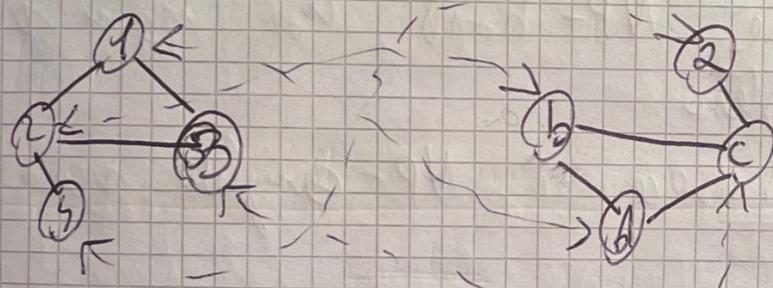


## Graph Matching

Graph nel senso informatico fra grafi.  
Dobbiamo verificare se 2 grafi hanno lo stesso contenuto.

$G_1$  e  $G_2$  sono isomorfi se esiste  $\rho: V_1 \rightarrow V_2$  biunivoco tale che  $(u, v) \in E_1$  e se e solo se  $(\rho(u), \rho(v)) \in E_2$ .

In altri termini si parla di isomorfismo se i nodi di  $G_1$ , coi nomi di  $G_2$  mantengono le corrispondenze tra gli altri.



| Mapping           |
|-------------------|
| 1 $\rightarrow$ a |
| 2 $\rightarrow$ b |
| 3 $\rightarrow$ c |
| 4 $\rightarrow$ d |

Cioè tramite la funzione mapping si applica a  $G_1$  otengo  $G_2$  e viceversa.

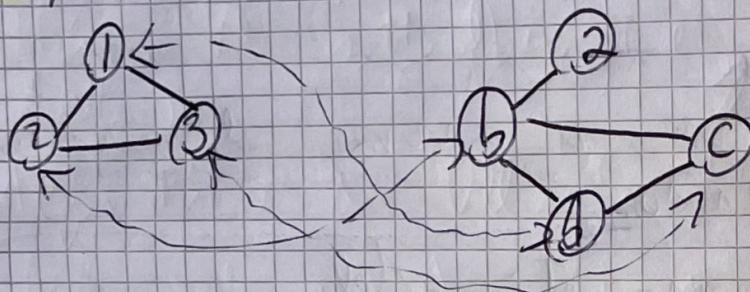
La funzione è biunivoca.

## Sub-graph matching

$G_1$  è sottografo isomorfo di  $G_2$  se esiste

una funzione iniettiva  $f: V_1 \rightarrow V_2$  detta mapping tale che  $(u, v) \in E_1$ , e  $(f(u), f(v)) \in E_2$

La funzione è <sup>non</sup> iniettiva, non è necessario che tutti i nodi di  $G_2$  (il grande) vengano mappati. Imposta che vengano mappati solo i nodi di  $G_1$ .

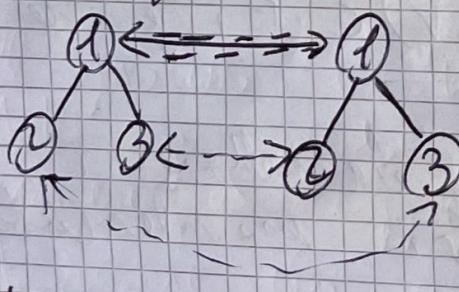


Mapping  
 $1 \leftrightarrow a$   
 $2 \leftrightarrow b$   
 $3 \leftrightarrow c$

Potrebbero esserci più soluzioni in quanto potrei mappare 1 in b e 2 in a. Più soluzioni si potrebbero avere anche nel graph matching classico.

## Automo spino

homomorfico tra un grafo  $G$  e un altro.



Il graph matching è NP-hard ma meno se il grafo è NP-completo.

Sub-graph matching invece è NP-completo risolvere.

## Algoritmi

Si triviscono la forma canonica del grafo per  
visualizzare ma rappresentazione  
La rappresentazione è chiave ed è una struttura.

Questi algoritmi sono efficienti perché trovare la  
forma canonica è semplice

Un granli tool Nauty è il migliore algoritmo  
ma in generale non c'è un algoritmo  
migliore di altri. Nauty è detto anche Trivial

Molti di questi tool / algoritmi hanno utility  
interessanti (Fis: generare tutti i possibili grafi  
connessi su  $n$  nodi)

## Soluzione Brute-force ← → Subgraph Matching

E' ottenere un albero con root (non radice)  
e non binario.

- 1) 1° nodo lo mappa con ogni possibile nodo  
del 2° grafo.
- 2) 2° nodo lo mappa con ogni possibile nodo  
risparmiato (nel sottoalbero)
- ⋮

La soluzione brute-force cerca tutte le  
possibili combinazioni per poi verificare solo  
quelle i buone

In realtà non c'è bisogno di costruire un  
sottoalbero se non sono utilizzate le condizioni

Il metodo brute-force ha complessità polinomiale  
Dobbiamo evitare di esplorare tutto l'albero  
tramite qualche strategia.

