Advanced Data Management (CSCI 490/680)

Visualization and Databases

Dr. David Koop
Graphs

- In computing, a graph is an abstract data structure that represents set objects and their relationships as vertices and edges/links, and supports a number of graph-related operations.
- Objects (nodes): \{A, B, C, D\}
- Relationships (edges):
  \{ (D, B), (D, A), (B, C), (B, A), (C, A) \}
- Operation: shortest path from D to A

[K. Salama, 2016]
Graphs with Properties

- Each vertex or edge may have properties associated with it
- May include identifiers or classes

```
Person
name = 'Tom Hanks'
born = 1956

Person
name = 'Robert Zemeckis'
born = 1951

Movie
title = 'Forrest Gump'
released = 1994
```

D. Koop, CSCI 680/490, Spring 2022
What is a Graph Database?

- A database with an explicit graph structure
- Each node knows its adjacent nodes
- As the number of nodes increases, the cost of a local step (or hop) remains the same
- Plus an Index for lookups
Graph Databases Compared to Relational Databases

Optimized for aggregation

Optimized for connections

[M. De Marzi, 2012]
Graph Databases Compared to Relational Databases

• Relational Databases (querying is through joins)
  - In effect, the join operation forms a graph that is dynamically constructed as one table is linked to another table.
  - Must be inferred through a series of index-intensive operations

• Graph Databases (querying is through traversal paths)
  - There is no explicit join operation because vertices maintain direct references to their adjacent edges
  - Structures are "hard-wired", not computed at query time

[Rodriguez & Neubauer via Lembo & Rosati]
Example: Friend of Friends Query

• Relational:

• Graph:

[Lembo & Rosati]
Graph Databases Compared to Key-Value Stores

- Optimized for simple look-ups
- Optimized for traversing connected data

[M. De Marzi, 2012]
Storing and Traversing Graphs

• Storage:
  - Adjacency List: nodes store their neighbors
  - Incidence List: nodes store edges and edges store incident nodes
  - Adjacency Matrix: adjacency list in matrix form (rows & cols are nodes)
  - Incidence Matrix: rows are vertices, columns are edges

• Traversal:
  - Breadth-first Search
  - Depth-first Search
Adjacency List vs. Incidence List

**Adjacency List**

- **V1**
- **V2**
- **V3**
- **V4**

Properties:
- Storage: $O(|V| + |E| + |L|)$
- $Adjacent(G, x, y): O(|V|)$
- $Neighbors(G, x): O(|V|)$
- $AdjacentEdges(G, x, y): O(|V| + |E|)$
- $Add(G, x, y, l): O(|V| + |E|)$
- $Delete(G, x, y, l): O(|V| + |E|)$

**Incidence List**

- **V1**
- **V2**
- **V3**
- **V4**

Properties:
- Storage: $O(|V| + |E| + |L|)$
- $Adjacent(G, x, y): O(|E|)$
- $Neighbors(G, x): O(|E|)$
- $AdjacentEdges(G, x, y): O(|E|)$
- $Add(G, x, y, l): O(|E|)$
- $Delete(G, x, y, l): O(|E|)$

Simplified version: each edge has a different label

From [Acosta et al.]
## Adjacency Matrix vs. Incidence Matrix

**Adjacency Matrix**

<table>
<thead>
<tr>
<th></th>
<th>V1</th>
<th>V2</th>
<th>V3</th>
<th>V4</th>
</tr>
</thead>
<tbody>
<tr>
<td>V1</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>•</td>
</tr>
<tr>
<td>V2</td>
<td>•</td>
<td>•</td>
<td></td>
<td>•</td>
</tr>
<tr>
<td>V3</td>
<td>•</td>
<td></td>
<td>•</td>
<td>•</td>
</tr>
<tr>
<td>V4</td>
<td>•</td>
<td>•</td>
<td></td>
<td>•</td>
</tr>
</tbody>
</table>

Properties:
- *Storage:* $O(|V|^2)$
- *Adjacent(G,x,y):* $O(1)$
- *Neighbors(G,x):* $O(|V|)$
- *AdjacentEdges(G,x,y):* $O(|E|)$
- *Add(G,x,y,l):* $O(|E|)$
- *Delete(G,x,y,l):* $O(|E|)$

From [ABFRV14]

