
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 ed il loro conteggio è un problema ben noto del graph mining e dell'analisi dei social network. Più precisamente, 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 molteplici tipi di analisi applicate, per esempio, a reti sociali [1, 2, 3], biologiche [4], .... Poichè il conteggio esatto del numero di occorrenze dei graphlet risulta computazionalmente impegnativo, di solito ci si accontenta di obiettivi meno ambiziosi come la stima *approssimata* della loro frequenza: per ogni sottografo di $k$ nodi si richiede di stimare, nel modo più accurato possibile, la sua frequenza relativa rispetto a tutti i sottografi della stessa dimensione (i.e., numero di nodi). Dal momento che il numero di motif cresce più che esponenzialmente rispetto al numero di nodi $k$, spesso si restringe l'attenzione al problema della stima della frequenza relativa dei soli sottografi che compaiono il maggior numero di volte nel grafo input. Ci sono due approcci principali per ottenere tali stime: Il primo utilizza i metodi di Monte Carlo basati sulle catene di Markov (MCMC), 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 che, seppure l'elevata complessità spaziale delle tecniche basate sul color coding può limitarne l'utilizzo, tali

tecniche forniscono garanzie di accuratezza migliori rispetto agli approcci MCMC. Per tale motivo, nel resto di 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 inoltre 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 a quello di campionare $k$-alberi, ossia 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 "estraggono" degli alberi dall'urna fino ad ottenere delle stime accurate per i graphlet di interesse.

## 1.1 Contributo della tesi

In questo lavoro di tesi l'attenzione si è concentrata principalmente sull'ottimizzazione di un algoritmo basato sulle tecnica del Color Coding per il conteggio delle occorrenze di tutti i $k$-treelet presenti nel grafo $G$ in input, dove per $k$-treelet si intendono alberi con $k$ nodi e ed un occorrenza di un $k$-treelet $T$ in $G$ è un sottografo (non necessariamente indotto) isomorfo a $T$. Ciò corrisponde alla "fase di costruzione" descritta in precedenza. Tale fase può essere descritta mediante un algoritmo basato sulla tecnica della programmazione dinamica. In primo luogo, tramite il lavoro svolto, si è implementato l'algoritmo descritto in [6] in Java. L'algoritmo, supposto di voler conteggiare i treelet di dimensione $k$ di un grafo, lavora in esattamente $k$ fasi. Nell'$i$-esima fase (con $i \in \{1, \ldots, k\}$), vengono conteggiati i treelet di dimensione $i$. Tali conteggi saranno ottenuti in funzione della struttura del grafo e del numero di occorrenze dei treelet con un numero di nodi compreso tra 1 ed $i-1$. Ne segue che per poter calcolare il numero di occorrenze dei 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 circa un terzo dei conteggi precedenti. Tale ottimizzazione permette di eseguire solo le prime $\frac{2}{3}k + O(1)$ (si veda la sezione TODO per maggiori dettagli) fasi prima della fase $k$, comportando un risparmio notevole di tempo.

TODO Dì che facciamo un analisi sperimentale

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 tesi è 'e strutturata nel seguente modo. Nel capitolo 2 verrà descritta la tecnica del color coding e il suo utilizzo per il conteggio degli alberi. Si vedrà l'algoritmo di [6], la sua formulazione e le scelte adottate in fase implementativa. 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 dell'analisi sperimentale delle performance dell'algoritmo ottimizzato confrontandolo con l'implementazione della versione di [6]. Infine, nel capitolo 5, veranno discusse le possibili estensioni del presente lavoro di tesi al problema più generale della stima del numero di occorrenze dei graphlet.

# CAPITOLO 2

# COLOR CODING

In questo capitolo verrà descritta la tecnica del Color Coding utilizzata in questa tesi.

La tecnica fu introdotta nel 1995 da Alon, Yuster e Zwick [5]. In generale, dati due grafi $G$ ed $H$ , il problema di individuare un sottografo indotto di $G$ isomorfo ad $H$ è un problema $NP$-completo, ma può essere risolto in tempo polinomiale tramite un algoritmo randomizzato per classi particolari di grafi $H$, usando la tecnica del color coding.

