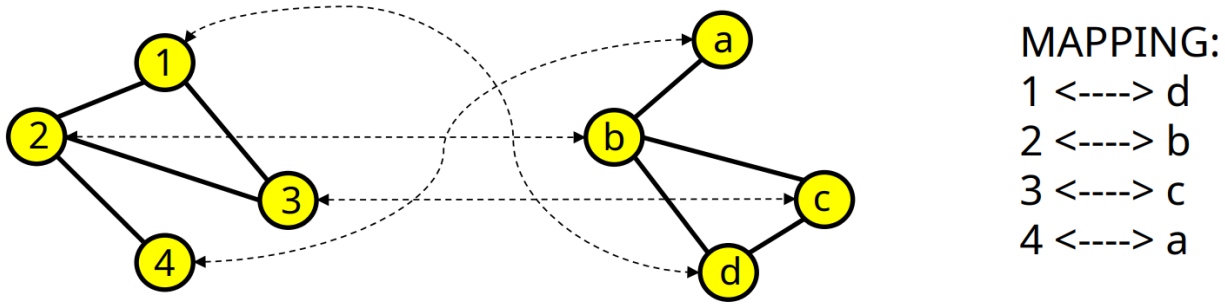
Lezione 7

Status	In progress
Attach file	7_SubgraphMatching.pdf

1. Introduzione

L'isomorfismo tra grafi (o graph matching) consiste nel verificare se due grafi sono uguali tra loro, ovvero hanno una corrispondenza.
Formalmente, due grafi $G1 = (V1, E1)$ e $G2 = (V2, E2)$ si dicono isomorfi se esiste una funzione biiettiva $f : V1 \to V2$, detta anche mapping, tale che:

$$(u, v) \in E_1 \Leftrightarrow (f(u), f(v)) \in E_2$$



L'isomorfismo tra un grafo G e se stesso è detto automorfismo di G.
Un grafo può avere più automorfismi.

Il subgraph matching consiste nel verificare se un grafo (detto grafo query) è contenuto in un altro grafo più grande (detto grafo target).

> 🌴 Il grafo $G1 = (V1, E1)$ è sottografo-isomorfo a $G2 = (V2, E2)$ se esiste una funzione iniettiva , chiamata mapping, tale che $(u, v) \in E_1 \Leftrightarrow (f(u), f(v)) \in E_2$.

Essendo iniettiva, non è necessario che tutti i nodi del grafo target siano mappati. L'importante è che siano mappati tutti i nodi del grafo query. Nel graph matching, invece, la funzione è biettiva. In generale, la soluzione del matching NON è unica.

Il graph matching è un problema NP-hard, ma non si sa se è NP-completo. Il subgraph matching è NP-completo.
Gli algoritmi di graph matching più popolari sono: Bliss, Saucy, Conauto, Nauty (ora Traces).
Questi tool trasformano grafi in una rappresentazione standard, chiamata forma canonica, in modo tale che due grafi isomorfi abbiano la stessa forma canonica.

2. Subgraph matching di un grafo

Soluzione brute-force

La soluzione brute-force per il subgraph matching consiste nell'esplorare tutti i possibili mapping tra i nodi della query e quelli del target.

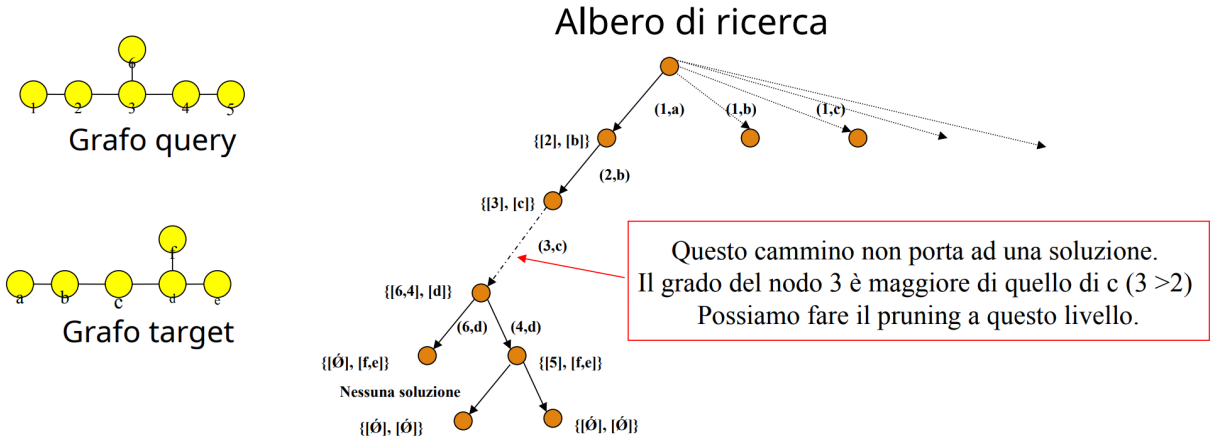
Le soluzioni vengono rappresentato tramite un albero di ricerca. Ogni cammino dalla radice a una delle foglie rappresenta un possibile mapping.

