Business Analytics II

Lecture 3
Network Analytics

Pedro Hernandez Serrano
p.hernandezserrano@maastrichtuniversity.nl

April 10, 2018

BISS Institute

Today's session

- Density Based Clustering
  - DBSCAN implementation
- Network Analytics
  - Some History
  - Some Use Cases
  - Graph Databases
  - Neo4j Overview
  - Graph Theory
  - Basic Network Analysis
- Hands-on during the session on Jupyter Notebooks
  - Cluster example notebook
  - Neo4j Sandbox
  - Networkx library implementation

Density Based Clustering

DBSCAN

Short for Density-Based Spatial Clustering of Applications with Noise, is the most popular density-based clustering method. We attempt to capture our intuition that a cluster is a region of the data space where there are lots of points, surrounded by a region where there are few points.

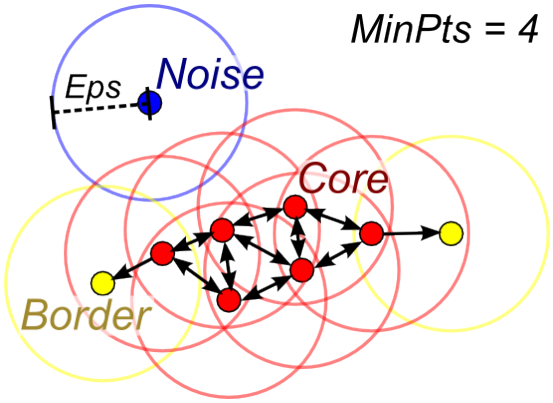
The goal is to identify dense regions, which can be measured by the number of objects close to a given point.

Parameters:
Epsilon (ϵ): Is the radius of neighborhood around a point x. It is called called the $\epsilon - neighborhood$ of x.
Minimum points (*MinPts*): Is the minimum number of neighbors within ϵ radius.

Density Based Clustering

DBSCAN

Any point x in the dataset, with a neighbor count greater than or equal to *MinPts*, is marked as a **core point**. We say that x is border point, if the number of its neighbors is less than *MinPts*, but it belongs to the $\epsilon - neighborhood$ of some core point x_i. Finally, if a point is neither a core nor a border point, then it is called a noise point or anomalous.



DBSCAN

DBSCAN tends to be slower than K-means because it requires computation of the similarity graph on the input dataset, but it has several major conceptual advantages:

- The number of clusters does not need to be known a priori.
- Recovers much more flexible cluster shapes than K-means, which can only find spherical clusters.
- Intrinsically finds and labels outliers as such, making it a great solution for anomaly detection.
- It works with any distance function. (e.g. "euclidean", "manhattan", "jaccard", and "levenshtein")

Example

- Go to **dbscan points** notebook
- Create a k means model for crime data

Use the methods:

```
dbscan_model = tc.dbscan.create(sf, radius=0.2,
                                min_core_neighbors=15)
dbscan_model.summary()
dbscan_model.cluster_id

sf.head()
sf.tail()
```

Network Analytics

Some History

Network analytics has its basis in joint field between discrete mathematics and topology called **Graph theory**. This field was born out of a very practical problem. (Konigsberg bridge problem, 1736)

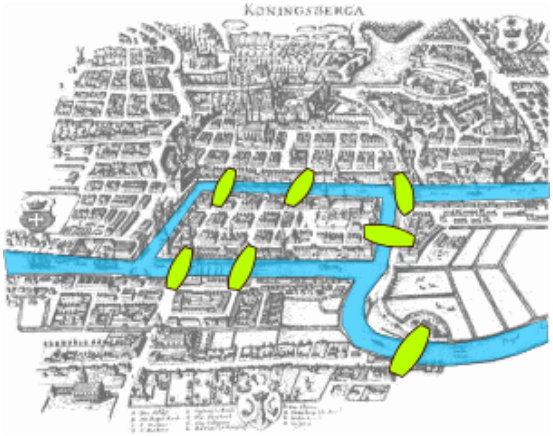
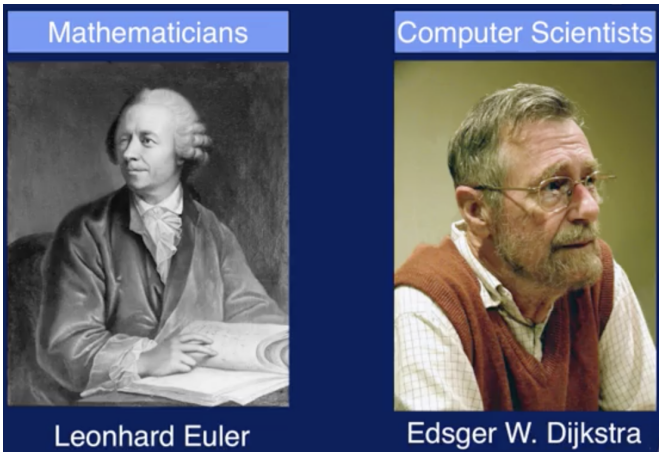


