UNIVERSITÀ DEGLI STUDI DELL'AQUILA

DIPARTIMENTO DI INGEGNERIA E SCIENZE DELL'INFORMAZIONE E MATEMATICA

CORSO DI LAUREA IN INFORMATICA

Stima efficiente del numero di Network Motifs tramite Color-Coding e decomposizioni bilanciate

Relatore:

Dott. Stefano Leucci

Candidato:

Giulia Scoccia

Correlatore:

Prof. Guido Proietti

Matricola:

249503

ANNO ACCADEMICO 2019–2020

INDICE

CAPITOLO 1

INTRODUZIONE

I Motif, anche chiamati Graphlet o Pattern, sono piccoli sottografi connessi indotti di un grafo, la conta dei motif è un problema ben noto del graph mining e dell'analisi dei social network.
Dato in input un grafo G e un intero positivo k il problema richiede di contare, per ogni graphlet H di k nodi, il numero di sottografi indotti di G isomorfi ad H. Comprendere la distribuzione dei motif in un grafo fa luce sul tipo di strutture locali presenti in esso, che possono essere usate per una miriade di analisi, come ad esempio per l'analisi di reti sociali [2, 3, 4], reti biologiche [1], etc... Poichè il conteggio dei graphlet può risultare computazionalmente impegnativo, di solito ci si accontenta di obiettivi meno ambiziosi. Uno di questi è la stima approssimata della frequenza: per ogni sottografo si richiede di stimare, nel modo più accurato possibile, la sua frequenza relativa rispetto a tutti i sottografi della stessa dimensione. Ancora meno ambiziosamente, visto che il numero di motif cresce rispetto al numero k, si restringe l'attenzione al problema della stima della frequenza relativa solo ai sottografi che compaiono il maggior numero di volte nel grafo input. Ci sono due approcci per ottenere tali stime. Il primo utilizza i metodi Monte Carlo,basati sulle catene di Markov, mentre il secondo sfrutta la tecnica del Color Coding introdotta da Alon, Yuster e Zwick [5]. Studi recenti analizzano le differenze tra i due approcci concludendo come regola generale che il Color Coding è preferibile quando la garanzia di accuratezza sui dati risulta fondamentale, nonostante la complessitá spaziale che può limitarne

l'utilizzo [6]. In questa tesi ci concentreremo solo sulla tecnica del Color Coding. Tale tecnica è stata introdotta per risolvere in maniera randomizzata il problema di determinare l'esistenza di sottografi isomorfi a cammini ed alberi con treewidth limitata nel grafo G in input. Un'estensione di questa tecnica consente di ottenere garanzie statistiche forti per il problema del Motif Counting, da cui le frequenze possono essere facilmente derivate. Tale estensione si basa su due osservazioni chiave. La prima è che il Color Coding può essere usato per costruire "un'urna" astratta che contiene un sottoinsieme statisticamente rappresentativo di tutti i sottografi di G (non necessariamente indotti) che hanno esattamente k nodi e sono alberi. La seconda osservazione è che il compito di campionare k-graphlet, ossia graphlet con k nodi, può essere ridotto, con un overhead minimale, a campionare k-alberi, alberi con k nodi, dall'urna. Si può così stimare il numero dei motif in due fasi: la "fase di costruzione", in cui si crea l'urna da G e la "fase di campionamento", dove si campionano i graphlet fino ad ottenere delle stime accurate per quelli di interesse.

1.1 Contributo della tesi

In questo lavoro di tesi, l'attenzione è stata concentrata sull'ottimizzazione di un algoritmo basato sulle tecnica del Color Coding per il conteggio di k-treelet all'interno di grafi più o meno grandi. Per k-treelet si intendono alberi (non necessariamente indotti) in un grafo con k nodi. È stato visto in uno studio del 2008 [1] su una rete PPI (Protein-Protein Interaction) quanto l'individuazione di k-treelet in un grafo può essere utile per la ricerca della frequenza di particolari strutture biomolecolari, lo studio é stato svolto in particolare su reti di organismi unicellulari, che risultano tutti molto simili, e reti di organismi pluricellulari,che invece possono variare molto tra di loro.

Per effettuare il conteggio è stato necessario concentrarsi sulla fase costruttiva descritta in precedenza.