L'algoritmo brute-force termina quando:

- Tutti i nodi del grafo query sono stati mappati;
- Tutte le alternative per fare matching con un nodo del grafo target sono state esplorate con successo.

Se un grafo ha n nodi, ci sono n! possibili match.
Occorrono strategie di ricerca per velocizzare la computazione:

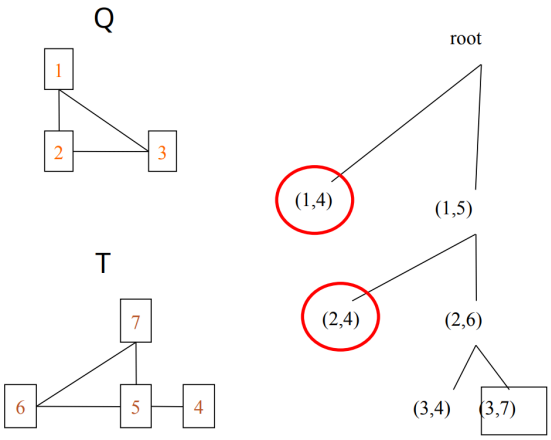
- Look-ahead: predire anticipatamente che il matching parziale prodotto non porterà ad una soluzione finale;
- Backtracking: abbandonare soluzioni parziali nel momento in cui ci accorgiamo che non portano a soluzioni finali.



Algoritmo di Ullmann

L'algoritmo di Ullmann utilizza il grado del nodo per fare look-ahead e backtracking e flitrare lo spazio delle possibili soluzioni.

Se il nodo della query ha un grado maggiore del nodo corrispondente del grafo target, allora si effettua il pruning.



Algoritmo VF

L'algoritmo VF (Vento-Foggia) si basa sul concetto di State Space Representation (SSR).
Il processo di matching è visto come una successione di stati dove ad ogni stato s è associato un insieme M(s) di coppie di nodi già mappati (detto mapping parziale).

L'aggiunta di una coppia di nodi prduce il nuovo stato s+1.
Ogni nuova coppia da aggiungere deve soddisfare un insieme di regole, dette regole di fattibilità.

Regole di fattibilità - Grafi Indiretti

L'algoritmo VF definisce 6 insiemi di nodi che si aggiornano durante la transizione da uno stato ad un altro:

- $Match_Q(s)$: insieme dei nodi query già mappati allo stato s;
- $Match_T(s)$: insieme dei nodi target già mappati allo stato s;
- $Term_Q(s)$: insieme dei nodi query terminali, cioè non ancora mappati ma adiacenti a nodi query già mappati allo stato s;
- $Term_Q(s)$: insieme dei nodi target terminali, cioè non ancora mappati ma adiacenti a nodi target già mappati allo stato s;
- $Rem_Q(s)$: restanti nodi del grafo query allo stato s
- $Rem_T(s)$: restanti node del grafo target allo stato s

Insieme P(s) dei candidati: coppie (n,m) con $n \in Term_Q(s)$ e $m \in Term_T(s)$;

Una coppia (p,q) in P(s) è aggiunta al mapping parziale M(s) se e solo se:

1. Per ogni nodo query q′ ∈ MatchQ(s) connesso a q, il corrispondente nodo target t′ ∈ MatchT(s) è connesso a t (consistenza del nuovo stato);
2. Il numero di nodi query in TermQ(s) connessi a q è minore o uguale al numero di nodi target in TermT(s) connessi a t (regola look-ahead ad un livello);
3. Il numero di nodi query in RemQ(s) connessi a q è minore o uguale al numero di nodi target in RemT(s) connessi a (regola look-ahead a due livelli).