- Look-ahead: Prende due il matching puzzle molto  
uno prima e una soluzione finale
  - Backtrack King: abbando le soluzioni forzate se ci  
sono solo che non portano a soluzione
- Quante tecniche per evitare di esplorare zone (inclusi altri) i problemi basta molti gradi.

Se è un nodo mappa 2 in b, ~~se~~ ha un  
grado diverso, non ha nessun estremo per  
ottenere il matching (è vicino che non trovi  
nulla)

### Algoritmo di Ullman

Usa il grado del nodo per filtrare i  
nostri alberi.

Se il grado del nodo target è minore o  
uguale al grado del nodo di target estremo  
estremamente no.

### Algoritmo di VF lib

E' base sul concetto di State Space representation

Il processo è una successione di stati.

Ad ogni stato  $s$  è associato  $M(s)$ , un  
insieme di coppie di nodi già mappati

Aaggiungere una ~~coppia~~ coppia nuova di nodi da  
mappare in  $M(s)$  per trasmettere da uno stato  
all'altro

Ogni coppia ha ogni bene 2 vere un  
insieme di regole delle regole di fattibilità

$M(s) \equiv$  Matching puzzle

## Prembo colice VF lib

Match(1)

$S \equiv \epsilon$  uno stato

IF  $M(1)$  come tutti i modi del grafo giochi Then  
Then aggiungi  $M(1)$

Else Cerca  $P(1)$  ovvero i nuove possibili copie  
combinazione di modi da aggiungere a  $M(1)$

For Each  $P \in P(1)$

IF  $P$  soddisfa le regole di fattibilità

THEN Aggiungi  $P$  a  $M(1)$

Cerca stato  $\gamma'$  ottenuto dopo aver

aggiunto  $P$  a  $M(1)$

CALL Match( $\gamma'$ )

## Regole di Fattibilità

Per  $G_1 \Rightarrow M_1, T_1, V_1'$

$G_1 \equiv \text{Query}$

Per  $G_2 \Rightarrow M_2, T_2, V_2'$

$G_2 \equiv \text{Target}$

In  $M_1$  e  $M_2$  vi sono modi di rapporto

In  $T_1$  e  $T_2$  vi sono modi accentrati a quelli  
delle risposte (ma non ancora mappati)

In  $V_1'$  e  $V_2'$  vi sono gli altri modi

$P(1)$  è sostituito dalle copie  $(M, m)$  dove

$M \in T_1(1)$  e  $m \in T_2(1)$

$(M, m) \in P(1)$  viene aggiunto a  $M(1)$  se:

1)  $\forall M' \in M_1(1)$  corrisponde a  $m$ , il corrispondente  $M' \in M_2(1)$

2) è corrispondente a  $m$ . Regole di corrispondenza dello stato

il numero di modi corrispondenti a  $M$  in  $T_1(1)$  è minore  
o maggiore al numero di modi in  $T_2(1)$  corrispondenti a  $m$  (Regole look ahead a 1 livello)

3) Il numero di nodi in  $N_1'(1)$  è minore o uguale al numero di nodi in  $V_2'(1)$ .  
 Il numero di nodi in  $N_1'(1)$  è minore o uguale al numero di nodi in  $V_2'(1)$ .  
 Comunque a  $m$ . (Regole look ahead a 2 livelli)

### Nei passi Notti ci sono 3 regole

$(m, m)$  in  $P(1)$  è raggiunto in  $M(1)$  se:

- 1) Per ogni  $m' \in M_1(1)$  predecedente a  $m$ , il corrispondente  $m' \in M_2(1)$  è predecessore di  $m$ .
- 2) Per ogni  $m' \in M_1(1)$  predecessore di  $m$ , il corrispondente  $m' \in M_2(1)$  è predecessore di  $m$ .

3) Il numero di nodi in  $T_1^{in}(1)$  è minore o uguale al numero di nodi in  $T_2^{in}(1)$ .  
 Comunque a  $m$

4) Il numero di nodi in  $T_1^{out}(1)$  è minore o uguale al numero di nodi in  $T_2^{out}(1)$ .  
 Comunque a  $m$

5) Il numero di nodi in  $V_1'(1)$  è minore o uguale al numero di nodi in  $V_2'(1)$ .  
 Comunque a  $m$

Per concretizzare l'algoritmo e trovare la complessità  
 basta contare  $\leq 2 =$  mille regole.

### Combinazioni con $N$ nodi

Ogni nodo ha  $\Theta(N)$  adiacenti, allora il costo di esplorazione di un singolo nodo è  $\Theta(N)$ .

Caso migliore: ogni passo un solo nodo combinabile soddisfa le regole. Ci sono  $N$  passi.  
 $\Theta(N^2)$ , peggio:  $\Theta(N^3)$ .

Caso peggiore: devo esplorare l'intero albero

$\Theta(N!N)$ ,  $\Theta(N!N^3)$  per ultimo

c'è  $\frac{N!}{N!N!}$

Complessità spazio:  $\Theta(N^2)$ , Utente  $\Theta(N^3)$

Menziono le informazioni relative ad uno stato.  
Per ogni nodo ho una quantità di info ovvero  
mapping e appartennenza a un inserv.

