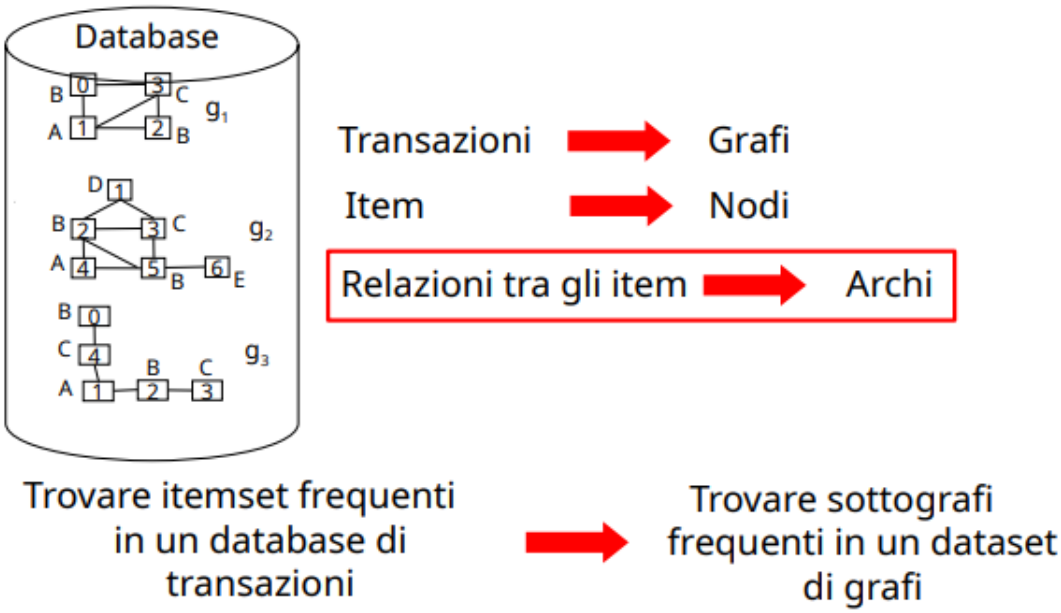
Lezione 8 - Frequent Subgraph Mining

Status	In progress
Attach file	8_FrequentSubgraphMining (1).pdf

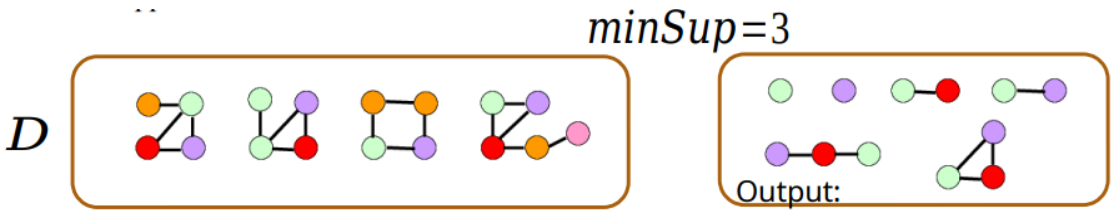
Introduzione al problema



Il frequent subgraph mining consiste nell'identificare tutti i sottografi che occorrono frequentemente in un database di grafi.

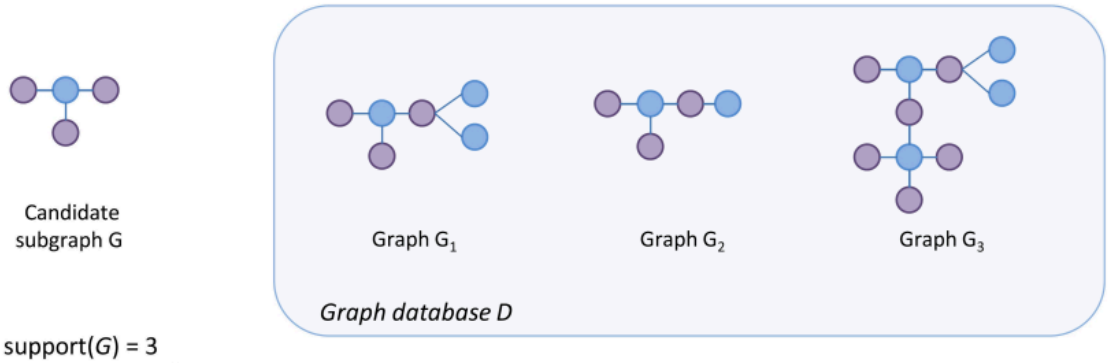
Si conta il numero di grafi del dataset che contengono il sottografo, chiamato anche **supporto**.
Se il supporto del sottografo è maggiore o uguale ad una soglia minima minSup, allora il sottografo è detto frequente.