La fase costruttiva, è descritta mediante un algoritmo di programmazione dinamica, è un processo che richiede un grande impiego di tempo e spazio. Il lavoro svolto ha portato, per prima cosa, ad un'implementazione in Java dell'algoritmo descritto in [6]. L'approccio dell'algoritmo supposto di dover conteggiare i treelet

di dimensione k di un grafo, lavora in esattamente k fasi. Nell'i-esima fase saranno conteggiati i treelet di dimensione i. Tali conteggi saranno ottenuti in funzione della struttura del grafo e del numero di occorrenze dei treelet di dimensioni minore di i. Perciò per poter calcolare i treelet di dimensione k, sarà necessario aver già calcolato quelli di dimensione fino a $k-1$. Tale dipendenza è evidenziata dalle formule di ricorrenza che descrivono l'algoritmo di programmazione dinamica, che verrà descritto nella sezione 2.1. Poichè il numero dei k-treelet cresce in maniera esponenziale rispetto a k, l'algoritmo puó essere eseguito solo per valori piccoli di k, prima che il tempo richiesto diventi eccessivo. A tal proposito nella tesi viene proposta un'ottimizzazione, basata su opportune decomposizioni "bilanciate" degli alberi, che consente di rendere indipendenti i conteggi dei treelet di dimensione k da $\frac{1}{3}$ dei conteggi precedenti. Questo consente di eseguire le prime $\lceil \frac{2}{3}k \rceil$ fasi prima della fase k, comportando un risparmio notevole di tempo. Ad esempio su un grafo sociale con 63731 nodi e 817090 archi, l'algoritmo non ottimizzato richiede DA VEDERE tempo per la ricerca dei treelet con DA VEDERE nodi, mentre quello ottimizzato richiede un tempo DA VEDERE.

1.2 Organizzazione del testo

La descrizione del lavoro è strutturata nel seguente modo. Nel capitolo 2 viene descritta la tecnica del color coding e il suo utilizzo per il conteggio degli alberi. Si vedrà l'algoritmo di [6] e la sua formulazione Si discuterà la scelta di adottata per l'implementazione e i suoi vantaggi. Nel capitolo 3 si discuterà in dettaglio la tecnica delle decomposizioni bilanciate ed il relativo impatto sull'algoritmo. Anche in questo caso si discuteranno le scelte effettuate in fase implementativa. Nel capitolo 4 verranno mostrati i risultati di un'analisi sperimentale delle performance dell'algoritmo ottimizzato rispetto alla versione di [6]. Infine, nel capitolo 5, veranno discusse le possibili estensioni del presente lavoro di tesi.

CAPITOLO 2

COLOR CODING

Nel capitolo viene descritta la tecnica del Color Coding utilizzata in questa tesi.

La tecnica fu introdotta nel 1995 da Alon, Yuster e Zwick [5]. In generale, dato un grafo $G = (V, E)$, il problema dell'isomorfismo dei sottografi di G è un problema NP-completo. Il metodo del Color Coding permette di risolvere sottocasi di questo problema in tempo polinomiale, tramite un algoritmo randomizzato.

Il primo algoritmo che loro descrissero, però, si limitava alla ricerca di sottografi indotti in un grafo, senza farne un conteggio del numero delle occorrenze totali.

È per questo motivo che in questo capitolo si presenta un'estensione dell'algoritmo descritto da Alon [5, 6], per effettuare un conteggio delle occorrenze dei treelet all'interno del grafo, che viene effettuato in maniera simultanea. Dati in input un grafo $G = (V, E)$ e un numero k, per prima cosa il color coding assegna uniformemente e indipendentemente per ogni nodo di G un'etichetta in $[k] := \{1, ..., k\}$, indicato come un colore. L'obiettivo è quello di conteggiare il numero di occorrenze di alberi colorati non indotti di k-nodi in G - chiamati $k-treelet$ - in cui ogni vertice dell'occorrenza ha un colore distinto.

Questo viene fatto in maniera efficiente mediante programmazione dinamica, una tecnica che identifica dei sottoproblemi del problema originario, procedendo dai problemi più piccoli verso quelli più grandi.

2.1 Algoritmo

In questa sezione viene descritta l'estensione dell'algoritmo del color coding che può contare i treelet (non indotti) colorati uniformemnte a caso.

L'algoritmo inizialmente prevede una fase di colorazione, dove ad ogni nodo $v \in V$ di G è assegnato un colore c_v, scelto indipendentemente e uniformemente a caso da $[k] := \{1, \ldots, k\}$.

L'obiettivo della fase di costruzione è quello di creare una tabella con il conteggio delle occorrenze dei treelet che si possono incontrare in G.

Per ogni nodo $v \in V$ e per ogni albero colorato T_C, con k nodi, si vuole calcolare il numero $c(T_C, v)$ del numero di copie di T_C in G che sono radicate in v (si noti che qui si intendono copie non indotte).

Si può immaginare T_C come una coppia (T, C) dove $C \subseteq 1, \ldots, k$.