Al massimo  $N$  stati (con le informazioni associate)  
stanno in memoria, i più numerosi  
con certi criteri dell'elaborazione

VF2 ottimizza lo spazio utile e ha complessità  
 $\Theta(N)$  spazio

Viamo sulle informazioni globali contiene traci  
per tutti gli ~~ri~~ si evita la ~~sovra~~ sovrapposizione  
di diverse sorgenti delle info sui nodi dello stato

• Core\_1, Core\_2 contengono il mapping corrente  
Core\_1 [ $M$ ] =  $m$  se  $m$  è in loro rapporto  
ovvero

• in\_1, out\_1, in\_2 e out\_2 descrivono  
l'appartenenza dei nodi agli inservi terminali  
Il valore memorizzato corrisponde alla probabilità  
nell'albero. Si fissa il nodo dello stato in cui il  
nodo è entrato nell'appartenenza corrispondente insieme

Così teniamo traccia sia dell'appartenenza del nodo  
agli inservi terminali e dello stato della  
computazione corrispondente.

E se fa backtracking: riguarda solo gli ultimi

## RI

L'algoritmo RI è in base all'ordine  $\sigma_m$  cui sono ordinati i nodi della query nell'altro di riga. Un ordinamento efficiente velocizza il matching. L'ordine  $\sigma$  è indipendente dal grafo Target in base static ordering.

Altimenti è fatto dynamic ordering.

RI è basa su static ordering dei nodi della query.

## Algoritmo RI

### Pre Processing

1) Ordina i nodi della query in modo che la probabilità di trovarne un comune nell'altro sia più grande.

2) Esplorazione, fatto sotto forma di ricerca:

Si segue l'ordine trovato nel pre processing.

Si mappano i nodi della query a nodi del target. Usando la regola del grafo (la stessa di Viterbi). Quando si trova un match completo  $\Rightarrow$  Menzione decisiva.

3) Ricerca 2 finché non vi è esplorato l'intero spazio.

## Ordinamento \*

Succede molto da innanzitutto è quello che ha

1) Max valore di  $|V_m, v_{IS}|$

2) Se c'è partita in 1: max valore di  $|V_m, n_{IS}|$

3) Partita in 2: max valore di  $|V_m, v_{mV}|$

4) Partita in 3: scelta arbitraria

Dalle loro le corrispondenze! (quindi entrambi gli nomi di calcolo si risolvono nell'caso vero oppure non sarà accettata)

( $u_i, M(u_i)$ ) è accettata  
se e solo se:

- 1) Né  $u_i$  e  $M(u_i)$  sono già stati mappati nella tabella principale
- 2) I due nodi hanno la stessa chiave primaria ( $\pi$  c'è)
- 3) Il numero di nodi comuni a  $M(u_i)$  è maggiore o uguale al numero di nodi comuni a  $u_i$

~~π~~  $U$  inoltre è il nodo con più nodi

\* Data  $O^{m-1} = (u_1, \dots, u_{m-1})$ , che è la sequenza ordinata, il conteggio di un nodo comune a  $U$  definisce nei 3 insiem

- 1)  $V_{M, V_1}$ : nodi in  $O^{m-1}$  simili a  $v$
- 2) ~~π~~  $V_{M, \text{match}}$ : nodi in  $O^{m-1}$  ~~comuni a~~ simili a  $v$  e altrimenti un nodo  $m_m$  in  $O^{m-1}$
- 3)  $V_{M, UV}$ : nodi comuni a  $v$  che non sono in  $O^{m-1}$  e  $m_m$  sono comuni a nodi in  $O^{m-1}$

### Graph Matching in DB & graph

Con un database di tipi app e un query Q  
Voglio trovare tutti i graph che appartengono a classe  $C$  (o come query)

Soluzione banale: graph matching da Q su ogni target, è poco efficiente

Per migliorare la cosa abbiamo indicizzare i graph sul database

Dunque con l'individuazione ottimale la ~~query~~  $\rightarrow$  query sui nodi del graph

## Individuazione base su features

Rappresenta il grafo con un insieme  $F$  di  
oggetti.

Dunque se un grafo non ha le stesse feature  
della query si può a priori che non può  
essere fatto dall'algorithmo.

## Individuazione non basata su features

Grafi memorizzati in uno hash tramite B-tree.

Questo metodo è indicato per database dinamico.

## Possibili features

- Ricorsi grafi
  - Alberi
  - Cammini (SING)
- ← più facilmente citabili

Esempio Cammin lungo 22a 2 slide

○ □ △ etichette nodi

## Algorithmi base

- 1) Preprocessing: da ogni grafo si estraggono le features.  
Per ogni feature viene creato quale grafo lo contiene  
(e quante volte)
- 2) Filtering: dalla query si estraggono le features e si trovano i grafi del database  
contenuti (ovvero quelli che hanno le features  
della query)
- 3) Matching: per ogni grafo contenuto si applica  
con algorithmi di graph matching per rientrare  
le contiene la query

