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 permette di avere una conoscenza delle interazioni tra le proprietà strutturali e i nodi del grafo e inoltre fa luce sul tipo di strutture locali presenti in esso, che possono essere usate per una miriade di analisi. 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 sottografi di una data dimensione cresce in modo esponenziale, 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 è basato sull'utilizzo delle catene di Markov Monte Carlo, mentre il secondo è quello della tecnica del Color Coding introdotta da Alon, Yuster e Zwick [1]. Studi recenti mettono in luce e studiano le differenze tra i due approcci [2]. In questa tesi ci concentreremo solo sulla tecnica del Color Coding. Tale tecnica è stata introdotta da Alon, Yuster e Zwick in [1], per risolvere in ma-

niera randomizzata il problema di determinare l'esistenza di cammini ed alberi in G. 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, tali tecniche sono state utilizzate per l'analisi di reti sociali e biologiche [2, 3, 4]. 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 i graphlet 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 la ricerca 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 [4] su una rete PPI (Protein-Protein Interaction)quanto la ricerca di k-treelet in un grafo può essere utile per la ricerca della frequenza di particolari strutture biomolecolari (unicellulari e pluricelluri). Per effettuare tale ricerca è stato necessario concentrarsi sulla fase costruttiva descritta in precedenza.

La fase costruttiva, è descritta mediante una programmazione dinamica , è un processo che però richiede un grande impiego di tempo e spazio. Il lavoro svolto ha portato, per prima cosa ad un'implementazione, in Java dell'algoritmo noto [2]. Il programma permette la ricerca delle occorrenze dei diversi k-treelet, all'interno del grafo. L'approccio dell'algoritmo utilizzato in questa tesi è bottom-up, per cui, supposto di dover conteggiare i treelet di dimensione k di un grafo, l'algoritmo lavora in esattamente k fasi. Nell'i-esima fase saranno conteggiati i treelet di dimensione i, ottenuti dalla composizione di tutti quelli con dimensione minore di i,perciò per

poter calcolare i treelet di dimensione k, sarà necessario aver già calcolato quelli di dimensione fino a k-1. Questo meccanismo rispetta ciò che viene dalle formule ricorsive della programmazione dinamica. Poichè il numero degli alberi cresce in maniera esponenziale rispetto a k l'algoritmo richiede, al crescere di k, sempre più tempo per essere eseguito. 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, circa $\frac{2}{3}$ k fasi prima della fase k, comportando un risparmio notevole di tempo. Ad esempio su un grafo 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 [2] e la sua formulazione "top-down". Si discuterà la scelta di adottare un approccio "bottom-up" 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 discuterà sulle scelte effettuate in fase implementativa. Nel capitolo 4 si discuteranno i risultati di un'analisi sperimentale delle performance dell'algoritmo ottimizzato rispetto alla versione di [2]. 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 questo studio.

La tecnica fu presentata per la prima volta nel 1995 da Alon, Yuster e Zwick [1]. 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.

Dati due grafi, $G = (V, E)$ e $H = (V_H, E_H)$, i vertici V di G, in cui viene cercato un sottografo isomorfo ad H, sono colorati casualmente di $k = |V_H|$ colori. Se $|V_H| = O(\log(V))$, allora tutti i vertici del sottografo di G isomorfo ad H, se esiste, sono colorati da colori distinti.

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 [1], per effettuare un conteggio delle occorrenze dei Motif all'interno del grafo. 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 alberi colorati non indotti di $k-nodi$ in G - chiamati *treelet* - i cui colori non sono ripetuti. Questo viene fatto in maniera efficiente mediante programmazione dinamica, una tecnica bottom-up che identifica dei sottoproblemi

del problema originario, procedendo logicamente dai problemi più piccoli verso quelli pi̇grandi.