Image: https://en.wikipedia.org/wiki/Graph_theory

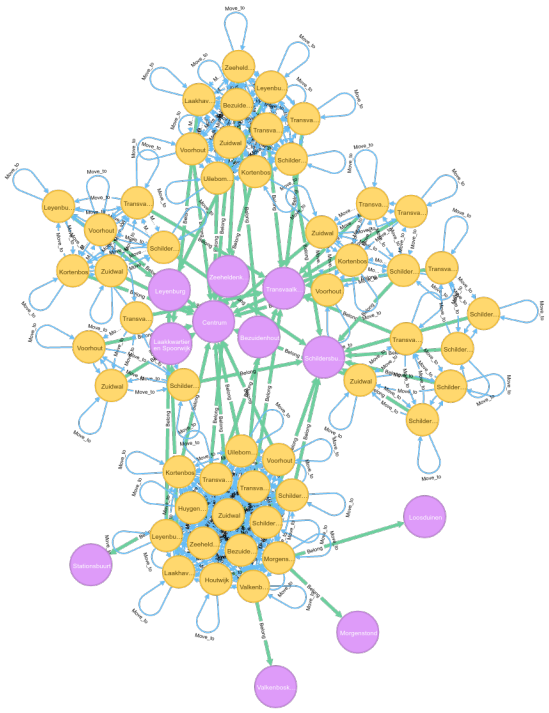
Network Analytics

Euler looked at this and figured out that you really cannot create such a walk. He said, it cannot be done, because there are an odd number of bridges, and proved it mathematically.

Edsger Dijkstra, a well-known computer scientist who has developed graph algorithms. His work has had far further impact on both the theoretical computer science and practical applications.

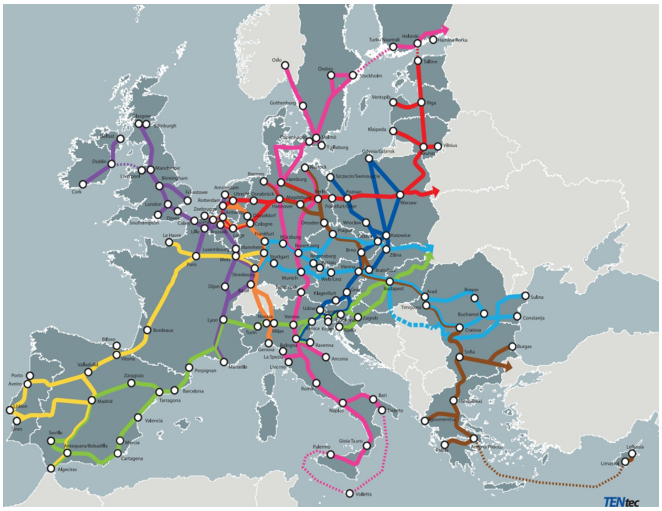


Network Analytics



- Demographic Movement
- Urban Nets for city planning
- Energy Transition
- Systems Biology
- Gene Pathways Networks
- Citations and Influence
- Social Networks (whole field)
- Scientific Representation in general
- Linguistics

Network Analytics



https://ec.europa.eu/transport/themes/infrastructure_en

- Logistics and Transport Network
- Optimization Cargo Route problems
- Driver on-demand (e.g. Uber, dijsktra to assign a driver)
- Product co-purchasing networks
- Communications Network (mails, semantic web)

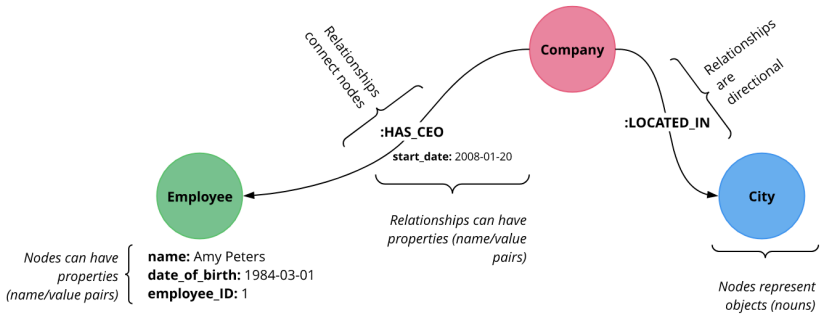
Graph Databases

What are Graph Databases?