Individuazione inversa del Preprocessing

$$p_1 \rightarrow g_1, g_2, g_3$$

$$p_2 \rightarrow g_1, g_3$$

$$p_3 \rightarrow g_3$$

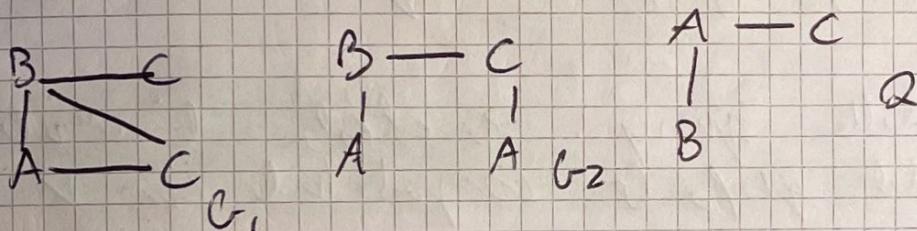
La  $p_i$  sta in  $g_1, g_2$  e  $g_3$

Nella cella di ogni lista oltre a un grido c'è l'occorrenza

Trovare le  $p$  della query e tramite operazioni di intersezione vediamo quali grafici contengono tutte le feature di  $Q$  (tenendo conto anche dell'occorrenza)

### Algoritmo SING

Agli ogni feature spazio associare non solo l'occorrenza ma anche il nodo "iniziale"



AB e AC stanno sia in  $G_1$  che  $G_2$  ma solo  $Q$  contiene  $Q$

Dunque succede perché AB e AC partono dallo stesso nodo in  $G_1$  ~~ma~~ e da nodi diversi in  $G_2$

Avremo 2 indici:

- indice inverso globale (quello  $G_1$ )
- indice inverso locale; ne abbiamo 1 per grafo, riportata come una bitmap

$$p_1 \rightarrow \begin{matrix} v_1 \\ v_3 \end{matrix}$$

$$p_2 \rightarrow \begin{matrix} 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ \nwarrow v_1 & \nearrow v_6 \end{matrix}$$

Indice da qual v parte la feature

## Procedura di query in SING

- 1) Filtering 1: Per ogni  $G \in Q$  prendi i grafici in cui  $P$  compare con numero di volte maggiore o uguale al numero di occorrenze in  $Q$ . Di questi inserisci facendo l'intersezione e lo chiamiamo  $R$ .
- 2) Per ogni  $G$  in  $R$  usare l'indice borse. Scartiamo i grafici per cui almeno un nodo di  $G$  non ha molti compatibili successori.
- 3) Matching: si può filtrare in entrambi i paesaggi graph matching

### Filtering 2

Dati una  $V$  della query:  $N$  classi. L'insieme dei nodi del grafo  $G$  compatibili con  $V$  è  $\{v \in G \mid v$  è compatibile con  $v \in V\}$  le feature che fanno da  $v$  in  $Q$  partono da  $v$  in  $G$ .

Sia  $S$  l'insieme delle feature che partono da  $V$ .

E classi nell'indice borse di cui AND basato tra i nodi borse associati alle feature in  $S$

### Mining di Grafi

Dai mobili trasformare posizioni di mobili in grafi

Transazione  $\rightarrow$  Grafo

Item  $\rightarrow$  Nodi

Relazioni  $\rightarrow$  archi

Trascurare i tempi frequenti  $\rightarrow$  trovare solo gli elementi presenti

## Frequent Subgraph Mining FSM

Consiste nel trovare i sottografi che c' sono frequentemente in un database di grafi.

Supporto a un sottografo: numero grafi in cui c'è.

Se il supporto supera una soglia  $\theta$ , il sottografo è detto frequente.

Output di FSM  $\Rightarrow$  insieme sottografi frequenti (gradi massimo)

La frequenza è espressa o in valori assoluti o in percentuale. Quindi  $\theta$  si potrebbe esprimere come un per centuale.

Supporto  $\Leftrightarrow$  Vlue ass.

Frequenza  $\Leftrightarrow$  supp / N

### Rappr. Aggreg.

Il rapporto di un sottografo S in un database non può essere maggiore di quello di S

Se S non è frequente neanche un grafo con S come sottografo sarà frequente.

### Schemi FSM

1) Cerca nodi e archi frequenti e metti in  $\emptyset$

2) Per  $K \geq 3$  fasi

a) Genera candidati: partendo dai sottografi frequenti con  $K-1$  nodi combinaci insieme le due sottografi candidati con  $K$  elementi/nod.

b) Pruning: elimina sottografi ridondanti e i restanti i candidati con sottografi da  $K-1$  nodi non frequenti.

c) Counting: calcola supporto di ogni candidato e inserisce nei sottografi frequenti con le nodi aggiungili.

3) Torna a 0

## Approccio BPS e DPS

Il primo approccio berentro è BFS (in arancione).  
Un'altra strategia è DFS che richiede meno memoria ma è meno efficiente.

BFS: serve per generare i combinatori con  $k+1$  nodi  
e calcola i sottografi frequenti con  $k$  nodi.

DFS: calcola rispetto di un sottografo combinabile  
di  $K$  nodi. Se è frequente è intermedio.

Come generare i combinatori

- a) Join-based
- b) Extend-based

### Join-based

Un esempio con  $k+1$  nodi si ottiene dalla  
unione di  $\mathcal{R}$  ~~node~~ graph con  $k$  nodi  
(frequenti).