L'output è l'insieme di tutti i sottografi frequenti con il loro supporto.



Piuttosto che fare riferimento al conteggio assoluto (supporto), spesso si fa riferimento al conteggio relativo (frequenza).
La frequenza di un sottografo S in un database di grafi D è il rapporto tra il supporto e il numero di grafi nel database.
La soglia minima σ in questo caso è una percentuale (da 0 a 100%). Un sottografo è frequente se è presente in almeno σ% grafi del database.

ESEMPIO:



support(G) = 3
frequency(G) = $\frac{3}{3}$ = 1 (100%)

Il frequent subgraph mining pone sfide più complesse rispetto alla ricerca degli insiemi frequenti.

- Quale strategia di ricerca adottare?
- Come generare in maniera efficiente i sottografi candidati?
- Come evitare o gestire le ridondanze nella generazione dei candidati

Esistono due tipologie di algoritmi per frequent subgraph mining:

1. Algoritmi Apriori: basati sulla regola Apriori e la generazione dei candidati tramite join di sottografi frequenti (es. algoritmi AGM, FSG);
2. Algoritmi pattern-growth: basati sulla generazione dei candidati tramite aggiunta di nodi e/o archi a sottografi frequenti (es. algoritmi gSPAN, FFSM, MoFa).

Algoritmo FSG

Regola Apriori per filtrare i candidati: il supporto di un sottografo S in un database di grafi non può essere maggiore del supporto dei sottografi di S.
Se un sottografo S non è frequente, allora nessun grafo che contiene S è frequente.
La regola Apriori è alla base dell'algoritmo FSG (Frequent SubGraph Mining).

ALGORITMO:

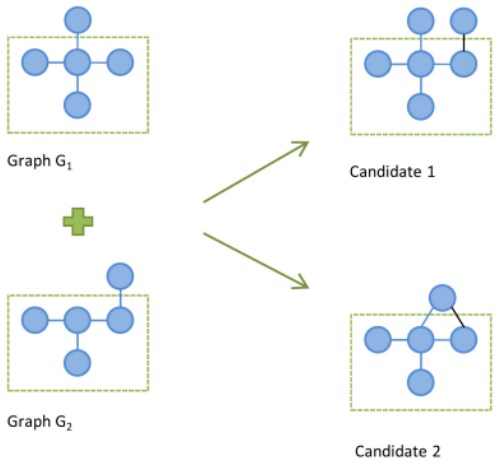
1. Calcola nodi e archi frequenti nel database di grafi e aggiungili all'insieme finale O dei risultati.

2. Per k ≥ 3, ripeti i seguenti passi:
    a. Candidate generation: a partire dall'insieme dei sottografi frequenti con k-1 archi, costruisci l'insieme C dei sottografi candidati con k archi;
    b. Pruning: verifica la regola Apriori, scartando tutti i candidati che contengono almeno un sottografo con archi che non è frequente;
    c. Counting: calcola il supporto di ogni grafo candidato e l'insieme dei sottografi frequenti con k archi. Aggiungi i sottografi frequenti trovati all'insieme finale O.
3. Restituisci in output O.

Generazione dei candidati

I sottografo candidati con k archi sono generati mediante unione (join) di due sottografi frequenti con k-1 archi.

