# What are some Knowledge Graph Inference Algorithms?

# Introduction

- Knowledge Graph Retrieval
  - Query languages
    - SPARQL and Cypher
- Knowledge Graph Inference
  - Drawing conclusions that are not explicit in the knowledge graph
    - E.g., shortest path, concluding new connections

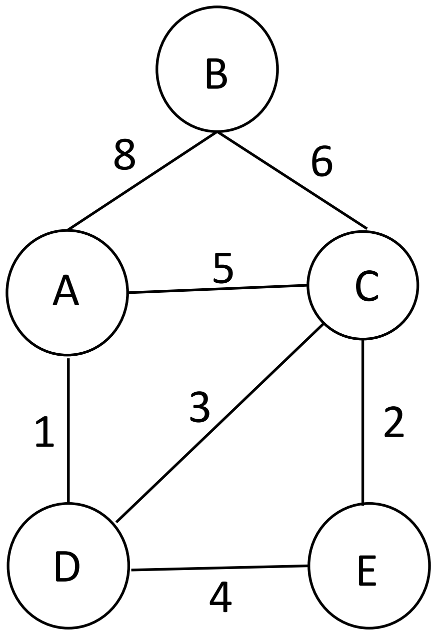
# Outline

- Knowledge Graph Inference
  - Graph-based inference algorithms
  - Ontology-based inference algorithms

# Graph-based Inference Algorithms

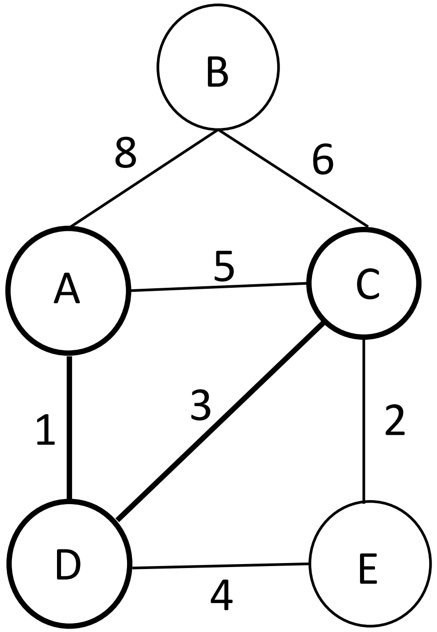
- Path finding
- Centrality Detection
- Community Detection

# Path Finding



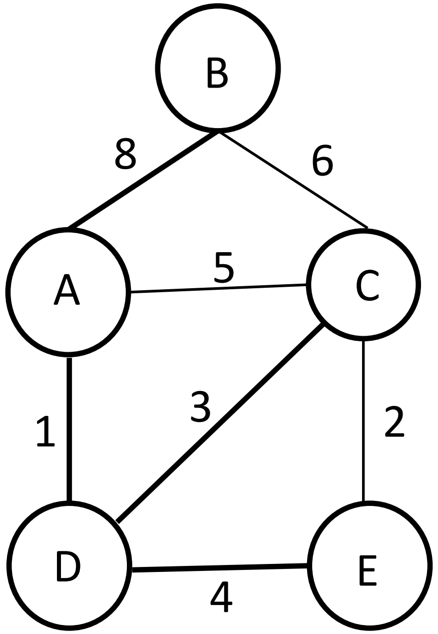
# Path Finding

- Shortest path
  - Optimal path in traffic planning



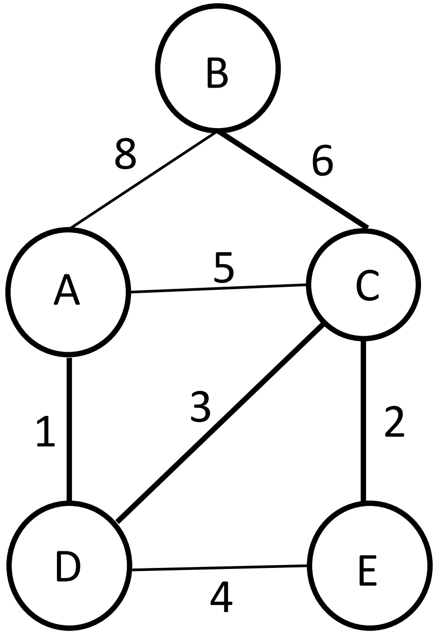
# Path Finding

- Single Source Shortest path



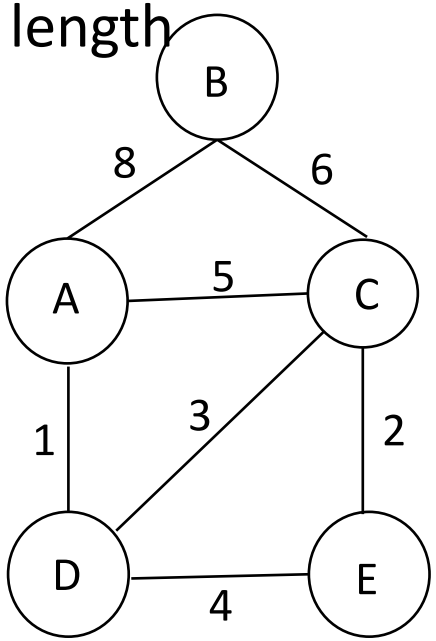
# Path Finding

- Minimum Spanning Tree
  - Trip planning



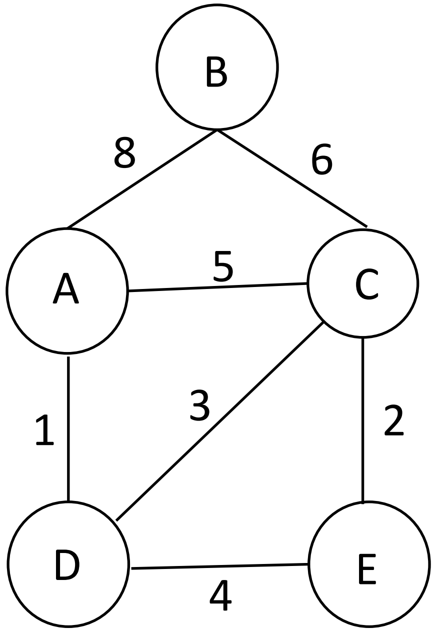
# Path Finding

- A* Algorithm
  - Maintain a tree of paths from the start node
  - Extend them until termination criteria is met
  - Extend based on estimated length
    - f(n)=g(n)+h(n)
    - f(n) = cost until now
    - g(n) – current cost
    - h(n) – estimated cost