Quello che si fa è per ogni nodo v si inizializza $c(T_{C_0}, v) = 1$, dove T è il treelet di 1 nodo e $C_0 = \{c_v\}$. Per ogni $h = 2, \ldots, k$ a turno, si considera ogni possibile albero radicato T di dimensione h e ogni possibile insieme di colori $C \subseteq [k]$ con $|C| = h$.

Quindi $\forall v \in V$ di G, si calcola come segue il numero $c(T_C, v)$ di occorrenze dei treelet (non indotti) radicati in v isomorfi a T e i cui colori giacciono nell'insieme C. Si divide idealmente T in due sottoalberi colorati unici T' di dimensione i con $i = 1, \ldots, h-1$ e T'' di dimensione $h - i$. I due alberi saranno tali che T'' è il sottoalbero radicato in uno dei figli della radice di T e T' è il sottoalbero di T, radicato in r contenente $|T| - |T''|$ nodi.

Per calcolare $c(T_C, v)$ si analizza $\forall v \in V$ di G l'insieme dei nodi $u \in V$ tale che la coppia $(u, v) \in E$. Quindi si prendono tutti i treelet colorati radicati in v isomorfi a $T'_{C'}$ precedentemente calcolati e tutti i treelet radicati in u isomorfi a $T''_{C''}$. Se $C' \cap C'' = 0$, si procede al conteggio delle occorrenze di T_C radicato in v nel modo seguente:

$$c(T_c, v) = \frac{1}{\beta_T} \sum_{(u,v) \in E} \sum_{\substack{C' \subset C \\ C'' = C \setminus C' \\ |C'| = |T'|, |C''| = |T''|}} c(T'_{C'}, v) \cdot c(T''_{C''}, u) \tag{1}$$

dove β_T è una costante di normalizzazione che é uguale al numero di alberi di T isomorfi a T'' radicati in un figlio di r.

La correttezza e la complessità di questa costruzione, non sono trattate qui, ma vengono dimostrate in [5].
Si può vedere l'algoritmo formalmente descritto in Algoritmo 1.

Algortimo 1: Fase di costruzione

input : Grafo $G = (V, E)$, dimensione del treelet k ;
for v *in* G **do**
    $C_0 = \{c_v\}$; viene assegnato un colore uniformemente a caso preso da $[k]$;
    $c(T_{C_0}, v) = 1$;
end
for $h = 2$ *to* k **do**
    for $v \in V$ **do**
        foreach $T : |T| = h$ **do**
            $c(T_C, v) = \frac{1}{\beta_T} \sum_{(u,v)\in E} \sum_{\substack{C',C''\subset C \\ C'\cap C''=0}} c(T'_{C'}, v) \cdot c(T''_{C''}, u)$
        end
    end
end

Come si puó notare, per calcolare le occorrenze di un albero T_C nell'algoritmo si sfrutta un approccio top-down, ossia a partire dall'albero T_C si identificano i due alberi T' e T'' in cui può essere scomposto, si considerano tutte le partizioni di $C', C'' \subset C$ e in seguito si procede al calcolo di $c(T_C, v)$ come indicato in (1).

In questa tesi, cosí come in [7], invece, si è sfruttato un approccio bottom-up. Infatti, per ogni nodo $v \in V$ di G e per ogni nodo $u \in adj(v)$, si prendono tutte le coppie possibili di treelet colorati radicati rispettivamente in u e v, $T'_{C'}$ e $T''_{C''}$, tali che $|T'| + |T''| = k$ e $|T'|, |T''| > 0$.

Dallo sviluppo dell'algoritmo 1, si noti che conteggia e salva per ogni nodo $v \in V$ di G il numero di occorrenze di ogni k-treelet colorati che si può raggiungere a partire da v. Lo scopo di questa tesi però é quello di conteggiare il numero di k-treelet colorati individuabili su tutto il grafo G e le occorrenze totali per ognuno. A tale scopo è neccessario aggregare gli alberi in un unica tabella contenente i k-treelet

e le rispettive occorrenze. Per evitare di conteggiare gli alberi un numero di volte superiori a quelle effettive, vengono classificati secondo una classe di equivalenza di isomorfismo. Dati due alberi T' e T'' si dicono isomorfi se esiste una funzione bijettiva $f : T' \to T''$. Perciò se due alberi sono isomorfi, verranno raggruppati nella stessa classe di equivalenza e le loro occorrenze saranno sommate. Così facendo otterremo i conteggi effettivi di tutti i k-treelet trovati in G e il numero di volte che occorrono.