L'unione è possibile solo se i due sottografi hanno almeno un sottografo con archi in comune, chiamato grafo core.

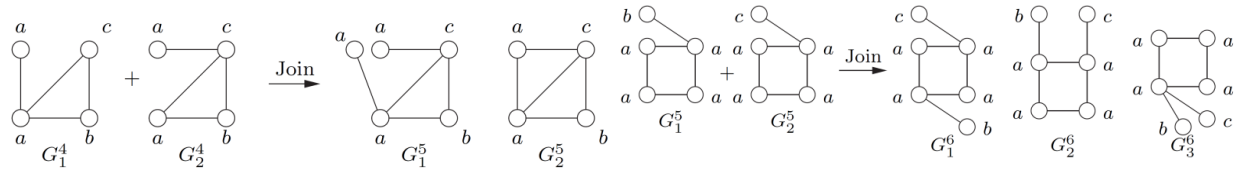


A differenza degli itemset la join di due sottografi può generare più sottografi distinti tra loro.
Tre scenari possibili:

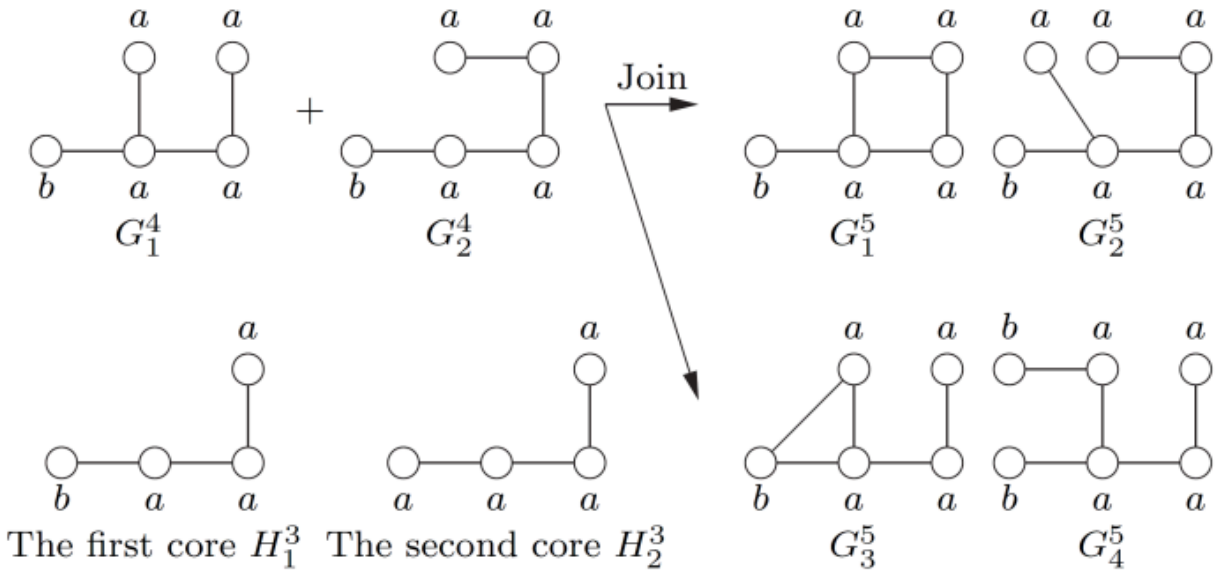
1. I due sottografi candidati differiscono per un nodo avente la stessa etichetta (nel caso di grafietichettati);
2. Il grafo core ha più automorfismi;
3. I sottografi candidati hanno più grafi core in comune.