2.1 Algoritmo

Qui viene descritta l'estensione dell'algoritmo del color coding che può contare e campionare treelet (non indotti) colorati uniformemnte a caso. L'algoritmo consiste in una fase di costruzione e una fase di campionamento, in questo studio però non si vede quest'ultima fase, poichè non rilevante ai fini di questa tesi, ma si concentra sulla prima fase. L'algoritmo inizialmente prevede una fase di colorazione, dove per ogni nodo $v \in V$ di G è assegnato un colore c_v, scelto indipendentemente e uniformemente a caso da $[k] := \{1, ..., k\}$. 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 e per ogni albero colorato T_C con k nodi, si vuole un conteggio $c(T_C, v)$ del numero di copie di T_C in G che sono radicate in v (si noti che qui si intendono copie non indotte). A questo fine per ogni nodo v si inizializza $c(T_C, v) = 1$, dove T è il treelet triviale di 1 nodo e $C = \{c_v\}$. Successivamente si esegue una programmazione dinamica per il conteggio di treelet di dimensione $h = 2, ..., k$. Per ogni h a turno, si considera ogni possibile albero radicato T con $h \leq k$ nodi e ogni possibile insieme di colori $C \subseteq [k]$ con $|C| = h$. Poi, $\forall v \in V$ 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, unici T' e T'' radicati rispettivamente nella radice r di T e in uno dei figli di r.
Perciò $c(T_C, v)$ è dato come segue:

$$c(T_c, v) = \frac{1}{\beta_T} \sum_{u \sim v} \sum_{\substack{C', C'' \subset C \\ C' \cap C'' = 0}} c(T'_{C'}, v) \cdot c(T''_{C''}, u) \tag{1}$$

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. Per calcolare $c(T_C, v)$ si passa attraverso tutti gli archi $u \sim v$ di G, combinando i conteggi di u e v. La correttezza e la complessità di questa costruzione, non sono trattate qui,ma vengono dimostrate in [1].

Algorithm 1: Fase di costruzione

input : Grafo G, dimensione del treelet k ;

for v *in* G **do**

  c_v = viene assegnato un colore preso da $[k]$;

  $c(T_{c_v}, v) = 1$;

end

for $h = 2$ *to* k **do**

  for v *in* G **do**

    foreach $T : |T| = h$ **do**

      $c(T_c, v) = \frac{1}{\beta_T} \sum_{u \sim v} \sum_{\substack{C', C'' \subset C \\ C' \cap C'' = 0}} c(T'_{C'}, v) \cdot c(T''_{C''}, u)$

    end

  end

end

Come si nota, l'algoritmo itera su tutte le coppie di conteggi $c(T'_{C'}, v)$ e $c(T''_{C''}, u)$ per ogni arco $u \sim v$ e, se $T'_{C'}, T''_{C''}$ possono essere unite in un albero colorato T_C, allora si aggiunge $c(T'_{C'}, v) \cdot c(T''_{C''}, u)$ al conteggio $c(T_C, v)$. Per fare questo è necessaria un'operazione di "controllo e unione", che risulta abbastanza costosa. Infatti, per calcolare $c(T_C, v)$, per ogni coppia di conteggi Una semplice analisi ha restituito il seguente limite di complessità:

Teorema 2.1.1. *([2] Teorema 5.1) La fase di costruzione richiede tempo $O(a^k|E|)$ e spazio $O(a^k|V|)$, per un qualche $a > 0$.*

La grandezza della tabella ottenuta da questa programmazione è il problema maggiore per l'algoritmo, infatti per $k = 6$ e $|V| = 5M$, sono necessari $45G$ di memoria.

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

In questo studio, invece, si è sfruttato un approccio bottom-up. Infatti, per ogni nodo $v \in V$ di G e per ogni nodo u adiacente a v, si prendono tutti le coppie possibili di treelet colorati radicati rispettivamente in u e v, $T'_{C'}$ e $T''_{C''}$, entrambi di dimensioni minori di h, con $h \leq k$. Se $C' \cap C'' = 0$ e la struttura di $T''_{C''}$ è minore della struttura del più piccolo sottoalbero radicato nella radice di $T'_{C'}$, secondo l'ordinamento totale dei treelet (vedere 2.2), allora i due treelet possono essere uniti a creare T_C di dimensione esattamente h le cui occorrenze saranno determinate come in 1.

2.2 Struttura dati

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 ha una rappresentazione unica, nella quale sono memorizzati anche i colori. Vengono codificati come stringhe binarie da 64 bit. I bit sono cosí suddivisi :

- i bit da 0-3 contengono la dimensione del treelet a meno della radice
- i bit da 4-7 contengono la dimensione del sottoalbero piú piccolo radicato nella radice del treelet
- i bit da 8-11 vengono usati per memorizzare il valore di β associato al treelet.
- i bit da 12-27 sono usati per indicare i colori dell'albero
- i bit da 28-58 sono usati per codificare la struttura del treelet .
- gli ultimi 4 bit sono lasciati a zero

Per codificare la struttura dell'albero si procede nel seguente modo. Si esegue una visita DFS trasversale su T partendo dalla radice r. Poi l'i-esimo bit della codifica è 1 (rispettivamente 0) se l'i-esimo arco si visita in direzione opposta a r (rispettivamente se si visita in direzione di r). In Figura 2.1 si può vedere un esempio di tale codifica.