Il primo algoritmo che Alon e i suoi colleghi descrissero in [5] , però, risolve tale problema quando $H$ è un cammino o un albero con treewidth costante limitandosi, inoltre, alla sola ricerca senza farne un conteggio del numero delle occorrenze totali.

In questo capitolo, si presenta un'estensione dell'algoritmo descritto da Alon [5, 6] che per effettuare un conteggio delle occorrenze di tutti i treelet all'interno del grafo.

## 2.1 Algoritmo

Dati in input un grafo $G = (V, E)$ ed un intero positivo $k$, per prima cosa il color coding assegna indipendentemente ed uniformemente a caso un *colore* $c_v \in [k] = \{1, ...., k\}$ ad ogni nodo $v \in V$ di $G$. Dato un sottografo $H$ di $G$, diremo che $H$ è *ben colorato* se ogni nodo $v$ di $H$ ha un colore $c_v$ distinto.

Il cuore dell'algoritmo consiste nel conteggiare il numero di occorrenze (non necessariament indotte) di alberi ben-colorati di $k$-nodi in $G$. Questo viene fatto in maniera efficiente mediante programmazione dinamica, una tecnica che risolve dei sottoproblemi del problema di interesse, procedendo dai problemi "più piccoli" verso quelli "più grandi" e ricostruendo le relative soluzioni in funzione delle soluzioni già calcolate per i problemi precedentemente risolti, fino a risolvere il problema originario.

Nel seguito deneoteremo con $T_C$ un *treelet colorato*, ovvero una coppia $(T, C)$ dove $T$ un albero radicato e $C \subseteq \{1, \ldots, k\}$. Per ogni nodo $v \in V$ e per ogni treelet colorato $T_C$, con $k$ nodi, si vuole calcolare il numero $c(T_C, v)$ di occorrenze (non indotte) di $T_C$ in $G$ che sono radicate in $v$.
Inizialmente per ogni nodo $v$ si inizializza $c(T_C, v) = 1$, dove $T$ è il treelet di 1 nodo e $C = \{c_v\}$. Dopodichè, per ogni $h = 2, \ldots, k$, si considera ogni possibile albero radicato $T$ di dimensione $h$, ogni possibile insieme di colori $C \subseteq [k]$ con $|C| = h$, ed ogni nodo $v \in V$. L'albero $T_C$ viene *decomposto* in due alberi $T'$ e $T''$ come segue: $T''$ è il sottoalbero radicato in uno dei figli della radice $v$ di $T$ e $T'$ è il sottoalbero di $T$ radicato in $v$ contenente $|T| - |T''|$ nodi. TODO Non va bene: ci sono tanti sottoalberi di $T$ che contengono $|T| - |T''|$ nodi. Quale prendi?

Ne segue che, per un'opportuna partizione dei colori $C$ in due insiemi $C$ e $C'$, ogni occorrenza di $T_C$ in $G$ implica l'esistenza due due occorrenze distinte $T'_{C'} = (T', C')$ e $T''_{C''} = (T'', C'')$ in $G$ radicate, rispettivamente nei nodi $v$ e $u$. Viceversa, ogni coppia di occorrenze $T'_{C'}$ e $T''_{C''}$ radicate, rispettivamente, in $v$ e $u$ induce un'occorrenza (non necessariamente distinta) di $T_C$ radicata in $v$. In particolare è facile ossevare che, chiamato $\beta_T$ è il numero di sottoalberi di $T$ radicati in un figlio di $r$ isomorfi a $T''$, esistono esattamente $\beta_T$ coppie di occorrenze $T'_{C'}$ e $T''_{C''}$ che inducono $T_C$.

