UNIVERSITY OF MARYLAND

UMD DATA605 - Big Data Systems

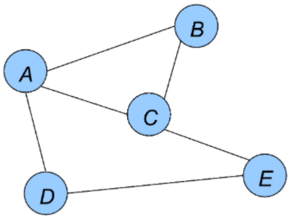
12.1: Graph Data Management

- **Instructor**: Dr. GP Saggese, gsaggese@umd.edu

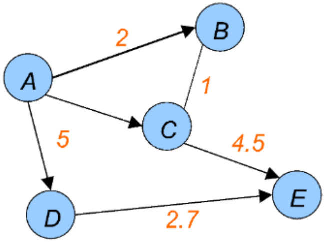
SCIENCE ACADEMY

Graphs: Background

- A **graph** represents entities and their connections
  - *Entities* are vertices (nodes)
  - *Connections* are edges (links, arcs, relationships)
- **Applications of graphs in many fields**
  - Social networks
  - Biological networks
  - Information networks
  - Infrastructure networks
  - . . .



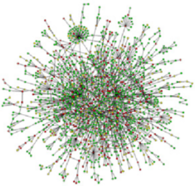
An undirected, unweighted graph



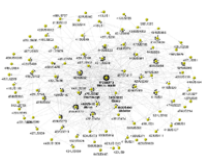
A directed, edge-weighted graph

Graph Data Structures: Motivation

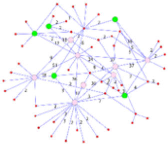
- **Graph data**:
  - Increasing volumes of data
  - Increasing interest in querying and reasoning
- **Sectors**
  - Healthcare
  - Finance
  - Logistics
- **Example applications**
  - Fraud detection
  - Recommendation systems
  - Network analysis



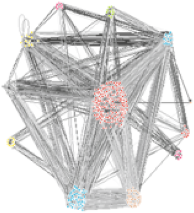
Protein-protein interaction network



Stock trading network



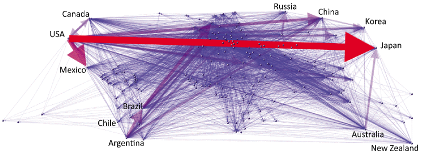
Supreme court citation network



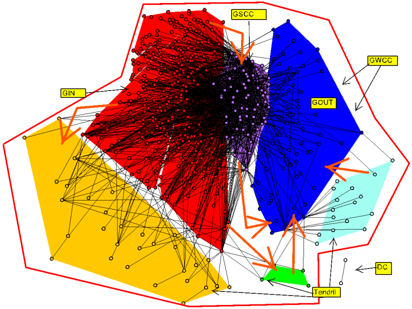
Social networks

Graph Data Structures: Motivation

- **Traditional tools** (e.g., relational DBs, NoSQL DBs) struggle with:
  - Storing and querying graph data
  - Processing graph-structured queries
- **Dedicated solutions** to:
  - Storing: Neo4j
  - Processing: Google Pregel, Apache Giraph, Spark GraphX



Global virtual trade network



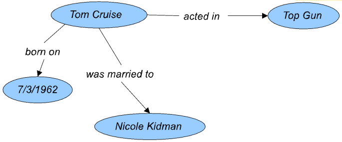
Federal funds networks

SCIENCE ACADEMY

Knowledge Graphs

- **Representation of knowledge in the form of graphs**
  - Capture entities, relationships, properties
  - Provide structured view of real-world information
  - E.g., Google Knowledge Graph, DBpedia, Wikidata
  - E.g., RDF or Property Graph models
- **Applications**
  - Enable machine understanding of complex domains
  - Support semantic search, recommendation, analytics
  - Used in industries for data integration, knowledge discovery, AI applications
- **Ontologies**
  - Provide formal and structure representation of knowledge
    - E.g., type of things and how they relate to each other
    - Like "meaning and rules" to interpret data into a knowledge graph
  - Promote interoperability across knowledge bases

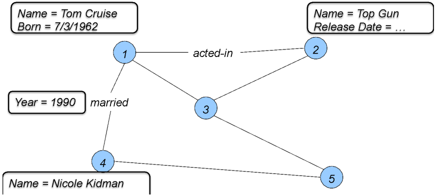
Graph Data Models: RDF



- Resource Description Framework
  - RDF uses triples: subject-predicate-object
  - Connects “subject” and “object” through “predicate”
  - E.g., “TomCruise-acted-TopGun”
- Used to represent knowledge bases
  - Queried through SPARQL
- Pros
  - Standardization
    - W3C standard to model data
    - Subject and object can be URIs in semantic web
  - Interoperability
    - Merge RDF data stores
  - Extensibility
    - Add new nodes and relationships
    - Support ontologies

Graph Data Models: Property Graph

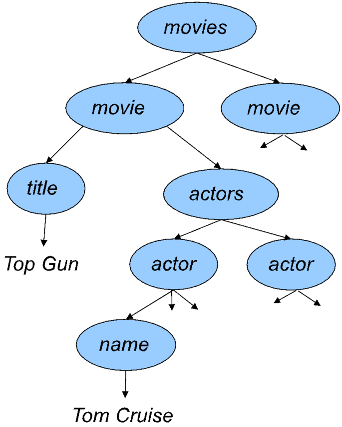
- Directed graph with nodes and edges having key-values *properties*
  - Similar expressive power to RDFs
- **No universal standard**
  - Less "schema"
  - Harder to interoperate
- **Examples of query languages**
  - Cypher for Neo4j
  - Gremlin for Apache TinkerPop