L'unione è possibile solo se i 2 sottografi  
hanno un simbolo 1 sottografo con  $k-1$  nodi  
in comune. Questo sottografo è detto

core graph

Lo stemo ragionamento si può fare con gli  
archi.

Così otterremo un grafo con un sottografo in più  
e optionalmente un nodo tra gli archi

Nella Join-based un join tra 2 grafici può  
essere già costituito dagli archi

- a) I due sottografi hanno già un core in comune
- b) Un core può avere già dei vettori morfismi

Da join direi riguarda generare lo stesso sottografo

## Nella ext - based

Un sottografo candidato con  $K+1$  nodi i generi estendendo un sottografo premente con  $K$  nodi aggiungendo un nodo.

Per evitare ribalzare a partire da un grafo l'estensione si fa nel cammino right-most

Il nodo è aggiunto se e solo se partire dai nodi nel cammino già e' finita nell'albero di ricerca operato da una DFS sul grafo

Anche qui i  $\mu$ , invece di aggiungere un nodo, aggiunge un arco

La operazione può partire ribalzare ovvero grafici inoltre.

Dobbiamo avere una rappresentazione di grafi che aiuti agli algoritmi di generare grafici isomorfi / ribalzanti

Non possiamo usare matrici o liste di adiacenze in quanto 2 grafi isomorfi potrebbero avere matrici diverse

## Stringa di adiacenza

Stringa ottenuta considerando le righe della matrice, se faccio nei grafi indetti solo alla fine faremo migliore.

Così con  $M$  nodi  $\Rightarrow M!$  possibili stringhe di adiacenza

Ci chiede la forma canonica del grafo ovvero la stringa di adiacenza lessicograficamente più vicina.

La forma canonica di un grafo è univoca

Perciò sottografi non abbiano libralzati né con stessa forma canonica

Dopo aver ottenuto i sothografi frequenti abbiamo  
ogni sì la significatività statistica.

Facciamo un post - procedimento mi limitati per i rimanenti  
gradi di libertà

La significatività si calcola fissando al 1% database  
di background contenuto in precedenza

### Post - procedimento

- 1) Partendo dal database D di N graphi, continuo il database di background B formato da N graphi  
random ottenuti da D
- 2) Rileggi i K volte, n'ottengono K database  
di background
- 3) Calcola rapporto  $\lambda$  del sothografo S in ogni database  
Ottieni una distribuzione di probabilità dei valori  $\lambda$
- 4) Calcola probabilità di osservare un  $\lambda \geq \lambda_0$  ovvero il rapporto nel database D.

Questa prob si dice p-value

- 5) Se p-value <  $\alpha$  allora S è un sothografo  
frequente significativo.

Di solito  $\alpha = 0,05$

### Algoritmo step - wise $\chi^2$

C'è neanche per generare il database B.

Permette di ottenere un grafico  $\chi^2$  con la stessa distribuzione di graphi di C

## Colice (G)

- 1) Pensi 2 ordi card di G,  $(a, b)$  e  $(c, d)$
1. Gl ordi sono disjunti ( $\cap = \emptyset$  m. in comune)
- Im b non ci sono  $(a, b)$  e  $(b, c)$
- 2) Scambia le vert. di due ordi
- 3) Itera 1 e 2 un certo numero di volte  
(d isto 100)

La scelta degli ordi in 1 garantisce che alla fine G e R abbiano la stessa distribuzione di gradi anche se sono grafici diversi.

## Algoritmo FSG

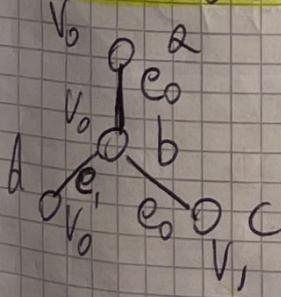
Usa una struttura BFS per cercare i sotto-graf frequenti. Generazione combattiva  $\Rightarrow$  Join board (mb) I sottografi sono rappresentati dalla forma canonica

### Pensando a b c in sillo

Ottiene la forma canonica usando la matrice di adiacenza.

1) Partizione per spazio

2) Partizione Ogni partizione per ticketta



|   | a  | b  | c  | d  | e  |
|---|----|----|----|----|----|
| a | 0  | 0  | 0  | 0  | ea |
| b | 0  | 0  | 0  | 0  | eb |
| c | 0  | 0  | 0  | 0  | ec |
| d | 0  | 0  | 0  | 0  | ed |
| e | ea | eb | ec | ed | 0  |

a e b nello stesso partizone perché stessi gradi e stessa ticketta

Per ogni combinazione partizione genera le combinazioni in questo caso

$$\begin{array}{l} a b c b \text{ e } b a c b \\ 000000, e_0 \quad 000000, e_0 \end{array}$$

$\equiv$  Quelle 2^m gradi le cui operazioni

## Esempio:

- Generazione core
- Join
- Cancellazione degli Attri  
Non presenti

## Ottimizzazione: parallel:

- Per ogni sottograph è necessario le forme canoniche dei suoi  $k-1$ -sotto graph frequenti
- Usa indicazione inversa
- Usa ~~list~~ cache di automorfismi di core frequenti

