

# Graph Analytics



# Contents

<b>1</b>	<b>Introduzione ai grafi</b>	<b>1</b>
1.1	Introduzione alla Graph Analytics . . . . .	2
1.2	Il problema dei ponti di Königsberg . . . . .	4
1.3	Proprietà dei grafi . . . . .	6
<b>2</b>	<b>Algoritmi di ricerca nei grafi</b>	<b>11</b>
2.1	Connettività di un grafo . . . . .	13
2.2	Graph Structure Analysis . . . . .	17
2.3	Centrality of a node . . . . .	18
2.4	Centrality Measures . . . . .	19
2.5	Community Detection . . . . .	20
2.6	Graph Partitioning . . . . .	20
<b>3</b>	<b>Graph Importance</b>	<b>22</b>
3.1	Importanza dei Nodi nei Grafi . . . . .	22
3.2	Random Walk . . . . .	22
3.3	Co-Citation and Bibliographic Coupling . . . . .	25
3.4	Hypertext Induced Topic Search (HITS) . . . . .	27
3.5	PageRank . . . . .	30
<b>4</b>	<b>Graph Databases</b>	<b>34</b>
4.1	Tipi di Databases . . . . .	37
4.2	Graph Landscape . . . . .	39
<b>5</b>	<b>Graph Modeling</b>	<b>41</b>
5.1	Capire il problema . . . . .	41
5.2	Creare un modello concettuale . . . . .	42
5.3	Costruire un graph data model . . . . .	42
<b>6</b>	<b>Machine learning e grafi</b>	<b>48</b>
6.1	Data: the true challenge . . . . .	48
6.2	Issues . . . . .	48
6.3	Performance . . . . .	48
6.4	Benefit of using graphs . . . . .	50
6.5	Ruolo dei grafi nel Machine Learning . . . . .	53
<b>7</b>	<b>Graph in Data Engineering</b>	<b>57</b>
7.1	Big Data lifecycle . . . . .	57
7.2	Lambda architecture . . . . .	59
7.3	Grafi per versioni fine-grained . . . . .	60
7.4	Vantaggi dei grafi . . . . .	61
7.5	Grafi come Master Data Management (MDM) . . . . .	61
<b>8</b>	<b>Recommendations con i Grafi</b>	<b>64</b>
8.1	Collaborative Filtering . . . . .	66
8.2	Fraud Detection . . . . .	68
8.3	Graphs for NLP . . . . .	70

<b>9 Knowledge Graphs</b>	<b>72</b>
9.1 Creating Knowledge Graphs . . . . .	73
9.2 Knowledge Graphs and Important Nodes . . . . .	75
9.3 Semantic Similarity . . . . .	75
9.4 Knowledge Graph Refinement . . . . .	76



## 1 Introduzione ai grafi

Un **grafo** è una struttura matematica utilizzata per rappresentare relazioni tra oggetti. È costituito da due elementi fondamentali: i **nodi** (o *vertici*) e gli **archi** (o *spigoli*).

- I **nodi** rappresentano gli oggetti o le entità. Possono essere, ad esempio, persone in una rete sociale, città in una mappa, oppure pagine web collegate tra loro.
- Gli **archi** rappresentano le connessioni o relazioni tra i nodi. Un arco collega due nodi e può essere:
  - **Orientato**, se ha una direzione (rappresentato graficamente come una freccia);
  - **Non orientato**, se non ha direzione (rappresentato come una semplice linea).

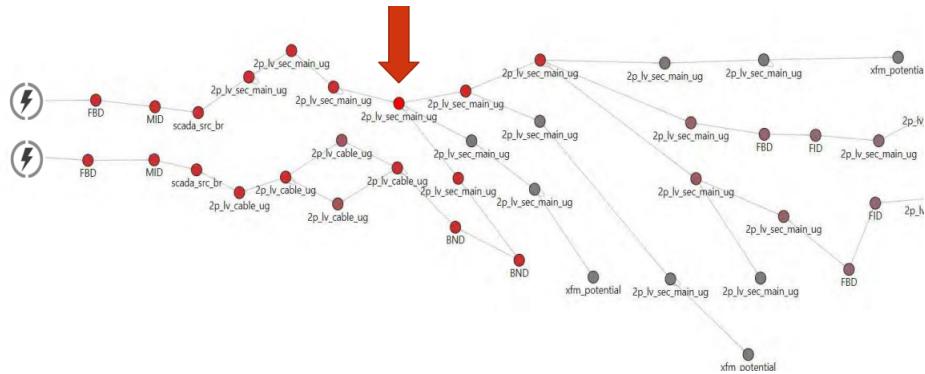
Ad esempio, in una rete di amici su un social network:

- Ogni persona è un nodo;
- Un'amicizia tra due persone è un arco.

I grafi sono strumenti molto potenti e vengono utilizzati in numerosi ambiti, come l'informatica, la logistica, la sociologia e l'ingegneria.

### L'importanza dei nodi

I nodi sono una parte fondamentale della struttura. Ed esistono nodi più importanti di altri, come ad esempio quelli che collegano tra loro più percorsi:



Immaginiamo che questa sia una rete di trasporti. Il nodo puntato dalla freccia fa da raccordo tra le due linee. Se succede qualcosa in quel nodo, allora ne risente tutto il sistema.

## 1.1 Introduzione alla Graph Analytics

La **Graph Analytics**, o analisi dei grafi, è un insieme di tecniche e algoritmi utilizzati per studiare e analizzare strutture a grafo, con l'obiettivo di estrarre informazioni significative dalle relazioni tra entità.

I grafi sono una rappresentazione naturale per molti sistemi complessi del mondo reale: reti sociali, infrastrutture di trasporto, sistemi informatici, reti biologiche, strutture dati nei motori di ricerca, blockchain e molto altro. L'analisi di queste strutture può rivelare pattern, tendenze, anomalie e proprietà globali o locali del sistema.

### Obiettivi dell'analisi dei grafi

L'analisi dei grafi mira a rispondere a domande come:

- Quali sono i nodi più importanti o influenti in una rete?
- Quali comunità o gruppi di nodi sono strettamente connessi tra loro?
- Esistono pattern o percorsi ricorrenti all'interno del grafo?
- Ci sono anomalie o comportamenti sospetti nella rete?

### Tecniche comuni

Tra le tecniche più comuni nella Graph Analytics troviamo:

- **Centralità:** misura l'importanza relativa di un nodo (es. grado, betweenness, closeness, PageRank);
- **Community detection:** individua gruppi di nodi fortemente connessi (es. algoritmo di Louvain, modularità);
- **Shortest path:** calcola i percorsi minimi tra coppie di nodi (es. algoritmo di Dijkstra, Bellman-Ford);
- **Link prediction:** predice la probabilità che si formi un nuovo arco tra due nodi;
- **Anomaly detection:** rileva nodi o connessioni insolite rispetto alla struttura generale del grafo.

### Applicazioni reali

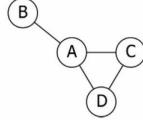
La Graph Analytics trova applicazione in numerosi settori:

- **Social network:** analisi dell'influenza, identificazione di comunità, suggerimenti di amicizie;
- **Cybersecurity:** rilevamento di attacchi o intrusioni in una rete di computer;
- **Biologia computazionale:** studio delle interazioni tra proteine o geni;
- **E-commerce:** raccomandazioni di prodotti basate sulla rete di acquisti e preferenze;
- **Finanza:** rilevamento di frodi analizzando i flussi di denaro come un grafo.

## Grafi matematici

Il grafo è una coppia  $G = (V, E)$ , dove  $V$  è una collezione di vertici  $V = \{V_i, i = 1, \dots, n\}$  e  $E$  una collezione di archi (edges) fatta in questo modo  $E_{ij} = \{(V_i, V_j)\}, V_i \in V, V_j \in V$ . Come disegnare un grafo:

Vertices: Points. Edges: Lines.



Vertices:  $A, B, C, D$   
Edges:  $(A, B), (A, C), (A, D), (C, D)$

### Vertici

*proprietà.* Un vertice può avere una label o essere unlabelled. Il numero dei vertici del grafo è anche detto **ordine del grafo**.

### Archi

*proprietà.* Un edge collega tra loro due vertici. I due vertici connessi prendono il nome di endpoints. Per definizione, se un edge esiste, allora esistono anche i suoi endpoints e sono 2. Se il grafo è diretto, allora esiste un outgoing edge (edge che esce) e un incoming edge (edge che entra). Il numero di edge stabilisce la **size** del grafo.

## 1.2 Il problema dei ponti di Königsberg

Il problema dei **ponti di Königsberg** è considerato uno dei problemi fondamentali che ha dato origine alla **teoria dei grafi**. Esso fu affrontato e risolto da **Leonhard Euler** nel 1736.

### Descrizione del problema

La città di Königsberg (oggi Kaliningrad) era attraversata dal fiume Pregel, che formava due isole collegate tra loro e con le sponde del fiume tramite **sette ponti**. Il problema era il seguente:

È possibile fare una passeggiata che attraversi ciascuno dei sette ponti **una sola volta** e che inizi e termini nello stesso punto?

### Modellazione tramite grafi

Euler semplificò la situazione rappresentandola con un **grafo**:

- Ogni **porzione di terra** (le due isole e le due sponde del fiume) viene rappresentata come un **nodo**.
- Ogni **ponte** viene rappresentato come un **arco** che collega due nodi.

In questo modo, il problema si traduce nel cercare un **ciclo euleriano**, ovvero un percorso chiuso che attraversi ogni arco del grafo **una sola volta**.

### Soluzione di Euler

Euler dimostrò che:

- Un grafo possiede un **ciclo euleriano** se e solo se è connesso e **tutti i nodi hanno grado pari**.
- Un grafo possiede un **cammino euleriano** (che non torna al punto di partenza) se e solo se ha **esattamente due nodi di grado dispari**.

Nel caso del grafo dei ponti di Königsberg, **tutti e quattro i nodi** hanno un **numero dispari di archi** (cioè grado dispari), quindi:

- **Non esiste** un ciclo euleriano;
- **Non esiste** nemmeno un cammino euleriano.

### Conclusione

Di conseguenza, non è possibile attraversare ciascun ponte una sola volta tornando al punto di partenza, né farlo iniziando e terminando in due punti diversi. Il problema dei ponti di Königsberg rappresenta quindi il primo esempio storico dell'applicazione della teoria dei grafi e ha segnato la nascita di questo importante campo della matematica.

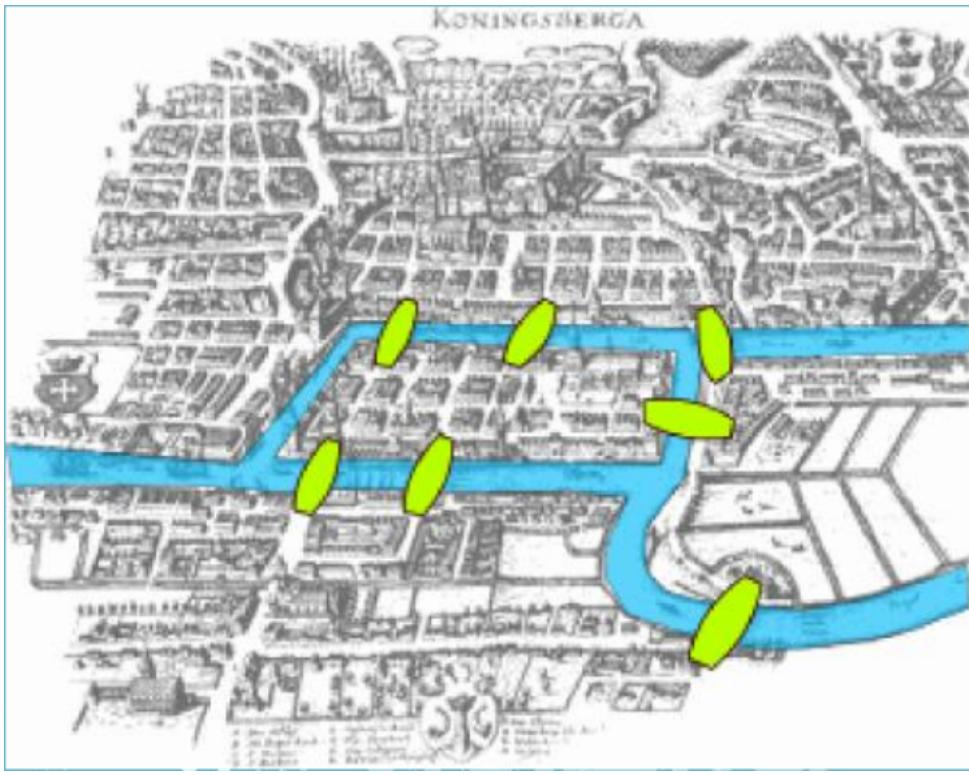
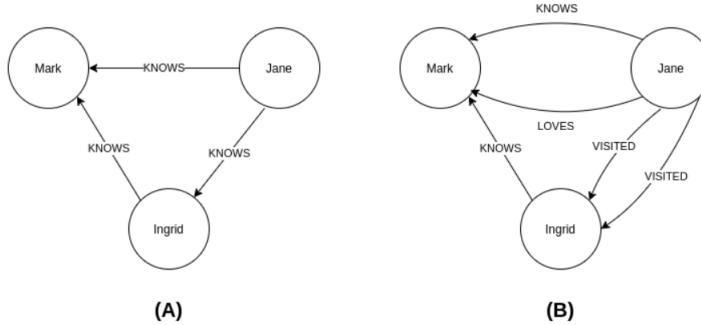


Figure 1: Immagine dei ponti di Koningsberg.

### 1.3 Proprietà dei grafi

#### Grafi semplici e multigrafo

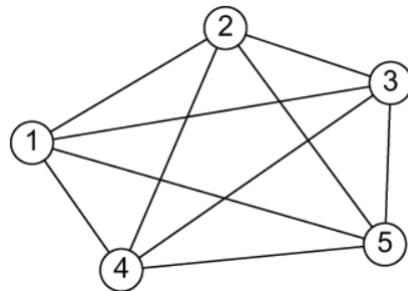
Un grafo semplice (A) permette solo relazioni semplici, ovvero una singola interaction tra due nodi. Se dovessi avere la necessità di usare più relazioni tra due nodi allora si parla di multigrafo (B).



#### Grafi completi

Due vertici si dicono **adiacenti** se sono connessi da un edge. Due edges invece si dicono **incidenti** se connettono lo stesso nodo ad altri nodi.

Posso quindi dire che se tutti vertici di un grafo sono a coppie adiacenti, il grafo è completo. In un grafo completo, tutti i nodi sono connessi tra loro.



#### Indegree e Outdegree

In un grafo diretto, il grado di un vertice V è diviso nell'**indegree** che consiste nel numero di edge per cui V è end node e **outdegree** che è il numero di edge per cui è start node.

Una delle proprietà più importanti di un vertice è il suo grado: il numero di edge incidenti a tal vertice, che coincide anche con il numero dei nodi a lui adiacenti.

#### Average degree

Il grado medio è definito come segue:

$$a = \frac{1}{N} \sum_{i=1, \dots, N} \text{degree}(V_i)$$

Dove N è il numero di vertici nel grafo.

## Percorsi

Una sequenza di vertici consecutivi connessi da degli edges si chiama **path** (percorso). Un path dove non ci sono vertici ripetuti è un **path semplice**. Un path dove il primo e l'ultimo vertice coincidono si chiama **cycle**.

Un grafo che ha almeno un edge ha necessariamente un path. Se i vertici sono connessi con degli edge diretti (ovvero il grafo è diretto) possiamo chiamarlo **directed path**. Esistono diverse tipologie di grafi con diversi percorsi:

- **weighted graphs:** sono dei grafi dove il passaggio da un vertice ad un altro ha un costo che è un valore numerico, che può essere utilizzato ad esempio in una rete di trasporti.
- **monopartite graphs:** grafi dove tutti i nodi sono della stessa classe (es: persone, animali)
- **multipartite graphs:** grafi dove abbiamo più di una classe. Un grafo che ha esattamente due classi è chiamato bipartite graph.

## La matrice di adiacenza

Una delle modalità più comuni per rappresentare un grafo in forma computazionale è la **matrice di adiacenza**. Si tratta di una matrice quadrata che descrive la connessione tra i nodi del grafo.

**Definizione** Data un grafo  $G = (V, E)$  con  $n = |V|$  nodi, la sua **matrice di adiacenza**  $A$  è una matrice  $n \times n$  tale che:

$$A_{ij} = \begin{cases} 1 & \text{se esiste un arco da } v_i \text{ a } v_j, \\ 0 & \text{altrimenti.} \end{cases}$$

Nel caso di grafi **non orientati**, la matrice di adiacenza è **simmetrica**, cioè  $A_{ij} = A_{ji}$ .

Nel caso di grafi **orientati**, la direzione degli archi è importante, quindi in generale  $A_{ij} \neq A_{ji}$ .

**Esempio** Consideriamo un grafo non orientato con tre nodi  $V = \{v_1, v_2, v_3\}$  e archi  $E = \{(v_1, v_2), (v_2, v_3)\}$ . La sua matrice di adiacenza è:

$$A = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

## Proprietà

- La riga  $i$  della matrice rappresenta tutti i nodi a cui è connesso il nodo  $v_i$ .
- Il numero di 1 nella riga (o colonna)  $i$  corrisponde al **grado** del nodo  $v_i$  (in grafi non orientati).
- Per grafi pesati, la matrice di adiacenza può contenere i **pesi** degli archi anziché solo 0 e 1.

## Vantaggi e svantaggi

- **Vantaggi:** accesso immediato all'informazione su una connessione tra due nodi; utile per grafi densi.
- **Svantaggi:** utilizza molta memoria per grafi sparsi (*sparse*), dove la maggior parte delle celle sono zeri.

## Graph density

La densità è un parametro che ci aiuta a stabilire quanto un grafo sia " pieno" tenendo conto del numero di edges che possiede e in relazione al numero dei suoi vertici. La densità di un grafo D ha questa forma  $D(V,E)$ ; questo valore oscilla tra 0 (grafo perfettamente sparse) e 1 (grafo completo, perfettamente denso). Per un grafo indiretto è:

$$D_U(V, E) = \frac{|E|}{Max_U(V)} = \frac{2 \times |E|}{|V| \times (|V|-1)}$$

$Max(V)$  è il massimo numero di edges. E dipende dall'ordine ma non dalla size del grafo. E possiamo definire questo valore in relazione all'ordine:

$$Max_U(V) = \frac{|V|(|V|-1)}{2}$$

La sua definizione possiamo estenderla ai grafi diretti:

$$Max_U(V) = |V|(|V|-1) = 2 \times M_U(V)$$

Se il grafo è diretto, il numero massimo di edges è il doppio del caso non diretto. Di conseguenza la densità nel caso diretto sarà la metà del caso non diretto.

**Sparse vs Dense** Un grafo sparse è un grafo dove la densità D è nel range più basso possibile di codominio ( $D$  compreso tra 0 e  $\frac{1}{2}$ ). Analogamente, un grafo denso ha un range del codominio più ampio ( $D$  compreso tra  $\frac{1}{2}$  e 1). Se  $D = \frac{1}{2}$  non viene considerato da nessuno dei due tipi.

## Condizioni generali

- la densità di un grafo di ordine 1 è indefinita, per ovvie ragioni
- tutti i grafi vuoti hanno densità 0
- tutti i grafi completi anno densità 1
- un grafo non diretto ha una densità di almeno  $\frac{2}{|V|}$  quindi è garantito che sia denso per  $|V| < 4$
- un grafo diretto tracciabile non è sempre detto che sia denso

**Efficiency in Graph Storage** La densità di un grafo influisce sull'efficienza dello storage del grafo in memoria. I grafi sono strutture molto complesse e devono essere salvate in maniera efficiente. Esistono quindi due tipi di rappresentazioni. lista di edges, o adjacency matrix. Come regola generale, se il grafo è sparso, usa lista di edges, altrimenti usa la adjacency matrix (per i più densi).

## Walk, Trail e Path

- walk: un walk è un cammino generico che possiamo percorrere nel nostro grafo.
- trail: un trail è un walk che non ripassa per lo stesso edge più di una volta. Se è un percorso chiuso, prende il nome di circuit.
- path: un path è un walk che non ripassa due volte per lo stesso vertice.

## Grafi euleriani, non euleriani e semi-euleriani

Nella teoria dei grafi, un problema classico consiste nel determinare se esiste un percorso che attraversi ogni arco del grafo una sola volta. Questo porta alla definizione di grafi **euleriani**, **semi-euleriani** e **non euleriani**, concetti introdotti da Leonhard Euler.

### Grafo euleriano

Un **grafo euleriano** è un grafo in cui esiste un **ciclo euleriano**, cioè un ciclo che:

- attraversa **ogni arco esattamente una volta**;
- **inizia e termina nello stesso nodo**.

**Condizione necessaria e sufficiente (non orientato):** Un grafo non orientato è euleriano se e solo se:

- il grafo è **connesso** (eccetto i nodi isolati);
- **tutti i nodi hanno grado pari**.

**Esempio:** Un ciclo chiuso in cui ogni nodo ha due archi è un grafo euleriano.

### Grafo semi-euleriano

Un **grafo semi-euleriano** è un grafo in cui esiste un **cammino euleriano**, cioè un percorso che:

- attraversa **ogni arco esattamente una volta**;
- **inizia e termina in nodi diversi**.

**Condizione necessaria e sufficiente (non orientato):** Un grafo è semi-euleriano se e solo se:

- è connesso;
- **esattamente due nodi hanno grado dispari**.

**Esempio:** Un grafo a forma di catena con due estremi di grado dispari è semi-euleriano.

### Grafo non euleriano

Un **grafo non euleriano** è un grafo che **non possiede né un ciclo euleriano né un cammino euleriano**. Questo accade quando:

- il grafo non è connesso (escludendo nodi isolati), oppure
- contiene **più di due nodi con grado dispari**.

**Esempio:** Il grafo dei *ponti di Königsberg* è un classico esempio di grafo non euleriano: ha quattro nodi con grado dispari, quindi non è possibile attraversare ogni ponte una sola volta, né con partenza e arrivo nello stesso punto, né in punti diversi.

### Nota sui grafi orientati

Nei grafi **orientati**, le condizioni per l'esistenza di un cammino o ciclo euleriano sono leggermente diverse e si basano su:

- **Grado entrante** (*in-degree*) e **grado uscente** (*out-degree*) dei nodi;
- **Forte connessione** del grafo.
- Un grafo orientato ha un ciclo euleriano se è fortemente connesso e ogni nodo ha in-degree uguale a out-degree.
- Ha un cammino euleriano (semi-euleriano) se è connesso e:
  - Un nodo ha out-degree = in-degree + 1 (inizio del cammino),
  - Un nodo ha in-degree = out-degree + 1 (fine del cammino),
  - Tutti gli altri nodi hanno in-degree = out-degree.