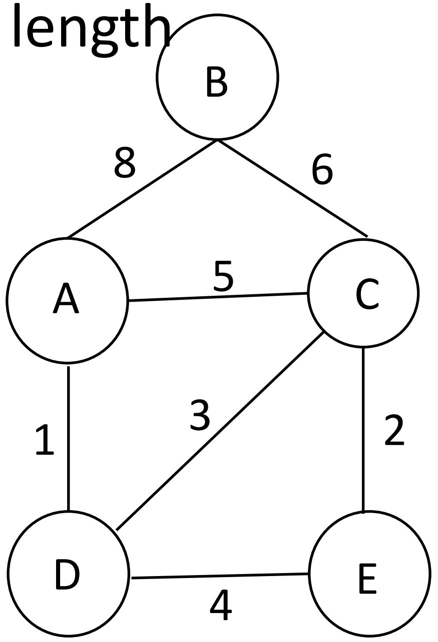
# Path Finding

- A* Algorithm
  - Well-known path finding algorithm
  - Originally developed for AI planning



# Path Finding

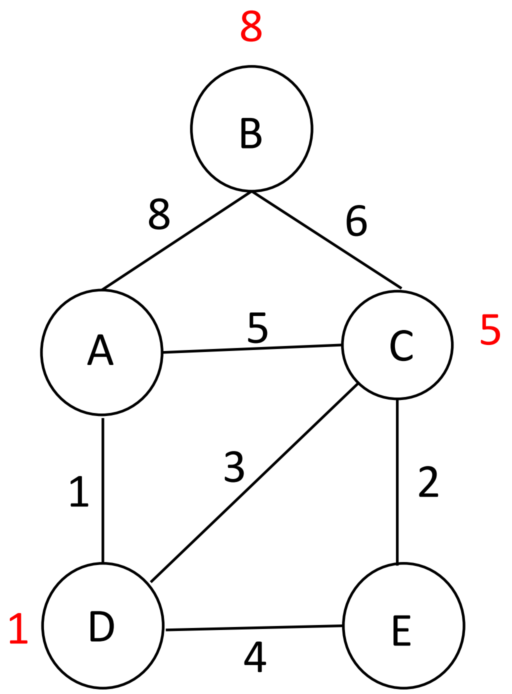
- A* Algorithm
  - Maintain a tree of paths from the start node
  - Extend them until termination criteria is met
  - Extend based on estimated length
    - f(n)=g(n)+h(n)
    - f(n) = cost until now
    - g(n) – current cost
    - h(n) – estimated cost




Admissible heuristic
- Never over estimate

# Path Finding

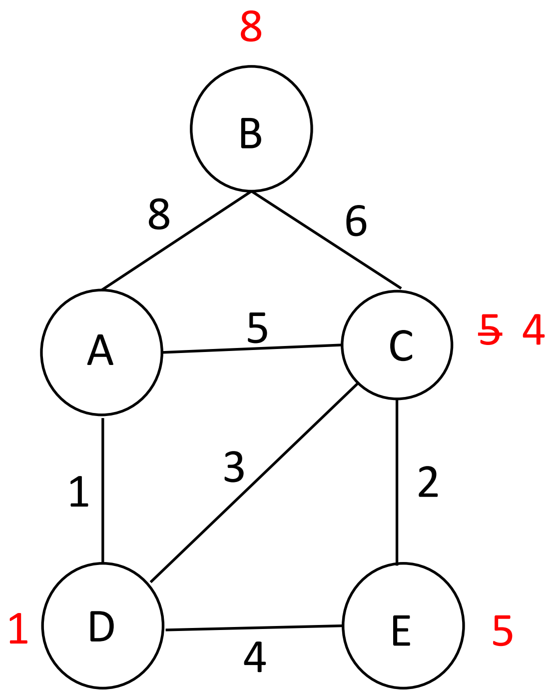
- A* Algorithm
  - Shortest path from A to E
  - h(n)=0
    - Breadth first search
    - Dijkastra’s algorithm




| | |
|---|---|
| A, B | 8 |
| A, C | 5 |
| A, D | 1 |

# Path Finding

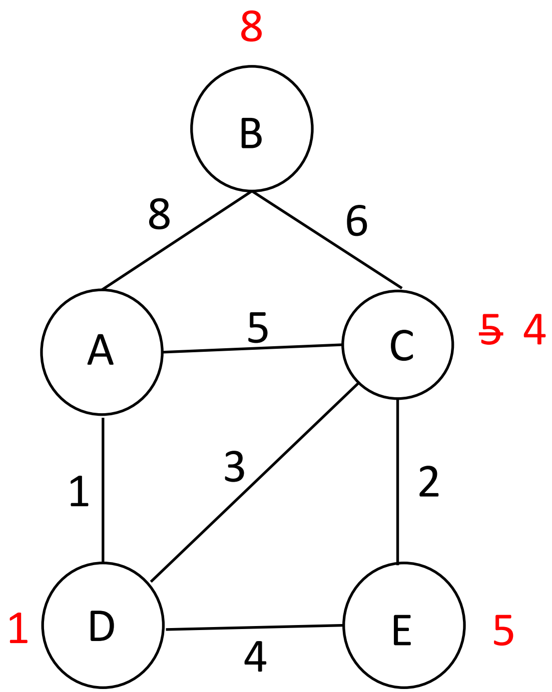
- A* Algorithm
  - Shortest path from A to E
  - h(n)=0
    - Breadth first search
    - Dijkastra's algorithm




| | |
|---|---|
| A, B | 8 |
| A, C | 5 |
| A,D | 1 |
| A, D, C | 4 |
| A, D, E | 5 |

# Path Finding

- A* Algorithm
  - Shortest path from A to E
  - h(n)=0
    - Breadth first search
    - Dijkastra's algorithm