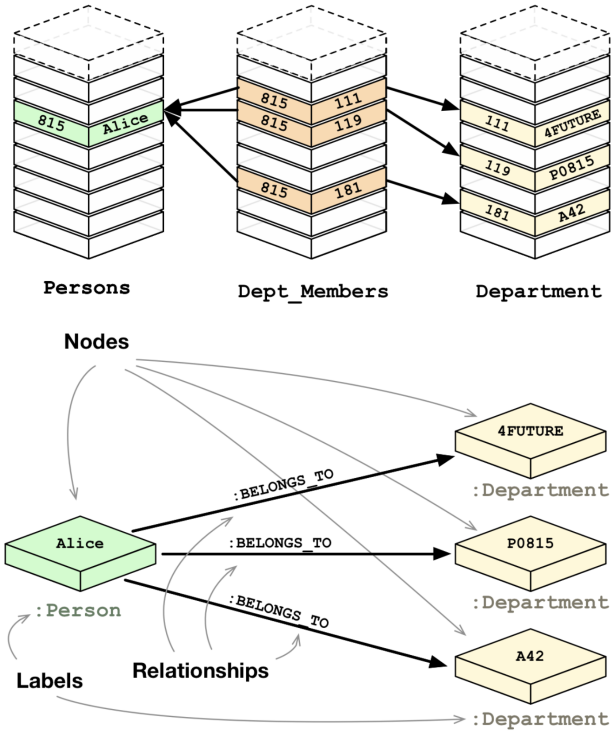
Graph Database is a system that represents and stores data in a graph structure and allows the execution semantic queries, directly retrieving related data.

- Node (Vertex): This is the most basic element. A node acts as the key in a dictionary. Note that every node is unique, and hey can hold any number of attributes
- Relationship (Edge): Are the elements where to store relations, they provide directed, named, semantically relevant connections between two node-entities. (e.g. direction, weight, distance, etc.)
- Attributes (Properties) are the information related to a node

- Improves the representation of relationships
- Possibility to retrieve another data analysis techniques, such as community detection, pattern recognition and centrality measures
- The flexibility in the data schema. While Relational Databases requires new tables or alterations in the existing ones to add new types of data, in a Graph Database we can add new type of vertices and edges without alterations in the previously stored data.

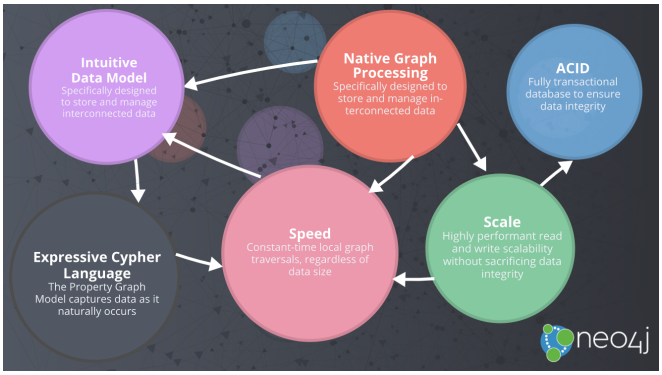


Graph DB vs Relational DB



https://neo4j.com/developer/graph-database/

Neo4j



- Open-source NoSQL native graph database
- Provides a compliant transactional backend for applications.
- Publicly available since 2007. Source code, written in Java and Scala
- The most popular graph-based database

Graph Query Language

- Retrieving data from a graphDB requires a query language
- Currently no single language has been universally adopted in the same way as SQL was for relational databases
- Some standardization efforts Gremlin, SPARQL, and Cypher.