ESEMPIO:

SCENARIO 1: SCENARIO 2:



SCENARIO 3:



Join

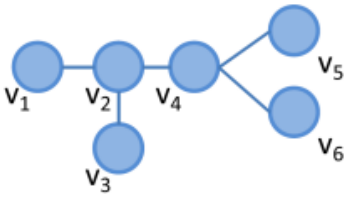
1. Sa F_k l'insieme dei sottografi frequenti con k archi e C^{k+1} l'insieme di tutti i sottografi candidati con k + 1 archi (inizialmente vuoto).
   Per ogni coppia di sottografi frequenti con k archi (G_i^k, G_j^k), con
   a. Calcola l'insieme dei core condivisi da G_i^k, G_j^k
   b. Per ogni core e per ogni automorfismo del core, effettua la join e aggiungi i sottografi candidati trovati a C^{k+1}
2. Restituisci C^{k+1}

Candidati ridondanti

La generazione dei candidati può produrre sottografi ridondanti, ovvero isomorfi. FSG calcola per ogni grafo un codice univoco, chiamato forma canonica (estratta tramite matrice di adiacenza), per gestire tale ridondanza.

Se due grafi hanno la stessa forma canonica, allora sono isomorfi, quindi uno dei due può essere scartato.

La stringa di adiacenza di un grafo è la stringa ottenuta concatenando le righe della matrice di adiacenza (triangolare superiore per grafi indiretti).



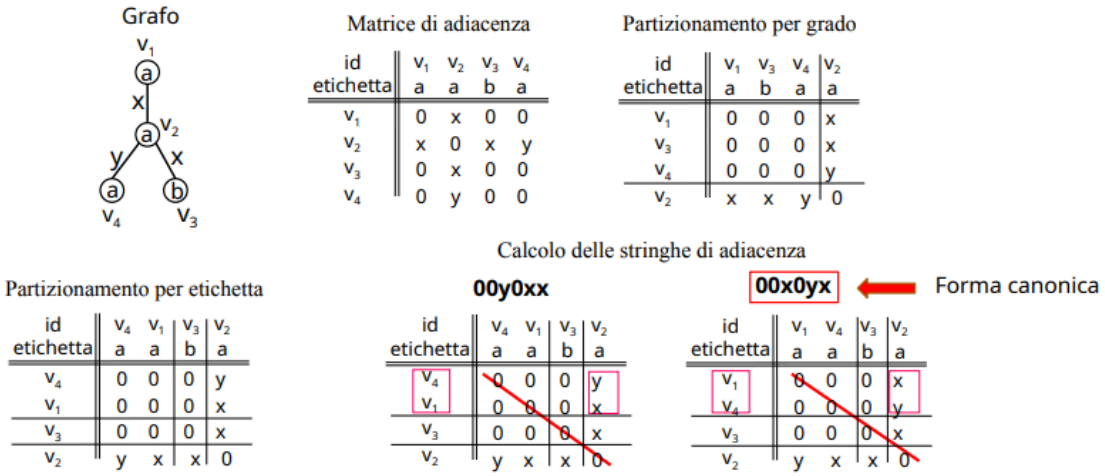
$$A(G) = \begin{array}{c} \\ v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \\ v_6 \end{array} \begin{array}{c} \begin{array}{cccccc} v_1 & v_2 & v_3 & v_4 & v_5 & v_6 \end{array} \\ \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix} \end{array}$$

S(G) = 100001100000110

Per un grafo G con n nodi, esistono n! stringhe di adiacenza, ottenute considerando tutte le possibili permutazioni dei nodi di G. Diverse permutazioni possono produrre la stessa stringa di adiacenza.
Come forma canonica di G consideriamo la stringa di adiacenza lessicograficamente più piccola (o più grande).
Calcolare le n! stringhe di adiacenza è oneroso. FSG sfrutta i gradi dei nodi e, se presenti, etichette associate a nodi e archi per ridurre il numero di stringhe di adiacenza da calcolare.

Come calcolo al forma canonica?

1. Partiziona i nodi in gruppi sulla base del grado;
2. Partiziona ogni gruppo ottenuto al passo precedente in sottogruppi sulla base delle etichette dei nodi;
3. Considera le k stringhe di adiacenza che è possibile ottenere permutando in tutti i modi possibili i nodi in ciascun sottogruppo.
4. Tra le k stringhe scegli quella lessicograficamente più piccola.