Dalla discussione seguente segue che, per calcolare $c(T_C, v)$ è sufficiente considerare tutti treelet colorati radicati in $v$ isomorfi a $T'_{C'}$ e tutti i treelet radicati in un vicino $u$ di $v$ isomorfi a $T''_{C''}$, tali che $C', C'' \subseteq [k]$ con $C' \cup C'' = C$ e $C' \cap C' = \emptyset$. Sia per $T'$ che per $T''$ è noto il numero di occorrenze $c(T'_{C'}, v)$ e $c(T_{C''}, u)$ in $G$, poichè $|T'|, |T''| \leq h - 1$. Pertanto:

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

La correttezza e la complessità di questo algoritmo, non sono trattate in questa tesi, ma vengono dimostrate da Alon in [5].

Si può vedere l'algoritmo formalmente descritto in Algoritmo 1.

---

**input** : Grafo $G = (V, E)$, dimensione del treelet $k$ ,insieme $[k]$ di colori;
**for** $v \in V$ **do**
    Sia $c_v$ un colore scelto uniformemente a caso in $[k]$;
    Sia $T$ il treelet contenente un solo nodo;
    $T_C = (T, \{c_v\})$;
    $c(T_C, v) = 1$;
**end**
**for** $h = 2$ *to* $k$ **do**
    **for** $v \in V$ **do**
        **foreach** $T : |T| = h$ **do**
            Suddivido $T$ in due alberi $T'$ e $T''$ come descritto in precedenza.
            $c(T_C, v) = \frac{1}{\beta_T} \sum_{(u,v)\in E} \sum_{\substack{C', C''\subset C \\ C'\cup C''=C \\ C'\cap C''=\emptyset}} c(T'_{C'}, v) \cdot c(T''_{C''}, u)$
        **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], si è sfruttato un approccio bottom-up. Infatti, per ogni nodo $v \in V$ di $G$ e per ogni nodo $u \in V$ con $(u, v) \in E$ si prendono tutte le coppie possibili di treelet colorati radicati in $v$ e $u$, rispettivamente $T'_{C'}$ e $T''_{C''}$, e si verifica se esiste un albero $T_C$ radicato in $v$ per cui la coppia $T'$ e $T''$ rappresenta

una decomposizione ammissibile e tale che $C'$ e $C''$ sono una partizione di $C$. Sarà, pertanto, necessario che dati due alberi $T'$ e $T''$ la sia tale che: $|T'|, |T''| > 0$ e che $|T'| + |T''| = k$.

Dallo sviluppo dell'algoritmo 1, si noti che i conteggi del numero di occorrenze di ogni $k$-treelet colorato raggiungibile da ogni nodo $v \in V$ di $G$, sono svolti e salvati in maniera separata. Lo scopo di questa tesi però é quello di stimare, per ogni $k$-treelet $T^*$, il numero $N(T^*)$ di sottografi (non necessariamente indotti) isomorfi $T^*$ nel grafo $G$. A tal proposito si noti che ogni occorrenza $O$ di un generico $k$-treelet $T$ in $G$ è responsabile dell'incremento di 1 di esattamente $k$ contatori $c(T_C, v)$. In particolare tali contatori si riferiscono ai nodi $v$ di $O$ e agli alberi $T_C = (T, C)$ per cui $C = \{1, \ldots, k\}$ e $T$ è ottenuto radicando $T$ nel nodo corrisponte a $v$.

Sia $\mathcal{T}$ l'insieme di tutti gli alberi isomorfi a $T$ e $\widetilde{N}(T)$ il numero di occorrenze ben colorate di $T$ in $G$. Dalla discussione precedente segue che: $\widetilde{N}(T) = \frac{1}{k} \sum_{T \in \mathcal{T}} \sum_{v \in V} c(T_C, v)$ dove $T_C = (T, \{1, \ldots, k\})$.