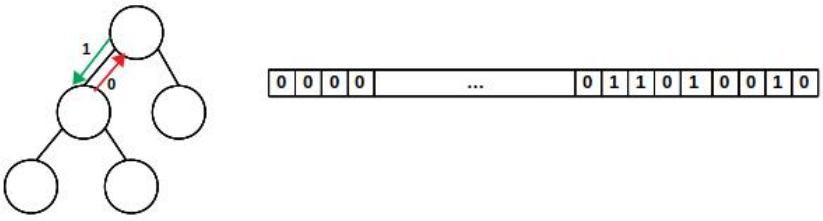

Figura 2.1: Un treelet radicato e la codifica della sua struttura

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 ben definita della struttura. Inoltre, in questo modo si garantisce un'operazione di unione molto rapida.

La codifica dei treelet supporta le seguenti operazioni :

- **singleton** (int color) : permette di inizializzare un treelet di un solo nodo con il rispettivo colore
- **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.

Un esempio di treelet colorato e della sua codifica è data nella Figura 2.2.

Per quanto riguarda i conteggi delle occorrenze dei treelet sono memorizzati in una tabella con k entrate ognuna delle quali composta da altrettante n tabelle, una per ogni nodo $v \in G$. La coppia $(T_C, c(T_C, v))$ è memorizzata in maniera strutturata nelle tabelle, tramite le istanze di una classe *Entry*, contenente gli elementi della

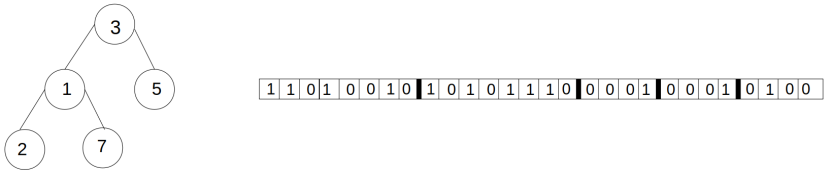

Figura 2.2: un treelet colorato e la sua codifica, mostrata per semplicità solo su 8+8+4+4+4=28 bit

la coppia, ordinate rispetto la dimensione dei treelet. Le tabelle sono costruite in maniera dinamica. La prima tabella, contiene $\forall v \in G$ la coppia $(T_0, 1)$ dove T_0 è il treelet triviale colorato. A seguire sfruttando la variante dell'algoritmo del color coding descritta in 2.1 vengono calcolate tutte le tabelle. Quindi l'i-esima tabella, con $2 \leq i \leq k$, sarà costruita sfruttando tutte le $h - 1$ tabelle già calcolate. Così facendo nella tabella con $i = k$, saranno contenuti tutti i k-treelet di G e le loro occorrenze.

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 questo paragrafo, che dato un albero T è sempre possibile ricavare una scomposizione bilanciata dell'albero.
Prima di poter enunciare il teorema e dimostrarlo 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 se:*

- $|A| \cup |B| = k$
- $A \cap B = r$.

Lemma 3.1.2. *Dato un albero T, e una sua decomposizione (A, B), affinchè tale decomposizione sia bilanciata dovrà risultare che:*

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

dove f(k) è il fattore di bilanciamento, varia in funzione al numero dei nodi dell'albero ed è pari a:

$$f(k) = \left\lceil \frac{2}{3}k \right\rceil$$

Dimostrazione. Per dimostrare il lemma si fa una dimostrazione costruttiva, mediante il seguente algoritmo. In input si ha, un albero T e due insiemi di nodi A e B entrambi inizialmente vuoti. Per prima cosa si inserisce la radice la radice r di T in entrambi gli insiemi. Si supponga, che il numero dei sottoalberi radicati nella radice di T sia $i \geq 2$ e che i T_i sottoalberi siano ordinati per dimensione, in ordine non crescente. Per prima cosa si calcola il valore di $f(k)$ applicato ai nodi dell'albero T. Il primo passo inserisce in A i nodi di T_1. Il secondo prova ad aggiunge ad A i nodi di T_2. Se $|A| + |T_2| \leq f(k)$ allora aggiorna A inserendo i nodi di T_2 e si ripete il passo due su T_3, altrimenti l'algoritmo si arresta inserendo tutti i nodi dei sottoalberi rimanenti in B. In realtá una volta trovato un T_i che non può essere aggiunto ad A l'algoritmo potrebbe continuare a vedere i sottoalberi a seguire, ma in questo caso si suppone che l'algoritmo si arresti. Una volta concluso l'algoritmo l'insieme A arriverà a contenere un numero di elementi tale che $|A| \leq \lceil \frac{2}{3}k \rceil$ □