Regole di fattibilità - Grafi Diretti

Nel caso di grafi diretti gli insiemi terminali si sdoppiano, producendo i seguenti insiemi:

- $TermOutQ(s)$: insieme dei nodi query terminali successori, cioè non ancora mappati ma successori di nodi query già mappati allo stato s;
- $TermInQ(s)$: insieme dei nodi query terminali predecessori, cioè non ancora mappati ma predecessori di nodi query già mappati allo stato s;
- $TermOutT(s)$: insieme dei nodi target terminali successori, cioè non ancora mappati ma successori di nodi target già mappati allo stato s;
- $TermInT(s)$: insieme dei nodi target terminali predecessori, cioè non ancora mappati mapredecessori di nodi target già mappati allo stato s;
- $RemQ(s)$: Restanti nodi non mappati del grafo query allo stato s

Una coppia (q,t) in P(s) viene aggiunta a M(s) se e solo se:

1. Per ogni nodo query q′ ∈ MatchQ(s) predecessore di q, il corrispondente nodo target t′ ∈ MatchT(s) è predecessore di t;
2. Per ogni nodo query q′ ∈ MatchQ(s) successore di q, il corrispondente nodo target t′ ∈ MatchT(s) è successore di t;
3. Il numero di nodi query in TermOutQ(s) connessi a q è minore o uguale al numero di nodi target in TermOutT(s) connessi a t;

4. Il numero di nodi query in TermInQ(s) connessi a q è minore o uguale al numero di nodi target in TermInT(s) connessi a t;
5. Il numero di nodi query in RemQ(s) connessi a q è minore o uguale al numero di nodi target in RemT(s) connessi a t.

Complessità Computazionale e Spaziale

Se vogliamo l'isomorfismo tra grafi, basta sostituire ≤ con = nelle regole look-ahead.
Complessità computazionale, nell'ipotesi che i due grafi abbiano N nodi:

- In media ogni nodo ha O(N) adiacenti, quindi il costo dell'esplorazione di un singolo stato è O(N);
- Caso migliore: ad ogni passo un solo nodo candidato soddisfa le regole di fattibilità.
  - N stati esplorati, complessità O(N2);
- Caso peggiore: devo esplorare l'intero albero di ricerca.
  - N! stati esplorati, complessità O(N!N);

Complessità spaziale:
Al più stati (e le informazioni associate) risiedono simultaneamente in memoria in un certo momento della computazione. Quindi O(N2)

Algoritmo VF2

VF2 ottimizza lo spazio utilizzato, fino ad ottenere una *complessità spaziale pari a O(N)* utilizzando strutture dati globali e condivise tra i vari stati.
Sei strutture dati introdotte:

- core_1 e core_2 contengono il mapping corrente core_1[n] = m se e solo se n e m sono mappati assieme.
- in_1, in_2, out_1, out_2 descrivono l'appartenenza dei nodi agli insiemi terminali. Il valore memorizzato corrisponde alla profondità (nell'albero di ricerca) dello stato in cui il nodo è entrato nel corrispondente insieme.

Obbiettivo: tracciare contemporaneamente l'*appartenenza del nodo* agli insiemi terminali e lo stato corrispondente della computazione. Quando si fa backtracking,

si ripristina il precedente valore di questi vettori.

Algoritmo RI

Le regole di fattibilità riducono lo spazio di ricerca ma sono costose da implementare.
L'algoritmo RI si concentra invece sull'ordine in cui i nodi della query vengono processati nell'albero di ricerca.
Un ordinamento efficace dei nodi velocizza molto il matching, anche in presenza di regole di pruning leggere.
In RI l'ordinamento è calcolato indipendentemente dal grafo target (static ordering), mentre in altri algoritmi è fatto anche sulla base del grafo target (dynamic ordering).

1. Ordina i nodi del grafo query in modo da massimizzare la probabilità che un cammino parziale nell'albero di ricerca venga tagliato prima possibile;
2. Seguendo l'ordinamento calcolato al passo 1), mappa nodi della query a nodi del target, verificando per ogni coppia candidata (q,t) che:
   a. q e t non siano già stati mappati ad altri nodi;
   b. Il grado di q sia minore o uguale al grado di t (regola di Ullmann).
3. Ripeti il passo 2) fino a quando l'intero spazio di ricerca non è stato esplorato.