## 2 Algoritmi di ricerca nei grafi

### DFS e BFS

#### Depth First Search (DFS)

Il *Depth First Search* (DFS) è un algoritmo di visita dei grafi che esplora il grafo andando il più in profondità possibile lungo ciascun ramo prima di fare *backtracking*.

- Funziona sia su grafi orientati che non orientati.
- Può essere implementato in modo ricorsivo o iterativo (usando uno **stack**).
- È utile per:
  - Rilevare cicli
  - Trovare componenti connesse
  - Ordinamento topologico (nei grafi aciclici orientati)

#### Algoritmo (versione ricorsiva):

```
DFS(G, u):  
    visited[u] = true  
    for each v in adjacency_list[u]:  
        if not visited[v]:  
            DFS(G, v)
```

#### Complessità:

- Tempo:  $\mathcal{O}(V + E)$
- Spazio:  $\mathcal{O}(V)$  per la memoria usata dai nodi visitati (più lo stack della ricorsione)

#### Breadth First Search (BFS)

Il *Breadth First Search* (BFS) è un algoritmo di visita dei grafi che esplora tutti i nodi vicini prima di procedere a quelli più lontani. Utilizza una **queue** (coda FIFO).

- Visita i nodi per livelli (distanza minima in termini di archi dal nodo sorgente).
- Può essere usato per:
  - Trovare il cammino minimo in grafi non pesati
  - Rilevare componenti connesse

---

### Algoritmo:

```
BFS(G, s):
    queue = [s]
    visited[s] = true
    while queue is not empty:
        u = queue.pop()
        for each v in adjacency_list[u]:
            if not visited[v]:
                visited[v] = true
                queue.append(v)
```

### Complessità:

- Tempo:  $\mathcal{O}(V + E)$
- Spazio:  $\mathcal{O}(V)$  per la memoria dei nodi visitati e la coda

## Shortest Path

Il concetto più importante nella scelta di un path è sicuramente la scelta del path più veloce, più comodo, di fatto più corto. Questo prenderà il nome di **shortest path**. Esso cambia in base al tipo di grafo:

- se il grafo non è pesato, basta trovare il numero minore di edges da percorrere.
- se il grafo è pesato, basta trovare il percorso (path) con il peso totale minore.

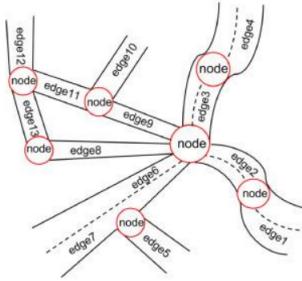
Ma quali sono i possibili percorsi? Dipende dalla richiesta:

- parto da un solo Vertice (nodo) → dato un nodo di partenza trova il percorso più veloce per connetterlo a tutti gli altri.
- data una partenza, trova la strada più veloce per la destinazione. Fondamentalmente è il discorso che fa Google Maps. **Se esiste il percorso.**
- trova tutti gli shortest path che collegano tra loro tutti i diversi nodi del grafo.

Il BFS come metodo di ricerca, è sicuramente il più complesso.

## Applicazioni

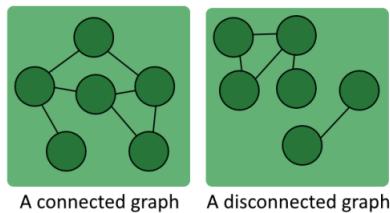
Sicuramente la ricerca del cammino più breve trova facile impiego nel campo della ricerca stradale. Infatti, è semplice pensare di attribuire ad ogni incrocio un nodo e le strade sono gli edges, come in figura qui sotto:



Potrebbe essere anche utile attribuire ad ogni edge un peso che può significare il tempo di percorrenza della strada, la lunghezza, ecc... Usare edge diretti può essere utile per rappresentare le strade a senso unico. Esiste un algoritmo chiamato Hub Labeling capace di calcolare lo shortest path di tutte le strade in EU o in USA in frazioni di secondo.

### 2.1 Connettività di un grafo

La connettività in un grafo è la qualità delle connessioni tra i suoi vertici. Si fa distinzione tra grafo connesso e disconnesso:



### Max-Flow Min-Cut Theorem

Sia  $G = (V, E)$  un grafo orientato con:

- una sorgente  $s \in V$ ,
- un pozzo (sink)  $t \in V$ ,
- una funzione di capacità  $c : E \rightarrow \mathbb{R}^+$ .

Un **flusso**  $f : E \rightarrow \mathbb{R}$  deve rispettare:

1. **Vincolo di capacità:**  $0 \leq f(u, v) \leq c(u, v)$
2. **Conservazione del flusso:** per ogni nodo  $v \neq s, t$ ,

$$\sum_{u \in V} f(u, v) = \sum_{w \in V} f(v, w)$$

**Il valore del flusso** è definito come:

$$|f| = \sum_{v \in V} f(s, v)$$

**Taglio**  $(S, T)$ : una partizione dei nodi  $V$  tale che  $s \in S$ ,  $t \in T$ , e  $S \cup T = V$ ,  $S \cap T = \emptyset$ .

La **capacità del taglio** è:

$$c(S, T) = \sum_{\substack{u \in S, v \in T \\ (u, v) \in E}} c(u, v)$$

**Teorema:** Il massimo flusso da  $s$  a  $t$  è uguale alla capacità minima di un taglio che separa  $s$  da  $t$ :

$$\max_f |f| = \min_{(S, T)} c(S, T)$$

**Conseguenze:**

- Ogni flusso massimo attraversa un taglio minimo.
- Il taglio minimo fornisce un limite superiore al flusso.
- Quando l'algoritmo trova un flusso massimo, ha anche trovato un taglio minimo.

Parlando per un grafo, questo significa cercare il minimo numero di edges che se rimossi, rendono il grafo disconnesso. Il flusso inteso qui può essere qualunque cosa, ad esempio anche un flusso di dati che passa attraverso un computer network come internet.

### Eccentricity

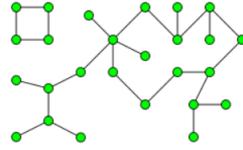
L'eccentricity di un nodo è la distanza massima da percorrere per raggiungere il nodo **più lontano** partendo da quello attuale. Tramite questa grandezza è possibile definire:

- raggio di un grafo: l'eccentricità minima di ogni nodo nel grafo.
- diametro di un grafo: la massima eccentricità di ogni nodo nel grafo.

### Connected components

Il componenti di un grafo sono tutte le partizioni separate dal grafo completo. Ovvero tutte le parti del grafo a sé stanti. Se un grafo ha un'unica componente, si dice che è un grafo **CONNESSO** altrimenti si dirà **DISCONNESSO** e si conteranno le partizioni chiamandole componenti.

### Disconnected w/ 2 components

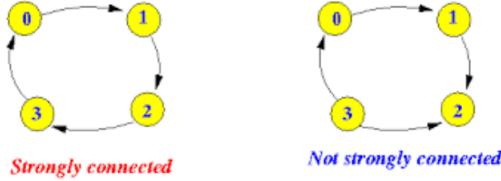


### Connected w/ 1 component



### Strongly connected components

Uno **strongly connected component** è una porzione di un grafo diretto dove c'è un path diretto tra tutti i nodi.



### Conduttanza nei Grafi

La **conduttanza** (*conductance*) è una misura che quantifica quanto bene un sottografo è separato dal resto del grafo. È molto usata nell'analisi di reti e nel clustering di grafi.

**Definizione:** Dato un grafo non orientato  $G = (V, E)$ , e un sottoinsieme non vuoto di nodi  $S \subset V$ , la conduttanza di  $S$  è definita come:

$$\phi(S) = \frac{|\partial S|}{\min(\text{vol}(S), \text{vol}(V \setminus S))}$$

dove:

- $\partial S$  è l'insieme degli archi che collegano un nodo in  $S$  a un nodo in  $V \setminus S$  (il bordo del taglio),
- $|\partial S|$  è il numero di tali archi,
- $\text{vol}(S) = \sum_{v \in S} \deg(v)$  è il volume di  $S$ , cioè la somma dei gradi dei nodi in  $S$ .

In un grafo non orientato bisogna considerare entrambe le direzioni di un edge.

### Interpretazione:

- Una bassa conduttanza  $\phi(S)$  indica che il sottoinsieme  $S$  è *ben separato* dal resto del grafo: pochi archi lo collegano al resto rispetto alla sua dimensione.
- Un valore elevato di  $\phi(S)$  indica invece che  $S$  è *fortemente connesso* al resto del grafo.

**Conduttanza del grafo:** La conduttanza del grafo è definita come la conduttanza del taglio migliore:

$$\phi(G) = \min_{\substack{S \subseteq V \\ 0 < \text{vol}(S) \leq \text{vol}(V)/2}} \phi(S)$$

Banalmente, data la partizione di un grafo in due set disgiunti  $G_1$  e  $G_2$ , la conduttanza consiste nella somma degli edges che vanno da  $G_1$  a  $G_2$  diviso il minimo tra gli edges che vanno da  $G_1$  a  $G_2$ .

### Applicazioni:

- Clustering e rilevamento di comunità in grafi
- Algoritmi di partizionamento dei grafi
- Analisi della connettività di reti complesse

### Clique nei Grafi

Sia  $G = (V, E)$  un grafo non orientato.

Un **clique** è un sottoinsieme di vertici  $C \subseteq V$  tale che ogni coppia distinta di vertici in  $C$  è connessa da un arco in  $E$ , ovvero:

$$\forall u, v \in C, \quad u \neq v \Rightarrow (u, v) \in E$$

### Tipi di clique:

- **Clique massimale:** un clique che non può essere esteso includendo un altro nodo adiacente (cioè non è contenuto in nessun altro clique più grande).
- **Clique massimo:** un clique con il massimo numero possibile di nodi nel grafo.
- **k-clique:** un clique composto da esattamente  $k$  nodi.

**Esempio:** Nel grafo seguente (non rappresentato), se i nodi  $\{1, 2, 3\}$  sono tutti connessi tra loro, allora formano un 3-clique.

### Applicazioni:

- Analisi di reti sociali (gruppi di amici tutti connessi tra loro)
- Bioinformatica (moduli funzionali nelle reti genetiche)
- Ottimizzazione e intelligenza artificiale

**Complessità computazionale:** Il problema di trovare un clique massimo è **NP-completo**. Trovare tutti i cliques massimali è computazionalmente costoso per grafi grandi, ma esistono algoritmi efficienti per casi particolari (es. algoritmo di Bron–Kerbosch). [Fonte GPT]

### K-Core Decomposition

Graph algorithm che serve per cercare il piú grande sottografo dove tutti i nodi hanno almeno grado k. Il sottografo massimo prende il nome di **k-core** del grafo.

L'algoritmo iterativamente rimuove i nodi con un grado minore di k e tutti i nodi incidenti ad essi finché non rimangono solo nodi con degree almeno uguale a k.

## 2.2 Graph Structure Analysis

Nell'analisi delle strutture dei grafi, é utile evidenziare due componenti principali:

- **Average Diameter L:** misura media dello shortest path che collega tra loro due nodi qualsiasi.  
Misura quanto i nodi siano "lontani" tra loro.

$$L = \frac{1}{N(N-1)} \sum_{i \neq j} d(i,j)$$

Dove N é il numero di nodes e d(i,j) shortest path tra il nodo i e j.

- **Clustering Coefficient C:** media della local density. È equivalente alla densitá del sottografo quando si considerano solo i vicini di n, escludendo n. Questo misura quanto i suoi vicini siano tendenti ad essere un clique (sottografo completo) e quindi misura quanto due nodi vicini al nodo target possano essere connessi.

**Directed:**

$$C_n = \frac{|E|}{|V|(|V|-1)}$$

**Undirected:**

$$C_n = \frac{2|E|}{|V|(|V|-1)}$$

**Clustering Coefficient of the entire graph:**

$$C = \frac{1}{|N|} \sum_{n \in N} C_n$$

### Wiener Index: closeness of a graph

Il wiener index é una misura della complessitá topologica definita come la somma a coppie degli shortest paths tra i nodi del grafo.

### 2.3 Centrality of a node

È la misura dell'importanza di un nodo all'interno di un grafo. La banale intuizione è che più un nodo è fondamentale nel mantenere un grafo connesso, più il nodo è importante. Vengono utilizzate diverse metriche per indicare la **centrality** di un nodo:

- **Degree centrality**: numero dei vicini del nodo v
- **Closeness centrality**: reciproco della distanza tra un nodo qualsiasi v e tutti gli altri nodi:

$$C_c(v) = \frac{1}{\sum_{u \in V} \delta(u, v)}$$

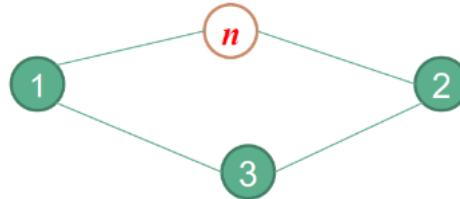
Dove il termine in delta è la distanza tra il nodo u e v.

- **Betweenness centrality [0-1]**: rapporto tra il numero di shortest paths passanti per il nodo v e tutti gli shortest paths del grafo:

$$C_B(v) = \sum_{s \neq t \neq v \in V} \frac{\sigma_{st}(v)}{\sigma_{st}}$$

Dove sigma indica il numero di shortest paths (in particolare quello di v indica quelli passanti per v)

**Esempio.** Il nodo di cui valutare l'importanza è n. Grafo indiretto, 4 nodi totali. I parametri:



- **Degree centrality**=2
- **Closeness**  $C_C(n) = \frac{1}{1+1+2}$  distanza di n dagli altri nodi. Risultato: 0.25
- **Betweenness**  $C_B$ : Per questo punto, è necessario scrivere tutti gli shortest paths: Casi totali:

CASO	Shortest Path
P[1,n]	(1, n)
P[2, n]	(2, n)
P[3, 1]	(3, 1)
P[3, 2]	(3, 2)
P[1, 2]	(1, 3, 2), (1, n, 2)
P[3, n]	(3, 2, n), (3, 1, n)

8. Quanti contengono n? 5. Quindi il ratio risulta  $\frac{5}{8}$ .

## 2.4 Centrality Measures

Il concetto di "importanza" all'interno di una rete può essere interpretato in molti modi diversi, e per questo esistono varie misure di centralità che permettono di quantificare tale importanza in base alla struttura del grafo e al ruolo che ogni nodo ricopre nel contesto della rete. Tra le più comuni troviamo:

- **PageRank:** Questa misura di centralità è stata originariamente sviluppata da Google per classificare le pagine web. L'algoritmo assegna un punteggio a ciascun nodo del grafo basandosi sia sul numero di archi (link o connessioni) che entrano nel nodo, sia sulla qualità dei nodi da cui provengono tali connessioni. In sostanza, un nodo riceve un punteggio elevato se è collegato da altri nodi che sono anch'essi importanti. Questo tipo di centralità è particolarmente utile in contesti in cui la popolarità o l'autorevolezza gioca un ruolo fondamentale.
- **Centralità di intermediazione (Betweenness Centrality):** Questo tipo di misura valuta quanto un nodo sia "strategico" nella rete, ovvero quanto spesso si trova sui cammini minimi (shortest paths) tra coppie di altri nodi. Un nodo con alta centralità di intermediazione agisce come ponte tra diverse parti del grafo e può quindi avere un ruolo critico nel facilitare (o controllare) il flusso di informazioni. È utile per individuare nodi influenti nel collegare comunità o sottoreti diverse.

Queste misure sono fondamentali in numerose applicazioni di analisi dei grafi, come la rilevazione di influencer nei social network, l'identificazione di nodi critici in reti infrastrutturali, la valutazione dell'impatto di un'entità all'interno di un ecosistema complesso o l'ottimizzazione della diffusione di informazioni.

**TextRank** L'algoritmo TextRank, introdotto da Mihalcea e Tarau nel 2004, è un metodo di classificazione basato su grafi utilizzato per l'elaborazione del linguaggio naturale, in particolare per compiti come l'estrazione automatica di parole chiave e la sintesi automatica. L'applicazione dell'algoritmo su testi scritti prevede i seguenti passaggi:

1. **Identificazione delle unità testuali rilevanti:** Vengono individuate le unità testuali pertinenti al compito specifico (ad esempio, parole o frasi significative) e aggiunte al grafo come nodi.
2. **Identificazione delle relazioni tra le unità testuali:** Si determinano le relazioni che collegano le unità testuali. Gli archi del grafo possono essere diretti o non diretti, e pesati o non pesati, a seconda del tipo di relazione e della sua intensità.
3. **Esecuzione iterativa dell'algoritmo di ranking:** L'algoritmo di ordinamento basato su grafo viene applicato in modo iterativo fino al raggiungimento della convergenza (ovvero, quando i punteggi dei nodi si stabilizzano) oppure fino al raggiungimento del numero massimo di iterazioni.
4. **Ordinamento e selezione:** I nodi vengono ordinati in base ai punteggi finali ottenuti, e tali punteggi vengono utilizzati per effettuare decisioni di ordinamento o selezione. A questo punto è possibile anche unire due o più unità testuali in un'unica parola chiave o frase chiave.

Questo approccio consente di estrarre in modo efficace informazioni rilevanti da un testo, sfruttando la struttura delle relazioni tra le parole piuttosto che un'analisi puramente statistica.

## 2.5 Community Detection

Una community all'interno di un grafo corrisponde ad una porzione interna con un'alta connettività spesso chiamata anche **cluster**. Questi gruppi si riconoscono perché hanno una bassa connettività con gli altri nodi del grafo, ma un'alta connettività tra loro.

Per fare community detection ovvero trovare le community all'interno di un grafo, è necessario utilizzare l'algoritmo di **Girvan-Newman**:

1. **Calcolare l'edge betweenness:** trova gli edge più centrali all'interno del grafo.
2. **Elimina:** elimina gli edges con alta betweenness.
3. **Trova:** tutti i componenti connessi rimanenti sono communities. Processo che può essere iterato.

Vengono cercati gli edges più importanti, e vengono prunati. In questo modo, avanzano quelli un po' meno rilevanti, non abbastanza da diventare i più centrali. Ma quelli che comunque hanno tante connessioni: quindi sono edge che connettono tanti nodi insieme in quelle che chiamiamo communities.

## 2.6 Graph Partitioning

Nei paragrafi precedenti è stato introdotto il concetto di "dividere" il grafo in più parti, tagliando degli archi.

**Definizione** L'idea è quella di mantenere le parti in cui dividiamo densamente connesse e di usare il minor numero di tagli possibile. **Graph Cut** è proprio ricercare di avere dei grafi distinti dopo il taglio. Il nostro cut quindi dovrà minimizzare la conduttanza:

$$C(G_1, G_2) = \frac{|\mathbb{C}|}{\min\{\mathbb{C} + |G_1|, (\mathbb{C} + G_2)\}}$$

Dove  $\mathbb{C}$  è il Cut:

$$|Cut(G_1, G_2)|$$

## Proximity Prestige nei Grafi

Il **Proximity Prestige** è una misura di centralità usata per valutare il prestigio di un nodo all'interno di un grafo diretto, tenendo conto sia della quantità di nodi che possono raggiungerlo, sia della distanza con cui lo fanno. È un'estensione del concetto di *prestige* (cioè di essere destinatario di connessioni), che integra l'idea di *prossimità*.

**Definizione** Dato un grafo diretto  $G = (V, E)$ , il **Proximity Prestige** di un nodo  $v \in V$  si definisce come:

$$\text{ProxPrestige}(v) = \frac{|\text{R}(v)|}{n-1} \cdot \frac{1}{\frac{1}{|\text{R}(v)|} \sum_{u \in \text{R}(v)} d(u, v)}$$

dove:

- $\text{R}(v)$  è l'insieme dei nodi che possono raggiungere  $v$  tramite un cammino diretto (cioè i predecessori di  $v$ ) anche chiamato **Influence Domain**,

- $d(u, v)$  è la lunghezza del cammino più breve da  $u$  a  $v$ ,
- $n$  è il numero totale di nodi nel grafo.

La formula può essere letta come:

$$\text{ProxPrestige}(v) = \text{Coverage}(v) \cdot \frac{1}{\text{Average Distance to } v}$$

### Interpretazione

- Il primo termine ( $\frac{|\mathcal{R}(v)|}{n-1}$ ) misura la *copertura*, ossia la percentuale di nodi che possono raggiungere  $v$ .
- Il secondo termine valuta l'*efficienza con cui* tali nodi raggiungono  $v$ , ovvero quanto in media sono vicini.

Un nodo ha **alto proximity prestige** se è raggiunto da molti altri nodi e questi lo raggiungono con cammini brevi.

**Applicazioni** Questa misura è utile in reti sociali, reti di citazioni e sistemi di comunicazione, per identificare gli attori più influenti non solo in base alla quantità di connessioni in entrata, ma anche alla rapidità con cui l'informazione può arrivare a loro.

## 3 Graph Importance

### 3.1 Importanza dei Nodi nei Grafi

In molte applicazioni che coinvolgono grafi, come le reti sociali, i motori di ricerca, i sistemi di raccomandazione e le reti di trasporto, è fondamentale determinare quanto un nodo sia *importante* o *centrale* all'interno della struttura della rete. Esistono diversi approcci per quantificare tale importanza, e uno dei più efficaci è basato sul concetto di **passegiata casuale** (*random walk*) su un grafo.

In questa sezione, esploreremo le principali tecniche per valutare l'importanza dei nodi utilizzando strumenti probabilistici, concentrando in particolare su:

- il concetto di **random walk** su grafi diretti e non diretti;
- la **probabilità di transizione** tra nodi, che rappresenta la probabilità che un cammino casuale si sposti da un nodo all'altro;
- le misure di importanza derivate da questi concetti, come il *PageRank*.

Questi strumenti permettono di modellare dinamiche realistiche nei grafi e di identificare i nodi che, per posizione o connettività, svolgono un ruolo cruciale nella diffusione dell'informazione, nell'accessibilità o nel controllo della rete.

### 3.2 Random Walk

É una tecnica che consiste nel percorrere il grafo **casualmente**. Partendo da un nodo qualsiasi, cominciamo a scorrere il grafo (se diretto, nella direzione corretta) e ci fermiamo quando vogliamo, contando però le ripetizioni di tutti i nodi (ovvero quante volte incontriamo ogni nodo nel nostro cammino).

#### Transition Probability in a Graph

Dato un nodo  $N_i$ , qual é la probabilitá che percorrendo K passi casuali si arrivi ad un nodo  $N_j$ ? Chiamiamo questa misura **transition probability**.