**Incidence Matrix**

<table>
<thead>
<tr>
<th></th>
<th>L1</th>
<th>L2</th>
<th>L3</th>
</tr>
</thead>
<tbody>
<tr>
<td>V1</td>
<td>destination</td>
<td>destination</td>
<td></td>
</tr>
<tr>
<td>V2</td>
<td></td>
<td>source</td>
<td>source</td>
</tr>
<tr>
<td>V3</td>
<td></td>
<td></td>
<td>destination</td>
</tr>
<tr>
<td>V4</td>
<td>source</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Properties:
- *Storage:* $O(|V| \times |E|)$
- *Adjacent(G,x,y):* $O(|E|)$
- *Neighbors(G,x):* $O(|V| \times |E|)$
- *AdjacentEdges(G,x,y):* $O(|E|)$
- *Add(G,x,y,l):* $O(|V|)$
- *Delete(G,x,y,l):* $O(|V|)$

From [Acosta et al.]
Graph Models: Relational Model

<table>
<thead>
<tr>
<th>NAME</th>
<th>LASTNAME</th>
<th>PERSON</th>
<th>PARENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>George</td>
<td>Jones</td>
<td>Julia</td>
<td>George</td>
</tr>
<tr>
<td>Ana</td>
<td>Stone</td>
<td>Julia</td>
<td>Ana</td>
</tr>
<tr>
<td>Julia</td>
<td>Jones</td>
<td>David</td>
<td>James</td>
</tr>
<tr>
<td>James</td>
<td>Deville</td>
<td>David</td>
<td>Julia</td>
</tr>
<tr>
<td>David</td>
<td>Deville</td>
<td>Mary</td>
<td>James</td>
</tr>
<tr>
<td>Mary</td>
<td>Deville</td>
<td>Mary</td>
<td>Julia</td>
</tr>
</tbody>
</table>

Figure 1: Example of a genealogy expressed in the relational model (i.e. as tables on the left) and a diagram of its scheme on the right.

Of node, by allowing nesting graphs inside nodes. As drawbacks, both models use complex data structures which make it less intuitive their use and implementation.

Regarding simplicity, one of the most popularized models is the semi-structured model, which uses the most simple version of a graph, namely a tree, the most common and intuitive way or organizing our data (e.g. directories).

Finally, the most common models are slightly enhanced version of the plain graphs. One of them, the RDF model, gives a light typing to nodes, and considers edges as nodes, giving uniformity to the information objects in the model. The other, the property graph model, allows to adds properties to edges and nodes.

Next, we will present these models and show a paradigmatic example of each. We will use the genealogy toy example modeled as tables and a simple schema in Figure 1.
Property Graph Model (Cypher in neo4j)

- Directed, labelled, attributed multigraph
- Properties are key/value pairs that represent metadata for nodes and edges

![Property Graph Model Diagram](image)

[R. Angles and C. Gutierrez, 2017]
Hypergraph Model (Groovy)

- Notion of edge is extended to **hyperedge**, which relates an arbitrary set of nodes
- Hypergraphs allow the definition of complex objects (undirected), functional dependencies (directed), object-ID and (multiple) structural inheritance

![Hypergraph Diagram]

Figure 3: GROOVY. At the schema level (left), we model an object `PERSON` as an hypergraph that relates the attributes `NAME`, `LASTNAME` and `PARENTS`. Note the value functional dependency (VDF) `NAME,LASTNAME`\!→\!`PARENTS` logically represented by the directed hyperedge \{\{NAME,LASTNAME\}\}\{\{PARENTS\}\}. This VFD asserts that `NAME` and `LASTNAME` uniquely determine the set of `PARENTS`.

3.3 Nested graphs: The Hypernode model

A hypernode is a directed graph whose nodes can themselves be graphs (or hypernodes), allowing nesting of graphs. Hypernodes can be used to represent simple (flat) and complex objects (hierarchical, composite, and cyclic) as well as mappings and records. A key feature is its inherent ability to encapsulate information.

The hypernode model which we will use as example was introduced by Levene and Poulovassilis [104]. They defined the model and a declarative logic-based language structured as a sequence of instructions (hypernode programs), used for querying and updating hypernodes. A more elaborated version [123] includes the notion of schema and type checking, introduced via the idea of types (primitive and complex), that are also represented by nested graphs (See an example in Figure 4). It also includes a rule-based query language called **Hyperlog**, which can support both querying and browsing with derivations as well as database updates, and is intractable in the general case. A third version of the model [102] discusses a set of constraints (entity, referential and semantic) over hypernode databases. In addition it presents another query and update language called **HNQL**, which uses compounded statements to produce HNQL programs.