Come si può capire dalla dimostrazione del Lemma 3.1.2, le decomposizioni bilanciate così come sono definite fino ad ora non possono essere applicate su tutti gli alberi, ma solo su alcuni. Occorrono, perciò ulteriori definizioni.

Definizione 3.1.3. *Per ogni nodo v di un albero T, le diramazioni di T rispetto a v, sono tutti i sottoalberi massimali, radicati nei figli di v. Sia $\alpha(v)$ il numero di nodi della massima diramazione di v.*

Un nodo u di un albero T con n nodi, è un nodo centroide se $\alpha(u) \leq \frac{n}{2}$.

Il centroide di un albero non è necessariamente unico, infatti Jordan [5] ha dimostrato che o (i) T ha un singolo centroide v e $\alpha(v) < \frac{n}{2}$ oppure (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 da noi usato è l'algoritmo di Jordan [5] che ha una complessità temporale lineare al numero di nodi O(n).

Il primo passo da fare è determinare $\alpha(v)$ $\forall v \in T$.

Per poter calcolare $\alpha(v)$ $\forall v \in T$, inizialmente si effettua una visita DFS (Depth First Search) dell'albero, in modo da definire la cardinalità di ogni possibile sottoalbero a partire dalla radice, procedendo su ogni suo figlio . I sottoalberi ottenuti dai nodi foglia e dal nodo radice, sono alberi banali che avranno rispettivamente cardinalità 1 e $|T|$.

A questo punto si può procedere con l'individuazione di $\alpha(v)$, ossia $\forall v$ della diramazione con un maggior numero di nodi.

Per farlo, si considera ogni nodo v di T e si calcola la cardinalità dei sottoalberi radicati in ognuna delle sue possibili diramazioni. $\alpha(v)$ sarà il valore massimo tra tutte le quantità individuate. Una volta determinato $\alpha(v)$ $\forall v \in T$, si procede alla ricerca del centroide, che sarà il nodo di T per cui vale la seguente disuguaglianza:

$$\alpha(v) \leq \left\lfloor \frac{n}{2} \right\rfloor$$

Nell'esempio 3.1.4 si può vedere l'applicazione dell'algoritmo per la ricerca del centroide.

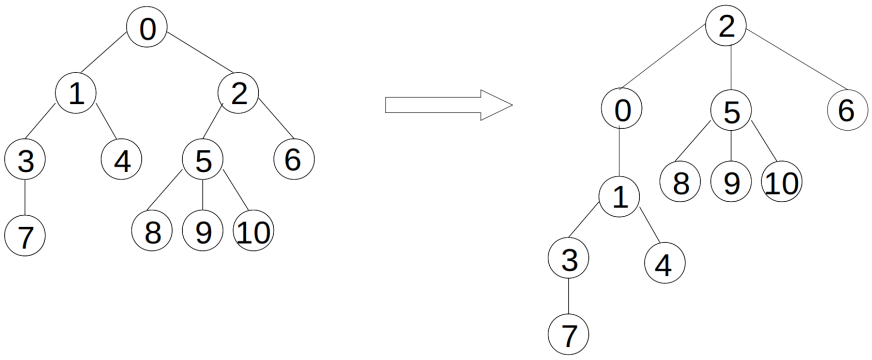

Figura 3.1: Dato un albero T, si vede come può cambiare la struttura se radicato nel suo centroide

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

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

Poiché $\left\lfloor \frac{n}{2} \right\rfloor = \left\lfloor \frac{11}{2} \right\rfloor = 5$*, basta verificare per quale nodo v di T , vale che* $\alpha(v) \leq \left\lfloor \frac{n}{2} \right\rfloor$*. L'unico nodo per cui tale disuguaglianza risulta vera è il nodo 2, infatti* $5 \leq 5$*, e sarà l'unico centroide dell'albero T (figura* **??***).*

Come giá accennato l'algoritmo ha complessità lineare sul numero dei nodi. Infatti, occorre calcolare il grado di ogni nodo e successivamente verificare che ci siano le condizioni affichè risulti un centroide e si ha:

$$\sum_v (1 + \delta(v)) = n + \sum_v \delta(v) = n + n - 1 = 2n - 1 = O(n)$$

In base a tutte le nozioni illustrate in precedenza possiamo passare ad enunciare il teorema di seguito.