## Ottimizzazione: Inverted List

Quando vogliamo contare le frequenze di un  $k$ -sotto graph, esistono due stratezie: o come fatto dalla cancellazione e aggiunta in pratica risolvere  $M \times M_K$  problemi di subgraph matching ( $M$ : grafici nel DB)

Con le inverted list riduciamo il costo

- 1) Ad ogni  $S$  (sotto graph frequente) associa la transizione identificata (TID). E' la lista delle transizioni contenenti  $S$ .
  - 2) Calcolare freq di un  $k+1$ -sotto graph somigliante intersecare le TID dei suoi sotto graph frequenti.
  - 3) Se la m intersezione < mappa: i conta il sotto graph.
- Se no i conta la sua frequenza solo nelle transizioni della sua intersezione.

## Algoritmo di SPAN

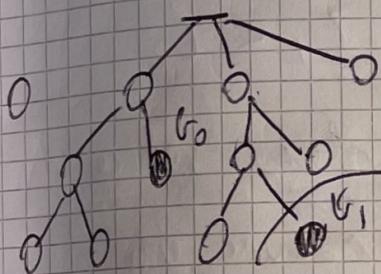
Usa un albero DFS per ricercare i rotoli frequenti

Rappresentazione rotoli = DFS sbe

Spatio di ricerca = albero costituito dai DFS sbe dei rotoli

I rotoli frequenti sono letti aggiungendo caselli alla volta con extend-based

## Algoritmo di ricerca



Span di SPAN via un ordinamento lexicografico per i vertici. Vertice al livello  $m$  rappresenta un rotolo con  $m$  simboli.

I vertici con DFS sbe più piccoli sono resenti più. Se siamo arrivati a  $G_0$  con DFS sbe X non serve più tornare a  $G_1$  con DFS sbe X logico. Così arriviamo a finire in "verti binari".

Il DFS ~~sceglie~~ sarà la lista dei nodi resi presenti in un grafo (rispetto un albero).

Saranno un albero in pratica.

Se eseguiamo DFS: sceglierà tronca ricerca in profondità.

Viamo da destra verso sinistra una sequenza di simboli di simboli fatta colDFS DFS

Definito l'ordine tra simboli DFS succede al grafo il colDFS più piccolo (un grafo può avere più alberi DFS)

- Per:
- vertice right-most: ultimo nodo without
  - coming right-most: cammino dal vertex - node right-most

$$E_{l,T} = \{e \mid \forall i, s_i < j \quad e = (v_i, r_j)\} \quad \text{2nd forward}$$

$$E_{b,T} = \{e \mid \forall i, s_i > j \quad e = (v_i, r_j)\} \quad \text{2nd backward}$$

La regola ordinata Norberto nel codice DFS  
ha queste proprietà

1) Dato  $v \in \text{ris}$  sarà eseguito backward visiti davanti  
a quelli prima di forward

2) Tra forward ha precedenza l'uno con  
festivazione un modo vincolo prima

$$(i, V_{J_1}) < (i, V_{J_2}) \quad \text{se } J_1 < J_2$$

3) Tra backward ha precedenza l'uno con  
sorgente un modo vincolo prima.

$$(i_i, V_{J_1}) < (i_i, V_{J_2}) \quad \text{se } i_i < i_2$$

4) Se due backward partono dalla stessa radice  
ha precedenza l'uno con festivazione un  
modo vincolo prima

$$(i, V_{J_1}) < (i, V_{J_2}) \quad \text{se } J_1 < J_2$$

Ordine lexicografico tra codici DFS

- E' grande il codice sottostante la regola minima
- Se i tempi sono uguali: ci sono basse nel  
ordine lexicografico dei tentativi compiuti  
(Occhiali 1° vertice, occhiali arco e 2° vertice)

### Extensione notazioni

Dato  $G$  e l'altro DFS  $T$  su slice minima  
ci sono 2 possibili estensioni

- Estensione backward: aggiungi uno step il vertice rightmost e uno al vertice del cammino rightmost
- Estensione forward: aggiungi nodi v e s negati  
o un qualunque nodo del cammino rightmost  
di SPAN opera verso condizioni di FSA

Con queste tecniche di espansione il codice DFS  
del figlio generato sarà un'estensione del codice  
del padre

## Mining in un singolo gfo

Dal G Thorson tutti i sottografi frequenti  
la frequenza è definita in base al numero di  
occorrenze in G

Sottografo frequente in G = "Motivo" di G

Problema: **overlaps** di occorrenze ovvero ci sono  
occorrenze con nodi in comune

In base alla situazione bisogna decidere se  
contare tra le occorrenze o no anche quelle  
della post.

Casi in cui non vogliamo sovrapposizioni: Vole la regola  
A prima

Caso contrario: non vole l'Aggiornamento  
(potrebbe non volere)

Di solito un sottografo ricorrente / frequente è  
detto motivo se è "significativo".