2.2 Dettagli implementativi: rappresentazione compatta dei treelet colorati

In questa sezione vengono descritte le strutture dati usate per implementare l'algoritmo in Java. Gli oggetti principali che vengono manipolati sono i treelet colorati e le occorrenze associate.

Ogni treelet colorato $T_C = (T, C)$ ha una rappresentazione unica, nella quale sono memorizzate la topologia di T, i colori in C, ed informazioni aggiuntive per facilitare la minipolazione di T_C. Tale rappresentazione richiede al più 59 bit e può pertanto essere memorizzata usando degli interi a 64 bit, un tipo di dati nativo in Java e nelle più comuni architetture moderne. I bit sono numerati da 0 a 63 e ordinati dal bit meno significativo a qullo più significativo.
Sono cosí suddivisi:

- i bit da 0-3 contengono il numero di nodi di T a meno della radice, che non è necessaria ai fini delle operazioni effettuate.
- i bit da 4-7 contengono la dimensione del sottoalbero radicato nell'ultimo figlio della radice di T.
- i bit da 8-11 vengono usati per memorizzare il valore di β_T associato a T.
- i bit da 12-27 sono usati per indicare i colori in C
- i bit da 28-58 sono usati per codificare la struttura del treelet, come descritti nel seguito.

- gli ultimi 5 bit sono lasciati a zero

Per codificare la struttura dell'albero si procede nel seguente modo.
Si esegue una visita DFS su T partendo dalla radice r e attraversando gli archi di T, in modo che, al termine della visita, ogni arco sia stato attraversato esattamente 2 volte, in direzioni opposte.

Sia $h = |T|$ e sia e_i, con $i = 1, 2, \ldots, 2(h-1)$, l'i-esimo arco attraversato dalla visita. L'i-esimo bit della codifica è 0 se e_i è attraversato in direzione di r in T e 1 in caso contrario. In Figura 2.1 si può vedere un esempio di tale codifica.

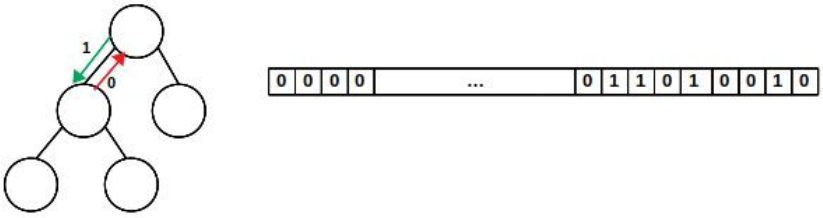

Figura 2.1: Un treelet radicato e la codifica della sua struttura

Per un qualunque $k \leq 16$ questa codifica richiede al massimo 30 bit. L'ordinamento lessicografico sulla struttura permette anche un'ordinamento totale sui treelet. Questo ordinamento è anche una regola decisiva per la visita DFS: i figli di un nodo vengono visitati nell'ordine dato dai sottoalberi radicati in esso. Ciò implica che ogni treelet T ha una codifica unica , ed ogni codifica valida corrisponde ad un solo treelet. Inoltre, in questo modo è possibile implementare rapidamente l'operazione di unione.

La codifica dei treelet supporta le seguenti operazioni :

- **singleton** (c) : permette di inizializzare un treelet di un solo nodo con il rispettivo $c \in 1, \ldots, k$
- **merge** (T',T'') : fa l'unione di due alberi T' e T'', se possibile creando un nuovo albero T che avrà come struttura la concatenazione delle strutture di T' e T''. Come dimensione, a meno della radice, T avrà la somma delle dimensioni

di T' e T'' piú 1, ossia il nodo radice di T''. I colori di T, sono dati dall'unione dei colori di T' e T''. In T come β viene memorizzato il valore corrispondente a quello di T' e se la struttura del sottoalbero piú piccolo di T' è uguale a quella di T'' viene incrementato di 1. Per finire, la dimensione del sottoalbero più piccolo radicato in T sarà esattamente la dimensione di $T'' + 1$.

- **normalization_factor** (T): restituisce la costante di normalizzazione β_T associata a T.

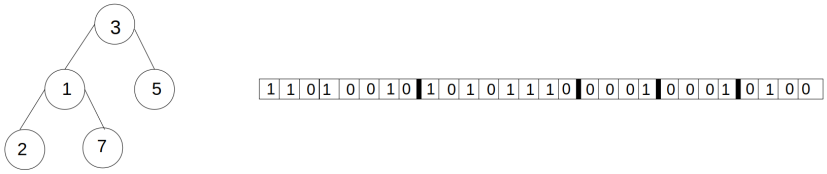

Figura 2.2: un treelet colorato e la sua codifica, mostrata per semplicità solo su 8+8+4+4+4=28 bit

Un esempio di treelet colorato e della sua codifica è mostrato in Figura 2.2.