Potremmo anche decidere di fare una pausa, fermarci per saltare ad un altro nodo: dovremo quindi aggiungere alla transition probability la **teleport probability**, ovvero la probabilitá di fare un salto. La somma delle due probabilitá deve andare a 1:

$$P_{\text{teleport}} + P_{\text{transition}} = 1$$

Normalmente la teleport probability viene indicata con  $\alpha$  ed é un parametro del random walk.

## Markov Model

Un **Modello di Markov** per grafi è una rappresentazione matematica del movimento stocastico (random walk) su un grafo, in cui la probabilità di passare da un nodo ad un altro dipende unicamente dallo stato corrente e non dalla sequenza di stati precedenti. Tale proprietà è nota come *Markov property*.

**Definizione formale** Sia  $G = (V, E)$  un grafo orientato, dove:

- $V = \{v_1, v_2, \dots, v_n\}$  è l'insieme dei nodi (o stati),
- $E \subseteq V \times V$  è l'insieme degli archi (o transizioni).

Si definisce una **matrice di transizione**  $P \in \mathbb{R}^{n \times n}$  tale che:

$$P_{ij} = \mathbb{P}(X_{t+1} = v_j \mid X_t = v_i)$$

cioè la probabilità di muoversi dal nodo  $v_i$  al nodo  $v_j$  in un singolo passo.

La matrice  $P$  è **stocastica per righe** (o per colonne, a seconda della convenzione): ogni riga (in questo caso) deve sommare a 1:

$$\sum_{j=1}^n P_{ij} = 1, \quad \forall i \in \{1, \dots, n\}$$

**Evoluzione temporale** Dato un vettore di distribuzione iniziale  $\pi^{(0)} \in \mathbb{R}^n$ , che rappresenta la distribuzione di probabilità iniziale sui nodi, l'evoluzione del sistema è data da:

$$\pi^{(t+1)} = \pi^{(t)} P$$

dove  $\pi^{(t)}$  rappresenta la distribuzione di probabilità sui nodi al tempo  $t$ .

**Teleportation e PageRank** In molte applicazioni reali (ad esempio nel *PageRank* di Google), si introduce una probabilità di **teleportation**  $\alpha \in [0, 1]$  per garantire che il modello sia ergodico (cioè che esista una distribuzione stazionaria unica).

La matrice di transizione diventa:

$$P' = \alpha P + (1 - \alpha) \cdot \frac{1}{n} \mathbf{1} \mathbf{1}^T$$

dove:

- $\alpha$  è il parametro di damping (solitamente  $\alpha \approx 0.85$ ),
- $\frac{1}{n} \mathbf{1} \mathbf{1}^T$  è la matrice di teleportation, che assegna probabilità uniforme a tutti i nodi.

**Distribuzione stazionaria** Una distribuzione  $\pi^*$  è detta **stazionaria** se:

$$\pi^* = \pi^* P$$

cioè, se applicando la matrice di transizione, la distribuzione resta invariata. In modelli ergodici, esiste una distribuzione stazionaria unica che rappresenta il comportamento a lungo termine del processo di Markov.

Data la matrice di transizione  $T$  e considerando la teleport probability  $\alpha=0$ , dato il vettore iniziale  $v$  di delle probabilitá di ogni nodo: uno step dal tempo  $t_i$  al tempo  $t_{i+1}$  corrisponde alla moltiplicazione  $T^T \times v_i$  dove  $T^T$  é la matrice trasposta e  $v$  é normalizzato sulle colonne:

	1	2	3	4
1	0	½	½	0
2	0	0	0	1
3	0	½	0	½
4	1	0	0	0

**T**

	<i>p</i>
1	¼
2	¼
3	¼
4	¼

**v<sub>o</sub>**

	1	2	3	4
1	0	0	0	1
2	½	0	½	0
3	½	0	0	0
4	0	1	½	0

**T<sup>T</sup>**

	<i>p</i>
1	¼
2	¼
3	¼
4	¼

×

	<i>p</i>
1	¼
2	¼
3	¼
4	¼ + ½

**v<sub>t</sub>**

Considerando invece  $\alpha \neq 0$ , e ripetendo lo stesso ragionamento, consideriamo uno step temporale dal tempo  $t_i$  al tempo  $t_{i+1}$  come la moltiplicazione:

$$(1 - \alpha)T^T \times v_i + \alpha \times v_0$$

$$(1 - \alpha) \cdot \begin{bmatrix} a_{11} & \dots & a_{1n} \\ \vdots & & \vdots \\ a_{n1} & \dots & a_{nn} \end{bmatrix}^T \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix}_{t_i} + \alpha \cdot \begin{bmatrix} \frac{1}{n} \\ \vdots \\ \frac{1}{n} \end{bmatrix} = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix}_{t_{i+1}}$$

E ci riferiamo a questo processo con il nome di **Power Iteration**.

### 3.3 Co-Citation and Bibliographic Coupling

Co-citation é uno degli ambiti dove si puó vedere l'applicazione del concetto di grafo. Indica la tecnica con cui si collegano ("linkano") due documenti. Quando un paper cita un altro, essi hanno una relazione, che puó essere studiata in **citation analysis**. Il grafo che connette piú documenti tra loro é un grafo diretto, aciclico:

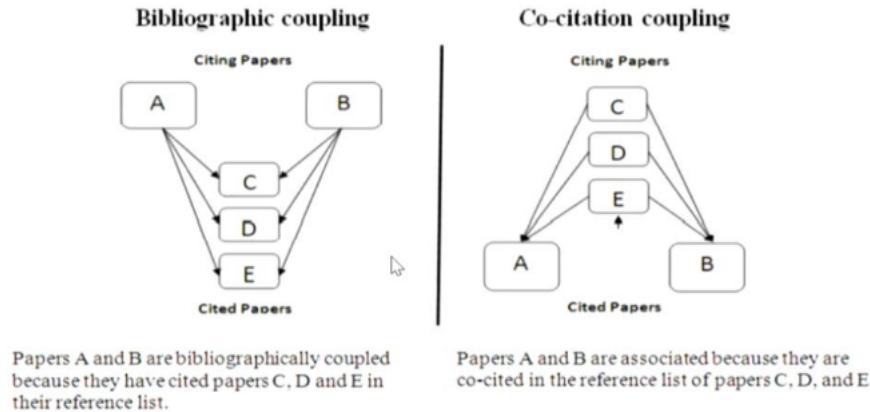


Figure-1: Bibliographic coupling and co-citation coupling

In un **citation network** ogni nodo corrisponde ad un documento e gli archi sono le connessioni (citations). É quindi possibile dedurre:

- se il nodo  $A$  e il nodo  $B$  sono connessi, e il nodo  $A$  e il nodo  $C$  sono connessi, é **probabile** che il nodo  $B$  e il nodo  $C$  siano connessi
- piú connessioni hanno in comune due nodi, piú quei nodi saranno simili, piú sarà forte la loro **relazione**

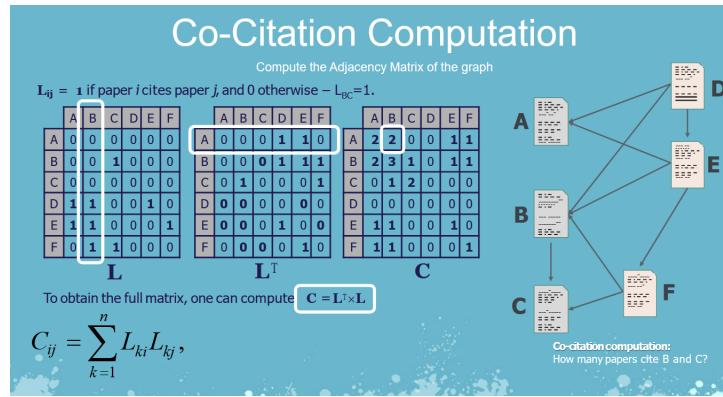
**Esempio** Costruiamo la matrice di adiacenza per un caso generale di co-citation. Come in un grafo non pesato indichiamo con 1 se é presente un arco, 0 se non é presente e chiamiamo la matrice  $L_{ij}$ .

Possiamo definire **Co-Citation**:

$$C_{ij} = \sum_{k=1}^n L_{ki} L_{kj}$$

come una misura di **similarity** basata sul numero di papers che co-citano  $i$  e  $j$ . Si puó inoltre generare una matrice  $\mathbb{C}$  quadrata sulla base di  $C_{ij}$  chiamata **co-citation matrix** che indica sulla diagonale principale quante volte un articolo é stato citato dagli altri.

$$C_{ij} = L^T \times L$$



### Bibliographic Coupling

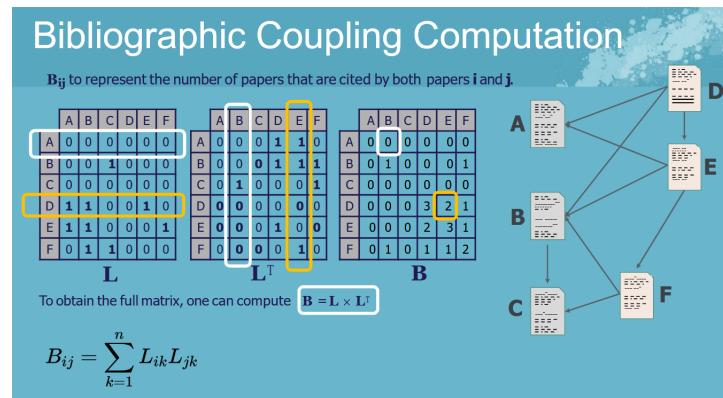
Si definisce in questo modo il caso in cui due diversi papers si riferiscono allo stesso paper nella loro bibliografia. Questo spesso indica che i due lavori siano in qualche modo correlati.

**Bibliographic Coupling** é simile a co-citation. Utilizziamo sempre una matrice e la chiamiamo  $B_{ij}$  definita come:

$$B_{ij} = \sum_{k=1}^n L_{ik}L_{jk}$$

Che rappresenti il numero di papers che sono citati sia da  $i$  che da  $j$ . Il termine  $B_{ii}$  é il numero di references che ha il documento  $i$ . Questa matrice viene chiamata **bibliographic coupling matrix** e si puó ottenere in maniera analoga a come viene ricavata la co-citation matrix:

$$B_{ij} = L \times L^T$$



La co-citazione tra articoli scientifici può essere utilizzata per molteplici finalità nel contesto dell’analisi della letteratura accademica. Può aiutare a **identificare ricerche correlate**, ovvero trovare articoli che esplorano tematiche o domande di ricerca simili. Inoltre, consente di **mappare i campi di ricerca**, visualizzando la struttura di un dominio scientifico e osservando come si connettono tra loro i diversi sottocampi. Può anche essere impiegata per **tracciare l’evoluzione scientifica**, studiando come si formano, evolvono o cambiano le comunità di ricerca e come si trasformano concetti, teorie o metodologie. Questo approccio permette di **trovare potenziali collaboratori**, identificando ricercatori che lavorano su problemi affini e aprendo opportunità di

collaborazione. Inoltre, contribuisce a **migliorare la scoperta della letteratura**, rivelando articoli fondamentali che arricchiscono il processo di revisione con intuizioni più ampie e approfondite. È utile per **identificare comunità di ricerca**, portando alla luce cluster di lavori connessi, e per **mappare l'influenza scientifica**, scoprendo i contributi fondativi di un campo. Infine, consente di **costruire reti di citazioni**, visualizzando la struttura di un'area di studio e identificando gruppi densi di articoli correlati.

### 3.4 Hypertext Induced Topic Search (HITS)

HITS è un algoritmo per link analysis creato nel 1999 da Jon Kleinberg come metodo alternativo al PageRank per il ranking di pagine web. L'algoritmo introduce 2 aspetti fondamentali:

- **Authority and Hub Scores:** HITS assegna ad ogni page due score, **authority score** e **hub score**. Il primo misura il valore della page in generale, il secondo misura il tipo di link che ha con le altre pagine
- **Mutual Reinforcement Relationship:** L'intuizione centrale dell'algoritmo HITS è il rafforzamento reciproco tra hub e autorità: alta autorità sono pagine che ricevono link da molte hub buone, buone hub sono pagine che puntano a molte autorità forti

HITS funziona sulla base di una query. L'utente immette una query di ricerca e HITS prima cerca con un motore, poi produce due liste: una per l'authority e una per l'hub.

Cosa sono effettivamente una pagina authority e una hub? Indichiamo con **hub** una pagina che ha tanti **out-links**, ovvero che è una pagina usata come organizer che linka a molte authorities, mentre un'**authority** è una pagina che ha molti **in-links** ovvero molte altre pagine linkano ad essa (discorso che non c'entra assolutamente nulla con i paper/documenti usati nel co-citation o bibliographic coupling).

La relazione principale che intercorre tra i due dev'essere:

- una buona authorities deve essere puntata da tante buone hub
- una buona hub deve puntare a tante buone authorities

Attenzione, ogni nodo ha due score quindi è ripetuto due volte in memoria (costo elevato). Procediamo iterativamente:

1. al tempo  $t_0$  abbiamo due score per ogni nodo assegnato (generico,  $\frac{1}{\sqrt{N}}$ )
2. al tempo  $t_1$  accumuliamo gli score di ogni authority da ogni hub e viceversa

Ripetendo questo processo ad ogni step, aggiornando ogni volta entrambi gli score.

**Convergenza** Nel contesto dell'algoritmo HITS (Hyperlink-Induced Topic Search), dire che converge a un punto stabile significa che, dopo un certo numero di iterazioni, i punteggi di hub e di autorità assegnati a ciascuna pagina non cambiano più in modo significativo: raggiungono un valore stabile (o oscillano attorno a un valore molto vicino).

#### Perché succede?

HITS funziona aggiornando i punteggi delle pagine in modo iterativo:

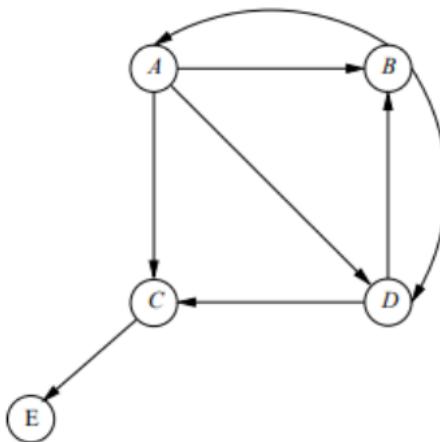
- Punteggio di autorità di una pagina = somma dei punteggi di hub delle pagine che la linkano.
- Punteggio di hub di una pagina = somma dei punteggi di autorità delle pagine a cui essa punta.

Dopo ogni aggiornamento, si normalizzano i punteggi (per evitare che crescano all'infinito) e si ripete il processo.

Questo processo è simile a moltiplicare ripetutamente un vettore per una matrice, e sotto certe condizioni (come la connettività del grafo), questa sequenza di aggiornamenti converge all'autovettore principale della matrice corrispondente. Alla fine, i punteggi smettono di cambiare in modo rilevante: si dice che l'algoritmo è convergente ad un punto. In pratica:

- Dopo poche iterazioni (tipicamente meno di 100), i punteggi si stabilizzano.
- I valori stabili rappresentano quanto una pagina è un buon hub o una buona autorità nel contesto del grafo di collegamenti.

**Esercizio** Calcola hub score e authority score del grafo seguente.



Facciamo le giuste assunzioni per risolvere correttamente l'esercizio:

- N web pages (numero di nodi nel grafo, = 5)
- vettori  $\mathbf{a} = (a_1, \dots, a_n)$  authority score e  $\mathbf{h} = (h_1, \dots, h_n)$  hub score
- $\mathbb{A}$  adjacency matrix (matrice di adiacenza chiamata link matrix of the web)  $N \times N$ , grafo non pesato quindi sulle righe e colonne ha 1 se il link c'è, 0 se il link non c'è

Attenzione, essendo che:

$$\mathbf{h}_i = \sum_{i \rightarrow j} a_j$$

Che vorrebbe dire che hub score della pagina  $i$  è la somma dei punteggi di authority delle pagine a cui essa punta, ma quindi riscrivendolo usando  $\mathbb{A}$ :

$$\mathbf{h}_i = \sum_{i \rightarrow j} a_j = \sum_{j=1}^N \mathbb{A}_{ij} a_j = \mathbb{A} \cdot \mathbf{a}$$

Discorso uguale identico per l'authority score, ovviamente invertito:

$$\mathbf{a}_i = \sum_{j \rightarrow i} h_j$$

Perché per definizione l'authority score sarebbe la somma dello score degli hub che puntano a questa authority:

$$\mathbf{a}_i = \sum_{j \rightarrow i} h_j = \sum_{j=1}^N \mathbb{A}_{ji} h_j = \mathbb{A}^T \cdot h$$

Quindi alla fine otteniamo che:

$$L = \begin{bmatrix} 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad L^T = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

Matrice dei link del web: partendo da uno score  $a, h = \frac{1}{\sqrt{N}} = \frac{1}{\sqrt{5}}$  si può calcolare lo score dello step successivo, fino ad arrivare a:

$$\mathbf{h} = \begin{bmatrix} 1 \\ 0.3583 \\ 0 \\ 0.7165 \\ 0 \end{bmatrix} \quad \mathbf{a} = \begin{bmatrix} 0.2087 \\ 1 \\ 1 \\ 0.7913 \\ 0 \end{bmatrix}$$

### 3.5 PageRank

**PageRank** é un algoritmo che misura l'importanza dei nodi di un grafo dal tipo di struttura che li collega. Creato originariamente dai fondatori di Google Larry Page e Sergey Brin nel 1996-1997 a Stanford, ha rivoluzionato il mondo del web con una categorizzazione accurata delle pagine.

**Basic concept** Una pagina é importante se riceve link da altre pagine importanti. Questo crea una definizione **ricorsiva** dove l'importanza di una pagina viene definita da quanto sono importanti le pagine che puntano ad essa.

**Modello matematico** Usa una struttura a grafo dove le pagine sono i nodi, i link tra le pagine sono archi e l'**importanza** di ogni pagina é distribuita su questi links.

Il link dalla pagina  $i$  alla pagina  $j$  é un **voto** da  $i$  a  $j$ ; i voti da pagine con tanti voti **contano di più**. In altre parole, ogni voto di ogni pagina é **proporzionale** all'importanza della pagina che lo emette. É quindi importante definire un **rank**:

$$\mathbf{r}_j = \sum_{i \rightarrow j} \frac{\mathbf{r}_i}{d_i}$$

Dove  $d_i$  é **out-degree** del nodo  $i$  ovvero il numero di archi uscenti dall'i-esimo nodo. Questo valore al denominatore serve a diluire il rank di quella pagina.

**Diluizione** Il valore di out-degree serve per distribuire equamente il pagerank della pagina  $i$  tra tutte le pagine a cui si collega. Questo significa che se  $\mathbf{r}_i$  é 1, e la pagina  $i$  é connessa ad altre 4, ad ognuna di esse arriverá  $\frac{1}{4}$ .

**Quindi out-degree diminuisce il valore di di rank della pagina  $i$ ?** No. Lo diluisce tra le i suoi collegamenti, ma se la pagina  $i$  ha un rank molto alto (dato dai rank delle pagine collegate a lei (in input) il valore  $d_i$  non cambierá di molto il suo rank. Serve solo a far capire che se avesse un solo link, darebbe a quel nodo 1, mentre avendone 4 lo distribuisce.

**Matrix Formulation** Definiamo una matrice stocastica sulle colonne (somma dei valori sulle colonne 1)  $M_{ij}$ , i cui valori sono 0 se due nodi non sono connessi,  $\frac{1}{d_i}$  se i nodi sono connessi e definiamo inoltre un rank vector  $r$  che ha un valore per ogni pagina (somma di tutti i suoi valori =1). L'**equazione del flusso** puó essere scritta in questo modo:

$$r = M \cdot r$$

Applicando il principio della **power iteration** all'equazione del flusso:

$$r^{t+1} = M \cdot r^t$$

Che si deve fermare quando:

$$|r^{t+1} - r^t| > \varepsilon$$

Questo modello é equivalente ad un random walk con nessun teleport, quindi  $\alpha = 0$ . Il risultato é una **distribuzione stazionaria**: il ranking finale delle pagine deriva dalla distribuzione stazionaria della catena di Markov che modella il comportamento di un utente che naviga il web seguendo solo i link. La distribuzione stazionaria é  $r$ , mentre la matrice di transizione é  $M$  (tornando alla pagina 22 é possibile notare l'analogia con la definizione di distribuzione stazionaria).

**Stationary distribution** Data una matrice di transizione, e seguendo la power iteration, arrivo sempre ad una fine dei calcoli?