## Calcolare i significativi di S

Crea varianti tamponi di G e contare frequenze di S  
nelle varianti

Il  $\chi^2$ -value ottenuto dalla distribuzione delle  
frequenze di S nelle varianti determina la  
significatività di S

Di recente sono stati sviluppati modelli che calcolano  
il p-value in maniera molto più efficiente  
non dovettero generare le varianti tamponi.

## Strategie di ricerca

Occhio come i sottografi (esempi)

Network-centric: ricercare tutti i punti sottografi  
con K molti filtri nella rete e poi ripetere l'operazione  
ovvero creare gruppi di grafi (in ogni gruppo  
grafi sono isomorfi)

Multip-centric: generare tutti i punti sottografi,  
con le molte cercate poi in un veloce  
la verifica di un sottografo è ~~lento~~, ma genera  
tutti i grafi punti con le molte cercate questo è  
molto costoso.

Set-centric: Un mixto dei primi 2.

Prima cercano alcuni sottografi e poi trovano  
i sottografi isomorfi nella rete e poi raggruppano  
quei sottografi della rete isomorfi

Ricerca esatta: Trova TUTTI i sottografi con K molti

Sampling: Campionamento random di un numero  
minimo di sottografi con le molte successive  
contagio frequenze nel campione

- 1) Seleziona un argo o modo random real
  - 2) Entendi il resto fino ad avere K molti
  - 3) Il sottografo campionato ha i K molti e  
sono reali
- S campionato  
hore ↴

Una variante del Sampling prevede di fare  
una ricerca detta di campione S in G

Pregi:

- Non dipende molto dalla dim. della rete
- Con molti molti campioni ottiene risultati efficaci e veloci

Difetti: può perdere di motivi

## Rete neurale

Punto sul comportamento dei neuroni.  
Ogni neurone riceve un input regolare da altri neuroni, elabora e fa un output.  
Una rete neurale ha un insieme di nodi che iniziano i neuroni.

Un nodo riceve un input, ognuno con un peso già da costante o bias.

Fa la somma e produce un output. Una funzione è quella di attivazione.

La rete neurale fa rettangolo come un insieme di layer dove:

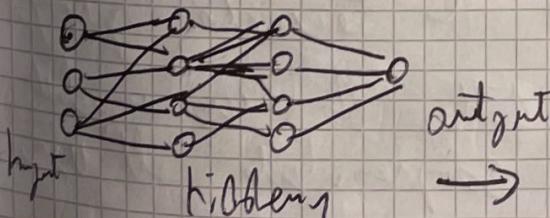
• **layer input**: vettore di n valori  $\vec{x} = x_1, \dots, x_n$

• **hidden layer**: costituiti da 1 o più nodi.

Ogni nodo riceve 1 o più input dal layer precedente e da 1 o più output al layer successivo.

• **output layer**: formato da 1 o più nodi che costituiscono il risultato.

E dice deep neural network se abbiano una rete con molti hidden layer.



## Tensione

Analisi multi-variale

Nel modello delle reti neurali:

• input e output di un nodo sono le tensioni

• i nodi di un layer producono le proprie tensioni o matrice

## Combinazioni

Rete densa: molte combinazioni

Si può comunque tollerare combini se c'è ogni  
particolare collegamento tra i 2

Random: finito in cui ogni nodo riceve output solo  
da un solo random precedente

Pooled: partizioniamo i nodi del layer in  $K$  cluster.  
Il layer successivo, detto pooled, sarà formato da  
 $K$  nodi. Il nodo associato a un cluster riceve tutti  
e solo gli output del layer precedente che  
che fanno parte di quel cluster.

Convolutional: vedi se i nodi di un layer si spostano  
su una griglia.

Il nodo di coordinate  $(i, j)$  riceve tutti e solo  
gli output dei nodi vicini allo stesso  $i, j$   
nel layer precedente

La rete neurale è un modello black-box,  
ovvero un osservatore vede il risultato di output  
verso presentarsi di come sia stato processato.

Explainable AI: una serie di tecniche per fare  
modelli che un risultato provetto da un modello  
come una rete neurale sia comprensibile all'uomo

Progettare una rete neurale

Fondamentale per costituire classificatori e predittori.

Primi step:

- Quanti layer?
- Come connettere molti layer?
- Quanti nodi in ogni layer?
- Che funzione usare per ogni nodo?
- (divisione)

Definire le strutture abbiano obiettivo (numero di training set) per trovare i valori ottimali dei pesi relativi a ogni input.

Definire la funzione loss  $F$ , abbiano trovare i pesi che minimizzano  $F$ .

Problema dei pesi / strutture

E' uno studio empirico, va molto a tentativi.