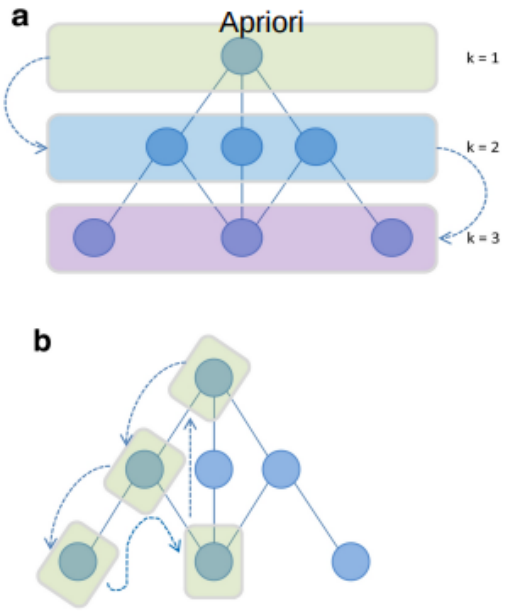
Per ogni sottografo candidato G con k archi si verifica la regola Apriori: se almeno uno dei suoi sottografi con k-1 archi non è frequente, G si può scartare perché non sarà frequente.
Ciò è fatto rimuovendo in tutti i modi possibili un arco da G e verificando se ogni sottografo ottenuto è frequente o no.
Per contare le **frequenze** degli m k-sottografi candidati al passo k, dovremmo risolvere n x k problemi di subgraph matching, dove n è il numero di grafi nel database.
La tecnica dell'indicizzazione inversa può essere utilizzata per ridurre il numero di problemi di subgraph matching da applicare.

Calcolo del supporto

1. Ad ogni sottografo frequente S associa la transaction identifier list (lista TID), ovvero la lista delle transazioni che contengono S;
2. Per ogni (k+1)-sottografo candidato, calcola l'intersezione delle liste TID dei suoi k-sottografi frequenti;
3. Se la cardinalità dell'intersezione della lista TID di un $(k+1)-\text{sottografo}$ candidato è minore del supporto minimo, allora scarta (perché non può essere frequente), altrimenti calcola (tramite subgraph matching) la sua frequenza solo nelle transazioni della lista intersezione.

Approccio Pattern-Growth

L'approccio **Apriori** è basato su visita dello spazio delle soluzioni in ampiezza, ovvero BFS: prima di generare i sottografi candidati con k+1 archi, si calcolano tutti i sottografi frequenti con k archi.
I pattern-growth usano una strategia DFS, ovvero in profondità: si calcola il supporto di un sottografo candidato con k archi, se è frequente si estende, quindi si controlla il supporto del sottografo esteso e così via.
L'approccio DFS richiede meno memoria, ma è meno efficace nel pruning.



Algoritmo gSPAN

gSpan è un algoritmo di ricerca di sottografi frequenti basato su approccio patterngrowth.
Ad ogni sottografo da ricercare è associato un codice, chiamato codice DFS minimo, calcolato a partire dalla visita DFS del sottografo e che funge da forma canonica.
Lo spazio di ricerca è strutturato tramite un albero, detto DFS code tree, costruito a partire dai codici DFS minimi dei sottografi.
I sottografi frequenti vengono ricercati effettuando una visita del DFS code tree secondo un ordinamento **pre-stabilito** basato sui codici DFS minimi.

Visita DFS

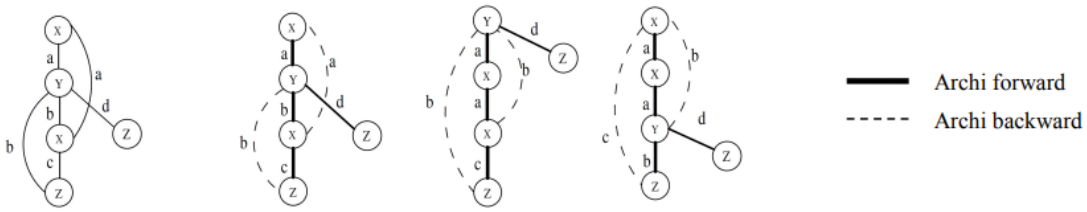
Nella DFS gli archi del grafo vengono esplorati a partire dall'ultimo nodo scoperto v che presenta ancora archi non esplorati uscenti da esso.