| | |
|---|---|
| A, B | 8 |
| A, C | 5 |
| A,D | 1 |
| A, D, C | 4 |
| A, D, C, E | 6 |
| A, D, E | 5 |

# Graph-based Inference Algorithms

- Path finding
- Centrality Detection
- Community Detection

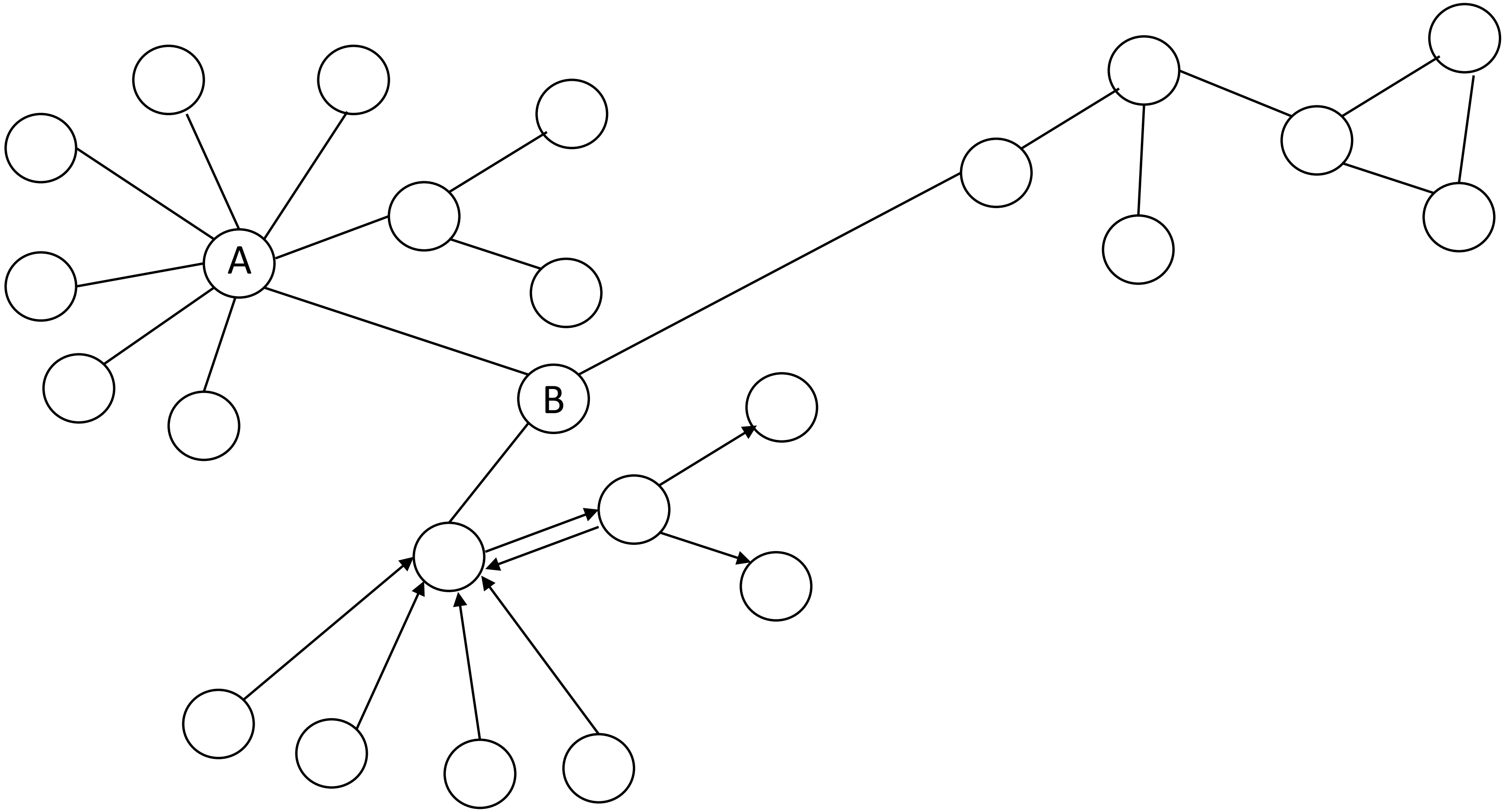
# Centrality Detection

- Helps in understanding the importance of a node in a network
  - Most important nodes
  - Bridges in a network

# Centrality Detection

- Helps in understanding the importance of a node in a network

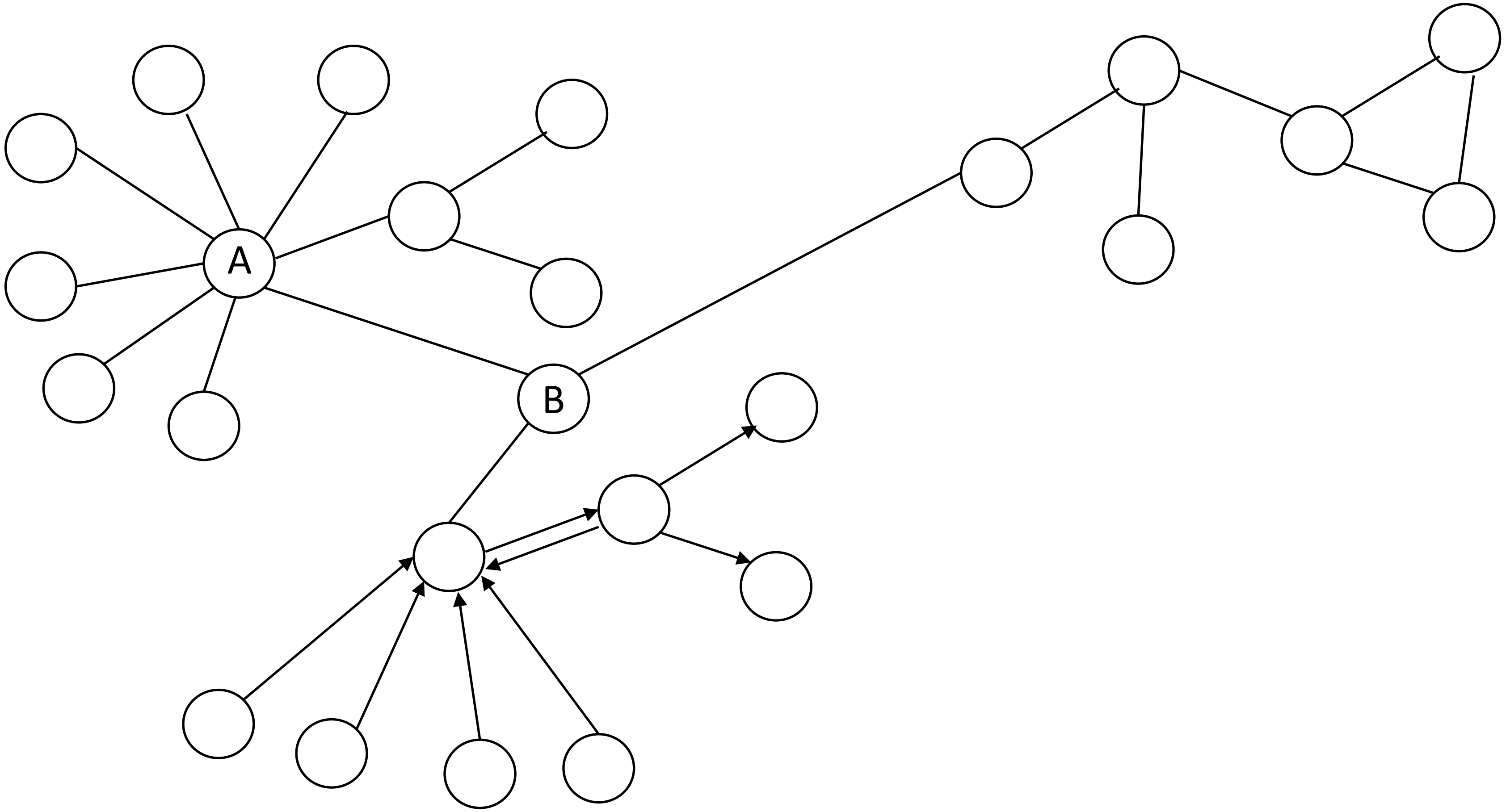
Degree Centrality
A has the highest degree