La risposta é **dipende**. Per grafi che soddisfano determinate condizioni eventualmente la raggiungono ed é unica; inoltre é indipendente dalla distribuzione di probabilitá iniziale al tempo 0. Esistono convergenze problematiche:

- **Dead-end**: nodo che non ha out-links, il random surfer si bloccherá lì facendo perdere informazione al next step (rimane fermo)
- **Spider Trap**: nodo con un ciclo ma senza uscita. Un loop, il random walk continuerá a camminare senza mai raggiungere la stabilitá

Per avere una convergenza e risultare una distribuzione stazionaria, é quindi necessario:

- la matrice di transizione é stocastica sulle colonne,
- la matrice é **irriducibile**: il grafo é fortemente connesso, andata e ritorno per ogni coppia di nodi,
- la matrice é **aperiodica**: é possibile tornare ad ogni nodo con un numero finito di passi

Come posso quindi far sì che il modello segua le condizioni di Markov? Aggiungendo un parametro che mi collega tra loro tutti i nodi: **damping factor**, o comunemente chiamato teleport, é la soluzione adottata da Google. Ad ogni step, si puó seguire un random link oppure saltare in un altro casuale evitando così spidertraps e deadends. L'equazione del PageRank risulta:

$$r_j = \sum_{i \rightarrow j} \beta \frac{r_i}{d_i} + (1 - \beta) \frac{1}{N}$$

$\beta$  é il damping factor e  $1 - \beta$  é la probabilitá di saltare ad un'altra pagina. La formulazione di Markov assume che non ci siano dead-ends: per funzionare, basta fare sì che se una colonna ha solo 0, si imposti un valore  $\frac{1}{N}$ . Riformulando:

$$r = \mathbb{A} \cdot r$$

dove

$$\mathbb{A}_{ij} = \beta M_{ji} + \frac{1 - \beta}{N}$$

Quindi:

$$\begin{aligned} r_j &= \sum_{i=1}^N \mathbb{A}_{ij} r_i = \sum_{i=1}^N [\beta M_{ji} + \frac{1 - \beta}{N}] \cdot r_i \\ r_j &= \sum_{i=1}^N \beta M_{ji} \cdot r_i + \sum_{i=1}^N \frac{1 - \beta}{N} \cdot r_i \end{aligned}$$

Ma nel secondo termine,  $\frac{1 - \beta}{N}$  é una costante e  $\sum r_i = 1$  per definizione (é una distribuzione di probabilitá che rappresenta la probabilitá che un surfer randomico si trovi in una pagina  $i$  in un determinato momento, quindi la somma di tutte deve necessariamente fare 1) allora otteniamo:

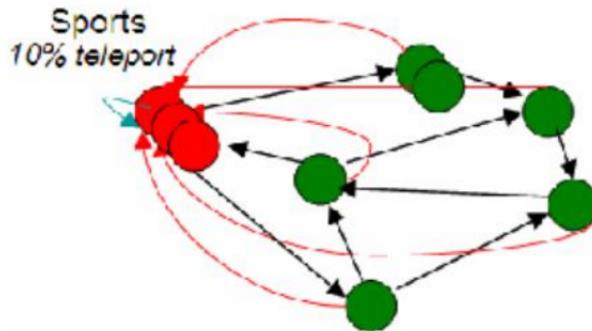
$$r_j = \sum_{i=1}^N \beta M_{ji} \cdot r_i + \frac{1 - \beta}{N}$$

### PageRank personalizzato: Topic-Specific PageRank

Il PageRank appena descritto misura la popolarità **generica** di una pagina, senza considerare un topic specifico. Per focalizzarsi su un argomento preciso, è necessario modificare la funzione di teleport:

- nel PageRank generico, il teleport ha un valore uguale per ogni altra pagina
- nel **topic-specific** PageRank la pagina è selezionata da un set a probabilità più alta a seconda della query di ricerca
- quando il walker fa teleport, va in una pagina presa dal set  $S$  di pagine specifiche
- per ogni insieme  $S$  (che rappresenta un topic) abbiamo un diverso vettore  $r_S$

**Esempio slide** Immaginiamo di prendere in considerazione un surfer che spende la maggior parte del suo tempo a visitare pagine inerenti allo sport. Immaginiamo ora che si sposti con teleport in una pagina randomica sempre inerente allo sport, a noi non interessa quale infatti non considereremo un ordine ma semplicemente puntiamo a non avere un  $S$  subset vuoto cosicché l'operazione di teleport sarà sempre possibile.



Dato che l'insieme  $S$  delle pagine relative allo sport non è vuoto, ne consegue che esiste un insieme non vuoto di pagine web  $Y \supseteq S$  sul quale la random walk ha una distribuzione stazionaria; indichiamo questa distribuzione di PageRank specifica per lo sport con  $\pi_S$ . Per le pagine web che non appartengono a  $Y$ , assegniamo un valore di PageRank pari a zero. Chiamiamo  $\pi_S$  il PageRank specifico per l'argomento "sport".

**Utilizzo topic-specific PageRank** Il PageRank specifico diventa una misura di similarity, e può trovare diverse applicazioni come ad esempio:

Qual è la probabilità di arrivare al nodo B, partendo dal nodo A?

Per rispondere a questa domanda potremmo passare per i nodi più rilevanti per il nodo A grazie al PageRank specifico. L'esempio più frequente è quello del movie recommendation: sapendo che ti è piaciuto un film (nodo) consigliami un altro (film che fanno parte di un subset  $S$  connesso al nodo attuale).

## Particle Filtering Approach

L'approccio a filtraggio a particelle (*Particle Filtering*) è una tecnica di campionamento stocastico utilizzata per stimare distribuzioni di probabilità in sistemi dinamici. Applicato ai grafi, esso consente di approssimare distribuzioni di PageRank o cammini di navigazione in modo efficiente, anche quando la dimensione del grafo è elevata o quando si desidera un PageRank specifico per un argomento.

Sia  $G = (V, E)$  un grafo diretto, dove  $V$  è l'insieme dei nodi (pagine) e  $E$  è l'insieme degli archi (link).

L'algoritmo si basa sui seguenti passaggi principali:

- **Inizializzazione:** vengono generate  $N$  particelle, ognuna rappresenta una posizione iniziale su un nodo  $v \in V$ , tipicamente campionato da una distribuzione iniziale  $p_0(v)$ , ad esempio una distribuzione uniforme o una distribuzione concentrata su un sottoinsieme  $S \subseteq V$  (per topic-specific PageRank).
- **Propagazione:** ad ogni passo  $t$ , ogni particella si sposta su un vicino del nodo corrente secondo la matrice di transizione  $P$ , definita da:

$$P_{ij} = \beta \cdot \frac{1}{\deg^+(v_i)} + (1 - \beta) \cdot \frac{1}{|V|}$$

dove  $\deg^+(v_i)$  è il numero di link uscenti dal nodo  $v_i$ , e  $\beta \in [0, 1]$  è il parametro di damping.

- **Resampling (opzionale):** per evitare la degenerazione (ossia la concentrazione eccessiva delle particelle in pochi nodi), si può applicare un'operazione di *resampling*, dove le particelle vengono ricampionate in base alla loro importanza o peso.
- **Stima:** dopo un numero sufficiente di iterazioni  $T$ , la distribuzione delle particelle sui nodi del grafo fornisce una stima empirica della distribuzione stazionaria desiderata:

$$\hat{\pi}(v) = \frac{\# \text{ particelle in } v}{N}$$

Questo metodo è particolarmente utile per calcolare il PageRank su grandi grafi o per versioni personalizzate e dinamiche del PageRank (es. topic-specific, temporal, personalized), dove i metodi tradizionali di moltiplicazione matriciale risulterebbero troppo costosi. [fonte GPT]

**Aggiunta slide** Nelle slide viene evidenziato il fatto che si tratti di un metodo che approssima l'importanza dei nodi di un grafo simulando il moto di particelle. Archi differenti hanno rilevanza diversa. Le particelle si posizionano in un nodo qualsiasi all'istante iniziale poi si diffondono seguendo la **edge importance**. La dissipazione assicura che le particelle eventualmente finiscano su nodi importanti.

## 4 Graph Databases

Un graph database è un data storage engine che combina le strutture base di un grafo con vertici e archi con un linguaggio di attraversamento (che finora abbiamo chiamato traversal) per creare un database per immagazzinare e avere rapido accesso a dati fortemente connessi.

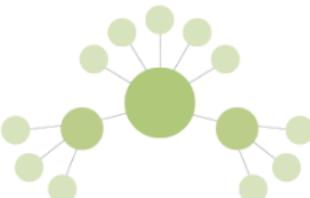
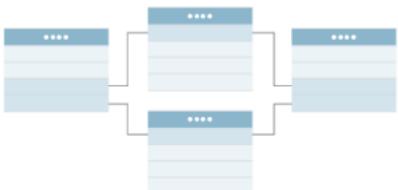
- entities e relationships hanno equal importance
- meno rigido rispetto ai comuni rdbms tabellari
- nessun limite al numero di relazioni che può avere un singolo nodo
- i nodi e gli archi possono avere più caratteristiche che li descrivono

I database relazionali sono poco adatti a rappresentare relazioni complesse. Le relazioni in questi database sono rappresentate da chiavi esterne, che costituiscono dei puntatori a chiavi primarie in altre tabelle.

Questi puntatori non sono entità che possiamo osservare o manipolare facilmente. Infatti, le chiavi esterne vengono seguite (al momento della query) da una riga a un'altra riga.

I database a grafo, invece, forniscono strumenti eccellenti per navigare tra le relazioni presenti nei dati. Attribuendo alle connessioni (gli archi) un'importanza pari a quella degli elementi (i nodi), i database a grafo rappresentano tali associazioni come costrutti a pieno titolo del database, che possono essere osservati e manipolati con facilità.

## Graph database vs. relational database

	Graph database	Relational database
FORMAT	Nodes and edges	Tables with rows and columns
RELATIONSHIPS	Considered data, represented by edges between nodes	Related across tables, established using foreign keys between tables
COMPLEX QUERIES	Run quickly and do not require joins	Require complex joins between tables
TOP USE CASES	Relationship-heavy use cases, including fraud detection and recommendation engines	Transaction-focused use cases, including online transactions and accounting
EXAMPLE		

**Advantages** Rispetto ad un RDBMS, un graph-based ha soluzioni più efficienti in:

- **query ricorsive:** per esempio, in un relational sono eseguite richiamandosi finché non arrivano al risultato desiderato; nei graph sono più semplici da gestire con gli edges.

**Esempio.** Sia data una query di questo tipo:

```
CREATE TABLE org_chart (
    manager_employee_id SM
    ALLINT NULL,
    employee_name
    VARCHAR(20)
    NOT NULL
);
```

Che genera la seguente tabella:

employee_id	Manager_employee_id	employee_name
1	3	You
2	3	Co-worker
3	4	Team Lead
4	5	Manager #2
5	8	VP
6	5	Manager #1
7	5	Manager #3
8	NULL	President/CEO

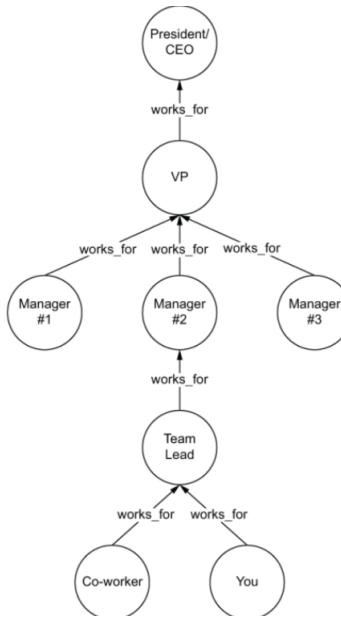
Per costruire una gerarchia aziendale da questa tabella usando una funzione ricorsiva, in SQL:

```
WITH RECURSIVE org AS (
SELECT employee_id, manager_employee_id,
employee_name, 1 AS level
FROM org_chart
UNION
SELECT e.employee_id, e.manager_employee_id,
e.employee_name, m.level + 1 AS level
FROM org_chart AS e
INNER JOIN org AS m
ON e.manager_employee_id = m.employee_id
)
SELECT employee_id, manager_employee_id, employee_name
FROM org
ORDER BY level ASC;
```

Mentre in un graph-based la traversal query:

```
g.V().repeat(
  'works_for')).path().next()
```

Con una struttura a grafo del tipo:



**Importante:** é un esempio. Non é necessario che sia corretto ma é per mostrare l'efficienza.

- **risultati di diverso tipo:** fa riferimento a quei casi in cui vogliamo come risultato dei dati che provengono da tabelle diverse, linkate, ottenuti simultaneamente in una nuova tabella risultato (esempio: lista acquisti insieme ai product information). In un graph based i dati sono contenuti in due nodi diversi, per ottenerli basta scorrere (traversal) il grafo.
- **paths:** questa é una feature esclusiva dei grafi; solo loro possono creare dei percorsi che possono essere utilizzati a nostro vantaggio per ottenere dati rapidamente.

**Potential Issues** I contro dei graph database sono principalmente:

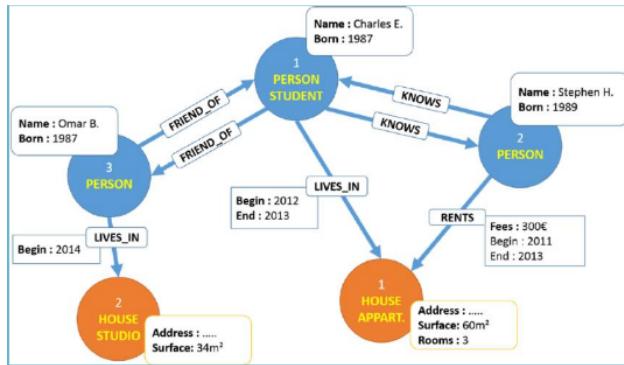
- I database a grafo richiedono implementazioni più rigorose delle misure di **sicurezza e di accesso**. Poiché sono orientati principalmente alla mappatura delle relazioni, questa struttura può anche essere sfruttata in modi che sollevano preoccupazioni in merito alla privacy. Ad esempio, essa può offrire una visione più esplicita di un cliente o utente, e potenzialmente di tutti gli altri clienti o utenti con cui è in qualche modo collegato.
- Per quanto riguarda **l'integrità dei dati**, i database a grafo semplificano le modalità con cui le informazioni si relazionano tra loro. In questo processo, accorciando o condensando il percorso tra le informazioni (in confronto al dover attraversare molteplici tabelle in un database relazionale), diventa particolarmente importante garantire che tutti i dati presenti

nel grafo siano accurati. Una singola relazione mal configurata può condurre direttamente a dati errati, diversamente da un database relazionale dove un dato incorreto potrebbe generare un errore durante una query annidata, rivelando così il problema. Pertanto, l'integrità dei dati assume un'importanza particolarmente elevata quando si utilizzano database a grafo.

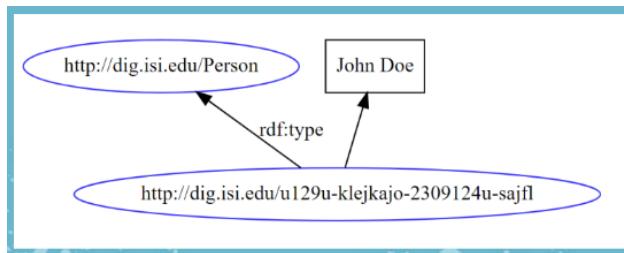
## 4.1 Tipi di Databases

Esistono diversi tipi di graph databases:

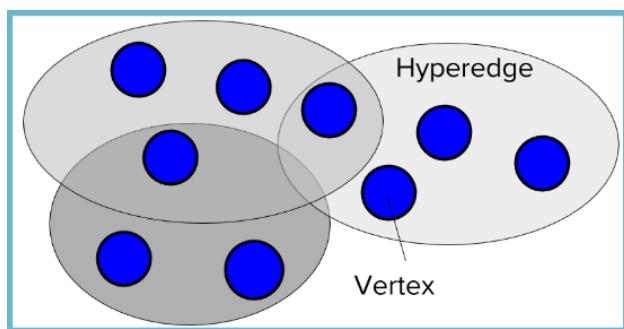
- **Property Graph Databases:** è il più comune, salva dati come nodi e archi e dà la possibilità di assegnare properties ad entrambi.



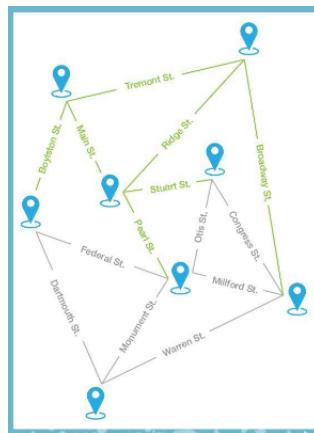
- **Resource Description Framework Graph Database:** usa appunto il resource description model, che rappresenta i dati come triplets formati da **nome**, **predicato** e **oggetto**.



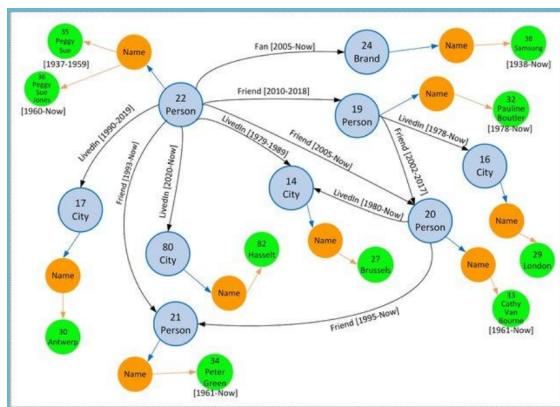
- **Hypergraph Database:** contiene archi e iperarchi, ovvero archi che collegano tra loro più di due nodi, cioè crea **insiemi** di nodi.



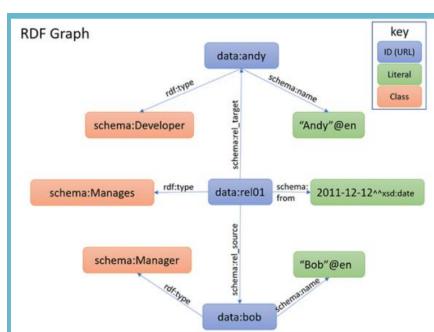
- **Spatial Graph Database:** progettato per rappresentare dati spaziali. Ad ogni nodo corrisponde una location e gli edges rappresentano relazioni tra due punti. Supporta spatial indexing e spatial querying.



- **Temporal Graph Databases:** progettato per rappresentare dati temporali, come time series o event logs. Ad ogni nodo corrisponde una entity e gli edges rappresentano relazioni tra due punti in uno specifico istante.



- **Property-Graph Triplestore Hybrid:** combina property e RDF.



## 4.2 Graph Landscape

Negli ultimi anni sono emersi numerosi progetti e prodotti per la gestione, l'elaborazione e l'analisi dei grafi. L'elevato numero di tecnologie rende difficile tenere traccia di tutti questi strumenti e comprendere in che modo si differenziano.

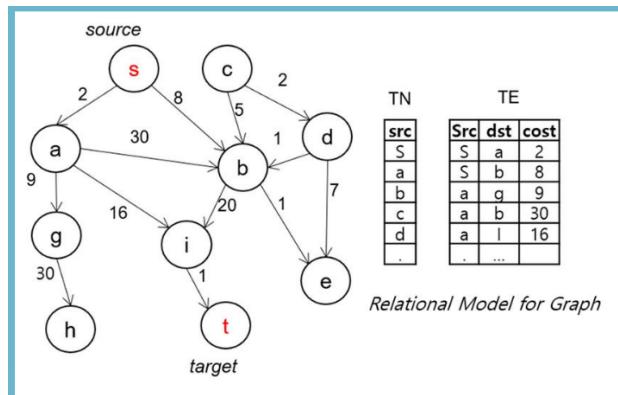
Possiamo suddividere lo spazio delle tecnologie grafiche in due categorie principali:

1. Le tecnologie utilizzate principalmente per la persistenza transazionale dei grafi in tempo reale, generalmente accessibili direttamente durante l'esecuzione. Queste sono chiamate *graph databases*, e sono l'equivalente dei database OLTP.
2. Le tecnologie utilizzate principalmente per l'analisi offline dei grafi, eseguita tipicamente come una serie di passaggi batch. Queste sono chiamate *graph compute engines*, e sono analoghe ai processi di data mining e OLAP.

### Il sistema di archiviazione sottostante

Alcuni database a grafo utilizzano un sistema di archiviazione nativo per i grafi, ottimizzato e progettato specificamente per memorizzare e gestire dati strutturati come grafi.

Tuttavia, non tutte le tecnologie di database a grafo adottano una memorizzazione nativa. Alcune serializzano i dati del grafo all'interno di un database relazionale, orientato agli oggetti o in un altro tipo di archivio dati generico.



### Aderenza senza indice nei database a grafo

Un database a grafo utilizza il concetto di *index-free adjacency*, ovvero l'adiacenza priva di indice, il che significa che i nodi connessi si "puntano" fisicamente l'un l'altro all'interno del database.

Ogni nodo fa riferimento diretto ai suoi nodi adiacenti, mantenendo una sorta di mini-indice locale verso i nodi vicini. Questo approccio rende le operazioni di attraversamento estremamente economiche in termini di prestazioni.

Adottiamo una visione leggermente più ampia: qualsiasi database che, dal punto di vista dell'utente, si comporta come un database a grafo — cioè espone un modello di dati a grafo attraverso operazioni CRUD — può essere considerato un database a grafo.

Riconosciamo tuttavia i significativi vantaggi in termini di prestazioni offerti dall'*index-free adjacency* e, per questo motivo, utilizziamo il termine *native graph processing* per descrivere quei database a grafo che sfruttano tale adiacenza priva di indice.

## Perché scegliere Graph-Based?

- **Prestazioni:** i database a grafo offrono prestazioni superiori per le query su dati connessi, superando i database relazionali e NoSQL, specialmente con l'aumentare della quantità di dati. A differenza dei database relazionali, dove le query ricche di join rallentano con la crescita del dataset, le query nei grafi restano efficienti e localizzate.
- **Flessibilità:** la natura priva di schema dei grafi consente una modellazione dei dati adattiva, permettendo agli sviluppatori di evolvere dinamicamente la struttura senza necessità di complesse migrazioni o definizioni di schema anticipate.
- I database a grafo si allineano allo sviluppo software agile, supportando una crescita iterativa e modifiche fluide. (agility e evolving)
- La loro natura additiva consente di aggiungere relazioni, nodi e etichette senza compromettere le query esistenti.
- A differenza dei database relazionali, la governance viene applicata in modo programmatico attraverso approcci basati su test, garantendo affidabilità anche in applicazioni in continua evoluzione.

## 5 Graph Modeling

Quando si parla di **Data Modeling** si intende il processo per cui una entità reale di qualunque tipo viene trasformata nel suo corrispondente software. La misura in cui otteniamo rappresentazioni software accurate di questi elementi del mondo reale determina l'efficacia con cui affrontiamo il problema previsto. Suddividiamo il problema in 4 steps:

- **Capire il problema:** specificando i termini comuni e i principali modelli di accesso degli utenti ovvero come verranno fatte le query (core access pattern), le informazioni principali, ecc...
- **Creare un modello concettuale:** disegnare un diagramma che descriva in maniera approssimativa il problema da un punto di vista non tecnico e implementativo dal punto di vista degli obiettivi, delle esigenze e delle operazioni dell'organizzazione.
- **Creare un modello logico:** definire i vertici e gli archi con le loro proprietà.
- **Testare il modello:** verificare che il modello creato soddisfi il problema, che esistano le risposte alle nostre domande.

### 5.1 Capire il problema

Vediamo una serie di domande che possono far capire chiaramente il tipo di problema:

- Cosa deve fare il grafo per l'utente? (Dominio e scopo)
- Quali sono i building blocks fondamentali del nostro progetto? Come sono collegati tra loro? (Business entity)
- Come verrá usato il sistema? (Functionality)

**Esempio** Vediamo come applicare queste domande a **DiningByFriends**, applicazione per raccomandazioni gastronomiche.

Domanda 1: Cosa deve fare il grafo per l'utente?

DiningByFriends crea una lista di posti suggeriti dove mangiare personalizzata per l'utente. Deve quindi garantire che:

- ci si possa connettere con amici che usano l'applicazione
- si possa creare recensioni di piatti e ristoranti
- mettere un rating alle recensioni per capire quanto sono inerenti o importanti

Domanda 2: Quali sono i building blocks fondamentali del nostro progetto? Come sono collegati tra loro?

Sicuramente servirá un'autenticazione dell'utente, e i dati di tutti i ristoranti. Servirá tenere le recensioni, i ratings sia dei commenti che dei locali, la lista degli amici ecc...

Domanda 3: Come verrá usato il sistema?