Terminata l'esplorazione di tutti gli archi uscenti da v si ritorna indietro al nodo v' da cui si era andato verso v e si esplorano i rimanenti archi uscenti da v'.

La visita termina quando tutti i nodi sono stati visitati.

Abero DFS

La visita DFS di un grafo G produce un albero T (albero DFS), i cui nodi sono i nodi di G, mentre gli archi sono gli archi di G esplorati durante la visita e che hanno portato alla scoperta di nuovi nodi.
Gli archi di G presenti in T sono chiamati archi forward, i restanti sono archi backward. Un grafo può avere più alberi DFS dovuti al nodo di partenza nella visita DFS e all'ordine di esplorazione degli adiacenti di un nodo.



Codice DFS

Seguendo l'ordine di visita dei nodi nella DFS, è possibile costruire un codice DFS, formato da una sequenza ordinata di archi forward e backward stabilita secondo criteri precisi.
Ogni arco (u,v) nel codice DFS è rappresentato da una quintupla del tipo:

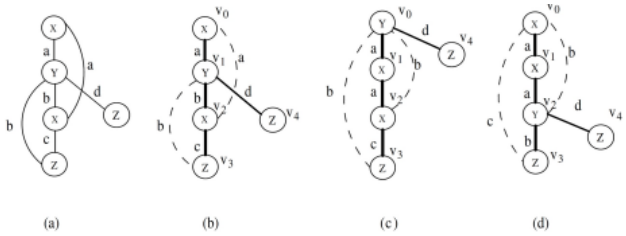
$$(t(u), t(v), lab(u), lab(u, v), lab(v))$$

dove t indica il tempo di visita di un nodo (ovvero la posizione di v nell'ordine di visita DFS) e lab l'etichetta (di un nodo o arco).
Se il grafo non è etichettato, le etichette possono essere omesse.

La sequenza ordinata di archi in un codice DFS è costruita seguendo questi criteri:

1. Dato un nodo v, tutti i suoi archi uscenti backward devono apparire prima degli archi forward;

2. Tra gli archi forward che partono dallo stesso nodo, si seleziona l'arco con destinazione un nodo già visitato prima, ovvero $(u, v) < (u, v')$ se $t(v) < t(v')$;
3. Tra archi forward che partono da nodi diversi, seleziona quello con sorgente un nodo visitato prima, ovvero $(u, v) < (u, v')$ se $t(u) < t(u')$;
4. Tra gli archi backward che partono dallo stesso nodo, seleziona quello con destinazione un nodo visitato prima, ovvero $(u, v) < (u, v')$ se $t(v) < t(v')$.
5. Tra gli archi backward che partono da nodi diversi seleziona quello con sorgente un nodo visitato prima, ovvero $(u, v) < (u, v')$ se $t(u) < t(u')$.



edge no.	(b) α	(c) β	(d) γ
0	$(0, 1, X, a, Y)$	$(0, 1, Y, a, X)$	$(0, 1, X, a, X)$
1	$(1, 2, Y, b, X)$	$(1, 2, X, a, X)$	$(1, 2, X, a, Y)$
2	$(2, 0, X, a, X)$	$(2, 0, X, b, Y)$	$(2, 0, Y, b, X)$
3	$(2, 3, X, c, Z)$	$(2, 3, X, c, Z)$	$(2, 3, Y, b, Z)$
4	$(3, 1, Z, b, Y)$	$(3, 0, Z, b, Y)$	$(3, 0, Z, c, X)$
5	$(1, 4, Y, d, Z)$	$(0, 4, Y, d, Z)$	$(2, 4, Y, d, Z)$