# Centrality Detection

- Helps in understanding the importance of a node in a network

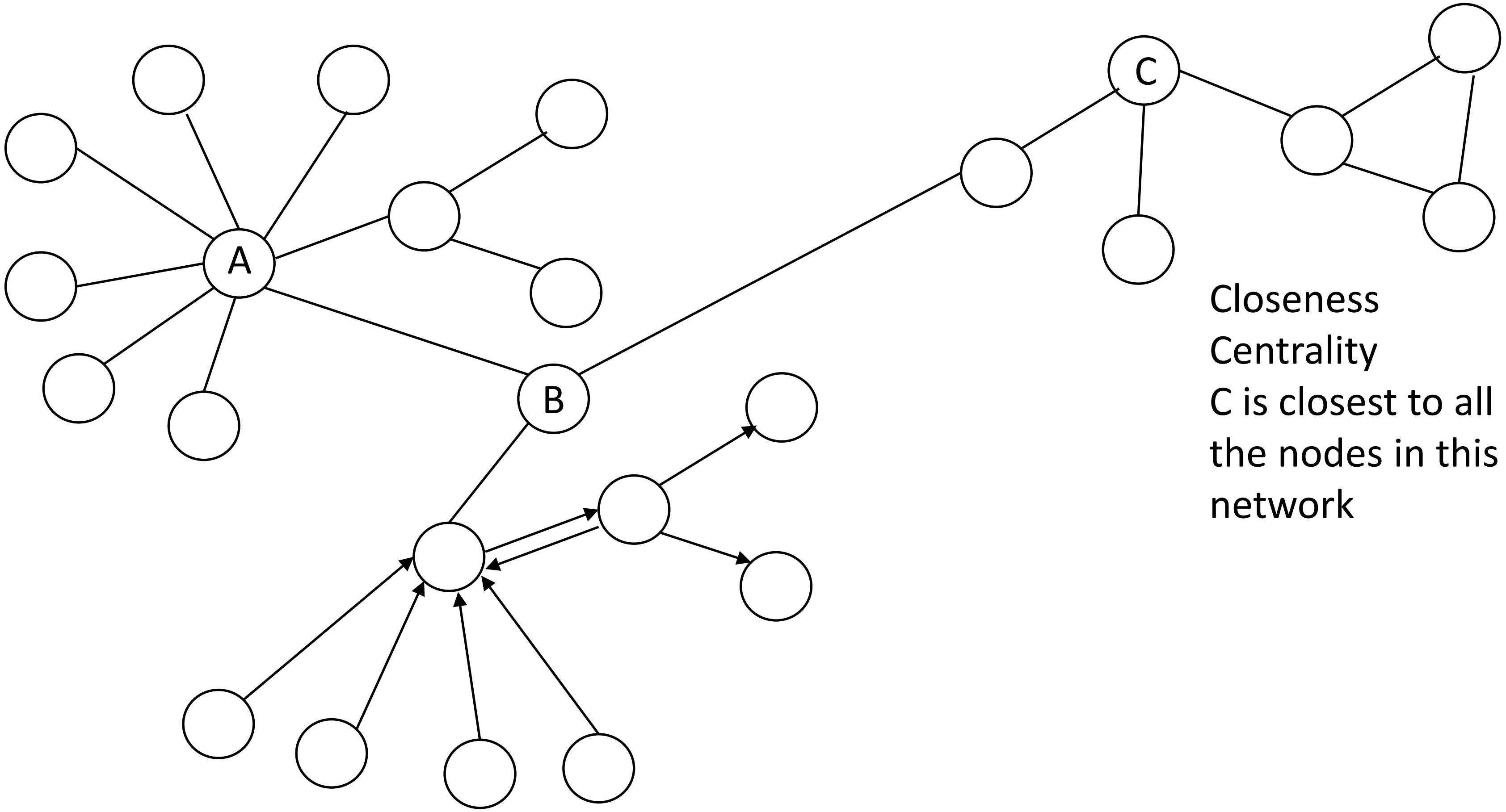
Degree Centrality
A has the highest degree




Betweenness Centrality
B has the largest number of shortest paths passing through it