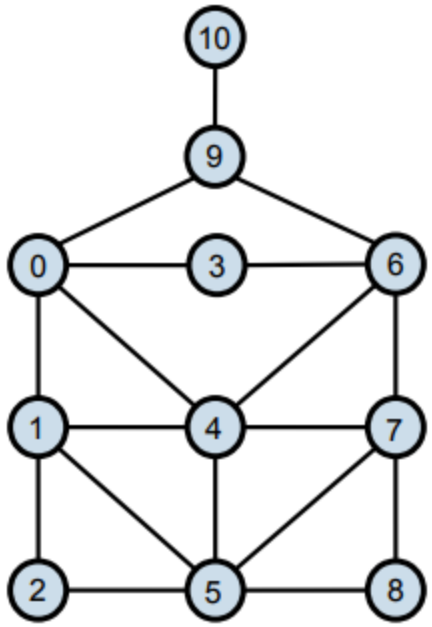
Ordinamento dei nodi della query

Dato un grafo query con n nodi, costruire una sequenza ordinata di nodi U = (u1, u2, ..., u3).
All'i-esimo passo, scegli il nodo con alto grado e con un elevato numero di connessioni con nodi già presenti nell'ordinamento Ui-1 .
Sia Ui-1 la sequenza ordinata parziale. Il punteggio di un nodo candidato v da inserire nell'ordinamento è definito sulla base di 3 insiemi:

- Vadiac(i): insieme di nodi in Ui-1 e adiacenti a v
- Vconn(i): insieme di nodi in Ui-1 adiacenti ad almeno un nodo che non appartiene a Ui-1 ed è connesso a v.
- Vrem(i): insieme di nodi connessi a v che non stanno in Ui-1 e non sono nemmeno connessi a nodi di Ui-1.

ESERCIZIO. Al passo 1, tutti gli insiemi sono vuoti, tranne $|V_{rem}(1, v)| = deg(v)$, vince il nodo di grado massimo (4). Lo stato dell'ordinamento diventa $U^1 = (4)$.

Al passo 2, scelgo il nodo 0, scelto arbitrariamente tra 0,5 e 6.



RI-DS

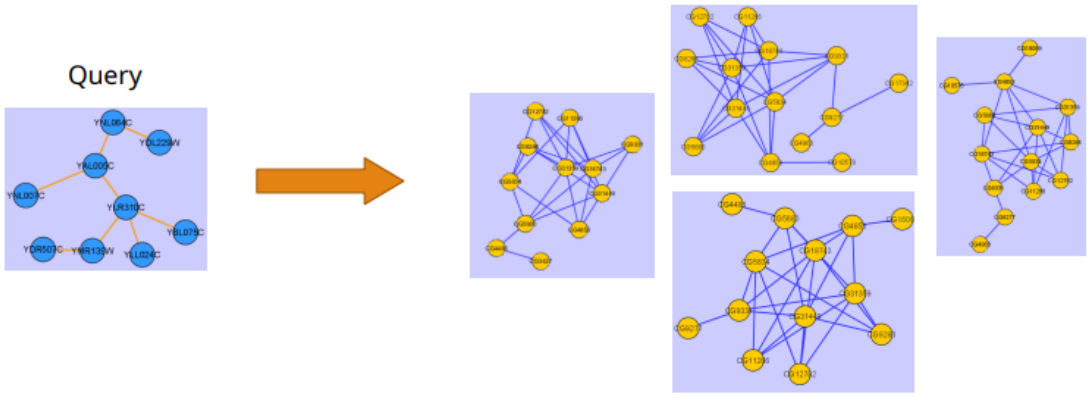
RI-DS è un'ottimizzazione di RI che consiste nel pre-calcolare, per ogni nodo query q, insiemi di nodi target, chiamati domini di compatibilità, che potrebbero essere mappaticon q.
Obbiettivo: applicare la regola del grado di Ullman in fase di matching una sola volta per una stessa coppia (q,t), e ridurre a monte le possibili coppie candidate.
Insieme di compatibilità per il nodo q:

$$Dom(q) = \{t \in V - T : deg(q) \leq deg(t)\}$$

1. Ordina i nodi del grafo query;
2. Calcola i domini di compatibilità per ogni nodo query;
3. Seguendo l'ordinamento calcolato al passo 1), mappa nodi della query a nodi del target, verificando per ogni coppia candidata (q,t) che:
   a. q e t non siano già stati mappati ad altri nodi;
   b. t appartenga al dominio di compatibilità di q.
4. Ripeti il passo 2) fino a quando l'intero spazio di ricerca non è stato esplorato.