[R. Angles and C. Gutierrez, 2017]
RDF (Triple) Model

- Interconnect resources in an extensible way using graph-like structure for data
- Schema and instance are mixed together
- SPARQL to query
- Semantic web

From the first perspective, an atomic RDF expression is a triple consisting of a subject (the resource being described), a predicate (the property) and an object (the property value). Each triple represents a logical statement of a relationship between the subject and the object, and one could enhance this basic logic by adding rules and ontologies over it (e.g. RDFS and OWL). A general RDF expression is a set of such triples called an RDF graph (see example in Figure 6), which can be intuitively considered as a semantic network. From the second perspective, the RDF model is the most general representation of a graph, where edges are also considered nodes. In this sense, formally it is not a traditional graph. This allows for self-referencing, reification (i.e., making statements over statements), and to be essentially self-contained. The drawback of all this niceties are the complexity that come with this generalization, particularly for efficient implementation.

SPARQL is the standard query language for RDF. It is able to express complex graph patterns by means of a collection of triple patterns whose solutions can be combined and restricted by using several operators.
Graph Query Languages: Cypher

- Implemented by neo4j system
- Expresses reachability queries via path expressions
  \[ p = (a)-[:knows*]->(b): \text{nodes from } a \text{ to } b \text{ following knows edges} \]
- \text{START } x=node:person(name="John")
  \text{MATCH (x)-[:friend]->(y)}
  \text{RETURN } y.name
Graph Query Languages: SPARQL (RDF)

• Uses SELECT-FROM-WHERE pattern like SQL
• SELECT ?N
  FROM <http://example.org/data.rdf>
  WHERE { ?X rdf:type voc:Person . ?X voc:name ?N }
Graph DBMS Building Blocks

- Property graph data model
- Graph query language
- Graph visualization
- Subgraph matching
- Relational queries
- Path queries
- Stored procedures

[D. Koop, CSCI 680/490, Spring 2022]

[P. Boncz, 2022]
Graph DBMS Problems

• performance
  - Slow loading speeds
  - Query speeds over magnitude slower than RDBMS

• scalability
  - Low datasize limit, typically << RAM
  - Little benefit from parallelism

• reliability
  - Loads never terminate
  - Query run out of memory or crash
  - Bugs

[Graph showing runtime comparison between Umbra prototype RDBMS (4 min), Hyper industry RDBMS (10 min), and every GDBMS we tested (>90 min timeout / crash)]

P. Boncz, 2022
Quiz Wednesday

- Read Nanocubes paper
- Quiz at the beginning of class
Data Exploration through Visualization
Transportation Data - NYC MTA
### MTA Fare Data Exploration