# Centrality Detection

- Helps in understanding the importance of a node in a network



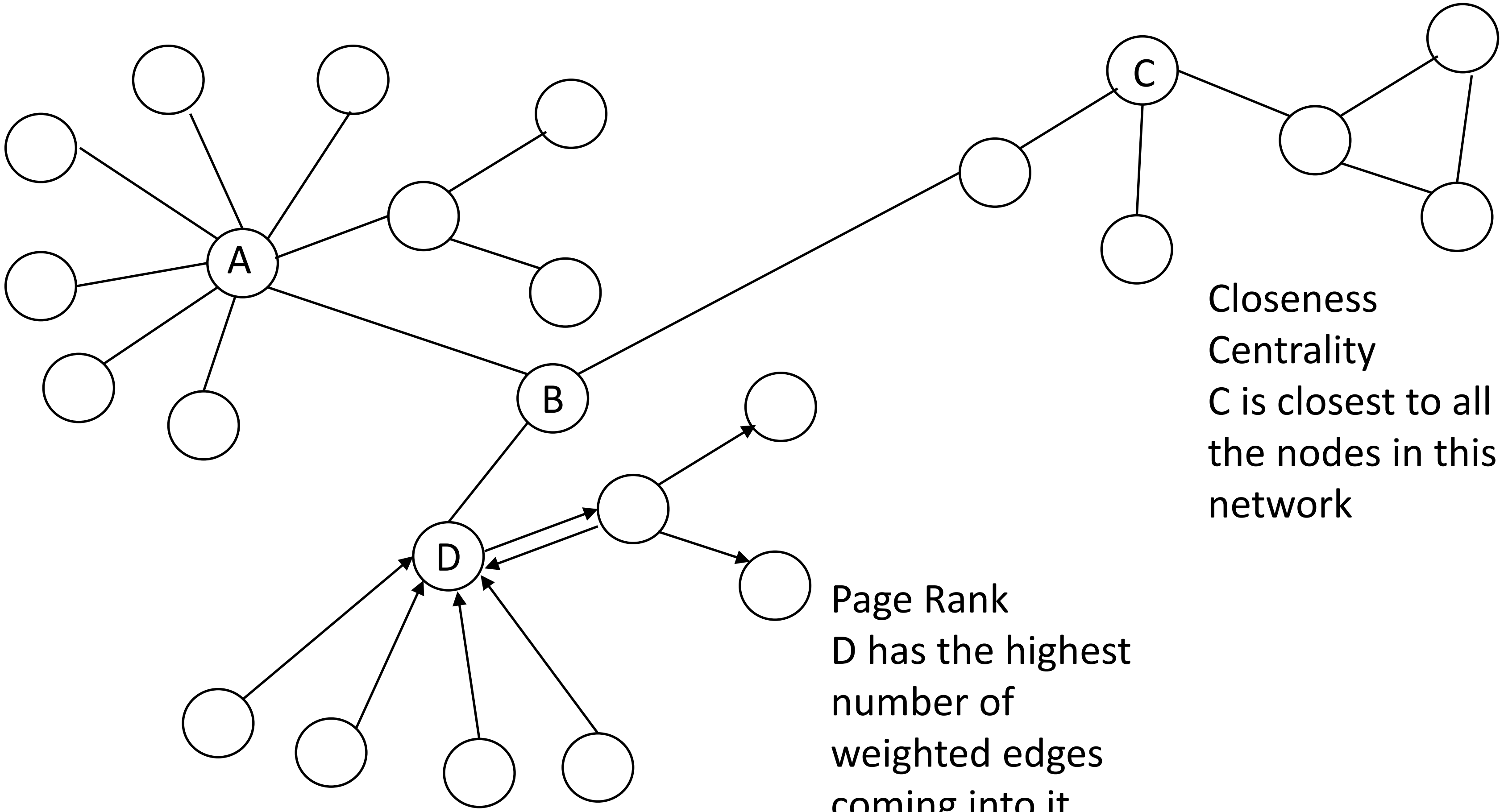

Degree Centrality
A has the highest degree

Betweenness Centrality
B has the largest number of shortest paths passing through it

Closeness Centrality
C is closest to all the nodes in this network

# Centrality Detection

- Helps in understanding the importance of a node in a network




Degree Centrality
A has the highest degree

Betweenness Centrality
B has the largest number of shortest paths passing through it

Closeness Centrality
C is closest to all the nodes in this network

Page Rank
D has the highest number of weighted edges coming into it

# Page Rank

- Measure the transitive influence of nodes
  - A node connected to a few influential nodes could be more important than a node connected to lots of unimportant nodes

$$PR(u) = (1 - d) + d * \left(\frac{PR(T1)}{C(T1)} + \ \dots + \ \frac{PR(Tn)}{C(Tn)}\right)$$

d – damping factor is usually set to 0.85
Set the values of PR for all nodes to same value, and iteratively improve it

# Graph-based Inference Algorithms

- Path finding
- Centrality Detection
- Community Detection

# Community Detection

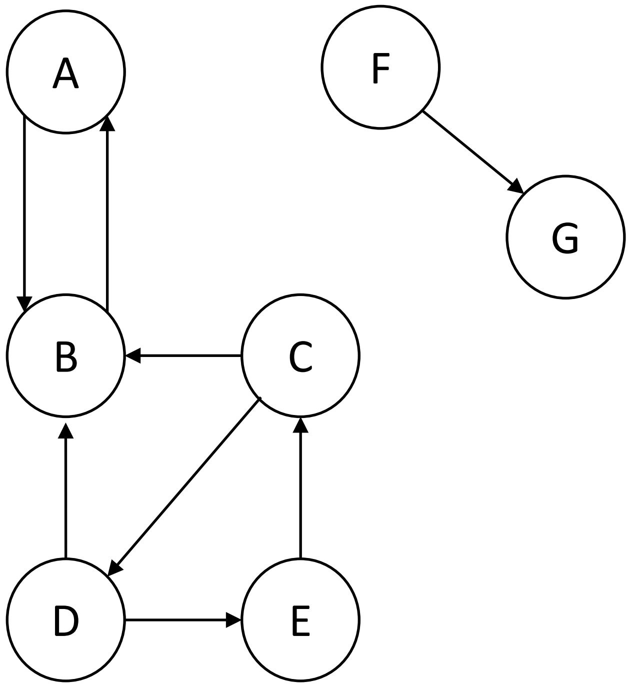
- Identify nodes in the graph that are closely connected to each other
  - More relationships between nodes within the community
- Could be the first step in understanding a graph
  - More in-depth analysis of nodes within the community