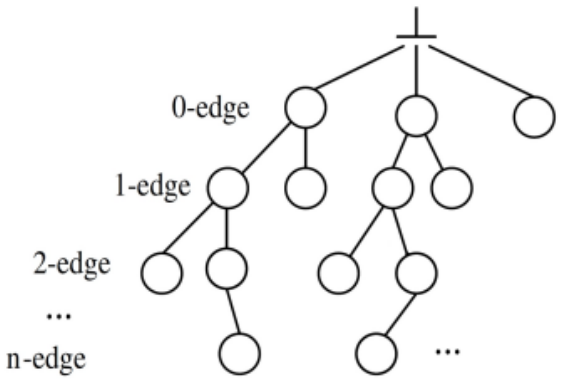
Tra tutti i possibili codici DFS di un grafo, gSpan seleziona quello lessicograficamente più piccolo, chiamato codice DFS minimo ($(d)\gamma$).

Essa è la forma canonica del grafo, usata per eliminare candidati ridondanti e semplificare la ricerca.

DFS CODE TREE

Il DFS code tree è un albero, rappresentantelo spazio dei candidati esaminati da gSpan, i cui nodi rappresentano un grafo identificato dal codice DFS minimo.

- Il livello 0 (radice) contiene grafi con 0 archi
- il livello 1 grafi con 1 arco, ecc.



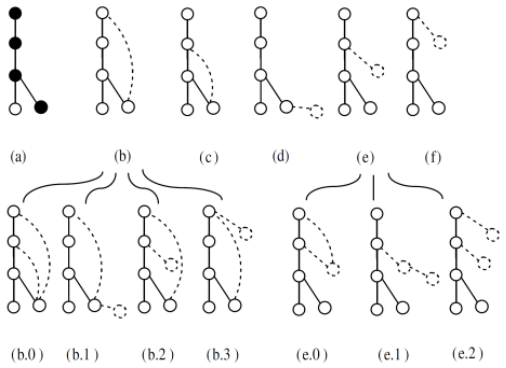
Un arco collega due nodi dell'albero se un grafo con k+1 archi è generato a

partire da uno con k archi mediante aggiunta di un arco (secondo opportuni criteri).

Generazione dei candidati

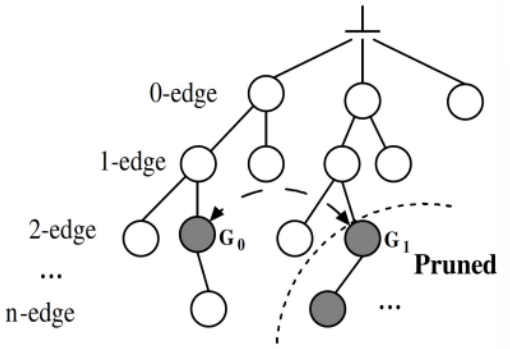
In gSpan un sottografo candidato con k+1 archi viene generato per aggiunta di un singolo arco (ed eventualmente anche di un nodo) a partire da un sottografo frequente con k archi.
Per non generare sottografi candidati ridondanti a partire da uno stesso sottografo e velocizzare il calcolo del codice DFS minimo, è consentito solamente aggiungere:

- un arco da un nodo che sta nel cammino più a destra dalla radice ad un nodo foglia (cammino rightmost) nell'albero DFS associato al codice DFS minimo.
- un arco dal nodo foglia del cammino right-most dell'albero DFS associato al codice DFS minimo a qualsiasi altro nodo dello stesso cammino.



L'estensione effettuata a partire dal cammino right-most permette disemplificare il calcolo del DFS minimo. Il codice DFS nel nuovo sottografo generato sarà ottenuto a partire dal codice DFS del sottografo di partenza aggiungendo la quintupla associata al nuovo arco.

I grafi candidati con k+1 archi generati a partire da un grafo frequente con k archi vengono processati seguendo l'ordine lessicografico dei loro codici DFS minimi.
Se generiamo un grafo candidato G1 con lo stesso codice di un grafo G0 già esaminato, possiamo fare un



pruning dell’intero sottoalbero radicato in G1.