SCIENCE ACADEMY

Graph Data Models: XML

- Common data model for flexible data representation
  - Directed labeled tree
  - Popular for non-tabular data exchange




```
<movies>
  <movie>
  <title>Top Gun</title>
  <actors>
    <actor>
     <name>Tom Cruise</name>
     <born>7/3/1962</born>
    </actor>
   <actor>
     ...
   </actor>
  </actors>
  </movie>
```

Comparison

Feature	RDF	Property Graph	XML
Core data model	Triples: (subject, predicate, object)	Nodes and edges with properties	Hierarchical tree of elements
How facts are stored	Fact is a separate triple	Properties on nodes/edges	Nested tags with attributes
Attributes	Modeled as triples	Key–value pairs	Attributes or child elements
Semantics	Formal (RDF, RDFS, OWL)	No built-in semantics	No built-in semantics
Ontology support	Native, standardized	Optional	None (only schema)
Reasoning & inference	Built-in (e.g., OWL)	Usually not supported	Not supported
Data integration	Excellent for heterogeneous data	Manual mapping	Hard across schemas
Query language	SPARQL	Cypher, Gremlin, GQL	XPath, XQuery
Query style	Pattern matching	Traversals	Tree navigation
Schema	Optional, supported	Labels, constraints	XSD, DTD
Standards	W3C standards	Vendor-specific	W3C standards
Interoperability	Very high	Limited	High for documents
Traversal performance	Moderate	Very fast	Poor
Use cases	Linked data, KBs	Fraud, recommendations	Docs, configs, data exchange
Examples	Wikidata, DBpedia	Neo4j, Neptune	SOAP, RSS, Office

Storing Graph Data

- **File systems**
  - Simple
  - No transactions, ACID compliance
  - Minimal functionality (build analysis/querying on top)
- **Relational databases**
  - Mature technology
  - SQL, transactions, ACID compliance, toolchains
  - Minimal functionality for graph data
- **NoSQL key-value stores**
  - Handle large datasets efficiently in a distributed fashion
  - Minimal native functionality for graph data
- **Graph databases**
  - Efficiently support complex queries/tasks (e.g., graph traversals)
  - Less mature than RDBMSs
  - Often lack declarative language like SQL
    - Write custom programs

Graph Databases

- Many specialized graph DB systems
  - E.g., Neo4j, Titan, OrientDB, AllegroGraph
- Key distinctions from relational / NoSQL databases
  - Store graph structure with pointers
    - Avoid joins
    - Simplify graph traversals
  - Manage and query graph-structured data
    - Write queries and graph algorithms (reachability, shortest paths)
  - Support graph query languages: SPARQL, Cypher, Gremlin
  - Declarative interfaces
  - Provide programmatic API for arbitrary graph algorithms

Query Languages for Graph Databases

Feature	Cypher	Gremlin	SPARQL
Data Model	Property Graph	Property Graph	RDF Triple
Query Style	Declarative	Imperative	Declarative
Syntax Example	SQL-like	Functional API	Triple patterns
Best For	Pattern matching	Complex traversals	Semantic data
Standardization	Neo4j-specific (OpenCypher)	Apache TinkerPop	W3C Standard
Backend Support	Mainly Neo4j	Multi-platform (TinkerPop)	RDF stores
Learning Curve	Low	High	Medium
Use Cases	Social graphs, fraud detection	Distributed graph processing	Linked data, knowledge graphs

Query Languages: Cypher

- **Purpose-Built for Property Graphs**
  - Nodes and relationships with key-value properties
- **Declarative Syntax**
  - Express *what* to retrieve, not *how*
- **Optimized for Pattern Matching**
  - Easily finds subgraph structures (e.g., friends of friends)
  - Limited in Graph Analytics
  - Not ideal for reachability, shortest paths, or centrality
- **Neo4j Native**
- Example:
  - *Find the names of people who know someone named "Alice"*

```
MATCH (person)-[:KNOWS]->(friend {name: "Alice"})
  RETURN person.name
```

Query Languages: Gremlin

- **Supports Multiple Models**
  - Compatible with both Property Graphs and RDF
- **Imperative Style**
  - Describes *how* to traverse the graph step by step
- **Traversal-Based Semantics**
  - Expresses computation as a flow of operations across vertices and edges
- Example:
  - *Find the names of people who know someone named "Alice"*

```
g.V().hasLabel('Person')
  .where(out('KNOWS').has('name', 'Alice'))
  .values('name')
```

Query Languages: SPARQL

- **SQL-Like Syntax**
  - Familiar structure: `SELECT`, `WHERE`, `FILTER`, etc.
- **Built for RDF Data**
  - Queries triples: subject–predicate–object
- **W3C Standard**
  - Core to Semantic Web and Linked Data applications
- Example:
  - *Find the names of people who know someone named "Alice"*

```
PREFIX foaf: <...>

SELECT ?personName
WHERE {
  ?person foaf:knows ?friend .
  ?friend foaf:name "Alice" .
  ?person foaf:name ?personName .
}
```