# Community Detection

- Two families of algorithms
  - Standard graph algorithms
  - Bottom up algorithms

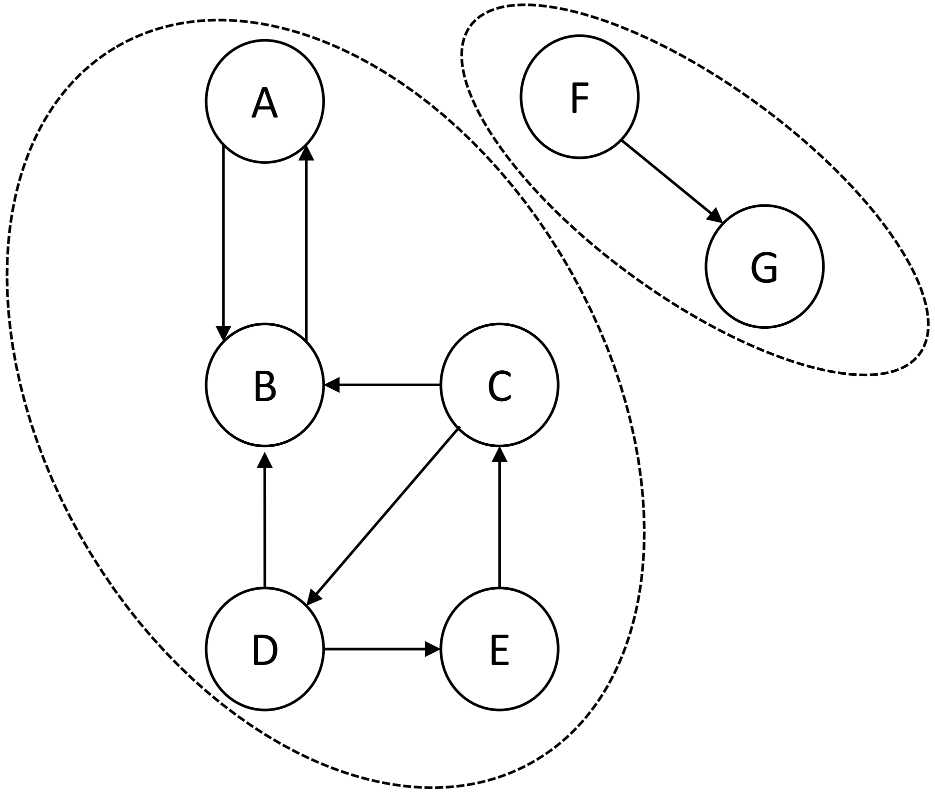
# Community Detection Algorithms

- Standard Graph Algorithms



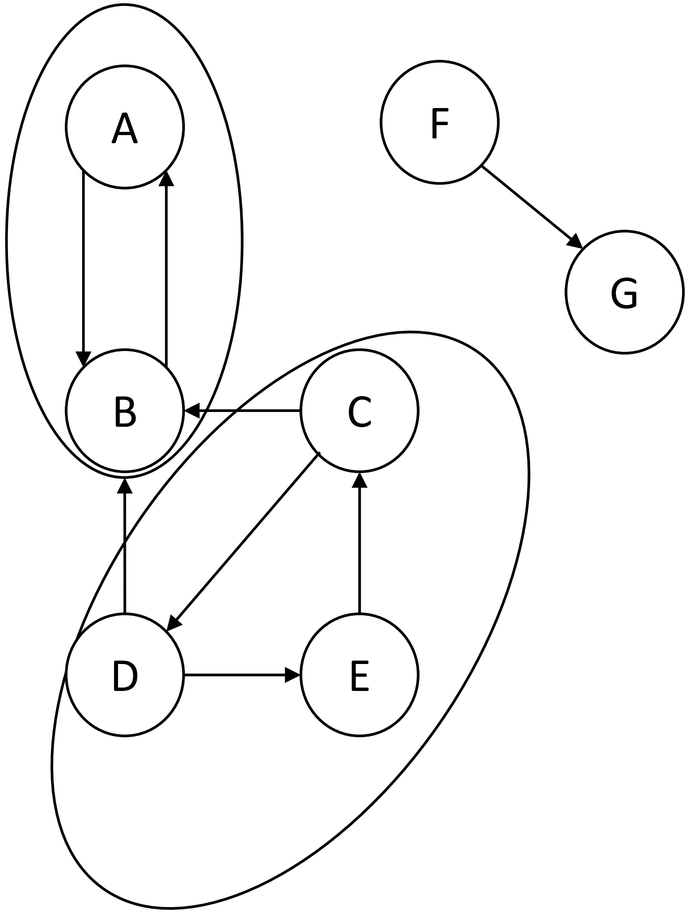
# Community Detection Algorithms

- Standard Graph Algorithms – Connected components



# Community Detection Algorithms

- Standard Graph Algorithms – Strongly connected components



# Community Detection Algorithms

- Bottom up algorithms
  - Label propagation
  - Unfolding

# Community Detection Algorithms

- Label Propagation
  - Assign each node to be in a different community
  - Examine all nodes in a fixed order
    - Update the community of a node that is shared by most of its neighbors
    - Break Ties in a random order
  - Terminate when each node is in a community shared by most of its neighbors

# Community Detection Algorithms

- Unfolding (Also known as Louvain)
  - Phase I
    - Assign each node into a separate community
    - Examine each node and its neighbors to test if there will be an overall gain in modularity by placing it in the same community as a neighbor
      - Modularity calculated by a formula
  - Phase II
    - Create a new graph in which each node represents a community from Phase I
    - If there are edges between nodes in a community, represent it as a self-loop
  - Repeat Phase I on the Phase II graph

# Outline

- Knowledge Graph Inference
  - Graph-based inference algorithms
  - Ontology-based inference algorithms

# Ontology-based Inference

- Key differentiator between a general graph and a knowledge graph
  - Associates classes with nodes
  - Defines semantic properties of relationships
  - Two major categories of inference
    - Class-based Inference or Taxonomic Reasoning
    - Rule-based inference

# Taxonomic Reasoning

- Applicable when it is useful to organize knowledge into classes
  - Classes are nothing but unary relations or types
    - Membership, specialization, disjointness, value restriction

