mode (may or may not be true in our case)
- Graph analysis is on data not transformed into a graph format
- Team needs and has expertise to code and implement their own algos
- Team uses graph algos infrequently
- Team prefers to keep all data and analysis within Hadoop

Neo4j - a tightly-integrated graph database and algo-centric processing center, optimized for graphs; popular for building graph-based applications and includes a graph algorithms library tuned for its native graph database

- Algo more iterative and require good memory locality
- Algos and results are performance sensitive
- Graph analysis is on complex data, requires deep path traversal
- analysis / results integrated into transactional workloads
- Results used to enrich an existing graph
- Needs to integrate with graph-based visualization tools
- Team prefers prepackaged and supported algos