3. Subgraph matching in un database di grafi

Dato un database di grafi e un grafo query , trovare tutti i grafi di che contengono come sottografo.



La soluzione banale consiste nell'applicare un algoritmo di subgraph matching (ad es. RI) su ciascun grafo del database, ma ciò richiederebbe troppo tempo.
Per ottenere tempi ragionevoli, occorre indicizzare i grafi del database e la query.
Per i grafi del database l'indicizzazione può essere effettuata off-line una volta sola.
Indicizzazione basata su feature: il grafo viene rappresentato tramite un insieme F di attributi (o feature). I grafi che non contengono tutte le feature della query vengono filtrate prima del matching. Esempi di algoritmi: SING, gIndex, TreePi, GraphFind.
Indicizzazione non basata su feature: i grafi del database vengono mappati in uno spazio metrico e memorizzati in un albero (B-tree oppure R-tree) in cui fare la ricerca. Questi sistemi sono più adatti per frequenti update del database. Esempi di algoritmi: Closure-tree, GCoding.

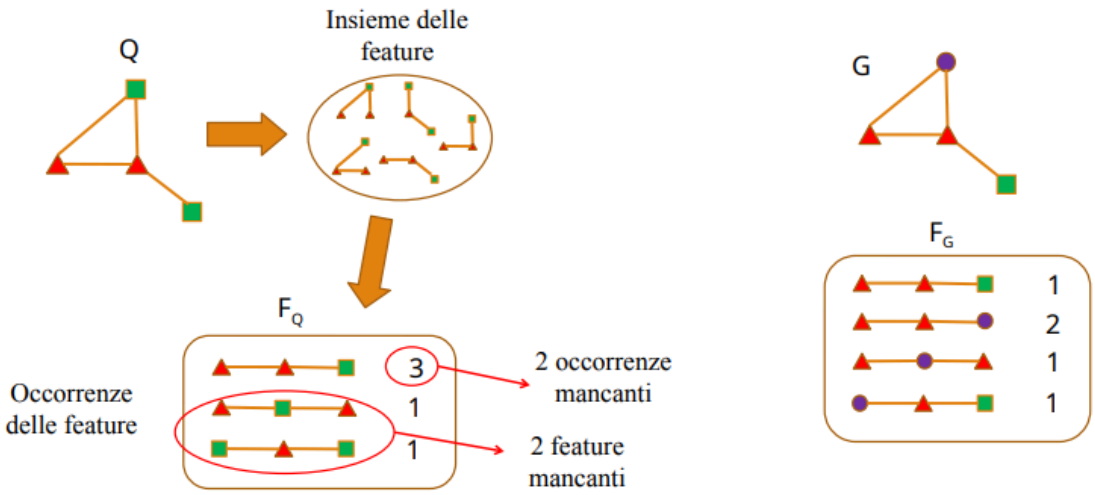
Feature dei grafi

Le feature da estrarre dai grafi possono essere:

- Piccoli grafi (algoritmi gIndex, FGIndex);
- Alberi (algoritmi TreePi);

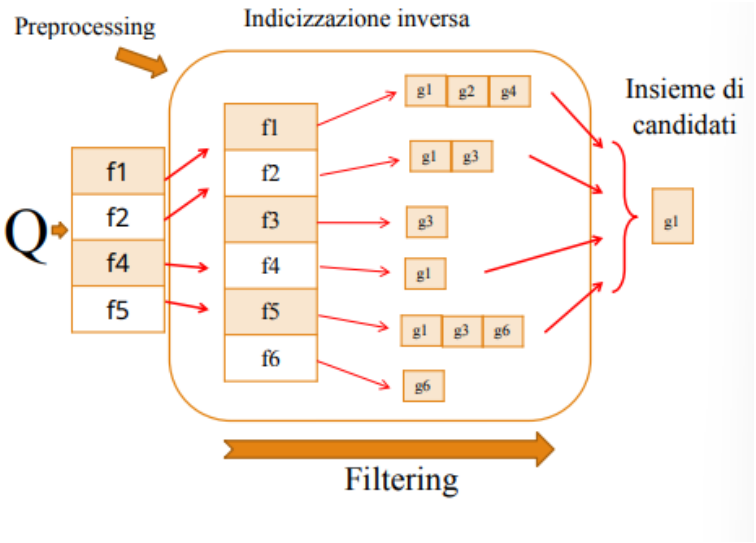
- Cammini (algoritmo SING).