- Both Property graph and RDF data models support classes
  - In property graphs model, node type is the same as a class
  - RDF has an RDF schema layer
    - More advanced systems use a full-fledged ontology language OWL

Our discussion here will be independent of either of the two models

# Example of classes

male(art)

male(bob)

male(cal)

male(cam)

female(bea)

female(coe)

female(cory)

class(male)
instance_of(art,male)

instance_of(bob,male)

instance_of(cal,male)

instance_of(cam,male)

class(female)
instance_of(bea,female)

instance_of(coe,female)

instance_of(cory,female)

# Class specialization

- We can organize classes into a hierarchy
  - e. g., male and female are subclasses of person

    subclass-of(female, person)

    subclass-of(male, person)
  - subclass relationship is transitive

    subclass_of(A,C) :- subclass_of(A,B) & subclass_of(B,C)
  - subclass and instance-of relationships are related

    instance_of(I,B) :- subclass_of(A,B) & instance_of(I,A)

# Disjoint classes

- Classes can be declared to be disjoint
  - e.g, disjoint(male,female)
  - i.e., they do not have any instances in common

```
illegal :- disjoint(A,B) & instance_of(I,A) & instance_of(I,B)
```

or

```
illegal("Disjoint classes cannot have an instance in common") :-
        disjoint(A,B) & instance_of(I,A) & instance_of(I,B)
```

# Class Definition

- Necessary properties of a class
  - Will have instance-of in the body of the rule

```
has_hair_color(X,brown) :-
        instance_of(X,brown_haired_person)
```

- Sufficient properties of a class
  - Will have instance-of in the head of the rule

```
instance_of(X,brown_haired_person) :-
        instance_of(X,person) &
        has_hair_color(X,brown)
```

# Value Restriction

- We can restrict the arguments of a relation to be instances of a specific class
  - domain is the restriction on the first argument

```
illegal :- domain(parent,person) &
           parent(X,Y) &
           ~instance_of(X,person)
```

  - range is the restriction on the second argument

```
illegal :- range(parent,person) &
           parent(X,Y) &
           ~instance_of(Y,person)
```

# Cardinality and Number Restrictions

- We can further restrict the values of relations by specifying cardinality and number restrictions
  - A cardinality restriction limits the number of values of a relation

    illegal :- instance_of(X,person) & ~countofall(P,parent(P,X),2)
  - A numeric range restriction limits the minimum and maximum value

    illegal :- instance_of(X,person) & age(X,Y) & min(0,Y,Y)

    illegal :- instance_of(X,person) & age(X,Y)& min(125,Y,100)

# Inheritance

- The relation values are said to inherit to the instances of a class
  - If art is an instance of the class brown-haired-person, we can conclude that art has brown hair

# Taxonomic Inference

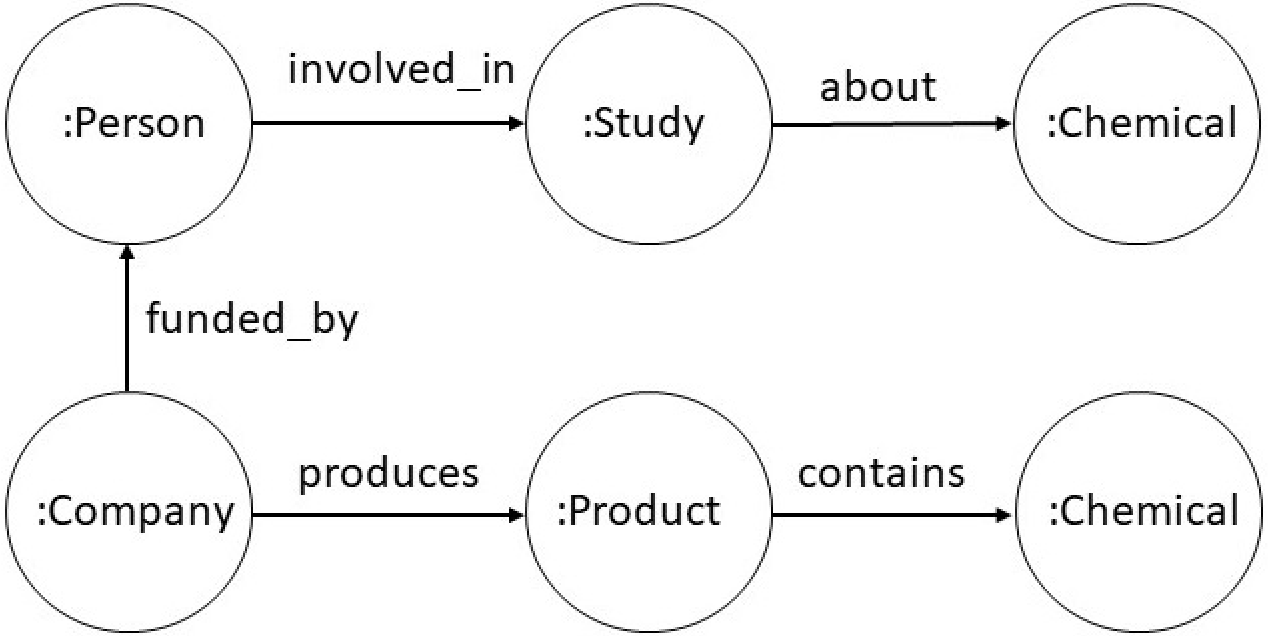
- Given two classes A and B, whether A is a subclass of B?
- Given a class A and an instance I, whether I is an instance of I?
- Given a ground relation atom determine whether it is true or false?
- Given a relation atom, determine values which values make it true?