Nell'implementazione i treelet individuati da ogni nodo e i rispettivi conteggi vengono salvati all'interno di una tabella indicizzata, rappresentata mediante strutture annidate di *ArrayList*. Inizialmente come nell'algoritmo 1 i treelet e i conteggi sono calcolati su ogni nodo $v \in V$ di G scorrendo tutti gli archi E. Solo una volta calcolati tutti i treelet su tutti i nodi v, questi verranno aggregati e i loro conteggi sommati, inoltre per ogni classe di equivalenza sarà identificato un unico rappresentante che è il k-treelet radicato nel centroide. Nella sezione **??** verrà data la definizione di centroide di un albero e l'algoritmo per la sua ricerca.
Inizialmente, perciò, quello che si ha è una tabella con k entrate, indicizzate da 1 a k ed associate al numero di vertici dei treelet.
L'i-esima entrata della tabella, con $1 \leq i \leq k$ generico, a sua volta, è un *ArrayList*, con una dimensione fissata, che varia a secondo della cardinalità di V, così che ad ogni entrata è associato un vertice $v \in V$ di G.
Per ognuna delle $|V|$ entrate, viene creato un *ArrayList* contenente tutti i treelet colorati di dimensione i raggiungibili dai differenti nodi in V, insieme al relativo

numero di occorrenze.
All'interno di quest'ultima lista i treelet sono ordinati in ordine non descrescente.
La tabella ha una costruzione dinamica, perciò l'i-esima entrata è costruita solo dopo aver terminato la costruzione della $i-1$-esima.

Per permettere una costruzione più rapida della tabella, nell'implentazione è utilizzato il *Multithreading*, cioè vengono utilizzati più thread che lavorano in parallelo. Un thread è un flusso di esecuzione indipendente all'interno di un processo. Il numero di thread utilizzato nell'implementazione è dipendente dal numero di processori presenti nella macchina. Supponendo di avere un pc con 4 processori, vengono usati 3 thread, uno in meno rispetto al numero di processori. Per garantire un funzionamento in parallelo senza conflitti è dovuto provvedere affinchè nessuna operazione necessitasse di semafori di mutua esclusione (mutex) e che ogni tipo di dato coinvolto fosse atomico, ossia non scomponibile.
Riguardo le operazioni sugli ArrayList non è stato necessario ricorrere alla mutua esclusione. Per le variabili è stato necessario, invece, garantire la visita dei vertici del grafo in maniera atomica e a tal proposito i vertici, in ogni ArrayList, sono rappresentati mediante *AtomicInteger*, ossia valori interi che possono essere aggiornati in maniera atomica.

CAPITOLO 3

TECNICA OTTIMIZZATA

In questo capitolo viene presentata un'ottimizzazione al problema descritto nel capitolo precedente. Tale ottimizzazione si basa sul principio delle decomposizioni bilanciate. Si fa vedere, inoltre, come l'utilizzo di quest'ultima permetta un notevole miglioramento delle performance.

3.1 Decomposizioni Bilanciate di un albero

Si vuole andare a dimostrare in questa sezione che dato un albero T è sempre possibile ricavare una scomposizione bilanciata dell'albero.
Prima di poter enunciare e dimostrare il risultato principale occorre dare delle nozioni preliminari.

Definizione 3.1.1. *Sia T_r un albero radicato nel nodo r, con k nodi. Diremo che la coppia (A,B), dove A e B sono insiemi contenenti i nodi di T_r, è una decomposizione per l'albero T_r se:*

- $|A| + |B| = k$
- $A \cap B = \{r\}$.

Definizione 3.1.2. *Dato un albero T con k nodi, diremo che (A, B) è una decomposizione bilanciata se:*

$$\max\{|A|, |B|\} \leq f(k)$$

dove f è una funzione definita su k.

Definizione 3.1.3. *Per ogni nodo v di un albero T, le diramazioni di T rispetto a v, sono tutti i sottoalberi massimali di T non contenenti v. Per ogni $v \in T$, si definisce $\alpha(v)$ come il grado della diramazione di v con il maggior numero di nodi. Un nodo v di un albero T con n nodi, è un nodo centroide se $\alpha(v) \leq \frac{n}{2}$.*

Il centroide di un albero non è necessariamente unico, infatti Jordan [8] ha dimostrato che, dato un albero T con n nodi:

i T ha un singolo centroide v e $\alpha(v) < \frac{n}{2}$;

ii T ha due nodi centroidi (adiacenti) v_1 e v_2 tali che $\alpha(v_1) = \alpha(v_2) = \frac{n}{2}$, in questo caso il numero di nodi n è pari.