I **cammini e gli alberi** sono feature più facili da estrarre rispetto ai sottografi.
Dal confronto tra le feature della query e le feature dei grafi del database, si può capire quali grafi possono contenere o no la query.



1. Preprocessing: per ogni grafo del database si estraggono tutte le features (cammini, alberi o grafi) che contiene;
2. Filtering: dalla query si estraggono tutte le features contenute, e viene costruito un insieme di grafi del db candidati che contengono tutte le features della query;
3. Matching: per ogni grafo candidato si applica un algoritmo di graph matching per verificare se contiene la query.

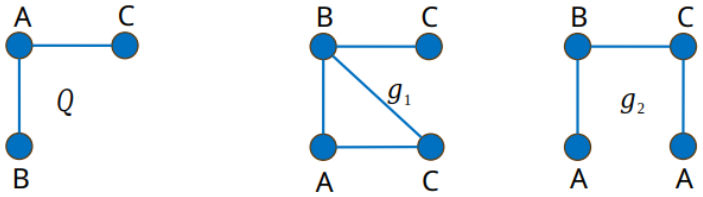
Per velocizzare il filtering si usa la tecnica dell'indicizzazione inversa. Pertanto, ad ogni feature (chiave) si associa la lista dei grafi che la contengono, con il relativo numero di occorrenze della feature.
L'insieme dei candidati per il matching si può ottenere per intersezione delle liste di grafi associati alle feature della query.

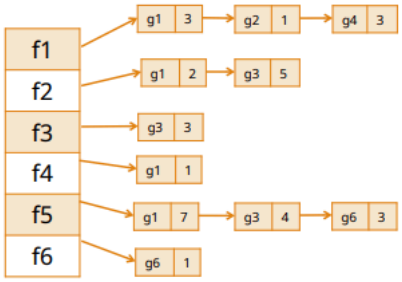


Algoritmo SING

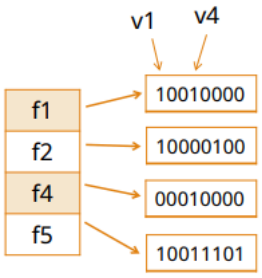
L'algoritmo SING (Subgraph search In Non-homogeneous Graphs) è un algoritmo di subgraph matching in un database di grafi basato su indexing tramite cammini di nodi.
Per migliorare l'efficacia dell'indicizzazione associa ad ogni feature la frequenza ma anche il nodo da cui parte.

AB e AC stanno in g1 eg2, ma solo g1 contiene Q. Questo perché AB e AC partono dallo stessonodo in g1 e da nodi diversi in g2.





Indice inverso globale: ad ogni feature si associa la lista dei grafi del database che la contengono e il relativo conteggio.



Indice inverso locale per ciascun grafo g: ad ogni feature F presente in g è associato un vettore binario dove l'i-esimo elemento è 1 se esiste un'occorrenza di F che parte dal nodo i, 0 altrimenti.

1. Filtering 1: per ogni feature F della query, prendi l'insieme dei grafi in cui occorre un numero di volte maggiore o uguale del numero di occorrenze nel grafo query. Calcola l'intersezione R di tutti questi insiemi;
2. Filtering 2: per ogni grafo G in R, usa l'indice locale di per calcolare gli insiemi di nodi compatibili (cioè mappabili) con i nodi della query. Scarta i grafi per cui almeno un nodo della query non ha nodi compatibili corrispondenti.
3. Matching: Per ogni grafo che ha superato i filtering 1 e 2, verifica se contiene il grafo query tramite algoritmo di subgraph matching.

Filtering 2

Dato un vertice v della query, l'algoritmo calcola l'insieme dei vertici di un grafo Gdel database che sono compatibili con v.
Un vertice x di G è detto compatibile con v se tutte le feature che partono dal nodo v nella query partono dal nodo x in G.
Se S è l'insieme di queste features che partono da v, è sufficiente calcolare, nell'indice locale del grafo G, un AND logico tra i vettori binari associati alle feature di S.
Se l'AND logico restituisce un vettore con almeno un 1, allora esiste un nodo di che è compatibile con