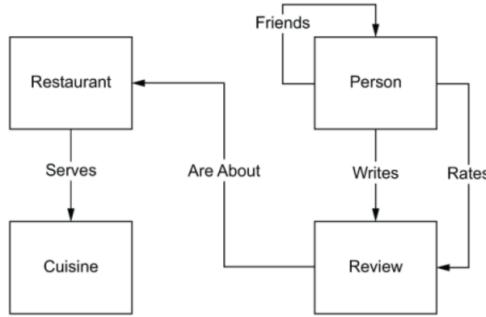
Tante le funzionalitá da tenere a mente, gli amici, i likes, i voti, i ristoranti... E poi fare classifiche, categorie, e molto altro.

## 5.2 Creare un modello concettuale

Uno dei primi step nella creazione del modello concettuale, collegandoci sempre all'esempio precedente e avendo già risposto a tutte le domande della prima fase, sarà quello di definire le entities: sicuramente ristoranti, i menu, le persone e le recensioni. Poi, lo step successivo sarà quello di identificare delle relazioni tra entities:

- relazione 1 potrebbe essere Persona-Amico-Persona
- relazione 2 potrebbe essere Persona-Scrive-Recensione

La relazione tra le entities è ovviamente "amico" nel primo caso e "scrive" nel secondo.



## 5.3 Costruire un graph data model

Per costruire il modello a grafo è necessario seguire questi passaggi:

1. Traduzione delle entità in nodi (vertici)
2. Traduzione delle relazioni in archi (edges)
3. Assegnare le varie proprietà ad archi e vertici
4. Fare un check finale del modello

### Step 1: Traduzione delle entità in nodi

Per la creazione dei vertici del modello abbiamo bisogno di due cose principalmente: **identificare** tutte le entities rilevanti dal conceptual model e **denominare** correttamente l'entity in modo che sia ben distinguibile all'interno del modello.

È best practice fare in modo che ogni label di ogni vertice sia singola poiché ogni vertice si riferisce ad una singola istanza di un elemento.

### Step 2: Traduzione delle relazioni in archi

Gli archi sono basati sulle relazioni trovate nel modello concettuale, ma per essere definite lo sforzo è maggiore rispetto ai vertici. Questo perché gli archi possiedono proprietà di **unicità** e **direzione** che non hanno una diretta controparte nei database relazionali. Le operazioni da svolgere nel caso di archi sono infatti: **identificare** le relazioni rilevanti e **denominarle**; assegnare una **direzione**

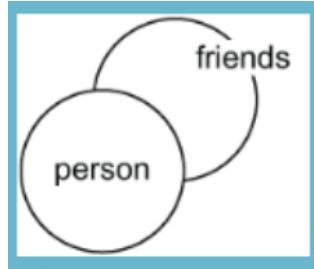
da un vertice ad un altro e **specificare** l'unicità di tale arco definendo il numero di volte che può esistere tra due diversi vertici (praticamente, specifichiamo quante volte questo tipo di relazione esiste).

**Esempio** Tornando a DiningByFriends, le domande che possono essere sorte nella fase di elaborazione del conceptual model, in particolare nella parte di social networking sono:

- chi sono i miei amici?
- chi sono gli amici dei miei amici?
- come l'utente X è associato all'utente Y?

Tutte queste domande dipendono dal tipo di **relazione** tra due persone. Quindi **Amicizia** sarà sicuramente un edge nel mio grafo, e collegherà due vertex **Persona**.

Ora che abbiamo deciso che dev'essere un arco, lo denominiamo "friend" e otteniamo una struttura di questo tipo:



Il prossimo passaggio sarà quello di assegnargli una direzione, definendo un vertex di uscita e uno di ingresso, e stabilendo che:

Se Marco → Daniele, Marco è amico di Daniele

E quindi Daniele è nella sua lista degli amici. Da slide: in un buon graph model, gli archi devono leggersi come frasi. Graficamente parlando:

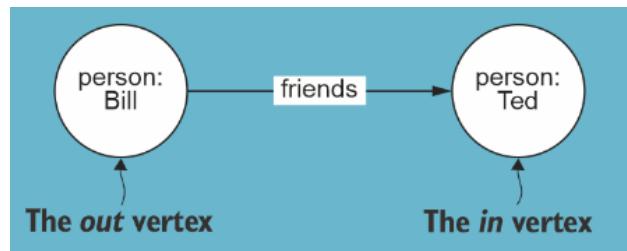
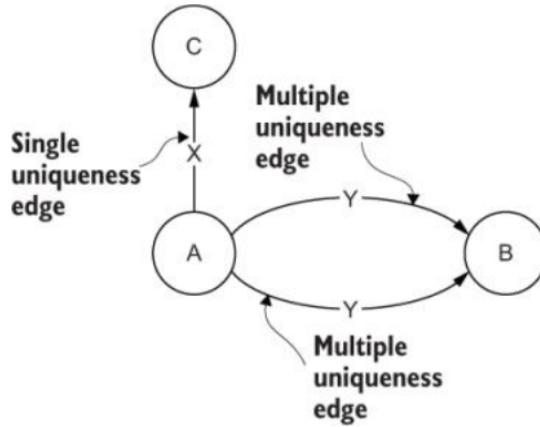


Figure 2: Bill è amico di Ted

Ora viene lo step piú complicato, assegnare l'**unicità** di un edge, che descrive il numero di volte che due vertici vengono connessi da un arco che ha lo stesso nome (guardando l'immagine sopra, quante volte usi l'arco friends).

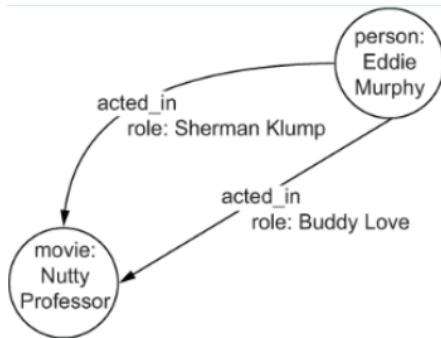


**Perché si parla di uniqueness e non di cardinality o multiplicity?** Quando si parla di metodi di modellazione dei dati, il termine cardinalità viene usato per indicare quante entità possono essere collegate tra loro. Ad esempio, si può avere una relazione tra ordine e cliente e dire che la cardinalità della relazione è uno-a-molti. Oppure si può dire che la cardinalità dei clienti per un ordine è zero-a-molti.

L'UML (Unified Modeling Language) evita il termine cardinalità, preferendo usare il termine molteplicità (multiplicity). Spesso, chi proviene da un background di modellazione dei dati ne rimane sorpreso, poiché il termine cardinalità è stato largamente usato in quel contesto.

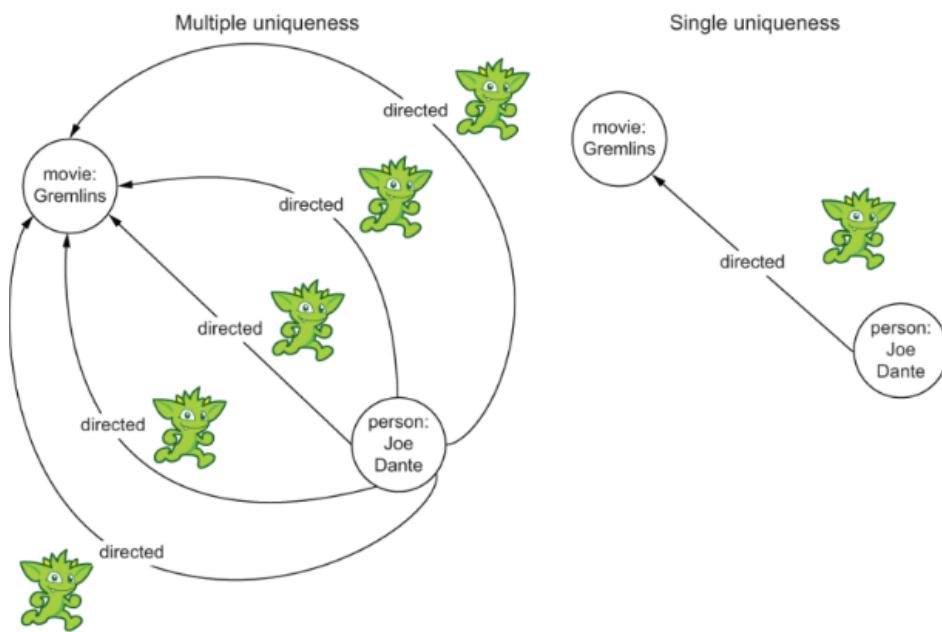
Il motivo di questo cambiamento è che, secondo la definizione del dizionario, la cardinalità è “il numero di elementi in un insieme o altro raggruppamento” (Oxford English Dictionary). In base a questa definizione, l’uso del termine cardinalità nella modellazione dei dati sarebbe in realtà scorretto. Nel suo eccellente UML Reference Manual, Rumbaugh definisce la molteplicità come “una specifica dell’intervallo di valori ammissibili di cardinalità – cioè la dimensione – che un insieme può assumere”. L'UML utilizza la molteplicità in vari contesti: per una proprietà (associazione o attributo) e anche per mostrare la molteplicità delle parti in una struttura composita. È formalmente definita come un limite inferiore e uno superiore. Un’associazione (l’equivalente UML di una relazione nella modellazione dei dati) ha una molteplicità per ciascuna direzione.

**Definizione** Si parla di **Multiple uniqueness** quando l'arco che collega tra loro due vertici si ripete più volte.



Questo è un chiaro esempio di multiple uniqueness, quando viene collegato Eddie Murphy a due ruoli nel film *Il professore matto*. Il problema di una uniqueness errata può essere davvero deleterio per un sistema:

- se abbiamo edges single unique, ma che dovrebbero essere multipli
- se abbiamo edges multiple unique, ma che dovrebbero essere singoli
- inoltre una query farà più fatica a ritornare tutti i risultati nel caso di una multiple uniqueness



### Step 3 : Assegnare proprietà a vertici e nodi

Le proprietà in un grafo sono coppie chiave valore che descrivono un attributo specifico di un vertex o un edge. Le principali domande da porsi quando devono essere assegnate delle proprietà sono:

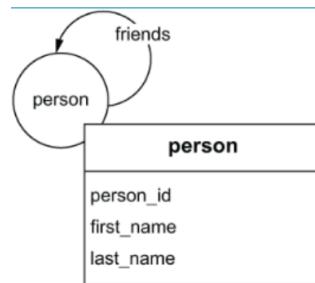
- che proprietà sono richieste?
- che nome gli diamo?
- qual è il loro data type?

Per rispondere a tali domande bisogna fare riferimento al modello concettuale preparato e vedere quali dati vanno salvati.

**Esempio** Guardando sempre il nodo "Persona" e arco "Amico", le domande sono sempre le stesse:

- chi sono i miei amici?
- chi sono gli amici dei miei amici?
- come l'utente X è associato all'utente Y?

Quindi sappiamo che sicuramente abbiamo bisogno di dare un nome ed un cognome ad ogni Persona e che possiamo inserirli come **proprietà**. Poi sicuramente servirà un identificatore unico per ogni persona, un id, per distinguere vari omonimi quando si vuole creare un collegamento di amicizia.



**Bivio** A questo punto se decidi di usare un qualunque graph model schemaless che esiste sul mercato, hai finito. Invece se ti è necessario esplicitare lo schema hai ancora uno step da fare: tradurre il tuo logical data model nel physical data model richiesto dal database scelto.

Quando progettisti un sistema, prima crei un modello logico dei dati – cioè una rappresentazione concettuale di come i dati sono organizzati e come sono collegati (per esempio: "Un cliente può fare più ordini"). Ma ogni database ha le sue regole per come quei dati devono essere effettivamente memorizzati. Questo si chiama modello fisico dei dati.

Due casi possibili:

1. Stai usando un database "schemaless" (senza schema fisso):

Alcuni database a grafo, come Neo4j o altri NoSQL, ti permettono di non specificare lo schema in anticipo. Puoi semplicemente iniziare a inserire dati, e il database li accetta così come sono.  
→ In questo caso sei a posto, non devi fare altro.

2. Stai usando un database che richiede schema esplicito (ad esempio, un RDBMS come PostgreSQL o un grafo con schema fisso):

Qui devi specificare in dettaglio quali sono i nodi, le relazioni, le proprietà, i tipi di dati ecc.  
→ Devi convertire il tuo modello logico (di concetti generali) in un modello fisico (concrete tabelle, colonne, tipi, vincoli...).

#### Step 4: Fare un check finale del modello

L'ultimo step nel create il nostro data model è quello di testare se possiamo dare una risposta alle domande prefissate in fase di progettazione.

**Esempio** Le domande:

- chi sono i miei amici? Possiamo rispondere partendo dal nodo di una persona specifica trovata tramite id unico e attraversando tutti gli edge "Amico" che partono da questa persona.
- chi sono gli amici dei miei amici? Partendo sempre da una persona specifica con il suo id, itero il processo ogni volta che raggiungo un amico guardo i suoi amici.
- come l'utente X è associato all'utente Y? Cerchiamo partendo dall'id dell'utente X di arrivare, attraversando tutti gli edges possibili, all'utente Y.

**Best practice** Alcuni controlli aggiuntivi di buone pratiche per assicurarci che il nostro modello di dati rappresenti un solido modello a grafo:

- I vertici e gli archi si leggono come una frase? Sì. Anche se non è un requisito assoluto, è un ottimo controllo generale per verificare che le etichette dei vertici rappresentino i nomi nel tuo modello e che le etichette degli archi descrivano le azioni o i verbi.
- Ho etichette di vertici o archi diverse con le stesse proprietà? No. In questo caso, vogliamo solo una singola etichetta di vertice e una singola etichetta di arco.
- Il mio modello ha senso? Sì. Anche se questo passaggio può sembrare un controllo ovvio, vale la pena prendersi il tempo per fare un passo indietro e verificare che il tuo modello di dati a grafo non si sia allontanato troppo dal modello concettuale e che abbia senso per il problema che stai cercando di risolvere.

A tutto questo va aggiunta una fase di testing per capire se effettivamente sono necessarie altre funzionalità, se quelle già presenti possono essere modificate, se servono nuove proprietà ecc...

## 6 Machine learning e grafi

La relazione principale tra ML e graph analytics si trova nei dati. Questo perché nella creazione di modelli hanno un ruolo fondamentale: migliori sono i dati, meglio si comporterà il modello.

### 6.1 Data: the true challenge

In Machine Learning sono richieste grandi quantità di dati per ottenere buoni risultati ma le data sources sono sempre piene di errori, valori anomali e rumore. **Perché dati sbagliati sono difficili da trattare?** Perché i modelli fanno molta fatica a riconoscere se un dato non va bene per poi scartarlo, quindi può succedere che imparino da esso imparando a sbagliare. ML è un processo di induzione: i modelli tenderanno sempre a comportarsi bene con cose che hanno effettivamente già visto e non supporteranno cose che non esistono nei set. Se sono presenti molti errori all'interno del dataset, il modello potrebbe overfittare sul training o creare dei bias, senza essere più in grado di generalizzare. Il modello però imparerà sicuramente bene da un set di feature rilevanti quindi è necessario trovare un buon compromesso tra tante feature totali, poche irrilevanti.

### 6.2 Issues

I problemi legati alla bassa qualità dei dati determinano una serie di vincoli, sul data management per i modelli:

- **Managing big data:** gestire dati provenienti da più fonti e combinarli in un unico set, migliora molto le prestazioni in fase di apprendimento
- **Designing a flexible schema:** cercare di creare uno schema del modello che fornisca la possibilità di unire più schemi eterogenei in una struttura dati unificata e omogenea che soddisfi i requisiti informativi e di navigazione. Lo schema deve evolversi facilmente in base al progetto.
- **Developing efficient access patterns:** velocizzare l'accesso ai dati porta ad un miglioramento del training process. Task come feature extraction, filtering, cleaning, merging e altre operazioni di preprocessing saranno più efficienti se potranno sfruttare una potenza di accesso ai dati più elevata.

### 6.3 Performance

Quando parliamo di **performance** nell'ambito del Machine Learning dobbiamo specificare a cosa ci stiamo riferendo:

- **Predictive accuracy**
- **Training performance**
- **Prediction performance**

#### Predictive accuracy

In ambito *Machine Learning*, l'**accuratezza predittiva** (*predictive accuracy*) è una metrica utilizzata per valutare quanto bene un modello è in grado di fare previsioni corrette su nuovi dati, ovvero su dati non visti durante la fase di addestramento.

Formalmente, nel caso di un problema di classificazione, l'accuratezza è definita come la proporzione di istanze correttamente classificate sul numero totale di istanze testate. Si calcola come:

$$\text{Accuracy} = \frac{\text{Numero di previsioni corrette}}{\text{Numero totale di previsioni}}$$

L'accuratezza è una metrica intuitiva e utile quando le classi sono bilanciate, ma può essere fuorviante in presenza di classi sbilanciate (ad esempio, se una classe rappresenta il 95% dei dati, un classificatore che predice sempre quella classe avrebbe il 95% di accuratezza ma sarebbe inutile). Nei casi in cui le classi sono sbilanciate, è consigliabile usare metriche alternative o complementari, come *precision*, *recall*, *F1-score* o l'*area sotto la curva ROC* (AUC).

### Training performance

Le **prestazioni dell'addestramento** si riferiscono al tempo necessario per elaborare il modello.

- La quantità di dati da elaborare e il tipo di algoritmo utilizzato determinano sia il tempo di elaborazione che la memoria necessaria per costruire il modello predittivo.
- Chiaramente, questo problema riguarda maggiormente gli algoritmi che producono un modello come risultato della fase di addestramento. Per gli algoritmi *instance-based*, i problemi di prestazioni emergono più avanti nel processo, ad esempio durante la fase di previsione.
- Nell'*apprendimento batch*, il tempo di addestramento è generalmente più lungo, a causa della maggiore quantità di dati da elaborare (rispetto all'approccio di *apprendimento online*, in cui l'algoritmo apprende in modo incrementale da quantità di dati più piccole).
- Anche se nell'apprendimento online la quantità di dati da processare è ridotta, la velocità con cui questi vengono elaborati influisce sulla capacità del sistema di rimanere aggiornato con gli ultimi dati disponibili, il che ha un impatto diretto sull'accuratezza delle previsioni.

### Prediction performance

Con il termine **prediction performance** intendiamo invece il tempo impiegato per fornire previsioni.

**Esempio** L'output del modello di machine learning potrebbe ad esempio essere un report statico una tantum per aiutare i manager a prendere decisioni strategiche o un servizio online per utenti finali. Nel primo caso, il tempo necessario per completare la fase di previsione e calcolare il modello non è un problema significativo purché il lavoro venga completato in un lasso di tempo ragionevole. Nel secondo caso, la rapidità è importante perché influisce sulla User Experience e l'efficacia della previsione stessa.

## 6.4 Benefit of using graphs

L'utilità dei grafi nel Machine Learning risiede nella possibilità di avere dei dati organizzati meglio, un accesso più rapido ad essi oltre ad alcuni algoritmi per implementare i parametri delle performances.

### Storing del modello

Nell'approccio model-based al Machine Learning, il sistema apprende un modello dai dati durante la fase di training. Questo modello va salvato per essere riutilizzato senza doverlo ricostruire ogni volta. La struttura del modello dipende dall'algoritmo usato: ad esempio, nel caso dei grafi, può consistere in:

- Matrici di similarità tra nodi (per modelli tipo nearest-neighbor)
- Mappature tra nodi e cluster (per modelli di clustering) Questo approccio è utile per compiti come la raccomandazione o la scoperta di comunità nei grafi.

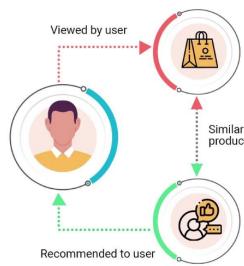
Le dimensioni dei due modelli differiscono enormemente. Si consideri un sistema contenente 100 elementi:

- Le similarità tra elementi richiederebbero l'archiviazione di  $100 \times 100$  voci. Sfruttando le ottimizzazioni, questo numero può essere ridotto per considerare solo i primi  $k$  elementi simili quindi  $100 \times k$  voci.
- La mappatura tra elementi e cluster richiede solo 100 voci; pertanto, lo spazio necessario per archiviare il modello in memoria o su disco potrebbe essere enorme o relativamente modesto. Inoltre, il tempo di accesso/interrogazione del modello influisce sulle prestazioni globali durante la fase di predizione.

Per questi motivi, la gestione dell'archiviazione dei modelli rappresenta una sfida significativa nel machine learning.

### Real Time

Molti sistemi ML richiedono una risposta real time per l'utente.



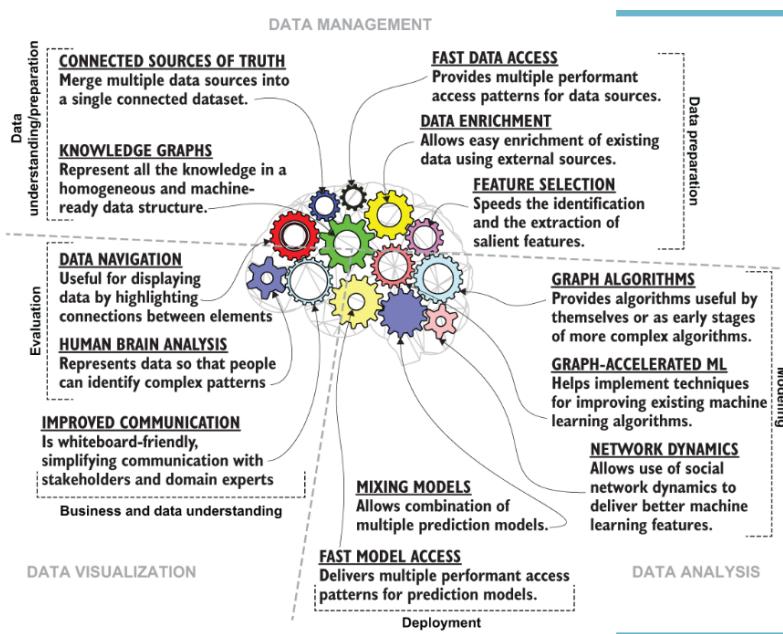
**Esempio** Semplici motori di raccomandazione che reagiscono agli ultimi clic dell'utente; auto a guida autonoma che sono state istruite a non ferire un pedone che attraversa la strada.

Anche se le conseguenze in caso di errore sono molto diverse tra i due casi, in entrambi è fondamentale la capacità del sistema di apprendimento di reagire rapidamente ai nuovi stimoli provenienti dall'ambiente, per garantire la qualità del risultato finale.