# Rule-based Inference

- Boundary between taxonomic inference and rule-based inference is not sharp
  - It is generally a matter of the implementation approach
  - Taxonomic inferences can be usually implemented using rules

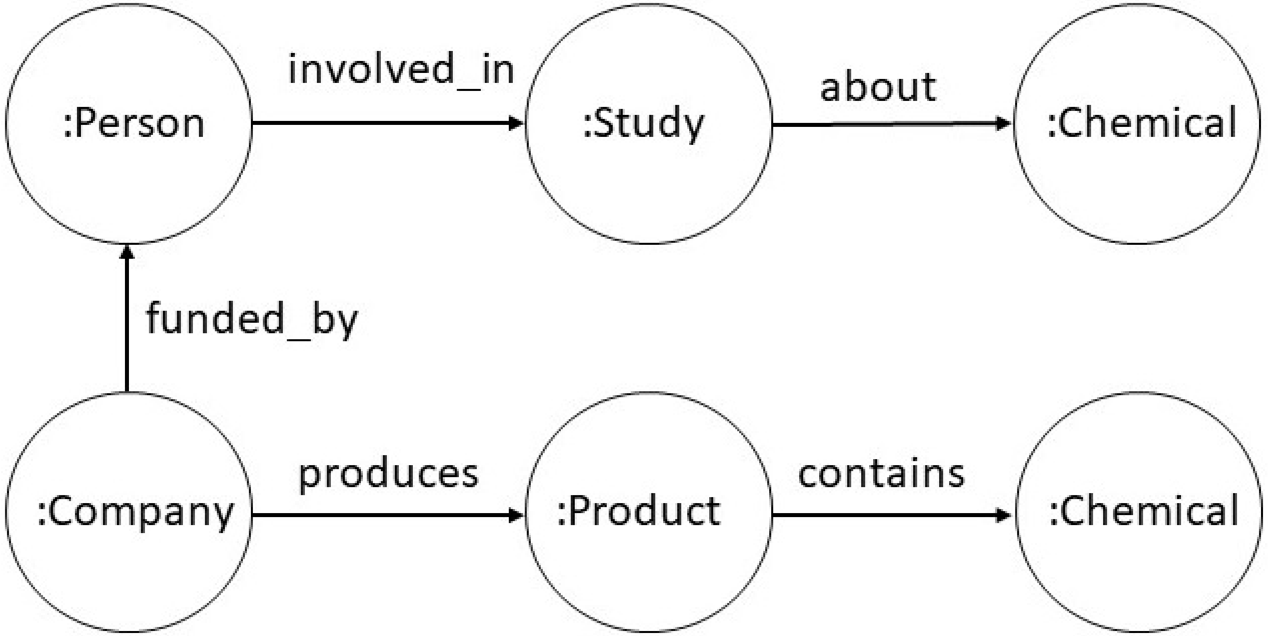
# Rule-based inference

- Example Scenario



# Rule-based inference

- Example Scenario

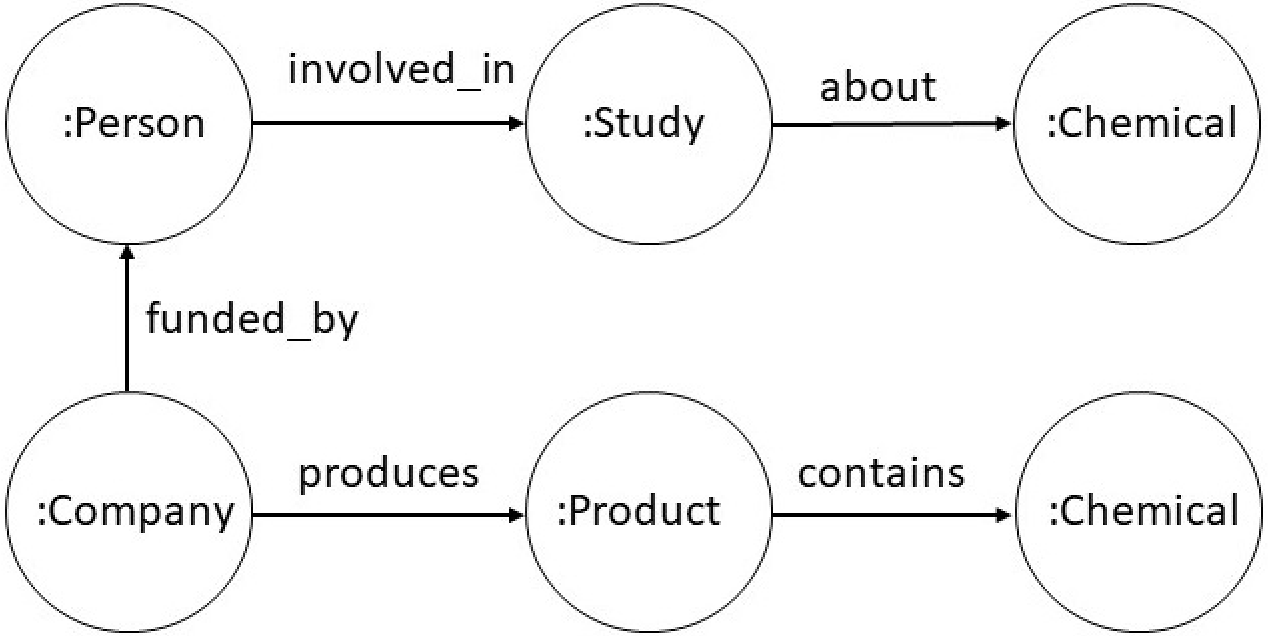



has_interest(X,Z) :- produces(X,Y) & contains(Y,Z)

coi(X,Y,Z) :-
involved_in(X,Y) & about(Y,P) &
funded_by(X,Z) & has_interest(Z,P)

# Rule-based inference

- Example Scenario




has_interest(X,Z) :- produces(X,Y) & contains(Y,Z)

coi(X,Y,Z) :-
involved_in(X,Y) & about(Y,P) &
funded_by(X,Z) & has_interest(Z,P)

∃c conflict_of(c,X) & conflict_reason(c,Y) &
conflict_with(c,Z) :-
involved_in(X,Y) & about(Y,P) &
funded_by(X,Z) & has_interest(Z,P)

# Rule-based Inference

- Approaches
  - Bottom up strategy (also known as Chase)
    - Apply the rules against the data, and add new facts
    - Ensure termination
    - Process queries as usual
  - Top-down strategy
    - Start from the query, and apply rules as needed
    - Requires tighter integration between the rule engine and query evaluation
    - Requires lot less space

Many efficient and scalable rule engines available today

# Summary

- Graph inference algorithms fall into two broad categories
  - Traditional Graph Algorithms
    - Path finding, centrality, community detection
  - Ontology-based algorithms
    - Taxonomic reasoning, Rule-based reasoning