Cypher

SQL Statement

```
SELECT name FROM Person
LEFT JOIN Person_Department
  ON Person.Id = Person_Department.PersonId
LEFT JOIN Department
  ON Department.Id = Person_Department.DepartmentId
WHERE Department.name = "IT Department"
```

Cypher Statement

```
MATCH (p:Person)<-[:EMPLOYEE]-(d:Department)
WHERE d.name = "IT Department"
RETURN p.name
```


Neo4j Example

- Go to https://neo4j.com/sandbox-v2/
- Create a new sandbox
- Comment the problem

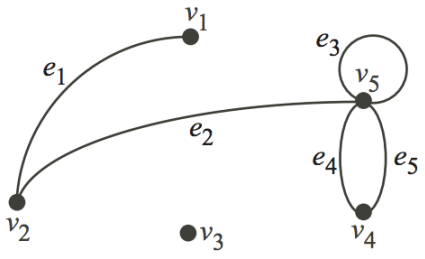
Graph Theory

Definition

A graph is a pair of sets V and E, where V is the set of vertices and E is the set of edges, formed by pairs of vertices.

$$G = (V, E) \quad \textit{where} \quad E = \{(v_i, v_j) | v_i, v_j \in V\}$$

Example
$V = \{v1, \ldots, v5\}$
$E = \{(v1, v2), (v2, v5), (v5, v5), (v5, v4), (v5, v4)\}$



Graph Properties

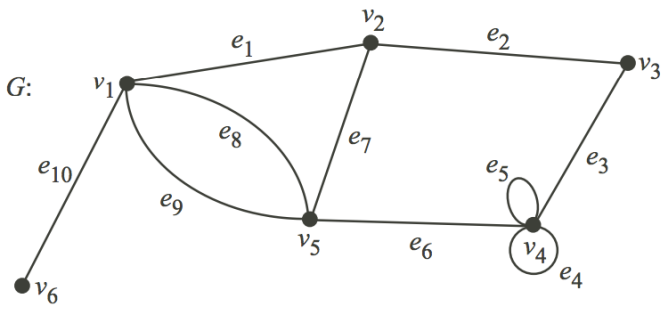
- The two vertices u and v are end vertices of the edge (u, v).
- Edges that have the same end vertices are parallel.
- An edge of the form (v, v) is a loop.
- A graph is simple if it has no parallel edges or loops.
- A graph with no edges (i.e. E are empty) is empty.
- A graph with no vertices (i.e. V and E are empty) is a null graph.
- A graph with only one vertex is trivial.
- Edges are adjacent if they share a common end vertex.

Graph Properties

- The degree of the vertex v, written as $d(v)$, is the number of edges with v as a vertex.
- **Minimum degree** in a graph G is denoted $\delta(G)$
- **Maximum degree** in a graph G is we write $\Delta(G)$
- A pendant (leaf) vertex is a vertex whose degree is 1.
- An isolated vertex is a vertex whose degree is 0.
- Every graph has an even number of vertices of odd degree.
- The graph $G1 = (V_1, E_1)$ is a **subgraph** of $G2 = (V_2, E_2)$ if V_1V_2 and Every edge of G_1 is also an edge of G_2.

Walks and Connectivity

A **walk** in the graph $G = (V, E)$ is a finite sequence with form $v_{io}, e_{j1}, v_{i1}, \ldots, e_{jk}, v_{ik}$ which consists of alternating vertices and edges of G. Vertices v_{it1} and v_{it} are end vertices of e_{jt} where $(t = 1, \ldots, k)$. k is the **length** of the walk. If each vertex of a graph is at least 2 then conaints a **circuit**.



Walks and Connectivity

- It is allowed to visit a vertex or go through an edge more than once
- A walk is **open** if $v_{io} \neq v_{ik}$. Otherwise it is closed
- A walk is a **path** if any vertex is visited at most once
- A closed path is a **circuit**
- A graph is circuitless exactly when there are no loops is called **tree**
- Only exists one path between two any give vertex in a **tree**
- A graph is **connected** if is possible to reach any vertex.
- If u and v are connected and v and w are connected, then u and w are also connected, i.e. if there is a $u - v$ walk and a $v - w$ walk, then there is also a $u - w$ walk.

Structure Properties

Given $G = (V, E)$ a graph:

- The **distance** $d(v_1, v_2)$ between two vertices $v_1, v_2 \in V$ as the length of the minimum path between v_1 and v_2
- The **eccentricity** of a vertex $\epsilon(v) = max_w(d(v, w))$ where $w \in V - w$ is the maximum distance from v to the rest of the vertices
- The **radio** $\rho(G) = min_v(\epsilon(v))$ of the graph is the minimum eccentricity
- The **diemeter** $\delta(G) = max_v(e(v))$ is the maximum eccentricity

Networkx

- Go to **network analytics** notebook
- Install Networkx and follow the examples
- Reproduce the analysis with Facebook data