## Graph powered machine learning

Di seguito vedremo quali sono quindi i benefici nell'utilizzare dei grafi per la rappresentazione dei dati elaborati dai modelli ML. Le funzionalità dei grafici sono raggruppate in tre aree principali:

- **Data management:** quest'area contiene le features fornite dai grafi che aiutano i modelli ML a rapportarsi con i dati
- **Data analysis:** quest'area contiene feature dei grafi e algoritmi utili per learning e predicting
- **Data visualization:** quest'area é quella di maggiore importanza per i grafi perché possono mettere in mostra la loro utilitá e intrerpretabilitá come visual tools



**Data management** I grafi permettono ai modelli di accedere ai dati molto piú velocemente oltre a permettere una maggiore efficienza in fase di pulizia e arricchimento. Normalmente i sistemi imparano da una singola tabella, mentre i graph based possono accedere a piú di una contemporaneamente. Vediamo delle features importanti in data management fornite dai grafi:

- **Tutte le risorse sono connesse:** I grafi permettono di fare merging di multiple data sources in un singolo dataset connesso, pronto per essere usato nella fase di training. Questo riduce la data sparsity, incrementa il numero di dati disponibili e semplifica il management.
- **Knowledge graph:** fornisce una data structure omogenea per combinare non solo fonti, ma anche modelli, fonti esterne... Il risultato é pronto per essere utilizzato come training, prediction o visualization.
- **Accesso rapido ai dati:** differenza fondamentale tra grafo e tabella. A paritá di condizioni, un grafo rende molto piú accessibile qualunque dato, con piú relazioni che lo legano agli altri, mentre una struttura rigida a tabella é molto piú complessa.
- **Data enrichment:** rende semplice estendere dati pre esistenti a risorse esterne ed unirli. Questo sicuramente grazie alla natura schemaless dei grafi.

- **Feature selection:** Identificare feature rilevanti è la parte fondamentale nei modelli e con un grafo hanno accesso molto rapido ad ogni tipo di dato.

**Data analysis** I grafi possono essere utilizzati per modellare e analizzare le relazioni tra entità, così come le loro proprietà. La flessibilità dello schema offerta dai grafi permette inoltre a modelli differenti di coesistere nello stesso dataset. Le funzionalità di analisi dei dati basate su grafi includono:

- **Algoritmi su grafi** — Diversi tipi di algoritmi, come il clustering, il page rank e l'analisi dei collegamenti, sono utili per ottenere informazioni dai dati e per l'analisi. Inoltre, possono essere utilizzati come fase di pre-elaborazione dei dati in processi di analisi più complessi.
- **Machine learning potenziato dai grafi:** l'estrazione di caratteristiche (feature extraction) basata su grafi è un esempio di come i grafi possano accelerare o migliorare la qualità di un sistema di apprendimento. I grafi aiutano a filtrare, pulire, arricchire e integrare i dati, sia prima che durante la fase di addestramento.
- **Dinamiche della rete:** essere consapevoli dei contesti circostanti e delle forze che agiscono sulle reti permette non solo di comprendere le dinamiche della rete, ma anche di sfruttarle per migliorare la qualità delle previsioni. **Esempio:** in un social network, la diffusione di contenuti dipende anche da eventi esterni (es. una notizia virale). Capire queste dinamiche può aiutare a prevedere i trend.
- **Modelli misti:** possono coesistere più modelli nello stesso grafo, sfruttando la flessibilità e l'accesso veloce ai dati, a condizione che possano essere fusi nella fase di previsione. Questa caratteristica migliora la precisione finale. Inoltre, uno stesso modello a volte può essere riutilizzato in modi diversi. Diversi modelli (es. un modello per raccomandazioni, uno per rilevare anomalie) possono essere combinati in un unico grafo. Questo è utile quando ogni modello vede un “aspetto” diverso dello stesso dato.
- **Accesso veloce al modello:** l'uso in tempo reale richiede previsioni rapide, il che implica un modello accessibile nel minor tempo possibile. I grafi offrono schemi di accesso adatti a questi scopi. I grafi permettono accesso efficiente a strutture complesse (es. neighborhood di un nodo, percorsi più brevi, comunità), rendendo le predizioni rapide e adattabili in tempo reale.

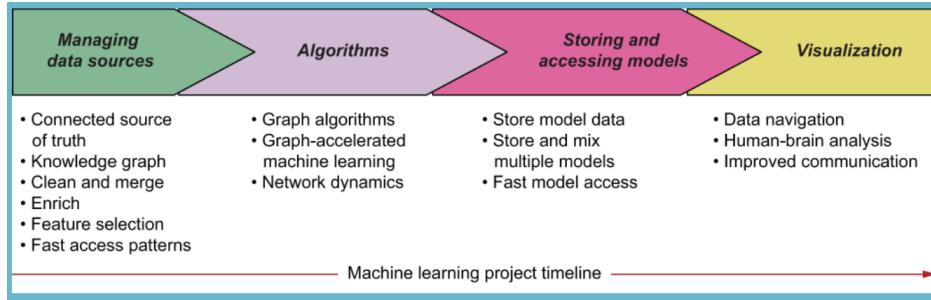
**Data visualization** I grafi hanno un elevato potere comunicativo e possono mostrare più tipi di informazione contemporaneamente in un modo che il cervello umano comprende con facilità. Le funzionalità di visualizzazione dei dati basata su grafi includono:

- **Navigazione dei dati:** i grafi sono molto utili per rappresentare i dati evidenziando le connessioni tra gli elementi. Possono essere usati sia come strumenti di navigazione visiva, che aiutano a esplorare i dati, sia come strumenti investigativi.
- **Analisi umano-macchina:** visualizzare i dati sotto forma di grafo permette di combinare le capacità del machine learning con l'intuito umano, facilitando l'elaborazione avanzata e il riconoscimento di pattern complessi.
- **Comunicazione efficace:** i grafi, in particolare i property graph, sono “whiteboard friendly”, cioè facili da rappresentare visivamente nello stesso modo in cui sono strutturati nel database. Questo riduce il divario tra la parte tecnica e la comunicazione verso esperti del dominio o stakeholder.

**Nota:** La visualizzazione dei grafi è particolarmente utile nei casi in cui le relazioni tra i dati contano più dei dati stessi, come nelle reti sociali, nell'analisi di frodi, o nei knowledge graph. [fonte GPT]

## 6.5 Ruolo dei grafi nel Machine Learning

Il ruolo dei grafi non comprende sempre tutti questi passaggi. Dipende dal motivo per cui vengono scelti, questa è una lista delle principali applicazioni. Riassumendo, potremmo vedere dei grafi in tutte e quattro le fasi del ML workflow:

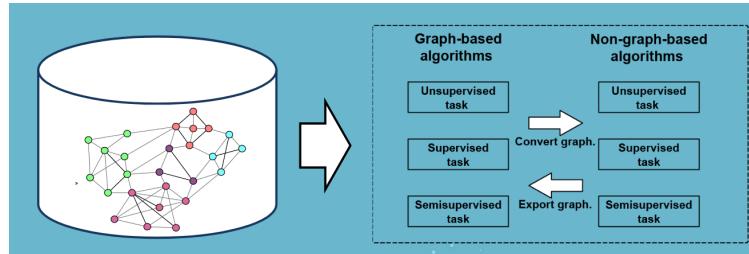


### ML pipeline: Managing data sources

La trasformazione dei dati in grafi può avvenire in diversi modi e possiamo definire due macro categorie:

- **Graph modeling:** Dati convertiti in una rappresentazione a grafo. Manteniamo le stesse informazioni.
- **Graph construction:** Costruiamo un nuovo grafo, aggiungendo informazioni ai dati.

### ML pipeline: Graph storage and learning process (Algorithms)



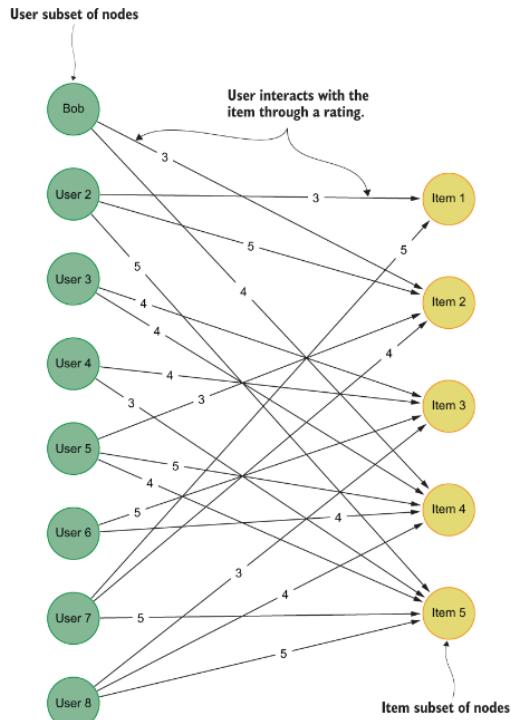
La rappresentazione a grafo è utile per le seguenti tasks:

- **Selezione delle caratteristiche:** Un grafo è facile da interrogare e può unire dati provenienti da più fonti, semplificando l'individuazione e l'estrazione delle variabili da utilizzare per l'addestramento.
- **Filtraggio dei dati:** Le relazioni facilmente navigabili tra oggetti permettono di filtrare rapidamente i dati inutili prima della fase di addestramento, velocizzando così la costruzione del modello.

- **Preparazione dei dati:** I grafi facilitano la pulizia dei dati, la rimozione di voci spurie e l'unione di dati provenienti da fonti diverse.
- **Arricchimento dei dati:** L'estensione dei dati con fonti esterne di conoscenza (come reti semantiche, ontologie e tassonomie) o l'integrazione dei risultati della fase di modellazione per costruire una base di conoscenza più ampia è semplice con un grafo.
- **Formattazione dei dati:** È possibile esportare i dati nel formato necessario: vettori, documenti, ecc.

Per selezionare l'algoritmo, decidiamo tra due approcci principali uno dove usiamo il grafo come algoritmo di learning e uno dove usiamo il grafo per semplificare una pipeline altrimenti molto più complessa.

**Fast computation of similarity using graphs** Normalmente, dati due items, dovrei calcolare la cosine similarity con ogni altro elemento del dataset per capire gli elementi simili. È possibile invece implementare un metodo che sfrutta la potenza dei grafi: usare un **bigrafo** (o **bigraph**). È una specie particolare di grafo i cui nodi possono essere divisi in due set U e V disgiunti e indipendenti; ogni nodo del set U si collegherà solo a nodi del set V.



**Perché conviene?** Utilizzando la rappresentazione a grafo, è facile utilizzare una semplice query per trovare tutti gli oggetti che hanno almeno un utente valutatore in comune. La similarità può quindi essere calcolata solo tra l'oggetto corrente e quelli sovrapposti, riducendo notevolmente il numero di calcoli necessari.

## ML pipeline: Storing and Accessing models

Il terzo step nel workflow consiste nel dare delle predizioni all'utente finale. Il modello deve essere salvato in modo da poter essere riutilizzato ogni volta che è richiesta una nuova prediction. La velocità con cui accediamo al modello limita la prediction performance: il tempo è un elemento di valutazione fondamentale per un modello di ML.

Nel bigrafo ad esempio è molto semplice salvare anche la matrice di similarità: basta infatti collegare tra loro i nodi con degli archi con il loro valore di similarity e per ridurre la complessità tenere solo i top k.

**Rete bayesiana** Una rete bayesiana è un grafo diretto dove ogni nodo è una variabile associata a delle informazioni probabilistiche. Questo tipo di grafo combina informazioni logiche con probabilistiche. Secondo le definizioni di Russel e Norvig:

- Ogni nodo corrisponde a una variabile casuale. Queste variabili possono essere grandezze osservabili, variabili latenti, parametri incogniti o ipotesi.
- Gli archi rappresentano dipendenze condizionali. Se esiste un arco dal nodo X al nodo Y, X si dice genitore di Y. Il grafo non ha cicli orientati (e quindi è un grafo aciclico orientato). I nodi non connessi rappresentano variabili condizionatamente indipendenti.
- Ogni nodo  $X_i$  ha una distribuzione di probabilità condizionata  $P(x_i, \text{Genitori}(x_i))$  che quantifica l'effetto dei genitori sul nodo. In altre parole, ogni nodo è associato a una funzione di probabilità che accetta (come input) un particolare insieme di valori per le variabili genitori del nodo e fornisce (come output) la probabilità, o la distribuzione di probabilità, se applicabile, della variabile rappresentata dal nodo.

**Rete bayesiana dinamica** Anche chiamata **Dynamic Bayesian Network**, è una rete bayesiana che modella l'evoluzione di variabili nel tempo. Significa che una DBN collega le variabili tra un tempo  $t$  e il tempo  $t + 1$ .

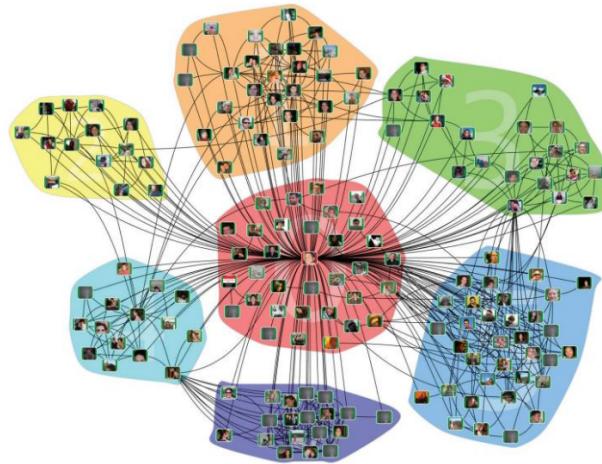
Ad esempio supponi una variabile Posizione di un robot: una DBN può rappresentare come la Posizione al tempo  $t + 1$  dipende dalla Posizione al tempo  $t$  e dall'Azione al tempo  $t$ . La versione più semplice per una DBN è una **Markov Chain**, ovvero una struttura a grafo dove ogni nodo attuale è causa del successivo e basta, senza avere una correlazione diretta con il passato.

## ML pipeline: Visualization

La visualizzazione viene presentata alla fine del processo, poiché visualizzare i dati dopo l'elaborazione iniziale è molto meglio che visualizzare i dati grezzi, ma la visualizzazione dei dati può avvenire in qualsiasi momento del workflow. L'importanza della visualizzazione dei dati risiede nell'analizzare dati complessi, identificare modelli ed estrarre informazioni preziose. Semplificare informazioni complesse e presentarle visivamente consente ai modelli di prendere decisioni informate ed efficaci in modo rapido e accurato. Inoltre la visualizzazione dei dati gioca un ruolo chiave perché ci consente di accedere e analizzare i dati da una prospettiva diversa.

**L'importanza della visualizzazione** Gli esseri umani sono creature naturalmente visive. I nostri occhi sono i nostri recettori sensoriali più potenti e presentare i dati attraverso la visualizzazione delle informazioni sfrutta al meglio le nostre capacità percettive. Molti set di dati oggi sono troppo grandi per essere analizzati senza strumenti computazionali che ne facilitino l'elaborazione e l'interazione.

Se esploriamo i dati sotto forma di grafico, è più facile individuare pattern, valori anomali e lacune. Il modello grafico espone relazioni che potrebbero essere nascoste in altre visualizzazioni degli stessi dati (come tabelle e documenti) e ci aiuta a individuare i dettagli importanti.



Scegliere la rappresentazione migliore per un grafo non é semplice, proprio perché dev'essere ben descrittiva e non in confusione. Qui sopra si può vedere la rappresentazione della rete di amicizie di un utente Facebook.

## 7 Graph in Data Engineering

### 7.1 Big Data lifecycle

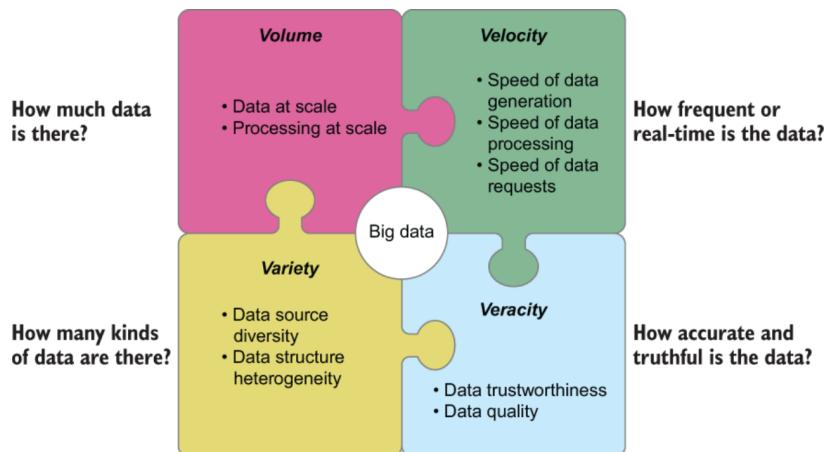
Il ciclo di vita dei sistemi che analizzano big data è composto da una serie di fasi:

- **Collect:** i dati vengono raccolti da molteplici fonti.
- **Store:** i dati vengono memorizzati in modo adeguato in un unico (o talvolta più di uno) archivio facilmente accessibile, per essere pronti alle fasi successive.
- **Clean:** i dati vengono unificati, puliti e, quando possibile, normalizzati usando uno schema coerente e omogeneo.
- **Access:** i dati sono resi disponibili. Vengono fornite diverse viste o modalità di accesso per semplificare e velocizzare l'accesso al dataset, che verrà poi utilizzato per l'addestramento.

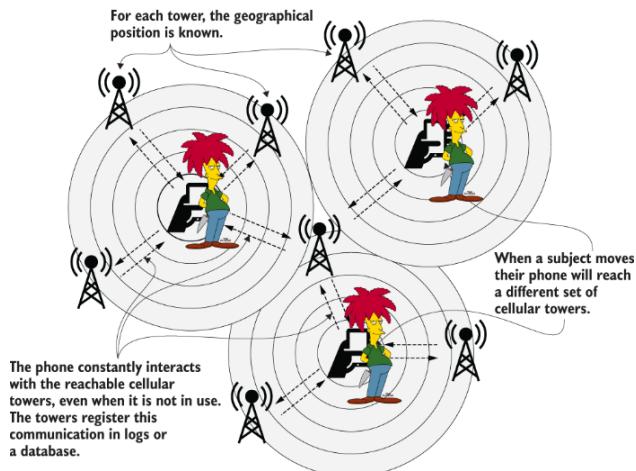
### Le 4 V dei Big Data

Le principali caratteristiche che definiscono i Big Data sono riassunte nelle cosiddette **4 V**:

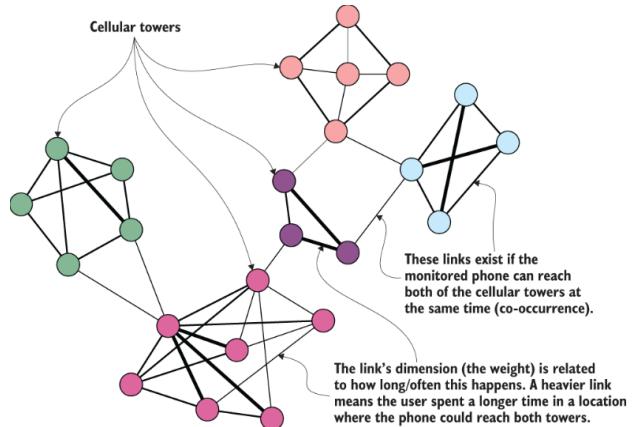
- **Volume:** rappresenta la quantità enorme di dati generata ogni giorno da fonti diverse come social media, sensori, dispositivi mobili, log di sistema, ecc. La gestione richiede infrastrutture scalabili per l'archiviazione e l'elaborazione.
- **Velocità:** indica la rapidità con cui i dati vengono generati, trasmessi ed elaborati. In molti casi (es. mercati finanziari, IoT) è necessario un trattamento in tempo reale o quasi.
- **Varietà:** i dati provengono da fonti eterogenee e possono essere strutturati (es. tabelle), semi-strutturati (es. JSON, XML) o non strutturati (es. testo libero, immagini, video, audio). L'integrazione e l'analisi di questi formati diversi è una sfida importante.
- **Veridicità:** si riferisce alla qualità, attendibilità e accuratezza dei dati. I big data possono contenere rumore, dati mancanti o contraddittori; è quindi fondamentale valutare l'affidabilità delle fonti e pulire i dati prima dell'analisi.



**Use case scenario (slides)** Sei un poliziotto e vuoi tracciare un sospettato dalle chiamate telefoniche potendo monitorare i segnali del cellulare alle torri. Il nostro scopo in questo caso é quello di creare un modello predittivo sulla base dei dati di monitoring del telefono e prevedere le mosse a partire dalla location attuale.



Per risolvere il problema possiamo fare uso dei grafi. Attraverso il loro utilizzo possiamo infatti generare i clusters che verranno successivamente analizzati da un modello.



Ognuno dei cellulari riesce a registrare le 4 torri piú vicine (quelle con il segnale piú forte). Una volta che vengono salvati questi dati possono essere rappresentati come CTN (cellular tower network) dove ogni nodo é una torre. I nodi sono presenti solo tra torri che compaiono contemporaneamente nello stesso record (quindi le 4 torri registrate dal singolo telefono saranno connesse) e ogni edge é pesato secondo una tempistica (quanto tempo la coppia é registrata insieme).

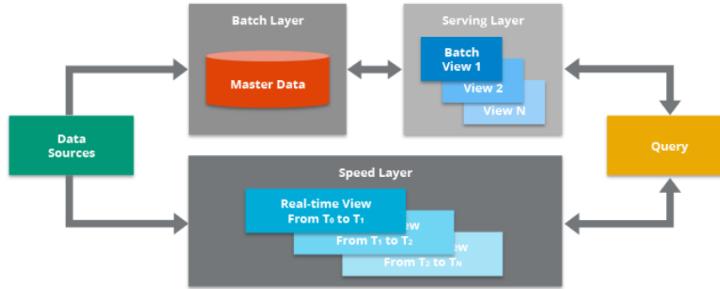
La **forza** di un nodo é misurata sulla somma dei pesi dei suoi edges. I nodi piú forti sono quelli piú vicini al cellulare. I cluster delle torri possono essere trasformati negli stati di un modello dinamico. Con i dati sugli spostamenti di un individuo, é infatti possibile prevedere con una buona precisione i suoi spostamenti futuri. Una volta che il modello sarà allenato, si potrà prevedere a partire dalla posizione attuale la prossima location.