C'è un conseguente buco progettazione che ci sembra di cogliere se la fitta progettazione è buona.

meno layer  $\Rightarrow$  meno allenamento e lessione minima

più layer  $\Rightarrow$  permettere di risolvere problemi decisionali più complessi

Troppi layer  $\Rightarrow$  Overfitting

3 layer  $\Rightarrow$  di solito abbastanza nello spazio

troppi nodi in un layer  $\Rightarrow$  permette di memorizzare pattern più complessi

Prevenire overfitting: partire da un numero basso di nodi nel layer e aumentarli finché non accade

## Funzioni di attivazione

Dati  $\vec{x}$ ,  $\vec{w}$  e  $b$  ovvero: vettore di valori di input, vettore dei pesi e costante di bias

Funzione di attivazione di un nodo è la funzione che prende in input 2 cose:

$$z = \vec{w} \cdot \vec{x} + b$$

In un layer tutti i nodi hanno lo stesso  $F$

Come scrivere la funzione di perda  
per minimizzare i pesi (minimizzare  
il metodo già visto è lo stesso del gradiente  
Per far funzionare al meglio l'algoritmo di ricerca  
del gradiente la funzione deve avere certe  
proprietà:

- 1) Continua e differenziabile in ogni punto (o quasi)
- 2) La derivata non deve tendere a 0.  
Derivate giuste tollerano la ricerca dei pesi ottimali
- 3) La derivata non deve tendere a infinito poiché  
creerebbe instabilità nella ricerca dei pesi

### Funzione step

$$step(z) = \begin{cases} 1 & z \geq 0 \\ 0 & z < 0 \end{cases}$$

La funzione restituisce valori binari, per questo motivo il solo è detto percezione.

La funzione non soddisfa ne la 2) né la 3)  
dunque non ha unico minimo allo stesso  
del gradiente.

La funzione step è usata per classificare binario  
multidimensionale. Esempio: Perception (output a 1 solo solo)

Se vogliamo una classificazione multidimensionale  
l'output sarà già solo (1 per classe)

L'output sarà un vettore  $\vec{o}$  anche un solo  
valore è già un vettore  $\vec{o}$  e gli altri 0

## Funzione logistica

$$\sigma(x) = \frac{1}{1+e^{-x}} = \frac{e^x}{1+e^x}$$

valore  $\frac{1}{2}$  se  $x=0$

per valori piccoli di  $x$   
 $\rightarrow 0$

per valori grandi  $\rightarrow 1$

Si ottiene un vettore

che i punti abbiano semplicemente applicare la funzione in ogni componente

$$G(\mathbf{x}) = (\sigma(x_1), \sigma(x_2), \dots, \sigma(x_n))$$

Periamo  $y = G(\mathbf{x})$

$$\frac{dy}{dx} = y(1-y)$$

Al centroide da  $x=0$   
in una delle tre direzioni  
lo diventa, tende a 0.

Non rigetta la 2)

Dato che la logistica tocca valori fra 0 e 1 (non binari) al contrario della ~~perceptron~~  
classificazione precedente poniamo ancora a ogni  
punto di output (quindi una classe) una  
probabilità.

## Tangente isterologica

$$\tanh = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

Faz parte delle funzioni sigmoidi

Ha una relazione con  
la logistica

$$\tanh(x) = 2\sigma(2x) - 1$$

E' in pratica una versione scalata e trasposta  
della logistica

Ha valore fra -1 e 1 ed è simmetrica rispetto  
all'asse  $y$

Ha le stesse proprietà della logistica

## Funzione soft max

Orafull'intervento (non comprende le propriez)

$$\vec{x}, \vec{p}(\vec{x}) = (p(x_1), p(x_2) \dots p(x_n))$$

$$p(x_i) = \frac{e^{x_i}}{\sum_{j=1}^n e^{x_j}}$$

Torna un valore tra  
0 e 1

Se sommiamo gli  $n$  risultati, il risultato è 1.

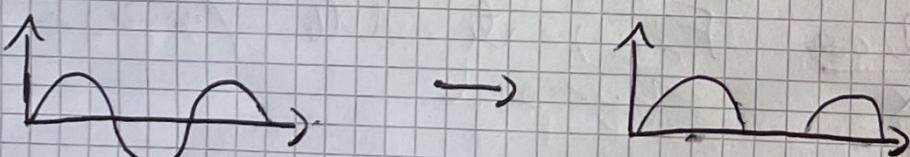
Questa funzione è ottima per ottenere una  
distribuzione di probabilità delle classi molto distin-

## Funzione ReLU Rectified Linear Unit

Prende segnale dai rilevatori e rigida  
rimanda uscita in elettronica.

Il circuito trasforma un segnale alternativo in  
un segnale unidirezionale.

In pratica sfida un'onda con valori positivi e negativi  
ottenendo un'onda con solo valori positivi (o negativi).



$$f(x) = \max(0, x) = \begin{cases} x & \text{se } x \geq 0 \\ 0 & \text{se } x < 0 \end{cases}$$

La funzione rigetta la 2\* quindi è molto  
più efficiente delle sigmoidi nella fase di  
training. \* Per x positivi

Il problema di p.e della derivata è molto  
semplice e veloce

Sopra si rigetta (non rigetta della 2) se  
i valori di x sono negativi

## Funzione Esponentiale Lineare Unit ELU

E' una variante della ReLU che evita il problema della saturazione con  $x$  negativo.

$$f(x) = \begin{cases} x & \text{se } x \geq 0 \\ \alpha(e^x - 1) & \text{se } x < 0 \end{cases}$$

$$\alpha \geq 0$$

ignorando  
vigne tempi

limi brontate  
il learning

Il learning è provato già  
che  $\alpha$  diverso  
per trovarne un valore buono