Esistono diversi algoritmi per la ricerca del centroide, quello utilizzato in questa tesi è l'algoritmo di Jordan [8] che ha una complessità temporale lineare nel numero di nodi.
Il primo passo da effettuare è determinare $\alpha(v)$ per ogni nodo $v \in T$.
Si definisce $\delta(z)$ come il fattore di diramazione di un albero. ossia il numero di nodi incontrati durante una visita DFS (in profondità) effettuata a partire dalla radice z. Siano $\delta(z_i)$ i fattori di bilanciamento ottenuti da tutte le possibili diramazioni i di T unite a v, che non lo contengono, allora, $\alpha(v) = \max\{\delta(z_i)\}$.

Una volta calcolato il valore di $\alpha(v)$ $\forall v \in T$ si verifica per quali valori risulta $\alpha(v) \leq \frac{n}{2}$. Nel caso ci fosse un unico nodo v che soddisfa la precedente espressione,, come in (i), allora tale nodo rappresenta l'unico centroide dell'albero T. Nel caso, invece, ce ne fossero due, come definito in (ii), per esempio v_1 e v_2, l'albero T conterrà due centroidi, rispettivamente v_1 e v_2.
Nell'esempio 3.1.4 si può vedere l'applicazione dell'algoritmo per la ricerca del centroide.

Esempio 3.1.4. *Si consideri l'albero T in figura 3.1 per la ricerca del nodo centroide. Per ogni nodo v di T, numerato da 0 a 10, viene calcolato $\alpha(v)$.*
Quello che si ottiene è :

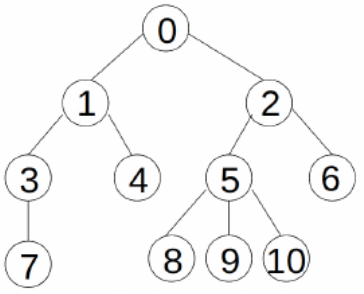

Figura 3.1: Albero T per la ricerca del centroide

$$\begin{array}{lll} \alpha(0) = 6 & \alpha(1) = 7 & \alpha(2) = 5 \\ \alpha(3) = 9 & \alpha(4) = 10 & \alpha(5) = 7 \\ \alpha(6) = 10 & \alpha(7) = 10 & \alpha(8) = 10 \\ \alpha(9) = 10 & \alpha(10) = 10 & \end{array}$$

Poiché $\left\lfloor \frac{n}{2} \right\rfloor = \left\lfloor \frac{11}{2} \right\rfloor = 5$*, l'unico nodo per cui la disuguaglianza risulta vera è il nodo 2, infatti* $5 \leq 5$*, ed è esattamente l'unico centroide dell'albero* T *(figura 3.1).* □

L'ultimo punto da considerare prima di poter enunciare e dimostrare il risultato principale di questa sezione riguarda la definizione di un algoritmo valido per comporre due insiemi di nodi, che chiameremo T' e T'', in maniera $f(k)$-bilanciata.
Siano dati in input un albero T, con k nodi, ed un fattore di bilanciamento definito da una funzione $f(k)$, poiché non considero la radice di T, la funzione viene valutata su $k-1$ piuttosto che su k. Si suppone inoltre, senza perdita di generalità, che i sottoalberi radicati nei figli della radice r di T siano ordinati con ordine non crescente. Da questo deriva che, supponendo che r abbia n figli, vi saranno al più n alberi radicati in ciascuno di essi tali che: $|T_i| \geq |T_{i+1}| \quad \forall i = 1, \ldots, n-1$.

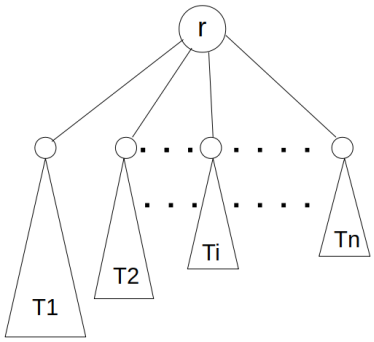

Figura 3.2: Esempio di albero T radicato in r, con k nodi, valido come input per l'algoritmo 2

Algortimo 2: Insiemi k-bilanciati

input : Albero T con k nodi, fattore di bilanciamento $f(k-1)$;
$|T'|, |T''| \longleftarrow$ due insiemi di nodi inizialmente vuoti;
for $i = 1, \ldots, n$ **do**
    if $\sum_{j=1}^{i} |T_j| > f(k-1)$ **then**
        $T' \longleftarrow$ sottoalbero di di T indotto da $\{r\} \cup \bigcup_{j=1}^{i-1} V(T_j)$.
        $T'' \longleftarrow$ sottoalbero di T indotto da $V(T) \setminus T'$.
    end
end
Restituisci (T', T'').