**Architectural problems** Alcuni aspetti rilevanti di questo data flow, possono andare ad intaccare l'architettura del modello di ML:

- **Eventi:** in questo caso i telefoni, quando mandano il segnale. Il signal non cambia e soprattutto avviene senza controllo. È importante fare lo storing dei dati una sola volta.
- **Molteplici viste:** create come funzioni (ad esempio in questo caso aggregazione, clustering) ma possono cambiare in base all'algoritmo di learning.
- **La creazione delle viste:** il processo normalmente lavora su tutti i dati disponibili, e può richiedere del tempo.
- **Real-time view:** avere una vista real-time si intende ovviamente **live**. È necessario un servizio streaming che legga gli eventi e registri nuovi dati subito.

## 7.2 Lambda architecture

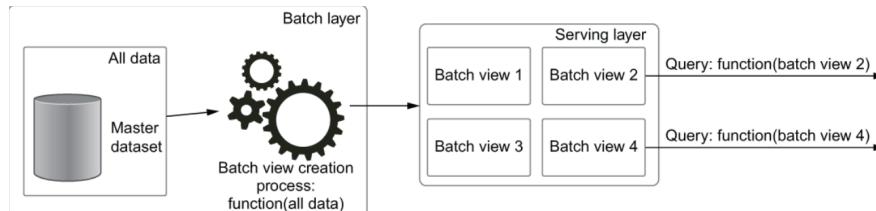
Tutti questi problemi possono essere risolti utilizzando Lambda Architecture, che suddivide il problema big data in 3 parti: batch, serving e speed.



Ogni layer si occupa di un subset di requisiti e lavora considerando le funzioni degli altri due. Contiene una pipeline batch ma anche una real-time per rispondere a richieste di ogni tipo.

### Layers di batch e di serving

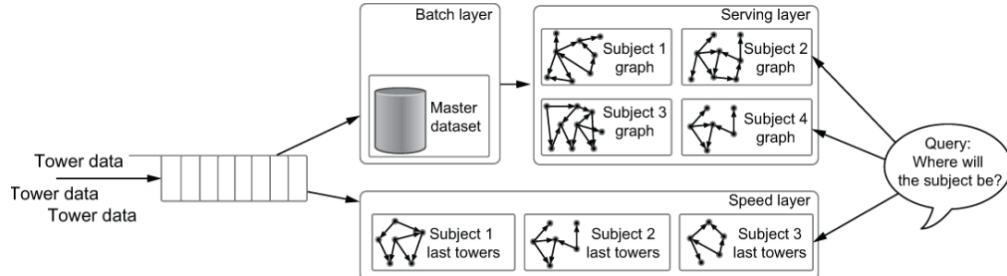
Tutti i dati nel formato raw sono salvati nel batch layer, oltre ad essere il responsabile dell'accesso a tali dati e al calcolo della batch view. Tutte le view risultanti sono salvate nel serving layer, dove sono indicizzate per essere più accessibili.



Nello scenario CTN, con le torri e i segnali, una prima batch view sarebbe il grafo con le torri. La parte di creazione dei cluster viene fatta all'interno del batch layer e le nuove views vengono inviate al serving layer.

## Speed layer

Lo speed layer fornisce delle view recenti, per coprire quel gap che c'è tra il batch layer e gli ultimi arrivati. Sfruttando sempre l'esempio del CTN:



Nel CTN, sia la batch view che la real-time sono grafi. Questa rappresentazione non solo permette di rispondere più velocemente a delle queries ma facilita ulteriormente l'analisi. La differenza sostanziale tra i due layer è che lo speed guarda solo gli ultimi arrivati. In questo scenario è lei che fornisce l'info sulla location attuale e su dove ci sposteremo.

**Conclusioni** Lambda architecture ha numerosi vantaggi architetturali che portano ad avere bassa latenza, buona data consistency, alta scalabilità e una buona gestione di ogni tipo di errore.

### 7.3 Grafi per versioni fine-grained

Esempio di fraud detection, vediamo gli approcci con graph:

#### Primo Approccio - Grafi come Viste Aggregate

I grafi vengono utilizzati per creare visualizzazioni sopra un dataset principale (*batch layer*) o rappresentazioni in tempo reale (*speed layer*) di porzioni dei dati disponibili. In questo caso, i dati transazionali risiedono altrove nel dataset principale, e questo approccio è utile quando si possono eseguire query e analisi su dati aggregati memorizzati nel *serving layer*.

#### Secondo Approccio - Grafi come Fonte Principale

Alcuni tipi di analisi non possono essere eseguite su versioni aggregate dei dati, ma richiedono algoritmi che necessitano di informazioni più dettagliate e granulari per essere efficaci. Anche in questo caso si può utilizzare il modello a grafo per rappresentare le connessioni e sfruttare gli *insight* dai dati.

#### Vantaggio Chiave

Comprendere le connessioni tra i dati e derivare significato da questi collegamenti offre capacità che non sono disponibili nei metodi analitici classici non basati sui grafi. In quest'ultimo caso, il grafo diventa la fonte principale di conoscenza e modella un'unica fonte di verità connessa.

**Conclusione:** In sostanza, il testo contrappone l'uso dei grafi come strumento di visualizzazione versus il loro utilizzo come struttura dati fondamentale per l'analisi.

## 7.4 Vantaggi dei grafi

Vediamo i vantaggi dei grafi nell'analisi dei dati nel dettaglio:

- **Integrazione di Fonti Multiple:** Diverse fonti di dati, come informazioni geografiche o GPS, dati dei social network, profili personali degli utenti, dati familiari e simili, possono essere unite in un'unica fonte di verità connessa.
- **Estensione con Fonti Esterne:** I dati esistenti possono essere arricchiti con fonti esterne di conoscenza (posizioni dei negozi, indirizzi delle persone, ecc.) o con informazioni contestuali (un nuovo negozio, altri reclami, ecc.) che possono essere utilizzate per migliorare l'analisi.
- **Supporto a Tecniche Multiple:** Lo stesso modello dati può supportare diverse tecniche di analisi. Ad esempio, per scoprire un *fraud ring* – ”un'organizzazione focalizzata sulla truffa delle persone. Falsificazioni, false dichiarazioni, furto d'identità, contraffazione di assegni e valute sono tutte attività fraudolente.”
- **Visualizzazione e Analisi Manuale:** I dati possono essere visualizzati come grafo per accelerare l'analisi manuale. L'analisi può essere estesa a molteplici livelli di interazione, considerando salti multipli (*multiple hops*).
- **Semplificazione delle Operazioni:** La struttura semplifica le operazioni di fusione e pulizia dei dati, grazie al pattern di accesso flessibile fornito dal modello a grafo.

## 7.5 Grafi come Master Data Management (MDM)

Con master data management si intende il ruolo che hanno i grafi nella gestione dei dati: la capacità di identificare, fare cleaning, salvare e controllare i dati. Questo perché si evolvono con il tempo, aggiungo nuove fonti di dati, aggiornano i dati attuali sulla base di fonti esterne, e fare data versioning per gestire al meglio i dati obsoleti.

Simili a DataWarehouses ma diversi. I graph based MDM hanno i seguenti vantaggi:

- **Flessibilità:** I dati acquisiti possono essere facilmente modificati per includere attributi e oggetti aggiuntivi.
- **Estensibilità:** Il modello consente la rapida evoluzione del modello di dati master in linea con le esigenze aziendali in evoluzione.
- **Capacità di Ricerca:** Ogni nodo, ogni relazione e tutte le loro proprietà correlate sono punti di accesso per la ricerca.
- **Capacità di Indicizzazione:** I database a grafo sono naturalmente indicizzati sia per relazioni che per nodi, fornendo un accesso più veloce rispetto ai dati relazionali.

In un progetto ML è necessario seguire diversi step per l'utilizzo dei grafi: ovvero come si fa modeling (come rappresentare i dati usando il grafo) come gestire lo storage (serve avere uno storage dei dati persistent, in modo da poter sempre accedere ai dati) e più importante, gestire la fase di processing dei dati. Quando si lavora con grandi quantitativi di dati inoltre è difficile gestirli contemporaneamente quindi su opta per tecniche di **sharding** dove i grandi set vengono suddivisi in shards. Ma con i grafi non è così semplice: una possibile soluzione è quella di mettere tutti i nodi connessi all'interno dello stesso shard, per ridurre la latenza nel caso di navigazione del grafo; questo comunque comporta grandi costi nel traversare il grafo soprattutto quando si cerca un'informazione in uno shard separato che comunque è connessa ad un nodo dello shard attuale (basta pensare alla struttura a di un grafo connesso e due nodi distanti ma connessi).

## Tecniche per lo sharding coi grafi

Esistono tre tecniche principali per gestire lo sharding che sono:

- Application Level Sharding: come indica il nome, viene tutto lasciato in mano all'applicazione che suddivide nel modo che piú aggrada.
- Utilizzo di cache sharding o aumento della memoria: aumentare la memoria puó garantire che tutto il grafo stia in memoria. Cache sharding i dati vengono distribuiti su multiple istanze di cache (shard) utilizzando una funzione di hash o altri algoritmi di partizionamento.

## Tecniche di replication

In particolare, parlando di replication, il problema si presenta piú complesso del previsto, perché entra in gioco anche la gestione delle repliche: una decisione fondamentale quando si fa sincronizzazione é scegliere quale delle parti viene aggiornata per prima; si parla di di:

- **Sistema centralizzato:** quando i primi updates sono fatti su una master copy, quelle che gestiscono tutte le operazioni.
- **Sistema distribuito:** quando i primi updates sono sulle repliche direttamente.

La differenza tra master e replica nella replication é semplicemente che come in ogni altro sistema di replication, il master é il primo nodo ad essere aggiornato e ha il compito di diffondere l'aggiornamento a tutte le sue repliche.

## Native e non-native graph databases

Ci sono numerosi metodi per rappresentare grafi all'interno di un database. **Native graph databases** sono progettati per creare un filesystem che supporti in ogni modo il grafo oltre a comprenderlo, rendendoli altamente performanti. Per essere piú precisi, ogni nodo mantiene un riferimento diretto ai suoi nodi adiacenti (**index free adjacency**).

**Non native graph dbms** Consiste in un dbms specializzato in un altro tipo di dato (colonne, relazionale, ecc...) quindi quando deve lavorare con dei grafi c'è sempre da fare anche una conversione dei dati in ingresso. Quindi un non native sarà sempre meno performante del native, appunto per questa caratteristica. Esistono due tipi di non native: quelli che inseriscono un layer graph API a monte di una struttura dati già esistente e quelli che integrano piú strutture dati in una con semantica multimodello.

**Vantaggi dei modelli native** Vediamo i pro:

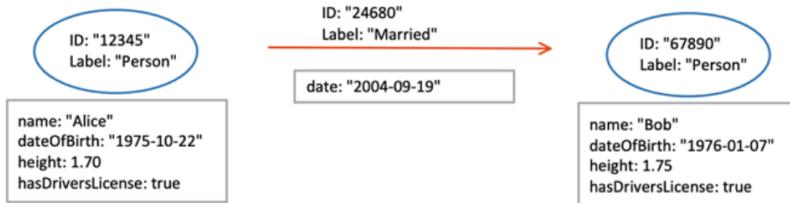
- **Prestazioni “da minuti a millisecondi”:** I database a grafo nativi gestiscono le query su dati connessi molto più velocemente rispetto ai database non nativi. Anche su hardware modesto, i database a grafo nativi possono facilmente gestire milioni di traversamenti al secondo tra nodi su una singola macchina e molte migliaia di scritture transazionali al secondo.
- **Efficienza in lettura:** I database a grafo nativi offrono traversamenti a tempo costante grazie all'adiacenza senza indice, senza la necessità di progettazioni complesse dello schema o ottimizzazioni delle query. Il modello intuitivo dei grafi a proprietà elimina la necessità di logiche applicative aggiuntive e spesso complesse per elaborare le connessioni.

- **Ottimizzazione dello spazio su disco:** Per migliorare le prestazioni in un grafo non nativo, è possibile denormalizzare gli indici o crearne di nuovi, o entrambe le cose, il che incide sulla quantità di spazio necessaria per memorizzare le stesse informazioni.
- **Efficienza in scrittura:** La denormalizzazione degli indici ha un impatto anche sulle prestazioni in scrittura, poiché tutte le strutture di indice aggiuntive devono essere aggiornate anch'esse.

## Label Property Graphs (LPG)

Altro metodo per la gestione big data nel caso di Graph Database Management. Modello per la struttura a grafo dove vengono labellati nodi e archi. Con questo modello è possibile sfruttare operazioni normalmente attribuibili ad altri DBMS, come proiezione, selezione, grouping, filtering... Secondo il progetto openCypher, un *label property graph* è definito come “un multigrafo diretto, con vertici etichettati, archi etichettati, archi che possono essere auto-collegamenti, in cui gli archi hanno una propria identità.” Un’**etichetta** è una stringa di testo che identifica un tipo, un insieme o una categoria di dati o oggetti. Nel mondo dei grafi, un’etichetta identifica un nodo o una relazione (vertice o arco).

- Un oggetto etichettato in un grafo è opzionale e non deve necessariamente essere unico.
- Un oggetto etichettato è simile a un tag che indica un certo tipo di raggruppamento o categoria.
- È possibile applicare più etichette a un oggetto nel grafo.



## Proprietà di un Label Property Graph (LPG)

Un *Label Property Graph* possiede le seguenti proprietà (definite in modo indipendente dalla piattaforma):

- Il grafo è costituito da un insieme di entità. Un’entità rappresenta un nodo oppure una relazione.
- Ogni entità ha un identificatore che la identifica univocamente all’interno dell’intero grafo.
- Ogni relazione ha una direzione, un nome che ne identifica il tipo, un nodo di partenza e un nodo di arrivo.
- Un’entità può avere un insieme di proprietà, solitamente rappresentate come coppie chiave/valore.
- I nodi possono essere etichettati con una o più etichette, che li raggruppano e indicano i ruoli che essi svolgono all’interno del dataset.

## 8 Recommendations con i Grafi

Negli ultimi anni, i sistemi di raccomandazione hanno assunto un ruolo centrale in numerosi ambiti applicativi, dall'e-commerce ai social media, dallo streaming musicale ai motori di ricerca. Tradizionalmente, questi sistemi si basano su approcci collaborativi o contenutistici; tuttavia, l'utilizzo della rappresentazione a grafo ha aperto nuove possibilità in termini di espressività, scalabilità e qualità delle raccomandazioni.

Un grafo permette di modellare in modo naturale le relazioni complesse tra utenti, item (prodotti, film, post, ecc.) e altri fattori contestuali, come tag, categorie o interazioni temporali. In questo contesto, i sistemi di recommendation basati su grafi sfruttano la struttura topologica per propagare preferenze, misurare somiglianze e generare suggerimenti più pertinenti anche in scenari con dati scarsi o altamente dinamici.

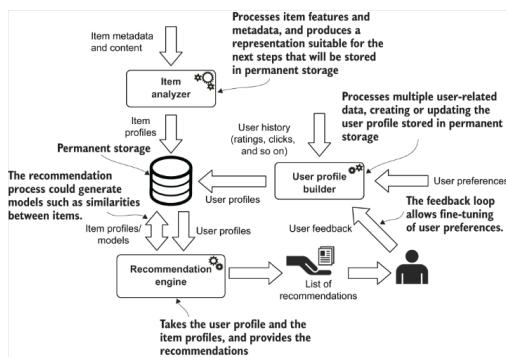
**Definizione** Il termine sistema di raccomandazione (RS) si riferisce a tutti gli strumenti software e le tecniche che, utilizzando le conoscenze che possono raccogliere sugli utenti e sugli elementi in questione, suggeriscono elementi che potrebbero essere di interesse per un particolare utente. Il motivo principale per cui questo sistema viene implementato è migliorare l'esperienza utente.

### Content Based Recommendation Systems (CBRS)

Sono sistemi che sfruttano le descrizioni di user e items per costruire delle rappresentazioni che abbiano una base semantica: collegare cose simili anche per descrizione potenzia la capacità di fare collegamenti utili. Processo base:

- Abbinando gli attributi del profilo utente target con gli attributi degli elementi per trovare elementi simili a ciò che l'utente ha apprezzato in passato, il risultato è un punteggio di pertinenza che prevede il livello di interesse dell'utente target per quegli elementi.
- In genere, gli attributi per descrivere un elemento sono caratteristiche estratte dai metadati associati a quell'elemento o caratteristiche testuali in qualche modo correlate all'elemento: descrizioni, commenti, parole chiave e così via.
- I CBRS utilizzano questi elementi ricchi di contenuto per effettuare confronti o dedurre gli interessi di un utente in base all'elenco di elementi con cui ha interagito.

**High-Level Architecture** Tre componenti di base nell'architettura di un CBRS:

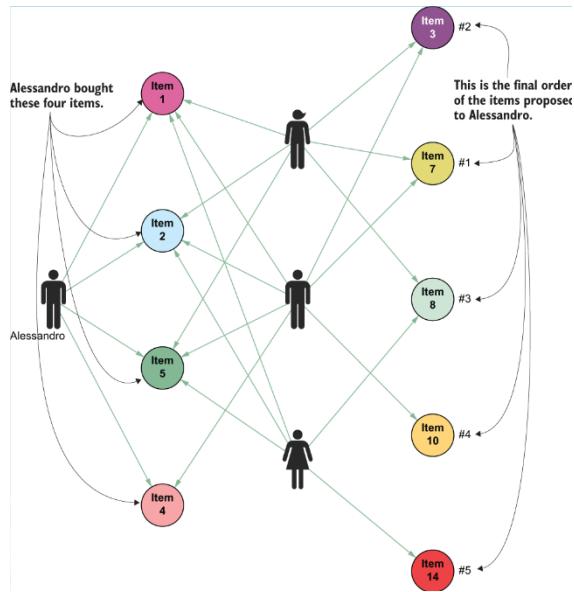


- **Item Analyser:** Estrarre o identificare le caratteristiche rilevanti (chiamate anche proprietà o attributi) e rappresentare gli elementi: prende come input il contenuto dell'elemento (il contenuto di un libro o la descrizione di un prodotto) e le metainformazioni (l'autore del libro, gli attori di un film...) da più di 1 fonte di informazioni e le converte in un modello di elemento che viene utilizzato in seguito per fornire raccomandazioni.
- **User Profile Builder:** Questo processo raccoglie dati rappresentativi delle preferenze dell'utente e ne deduce i profili utente, interrogando gli utenti sui loro interessi o raccogliendo feedback impliciti osservando e memorizzando il comportamento dell'utente. Il risultato è un modello, in particolare un modello grafico, che rappresenta l'interesse dell'utente per un elemento, una caratteristica dell'elemento o entrambi.
- **Recommendation Engine:** Sfrutta i profili utente e le rappresentazioni degli elementi per suggerire elementi pertinenti abbinando gli interessi dell'utente alle caratteristiche dell'elemento. In questa fase, viene creato un modello predittivo che viene utilizzato per creare un punteggio di pertinenza per ciascun elemento per ciascun utente. Questo punteggio viene utilizzato per classificare e ordinare gli elementi da suggerire all'utente.

In un CBR-S, esistono diversi metodi per raccogliere e modellare i profili utente. Il modello di progettazione selezionato varierà a seconda di come vengono raccolte le preferenze (implicitamente o esplicitamente) e del tipo di strategia di filtraggio o approccio di raccomandazione. Un modo semplice per raccogliere le preferenze dell'utente è chiedere direttamente all'utente. L'utente potrebbe esprimere interesse per generi o parole chiave specifici, o per attori o registi specifici. Da una prospettiva di alto livello, lo scopo del profilo utente e del modello definito è quello di aiutare il motore di raccomandazione ad assegnare un punteggio a ciascun elemento o caratteristica dell'elemento. Il punteggio aiuta a classificare gli elementi suggeriti all'utente specifico, ordinati dal più alto al più basso. Per questo motivo, i sistemi di raccomandazione appartengono all'area del machine learning chiamata "learning to rank".

## 8.1 Collaborative Filtering

Un’alternativa al filtraggio dei contenuti si basa esclusivamente sul comportamento passato degli utenti, come transazioni precedenti o valutazioni degli articoli, o sulle opinioni di una community di utenti esistente, per prevedere quali articoli saranno più probabilmente apprezzati o interessati agli utenti, senza richiedere la creazione di profili esplicativi sia per gli articoli che per gli utenti in base alle caratteristiche degli articoli. Questo approccio è noto come **Collaborative Filtering**, un termine coniato dagli sviluppatori di Tapestry [Goldberg et al., 1992], il primo sistema di raccomandazione. Il filtraggio collaborativo analizza le relazioni tra gli utenti e le interdipendenze tra gli articoli per prevedere nuove associazioni utente-articolo.



### Vantaggi

- È domain free
- Non ha bisogno di nessun dettaglio sugli items
- Può essere applicato ad una grande varietà di casi, e può trovare informazioni rilevanti difficili da vedere con content filtering

### Svantaggi

- Il problema dell'avvio a freddo è dovuto all'incapacità di fornire raccomandazioni ragionevoli (in termini di accuratezza) per nuovi elementi e utenti o quando sono disponibili dati di interazione limitati. Esistono tuttavia meccanismi che attenuano l'effetto del problema dell'avvio a freddo utilizzando diversi algoritmi, come l'approccio del grafo o altre fonti di conoscenza, come i social network.

Quindi per risolvere il problema del cold start, un primo approccio potrebbe essere quello di valorizzare informazioni aggiuntive sull'utente, come il gender, educazione, interessi... questo aiuterebbe nella sua classificazione (quindi aggiungere informazioni per migliorare la profilazione); il secondo approccio è creare sistemi per raccomandazioni ibridi, che uniscono diversi approcci in un singolo meccanismo di predizione.

**User-based approach** Se due utenti avevano gusti simili in passato, avranno gusti simili anche in futuro. Le preferenze degli utenti rimangono stabili e coerenti nel tempo. Sulla base di questa assunzione, possono succedere due cose:

- Interazione tra users e items non ha peso. Questo caso è chiamato modello booleano o binario.

$$score(a, b) = \frac{1}{|KNN(a)|} \times \sum_{b \in KNN(a)} r_{b,p}$$

- Interazione pesata. Con una formula che prevede i ratings che un user assegnerà ad un item.

$$pred(a, p) = \frac{\sum_{b \in KNN(a)} sim(a, b)}{\sum_{b \in KNN(a)} |sim(a, b)|} \times r_{b,p}$$

Quello che si vuole fare è una media pesata dei rating degli utenti simili.