<table>
<thead>
<tr>
<th>REMOTE</th>
<th>STATION</th>
<th>FF</th>
<th>SEN/DIS</th>
<th>7-D AFAS UNL</th>
<th>D AFAS/RMF</th>
<th>JOINT RR TKT</th>
<th>7-D UNL</th>
<th>30-D UNL</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>R011 42ND STREET &amp; 8TH AVENUE</td>
<td>00228985</td>
<td>00008471</td>
<td>00000441</td>
<td>00001455</td>
<td>00000134</td>
<td>00033341</td>
<td>00071255</td>
</tr>
<tr>
<td>2</td>
<td>R170 14TH STREET-UNION SQUARE</td>
<td>00224603</td>
<td>0011051</td>
<td>00000827</td>
<td>00003026</td>
<td>00000660</td>
<td>00089367</td>
<td>00199841</td>
</tr>
<tr>
<td>3</td>
<td>R046 42ND STREET &amp; GRAND CENTRAL</td>
<td>00207758</td>
<td>0007908</td>
<td>0000323</td>
<td>0001183</td>
<td>0003001</td>
<td>00040759</td>
<td>00096613</td>
</tr>
<tr>
<td>4</td>
<td>R012 34TH STREET &amp; 8TH AVENUE</td>
<td>00188311</td>
<td>00006490</td>
<td>00000498</td>
<td>0001279</td>
<td>0003622</td>
<td>00035527</td>
<td>00067483</td>
</tr>
<tr>
<td>5</td>
<td>R293 34TH STREET - PENN STATION</td>
<td>00168768</td>
<td>0006155</td>
<td>0000523</td>
<td>0001065</td>
<td>0005031</td>
<td>00030645</td>
<td>00054376</td>
</tr>
<tr>
<td>6</td>
<td>R033 42ND STREET/TIMES SQUARE</td>
<td>00159382</td>
<td>0005945</td>
<td>0000378</td>
<td>0001205</td>
<td>00000690</td>
<td>00058931</td>
<td>00078644</td>
</tr>
<tr>
<td>7</td>
<td>R022 34TH STREET &amp; 6TH AVENUE</td>
<td>00156008</td>
<td>0006276</td>
<td>0000487</td>
<td>0001543</td>
<td>0000712</td>
<td>00058910</td>
<td>00110466</td>
</tr>
<tr>
<td>8</td>
<td>R084 59TH STREET/COLUMBUS CIRCLE</td>
<td>00155262</td>
<td>0009484</td>
<td>0000589</td>
<td>0002071</td>
<td>0000542</td>
<td>00053397</td>
<td>00113966</td>
</tr>
<tr>
<td>9</td>
<td>R020 47-50 STREETS/ROCKEFELLER</td>
<td>00143500</td>
<td>0006402</td>
<td>0000384</td>
<td>0001159</td>
<td>0000723</td>
<td>00037978</td>
<td>00090745</td>
</tr>
<tr>
<td>10</td>
<td>R179 86TH STREET-LEXINGTON AVE</td>
<td>00142169</td>
<td>0010367</td>
<td>0000470</td>
<td>0001839</td>
<td>0000271</td>
<td>00050328</td>
<td>00125250</td>
</tr>
<tr>
<td>11</td>
<td>R023 34TH STREET &amp; 6TH AVENUE</td>
<td>00134052</td>
<td>0005005</td>
<td>0000348</td>
<td>0001112</td>
<td>0000649</td>
<td>00031531</td>
<td>00075040</td>
</tr>
<tr>
<td>12</td>
<td>R029 PARK PLACE</td>
<td>00121614</td>
<td>0004311</td>
<td>0000287</td>
<td>0000931</td>
<td>0000792</td>
<td>00025404</td>
<td>00065362</td>
</tr>
<tr>
<td>13</td>
<td>R047 42ND STREET &amp; GRAND CENTRAL</td>
<td>00100742</td>
<td>0004273</td>
<td>0000185</td>
<td>0000704</td>
<td>0001241</td>
<td>00022808</td>
<td>00068216</td>
</tr>
</tbody>
</table>
MTA Fare Data Exploration
MTA Fare Data Exploration
MTA Fare Data Exploration
MTA Fare Data Exploration
MTA Fare Data Exploration

East 161st Street and River Avenue

Date

Full Fares Purchased

08-02 08-09 08-16 08-23 08-30 09-06 09-13 09-20 09-27 10-04 10-11 10-18 10-25 11-01
MTA Fare Data Exploration

![Bar chart showing fare data for East 161st Street and River Avenue]

New York Yankees

- **August**
  - Full fares purchased for various dates.
  - Dates include 08-02, 08-09, 08-16, 08-23, 08-30, 09-06, 09-13, 09-20, 09-27, 10-04, 10-11, 10-18, 10-25, 11-01.

- **September**
  - Full fares purchased for various dates.
  - Dates include 09-02, 09-09, 09-16, 09-23, 09-30, 10-07, 10-14, 10-21, 10-28, 11-05, 11-12, 11-19, 11-26.

- **All games are Eastern Time.**

---

D. Koop, CSCI 680/490, Spring 2022

Northern Illinois University
Definition of Visualization

“Computer-based visualization systems provide visual representations of datasets designed to help people carry out tasks more effectively” — T. Munzner
Why do we visualize data?

Figures are richer; provide more information with less clutter and in less space. Figures provide the gestalt effect: they give an overview; make structure more visible.

Figures are more accessible, easier to understand, faster to grasp, more comprehensible, more memorable, more fun, and less formal.

- Total Bandwidth (millions of bits per second)

 deste: [Stasko et al. 1998]

[T. Nørretranders]
Why Visual?