Una volta concluso l'algoritmo 2 si avrà una coppia (T', T'') tale che : $\max(|T'|, |T''|) \leq f(k-1)$. Viene pertanto rispettata la definizione 3.1.2, perciò si ottiene una decomposizione $f(k-1)$-bilanciata.

In base a tutte le nozioni fino ad ora discusse si può dare il seguente risultato

.

Teorema 3.1.5. *Per ogni albero T di k nodi esiste un nodo r di T tale che l'albero T_r, ottenuto radicando T in r, ammette una decomposizione $(\lfloor \frac{2}{3}(k-1) \rfloor + 1)$-bilanciata ed, inoltre, sia $(|T'|, |T''|)$ tale decomposizione risulta che il* $\max\{|T'|, |T''|\} \geq 2 + \left\lfloor \frac{(k-1)}{3} \right\rfloor$

Dimostrazione. Sia T un albero di k nodi,con $k > 2$ (per $n \leq 2$ la proprietà è banalmente vera)
La prima operazione da compiere è individuare il nodo r di T su cui si andrà poi a radicare il nuovo albero T_r.
Si suppone che tale nodo sia un centroide dell'albero T quindi si applica l'algoritmo di Jordan per la ricerca del centroide, senza perdita di generalità si suppone di avere un unico nodo centroide, indicato con r.
Se il centroide r trovato non corrisponde alla radice dell'albero T, si modifica T radicandolo in r e l'albero così ottenuto verrà indicato con T_r.
Inoltre, si suppone che i sottoalberi radicati nei figli di r siano ordinati in maniera non crescente rispetto alla loro dimensione.
Si applichi a T_r l'algoritmo 2 precedentemente descritto per definire se è possibile ottenere una $f(k)$-decomposizione, con $f(k) = (\lfloor \frac{2}{3}(k-1) \rfloor + 1$.
Sia $\{T_i \mid i = 1, \ldots, n\}$ l'insieme dei sottoalberi radicati negli n figli di r e si consideri il primo valore di i tale che l'istruzione if di riga 4 nell'algoritmo 2 risulti vera (si noti che tale valore di i esiste sempre dal fatto che per $i = n$ la condizione è verificata).

Sia

$$S = \sum_{i=1}^{n} |T_i| = (k-1)$$

e sia

$$x = \sum_{j=1}^{i-1} |T_j|$$

Distinguiamo due casi

- **i>2** Si ha che

$$x + |T_i| > \frac{2}{3} \cdot S \tag{2}$$

Inoltre sapendo che

$$|T_i| \leq \frac{S}{i} \leq \frac{S}{3} \tag{3}$$

Sottraendo la disequazione (3) alla (2) si ottiene che

$$x > \frac{2}{3} \cdot S - \frac{S}{3} = \frac{S}{3} \tag{4}$$

- **i=2** Anche in questo caso come nel precedente vale la disequazione (2). Inoltre, essendo $i = 2$ per costruzione dell'albero T si può dire che

$$x = |T_1| \geq |T_2| = |T_i| \tag{5}$$

Pertanto, sfruttando la disequazione (5) combinata con la (2) si ha che

$$2x > \frac{2}{3} \cdot S \Rightarrow x > \frac{S}{3} \tag{6}$$

Per entrambi i casi otteniamo che

$$x > \frac{S}{3} \Rightarrow x \geq \left\lfloor \frac{S}{3} \right\rfloor + 1$$

Pertanto si avrà che

$$\left\lfloor \frac{S}{3} \right\rfloor + 1 \leq x \leq \left\lfloor \frac{2}{3} \cdot S \right\rfloor$$

Quindi

$$|T'| = 1 + x \leq 1 + \left\lfloor \frac{2}{3} \cdot S \right\rfloor = 1 + \left\lfloor \frac{2}{3} \cdot (k-1) \right\rfloor \tag{7}$$

$$|T''| = 1 + S - x = 1 + S - 1 - \left\lfloor \frac{S}{3} \right\rfloor = \left\lceil \frac{2}{3} \cdot S \right\rceil = \left\lceil \frac{2}{3} \cdot (k-1) \right\rceil \tag{8}$$