**Item-based approach** Questo tipo di approccio permette il calcolo in real-time. Ovviamente l'idea alla base è quella di usare similarità tra gli elementi e cercare elementi simili piuttosto che tra gli users. La formula per prevedere il rating di un elemento non ancora presente in un dataset è:

$$pred(a, p) = \frac{\sum_{q \in ratedItem(a)} (sim(p, q) \times r_{a,q} \times |KNN(q) \cap \{p\}|)}{\sum_{q \in ratedItem(a)} (sim(p, q) \times |KNN(q) \cap \{p\}|)}$$

Dove  $q$ , simboleggia ogni elemento considerato e valutato dall'user  $a$ . Per ogni  $q$ , moltiplica tre valori:

- la similarità tra  $q$  e il target product  $p$ ,
- il rating assegnato a  $q$  dall'user,
- $|KNN(q) \cap \{p\}|$  è 1 se  $p$  si trova nel set dei NN, altrimenti è 0

Il denominatore normalizza il valore per non eccedere nella valutazione del rating.

## Summary

Gli aspetti e i vantaggi principali dell'approccio basato su grafi ai motori di raccomandazione per collaborative filtering implementati con metodi NN sono i seguenti:

- Il dataset Utente-Elemento può essere facilmente rappresentato come un grafo bipartito, in cui il peso di ciascuna coppia utente-elemento è rappresentato come una proprietà opzionale della relazione.
- La rappresentazione a grafo bipartito del dataset Utente-Elemento non solo ha il vantaggio di consentire di memorizzare solo le informazioni rilevanti, evitando di sprecare memoria memorizzando zeri inutili, come nella rappresentazione a matrice, ma anche di velocizzare l'accesso durante la creazione del modello concentrandosi solo sui vicini potenzialmente rilevanti.
- È possibile estrarre diverse rappresentazioni vettoriali sia per gli elementi che per gli utenti dallo stesso modello di grafo.

- Il modello risultante, costituito da somiglianze tra utenti o elementi o entrambi, può essere rappresentato naturalmente come nuove relazioni che collegano utenti ed elementi. I nuovi grafi risultanti sono le reti del vicino più prossimo (k-NN).
- L'algoritmo, basato sul calcolo della similarity, per la creazione delle reti k-NN rappresenta una delle tecniche più potenti e ampiamente adottate per la costruzione di grafi. Le reti risultanti non solo sono facili da navigare durante il processo di raccomandazione, ma contengono anche nuove conoscenze, ricavate dal dataset Utente-Elemento esistente, che possono essere utilizzate per analizzare i dati da altre prospettive, come il clustering degli elementi, la segmentazione dei clienti e così via.
- Una tecnica ampiamente adottata per risolvere il problema della scarsità di dati e il problema dell'avvio a freddo si basa sulla rappresentazione, la navigazione e l'elaborazione dei grafi. Anche in questo scenario, è possibile combinare più algoritmi di raccomandazione in un singolo grafo e combinare la potenza di più approcci per fornire raccomandazioni.

## 8.2 Fraud Detection

Alcune delle tecniche di rilevamento delle frodi che vedremo provengono dal campo più generico del rilevamento di anomalie o valori anomali. Un'anomalia o un valore anomalo si riferisce a un data point che differisce significativamente dagli altri. Nel contesto delle frodi, questo si verifica in comportamenti (come le transazioni) che si discostano dal comportamento abituale di un individuo, il che può essere un indicatore di attività fraudolenta. Oltre a rivelare comportamenti sospetti o anomali in ambito finanziario, il rilevamento delle anomalie è fondamentale per individuare eventi rari come epidemie di malattie rare o effetti collaterali in ambito medico, con applicazioni vitali nella diagnosi medica. Un'altra applicazione del rilevamento delle anomalie è la pulizia dei dati, ovvero la rimozione di valori errati o rumore dai dati come fase di pre-elaborazione per consentire l'apprendimento di modelli più accurati dei dati.

Fraud is an uncommon, well-considered, time-evolving, carefully organized and imperceptibly concealed crime that appears in many different types and forms.

Quindi come si fa **Fraud detection**? Il rilevamento delle frodi si riferisce alla capacità di riconoscere o scoprire le frodi. Entra in gioco quando la prevenzione delle frodi fallisce, ma poiché non è sempre ovvio quando ciò accade, è necessario utilizzare costantemente misure di rilevamento delle frodi. Possiamo fare del nostro meglio per prevenire le frodi sulle carte di credito, ma se i dati di una carta vengono rubati, dobbiamo essere in grado di rilevare l'uso fraudolento il prima possibile. Quando parliamo di **Fraud prevention** facciamo invece riferimento a tutte quelle contromisure volte a limitare l'efficacia delle frodi.

### Rule-based engine

Un **rule-based engine** per la **fraud detection** è un sistema che rileva comportamenti sospetti applicando un insieme di *regole predefinite*. Queste regole sono definite da esperti del dominio e descrivono condizioni logiche che, se soddisfatte, indicano potenziali frodi.

Le regole possono essere formulate come condizioni del tipo:

- Se l'importo di una transazione supera i 10.000 € e proviene da un paese non abituale, allora segnalare come sospetta.
- Se un utente effettua più di 5 acquisti in meno di 5 minuti, allora indicare potenziale frode.

- Se si verificano più di 3 login falliti da un indirizzo IP sconosciuto, bloccare l'account.

Il processo tipico di un rule-based engine comprende i seguenti passi:

1. **Definizione delle regole:** Le condizioni vengono scritte in pseudocodice, linguaggi SQL, DSL o strumenti specifici come *Drools* o *CLIPS*.
2. **Input dei dati:** Il sistema riceve dati in tempo reale o in batch, comprendenti informazioni su utenti, transazioni, IP, dispositivi, ecc.
3. **Applicazione delle regole:** Le regole vengono valutate sui dati ricevuti. Se una regola è soddisfatta, si può bloccare l'operazione, inviarla per revisione o aggiornarne lo score di rischio.
4. **Logging e audit:** Ogni decisione è tracciata per garantire trasparenza e conformità.

I **vantaggi** di questo approccio includono:

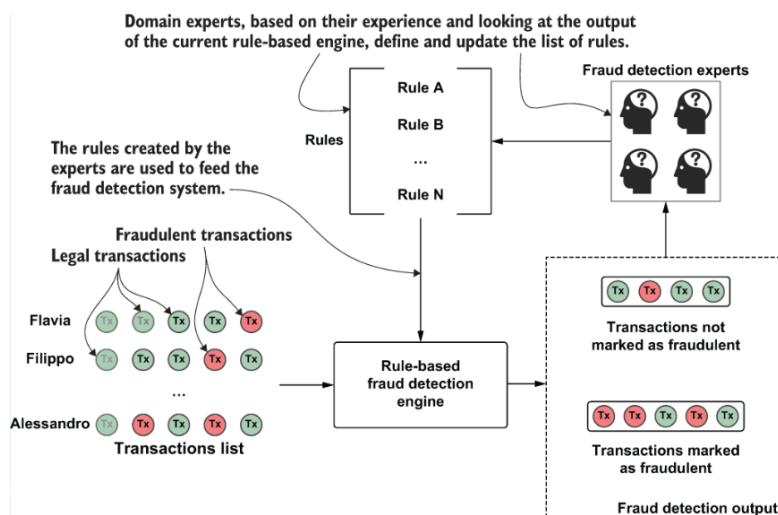
- Alta *spiegabilità*: è chiaro il motivo di ogni decisione.
- Semplicità di *manutenzione* e aggiornamento.
- Controllo totale sul comportamento del sistema.

Tuttavia, presenta anche dei **limiti**:

- *Rigidità*: non si adatta a nuovi schemi di frode.
- Gestione complessa con molte regole.
- *Falsi positivi* elevati se le regole sono troppo generiche.

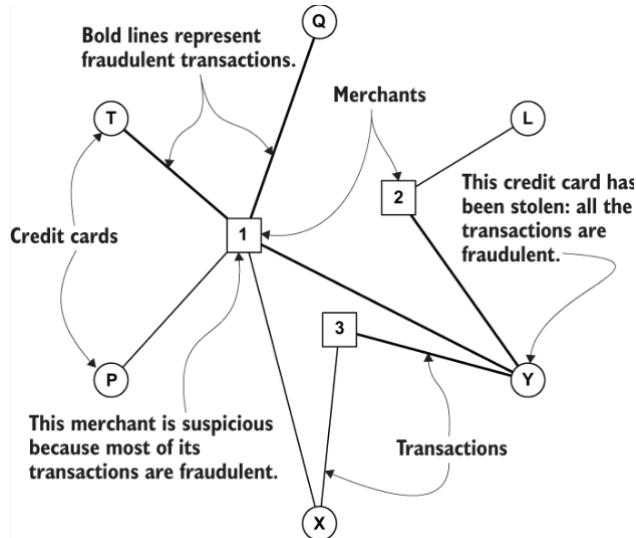
Nelle implementazioni moderne, spesso il rule-based engine è integrato con modelli di *machine learning*, formando un sistema ibrido dove:

- Le regole gestiscono i casi evidenti.
- I modelli predittivi analizzano pattern più complessi e adattivi.



## Graphs for Fraud Detection

I grafici forniscono un potente strumento di modellazione e analisi per catturare correlazioni a lungo raggio tra oggetti di dati interdipendenti, il che li rende particolarmente adatti allo scenario di lotta alle frodi. Le transazioni sui nostri conti bancari e sulle carte di credito seguono una logica legata a ciò che facciamo, a dove viviamo, a ciò che ci piace acquistare e così via. Inoltre, le frodi raramente si verificano in modo isolato: dietro ogni frode c'è un piano che prevede una preparazione, che spesso richiede la cooperazione tra più truffatori.

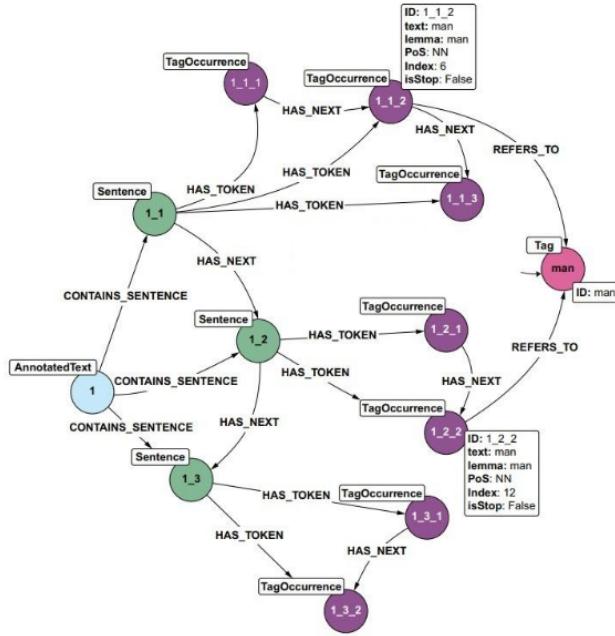


## 8.3 Graphs for NLP

I motori di ricerca, i sistemi di raccomandazione e gli assistenti vocali hanno i seguenti requisiti in comune: l'uso del testo per la creazione di una knowledge base, una rappresentazione adeguata della conoscenza che memorizzi tutte le informazioni necessarie per l'ambito dell'applicazione e interazione con l'utente via natural language. L'elaborazione del linguaggio naturale (NLP) è un campo interdisciplinare che utilizza concetti provenienti dall'informatica, dall'intelligenza artificiale e dalla linguistica allo scopo di elaborare e analizzare grandi quantità di dati in linguaggio naturale. I dati prodotti dall'attività di *Natural Language Processing* (NLP) hanno una natura altamente interconnessa. Memorizzare i risultati in un modello a grafo sembra essere una scelta logica e razionale.

- In alcuni casi, come nel caso delle **dipendenze sintattiche**, le relazioni vengono generate come output dell'attività di NLP e il grafo deve solo memorizzarle.
- In altri casi, il modello è stato progettato per servire più ambiti contemporaneamente, fornendo strutture dati facili da navigare.

Il modello a grafo qui proposto non solo memorizza i dati principali e le relazioni estratte durante il processo di *Information Extraction* (IE), ma consente anche un’ulteriore estensione aggiungendo nuove informazioni calcolate in una fase di post-elaborazione: *calcolo della similarità*, *estrazione del sentiment* e così via.



Di conseguenza, con uno sforzo relativamente minimo, possiamo soddisfare il caso d’uso del **suggerimento di parole**, così come esigenze di ricerca **più complesse** e persino la **risposta a domande** (“Come sono le mele?”).

## 9 Knowledge Graphs

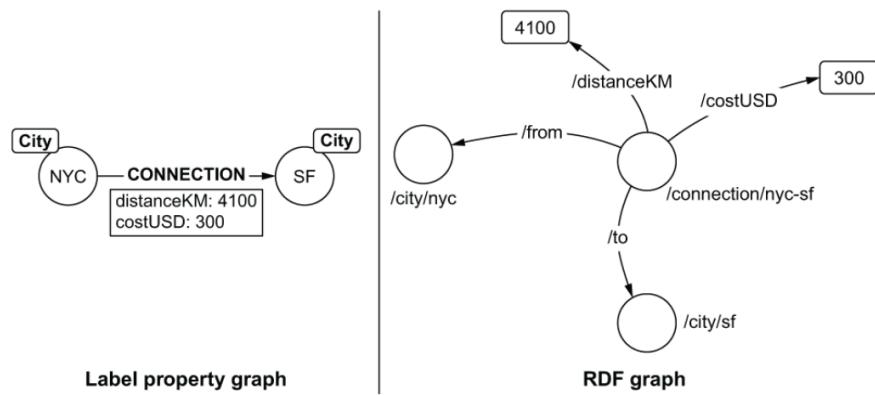
“A knowledge graph consists of a set of interconnected typed entities and their attributes.”

Nella rappresentazione e nel ragionamento della conoscenza, il knowledge graph è una base di conoscenza che utilizza un modello di dati o una topologia strutturata a grafo per integrare i dati. I knowledge graph sono spesso utilizzati per memorizzare descrizioni interconnesse di entità, oggetti, eventi, situazioni o concetti astratti, codificando anche la semantica sottostante la terminologia utilizzata. Sono ampiamente associati e utilizzati da motori di ricerca come Google, motori di conoscenza e servizi di domande e risposte come WolframAlpha, Siri di Apple e Amazon Alexa; e social network come LinkedIn e Facebook. Rispetto ad altri sistemi informativi orientati alla conoscenza, i knowledge graph si distinguono per la loro particolare combinazione di:

- Strutture di rappresentazione della conoscenza e ragionamento, come linguaggi, schemi, vocabolari standard e gerarchie tra concetti;
- Processi di gestione delle informazioni (come le informazioni vengono assimilate e trasformate in un knowledge graph);
- Modelli di accesso ed elaborazione, come meccanismi di interrogazione, algoritmi di ricerca e tecniche di pre e post-elaborazione.

**Definizione** Un knowledge graph è un grafo costituito da concetti, classi, proprietà, relazioni e descrizioni di entità. I dati possono essere aperti, privati o chiusi; possono inoltre essere originali, derivati o aggregati.

I knowledge graphs vengono generalmente rappresentati tramite l'utilizzo di Resource Description Frameworks (RDF). La struttura sottostante di qualsiasi espressione in RDF è una raccolta di triple, ciascuna composta da un soggetto, un predicato e un oggetto. Ogni tripla può essere rappresentata come un collegamento nodo-arco-nodo, chiamato anche grafo RDF.



Un Knowledge Graph è un insieme di dati ed è:

- **Strutturato** (sotto forma di una specifica struttura dati)
- **Normalizzato** (costituito da piccole unità, come vertici e archi)

- **Connesso** (definito dalle connessioni - anche distanti (non dirette) - tra gli oggetti)

Inoltre, i Knowledge Graph sono in genere:

- **Espliciti** (creati appositamente con un significato specifico)
- **Dichiarativi** (significativi di per sé, indipendenti da una particolare implementazione o algoritmo)
- **Annotati** (arricchiti con informazioni contestuali per registrare ulteriori dettagli e metadati)
- **Non gerarchici** (più di una semplice struttura ad albero)
- **Grandi dimensioni** (milioni anziché centinaia di elementi)

## 9.1 Creating Knowledge Graphs

La creazione di *knowledge graphs* (KG) può essere affrontata attraverso diversi approcci, che si distinguono per livello di automazione, qualità, e scalabilità. Di seguito vengono descritti i principali paradigmi.

### 1. Approcci curati (*Curated approaches*)

- I **tripli** (soggetto, predicato, oggetto) vengono creati manualmente da un gruppo chiuso di esperti.
- Tali approcci sono **altamente accurati**, grazie all'intervento umano esperto.
- Tuttavia, **non sono scalabili** a causa dell'alto costo e tempo richiesto.
- **Esempi** noti:
  - *Cyc/OpenCyc*
  - *WordNet*
  - *UMLS* (Unified Medical Language System)
  - *SNOMED CT* (Systematized Nomenclature of Medicine Clinical Terms)

### 2. Approcci collaborativi (*Collaborative approaches*)

- I triplici vengono creati manualmente da una comunità aperta di volontari.
- Questi approcci **scalano meglio** rispetto a quelli curati, grazie al contributo distribuito.
- **Esempi:**
  - *Wikidata*
  - *Freebase*
- **Criticità:**
  - *Incompletezza*: ad esempio, il 71% delle persone in Freebase manca del luogo di nascita obbligatorio.
  - *Crescita rallentata*: sia Wikipedia che Wikidata mostrano un tasso di crescita decrescente.

### 3. Approcci automatici da testo semi-strutturato (*Automated semi-structured approaches*)

- I triple vengono estratti automaticamente da testi semi-strutturati tramite:
  - Regole scritte manualmente
  - Regole apprese automaticamente
  - Espressioni regolari
- Si sfruttano fonti come le **infobox di Wikipedia**, che offrono dati strutturati in modo parziale.
- Questi approcci offrono un buon **compromesso tra accuratezza e scalabilità**.
- **Esempi:**
  - *YAGO*
  - *DBpedia*
  - *Freebase*
- **Accuratezza** stimata:
  - *YAGO2*: oltre il 95% (verifica manuale)
  - *Freebase*: circa il 99%
- **Limiti**: coprono solo una piccola parte delle informazioni presenti sul Web.

### 4. Approcci automatici da testo non strutturato (*Automated unstructured approaches*)

- I triple sono estratti automaticamente da testo **non strutturato**, come pagine Web o articoli.
- Si utilizzano tecniche di **machine learning** e **natural language processing (NLP)**.
- I sistemi imparano a identificare entità, relazioni e concetti direttamente da testi in linguaggio naturale.
- **Esempi** di sistemi:
  - *NELL* (Never-Ending Language Learning)
  - *Knowledge Vault*
  - *PATTY, PROSPERA*
  - *DeepDive, Elementary*
  - *ReVerb, OLLIE, PRISMATIC*
- **Sfide**:
  - Presenza di **rumore** (informazioni errate o imprecise).
  - La qualità può essere migliorata integrando **conoscenze da repository esistenti e affidabili**.

## 5. Collegamento di dataset esistenti (*Linking existing datasets*)

- Diversi dataset vengono interconnessi utilizzando principi di **linked data**.
- Questo approccio arricchisce il grafo di conoscenza, mantenendo coerenza e interoperabilità tra fonti diverse.

## 9.2 Knowledge Graphs and Important Nodes

Nei *knowledge graphs*, la determinazione dell'**importanza dei nodi** non si basa unicamente sulla loro connettività, ma può dipendere da ulteriori elementi semanticamente e strutturali. In particolare, possono influenzare l'importanza:

- Le **proprietà associate agli archi** (*edge attributes*), che rappresentano il tipo e il significato delle relazioni tra nodi. Ad esempio, un arco `hasSpouse` può essere considerato più rilevante di un arco `likesMusicGenre`.
- Le **etichette dei nodi** (*node labels*) o altri attributi semanticamente dei nodi stessi. Queste etichette possono fornire informazioni sul tipo di entità (es. `Person`, `Organization`) o sul loro ruolo nel grafo.
- La possibilità di **ignorare nodi o relazioni specifiche** che non aggiungono informazione utile. Ad esempio:
  - In un knowledge graph codificato in *OWL* (Web Ontology Language), ogni entità ha una tripla del tipo:

`:entity rdf:type owl:Thing`

- Poiché questa informazione è condivisa da tutte le entità, può essere considerata **ridondante** o **irrilevante** nel calcolo dell'importanza, e quindi ignorata.

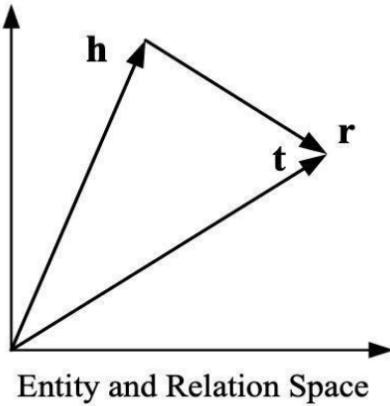
In sintesi, la valutazione dei nodi più rilevanti in un knowledge graph dovrebbe considerare non solo la struttura topologica, ma anche la semantica implicita nei dati e nei metadati associati.

## 9.3 Semantic Similarity

In un Knowledge Graph, entità simili sono rappresentate da nodi con connessioni simili; per identificare entità simili dobbiamo trovare strutture simili. L'idea è trovare gli embedding di nodi in un spazio vettoriale dimensionalmente più piccolo in modo che vettori simili siano nodi nello stesso embedding. Per farlo:

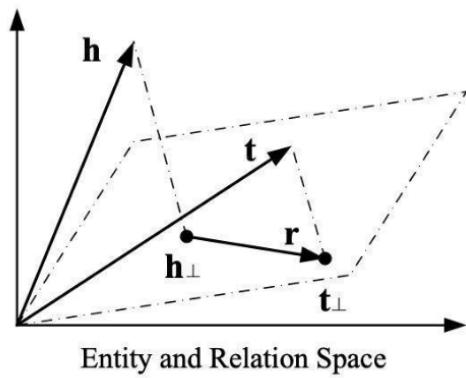
- **Translational methods:** TransE, TransH, TransR, TransEdge,... che utilizzano funzioni di score distance-based
- **Rotation based methods:** RotatE
- **Graph Convolutional Networks:** R-GCN, TransGCN
- **Walk-based methods:** DeepWalk, RDF2Vec

## TransE



Entità e relazioni sono mappati nello stesso spazio vettoriale. La relazione  $r$  viene considerata una traslazione dalla testa alla coda ma ha problemi con le funzioni simmetriche.

## TransH



Dallo spazio originale ad un iperpiano, TransH permette diversi ruoli di una entità all'interno di due relazioni diverse. Le entities  $h$  e  $t$  vengono proiettate in un iperpiano di relazione  $r$ .

## 9.4 Knowledge Graph Refinement

Come modello del mondo reale o di una sua parte, i grafi della conoscenza non possono ragionevolmente raggiungere una copertura completa, ovvero contenere informazioni su ogni singola entità dell'universo. È improbabile, in particolare se si applicano metodi euristici per la costruzione di un grafo della conoscenza, che il grafo della conoscenza sia completamente corretto.