<table>
<thead>
<tr>
<th></th>
<th>I</th>
<th>II</th>
<th>III</th>
<th>IV</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>y</td>
<td>x</td>
<td>y</td>
<td>x</td>
</tr>
<tr>
<td>10.0</td>
<td>8.04</td>
<td>10.0</td>
<td>9.14</td>
<td>10.0</td>
</tr>
<tr>
<td>8.0</td>
<td>6.95</td>
<td>8.0</td>
<td>8.14</td>
<td>8.0</td>
</tr>
<tr>
<td>13.0</td>
<td>7.58</td>
<td>13.0</td>
<td>8.74</td>
<td>13.0</td>
</tr>
<tr>
<td>9.0</td>
<td>8.81</td>
<td>9.0</td>
<td>8.77</td>
<td>9.0</td>
</tr>
<tr>
<td>11.0</td>
<td>8.33</td>
<td>11.0</td>
<td>9.26</td>
<td>11.0</td>
</tr>
<tr>
<td>14.0</td>
<td>9.96</td>
<td>14.0</td>
<td>8.10</td>
<td>14.0</td>
</tr>
<tr>
<td>6.0</td>
<td>7.24</td>
<td>6.0</td>
<td>6.13</td>
<td>6.0</td>
</tr>
<tr>
<td>4.0</td>
<td>4.26</td>
<td>4.0</td>
<td>3.10</td>
<td>4.0</td>
</tr>
<tr>
<td>12.0</td>
<td>10.84</td>
<td>12.0</td>
<td>9.13</td>
<td>12.0</td>
</tr>
<tr>
<td>7.0</td>
<td>4.82</td>
<td>7.0</td>
<td>7.26</td>
<td>7.0</td>
</tr>
<tr>
<td>5.0</td>
<td>5.68</td>
<td>5.0</td>
<td>4.74</td>
<td>5.0</td>
</tr>
</tbody>
</table>

[F. J. Anscombe]
# Why Visual?

<table>
<thead>
<tr>
<th>I</th>
<th>II</th>
<th>III</th>
<th>IV</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>y</td>
<td>x</td>
<td>y</td>
</tr>
<tr>
<td>10.0</td>
<td>8.04</td>
<td>10.0</td>
<td>9.14</td>
</tr>
<tr>
<td>8.0</td>
<td>6.95</td>
<td>8.0</td>
<td>8.14</td>
</tr>
<tr>
<td>13.0</td>
<td>7.58</td>
<td>13.0</td>
<td>8.74</td>
</tr>
<tr>
<td>9.0</td>
<td>8.81</td>
<td>9.0</td>
<td>8.77</td>
</tr>
<tr>
<td>11.0</td>
<td>8.33</td>
<td>11.0</td>
<td>9.26</td>
</tr>
<tr>
<td>14.0</td>
<td>9.96</td>
<td>14.0</td>
<td>8.10</td>
</tr>
<tr>
<td>6.0</td>
<td>7.24</td>
<td>6.0</td>
<td>6.13</td>
</tr>
<tr>
<td>4.0</td>
<td>4.26</td>
<td>4.0</td>
<td>3.10</td>
</tr>
<tr>
<td>12.0</td>
<td>10.84</td>
<td>12.0</td>
<td>9.13</td>
</tr>
<tr>
<td>7.0</td>
<td>4.82</td>
<td>7.0</td>
<td>7.26</td>
</tr>
<tr>
<td>5.0</td>
<td>5.68</td>
<td>5.0</td>
<td>4.74</td>
</tr>
</tbody>
</table>

| Mean of x | 9 |
| Variance of x | 11 |
| Mean of y | 7.50 |
| Variance of y | 4.122 |
| Correlation | 0.816 |

[F. J. Anscombe]
Why Visual?

[F. J. Anscombe]
Why Visual?

<table>
<thead>
<tr>
<th></th>
<th>Mean of x</th>
<th>Variance of x</th>
<th>Mean of y</th>
<th>Variance of y</th>
<th>Correlation</th>
</tr>
</thead>
<tbody>
<tr>
<td>$y_1$</td>
<td>9</td>
<td>11</td>
<td>7.50</td>
<td>4.122</td>
<td>0.816</td>
</tr>
<tr>
<td>$x_1$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$y_2$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$x_2$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$y_3$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$x_3$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$y_4$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$x_4$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

[F. J. Anscombe]
Visual Pop-out
Visual Pop-out

[C. G. Healey]
Visual Pop-out

[Image of a grid of red and blue squares and dots]

[C. G. Healey]
Visual Perception Limitations
Visual Perception Limitations

[C. G. Healey]
Databases and Visualization?
Scalable Visualization

J. Heer