Inoltre

$$|T'| = 1 + x \geq 1 + (1 + \left\lfloor \frac{S}{3} \right\rfloor) = 2 + \left\lfloor \frac{(k-1)}{3} \right\rfloor$$

Poichè

$$1 + \left\lfloor \frac{2}{3} \cdot (k-1) \right\rfloor \geq \left\lceil \frac{2}{3} \cdot (k-1) \right\rceil$$

Possiamo concludere che

$$2 + \left\lfloor \frac{(k-1)}{3} \right\rfloor \leq \max\{|T'|, |T''|\} \leq 1 + \left\lfloor \frac{2}{3} \cdot (k-1) \right\rfloor$$

ottenendo perció una decomposizione $(\lfloor \frac{2}{3}(k-1) \rfloor + 1)$-bilanciata, tale che $\max\{|T'|, |T''|\} \geq 2 + \left\lfloor \frac{(k-1)}{3} \right\rfloor$

□

Nel caso in cui si abbiano due centroidi, la scelta su quale radicare l'albero è deterministica.

Nel caso in cui gli alberi ottenuti radicando T in ognuno di essi abbiano strutture

differenti, viene scelto quello che tra i due ha una struttura più piccola. Altrimenti se gli alberi ottenuti sono identici, viene scelto uno dei due in maniera arbitraria.

3.2 Algoritmo

In questa sezione si vede come è stato utilizzato il risultato del paragrafo 3.1 per ottimizzare e migliorare l'algoritmo 1 descritto nel capitolo 2 paragrafo 2.1.
Molto brevemente, quello che si faceva precedentemente era la seguente cosa. Dato $T_C = (T, C)$, con T un albero colorato radicato di k nodi i cui colori giacciono in C, si procedeva a calcolare il numero di occorrenze $c(T_C, v) \quad \forall v \in V$ dividendo idealmente T in due sottoalberi T' e T'' con T'' il sottoalbero radicato in uno dei figli della radice di T e T' il sottoalbero di T , radicato in r contenente $|T| - |T''|$ nodi. Si procedeva al conteggio delle occorrenze di T_C nel seguente modo

$$c(T_c, v) = \frac{1}{\beta_T} \sum_{(u,v)\in E} \sum_{\substack{C'\subset C \\ C''=C\setminus C' \\ |C'|=|T'|,|C''|=|T''|}} c(T'_{C'}, v) \cdot c(T''_{C''}, u)$$

In questa nuova versione, invece, la divisione dell'albero T_C non è più scelta in maniera ideale, ma bensì vengono presi due alberi $T'_{C'}$ e $T''_{C''}$ tali che la decomposizione di T_C risulti bilanciata. Inoltre, a differenza della precedente versione, gli alberi $T'_{C'}$ e $T''_{C''}$ saranno radicati entrambi nello stesso nodo v su cui sarà poi radicato T_C.

BIBLIOGRAFIA

[1] Noga Alon, Phuong Dao, Iman Hajirasouliha, Fereydoun Hormozdiari, and S Cenk Sahinalp. Biomolecular network motif counting and discovery by color coding. *Bioinformatics*, 24(13):i241–i249, 2008.

[2] Mansurul A Bhuiyan, Mahmudur Rahman, Mahmuda Rahman, and Mohammad Al Hasan. Guise: Uniform sampling of graphlets for large graph analysis. In *2012 IEEE 12th International Conference on Data Mining*, pages 91–100. IEEE, 2012.

[3] Zhao Zhao, Maleq Khan, VS Anil Kumar, and Madhav V Marathe. Subgraph enumeration in large social contact networks using parallel color coding and streaming. In *2010 39th International Conference on Parallel Processing*, pages 594–603. IEEE, 2010.

[4] Paolo Boldi, Marco Rosa, Massimo Santini, and Sebastiano Vigna. Layered label propagation: A multiresolution coordinate-free ordering for compressing social networks. In *Proceedings of the 20th international conference on World wide web*, pages 587–596, 2011.

[5] Noga Alon, Raphael Yuster, and Uri Zwick. Color-coding. *Journal of the ACM (JACM)*, 42(4):844–856, 1995.

[6] Marco Bressan, Flavio Chierichetti, Ravi Kumar, Stefano Leucci, and Alessandro Panconesi. Motif counting beyond five nodes. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 12(4):1–25, 2018.

[7] Marco Bressan, Stefano Leucci, and Alessandro Panconesi. Motivo: fast motif counting via succinct color coding and adaptive sampling. *Proceedings of the VLDB Endowment*, 12(11):1651–1663, 2019.

[8] Camille Jordan. Sur les assemblages de lignes. *J. Reine Angew. Math*, 70(185):81, 1869.