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 bilanciata.*

Dimostrazione. Sia T un albero con più di due nodi(per $n \leq 2$ caso banale).
La prima operazione da compiere è individuare il nodo r di T, su cui si andrà poi a radicarsi l'albero. Banalmente, per come è stato definito, il nodo che si cerca, non è altro che un centroide dell'albero T, quindi si applica lâĂŹalgoritmo precedentemente descritto per la sua ricerca. Una volta trovato, questo sarà il nuovo nodo su cui sarà radicato l'albero T, che da questo punto sarà indicato con T_r.
Inoltre supponiamo, senza perdere di generalità, che i sottoalberi radicati nei figli di r siano ordinati in maniera non crescente rispetto alla loro dimensione.
È possibile ottenere una scomposizione, (A,B), dei k nodi di T_r, tale che un insieme,

ad esempio A, contenga al massimo i $\lceil \frac{2}{3} \rceil$ dei nodi dell'albero e B il restante di essi, come descritto nel Lemma 3.1.2. Ovviamente questo algoritmo terminerà, poiché il numero di nodi è finito. Inoltre l'insieme con il maggior numero di elementi non contiene più dei $\lceil \frac{2}{3} \rceil$ del totale.
Per dimostrarlo si osserva che A deve contenere almeno $\frac{1}{3}$ dei nodi totali.
Si supponga che dopo aver applicato l'algoritmo descritto nella dimostrazione del Lemma 3.1.2, A contenga i $\lceil \frac{2}{3} \rceil$ dei nodi totali. Sia x il primo elemento non in A e sia i la sua posizione (figura 3.2).

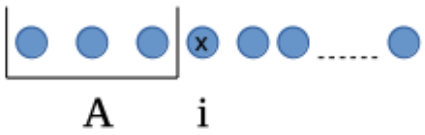

Figura 3.2

Si possono verificare due casi:

- (i=2) L'insieme A è formato da un unico elemento y:

Figura 3.3

Per come è costruito di A, si ha certamente che:

$$y + x > \frac{2}{3}k \tag{2}$$

Dividendo entrambi i membri di (1) per due, si ottiene:

$$\frac{x+y}{2} > \frac{k}{3} \tag{3}$$

Si nota che $\frac{x+y}{2}$ rappresenta esattamente il valore medio.
Dall'ordinamento dei sottoalberi di T_r, risulta che $y \geq x$ perciò si ha che:

$$y \geq \frac{x+y}{2} \tag{4}$$

unendo la (2) e la (3) si ottiene:

$$y > \frac{k}{3}$$

- ($i \geq 3$) In A vi sono almeno due elementi. Sia S il valore ottenuto dalla loro somma

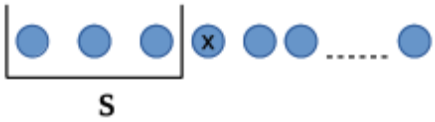

Figura 3.4

Si ha che:

$$S + x > \frac{2}{3} \tag{5}$$

Inoltre, per costruzione:

$$x \leq \frac{k}{3} \tag{6}$$

Sottraendo la (5) alla (4), ammissibile poiché rispetta le regole delle disequazioni, si ottiene:

$$S + x - x > \frac{2}{3}k - \frac{k}{3} \quad \text{ossia} \quad S > \frac{k}{3} \tag{7}$$

Perciò gli insiemi ottenuti dalla scomposizione di T_r hanno cardinalità compresa tra $\frac{1}{3}k$ e $\frac{2}{3}k$, garantendo così delle decomposizioni bilanciate.
Nel caso in cui si abbiano due centroidi, la scelta su quale radicare l'albero è deterministica e viene fatta prendendo quello che, tra i due,garantisce un'ordinamento corretto dei sottoalberi radicati nella radice oppure, a parità di ordinamento, ne viene scelto uno dei due in maniera arbitraria.

□

BIBLIOGRAFIA

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

[2] 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.

[3] 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.

[4] 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.

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