Inoltre, poichè ogni occorrenza di un $k$-treelet $T$ in $G$ è ben colorata con probabilità $\frac{k!}{k^k}$, uno stimatore unbiased del numero di $N(T^*)$ è $\frac{k^k}{k!} \cdot \widetilde{N}(T)$, cioè $\mathbb{E}\left[\frac{k^k}{k!} \cdot \widetilde{N}(T)\right] = N(T^*)$.

per ogni $k$-treelet $T_C = (T, \{1, \ldots, k\})$, è neccessario aggregare i conteggi $c(T_C, v)$ memorizzati nei vari nodi $v \in V(G)$ in modo da ottnere il numero $N(T)$ occorrenze complessivo per ogni possibile $k$-trelet.

Per evitare di conteggiare gli alberi un numero di volte superiori a quelle effettive, si considerano le classi di equivalenza dell'insieme degli alberi di $k$ nodi, in cui gli alberi di ogni classe sono isomorfi tra loro. Per ogni classe di equivalenza viene scelto un opportuno rappresentante , $\tau(T)$, che aggrega tutti i conteggi delle occorrenze relativi alla classe che rappresenta.

Dati due alberi $T' = (V', E')$ e $T'' = (G'', E'')$ si dicono isomorfi se esiste una funzione bijettiva $f : V' \to V''$ tale che $(u, v) \in E'$ se e solo se $(f(u), f(v)) \in E'$.

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 colorati trovati in $G$ e il numero di volte che occorrono nel grafo. Poichè quello che interessa è il conteggio di tutti i $k$-treelet in un grafo,

sarà necessario tener conto che per ogni nodo colorato $v \in V$ ogni albero viene conteggiato un numero $k$ superiore di volte, una volta in più per ogni colore differente di radicamento, quindi per calcolare il numero corretto sarà sufficiente dividere il numero dei conteggi delle occorrenze per $k$. Inoltre, poichè il numero delle occorrenze di un treelet colorato in un grafo subisce un fattore di normalizzazione pari a $\frac{k!}{k^k}$. Per calcolare il numero di occorrenze di tutti i $k$-treelet colorati sarà sufficiente fare quanto segue. Per ogni albero $T$ tale che $T_C$ è l'albero colorato in $C = \{1, \ldots, k\}$, si ha che $N(T)$, il conteggio delle occorrenze di $T$ in $G$ è pari a:

$$N(T) = \frac{1}{k} \cdot \frac{k^k}{k!} \cdot \sum_{v \in V} c(T_C, v)$$

## 2.2 Dettagli implementativi. Rappresentazione compatta dei treelet colorati e rispettivi conteggi

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 58 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 quello 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 $\beta_T$ 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 descritto di seguito.
- gli ultimi 5 bit sono lasciati a zero

Per codificare la struttura dell'albero si procede a partire da una visita DFS su $T$. Nella teoria dei grafi il DFS ($depth - first - search$), o ricerca in profondità, è un algoritmo di ricerca su alberi e grafi che ha la proprietà di essere intrinsecamente ricorsivo.
In questo caso la visita avviene partendo dalla radice $r$ di $T$ e attraversando tutti gli archi. Al termine ogni arco sarà 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.

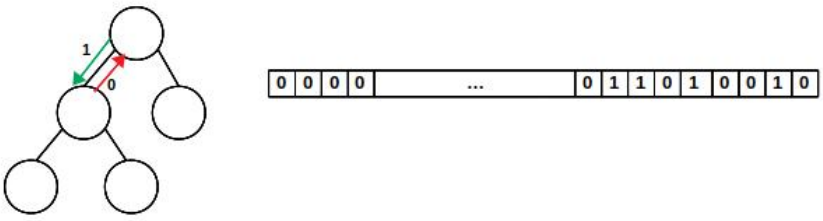


Figura 2.1: Un treelet radicato e la codifica della sua struttura

Per un qualunque $k \leq 16$ questa codifica richiede al massimo 30 bit. Come abbiamo precedentemente detto i sottoalberi radicati nei figli della radice dell'albero $T$ hanno un ordinamento non crescente, garantito anche nella struttura.
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 $T''$ e se possibile crea 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. I colori di $T$, sono dati dall'unione dei colori di $T'$ e $T''$. In $T$ come $\beta_T$ 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 $\beta_T$ associata a $T$.

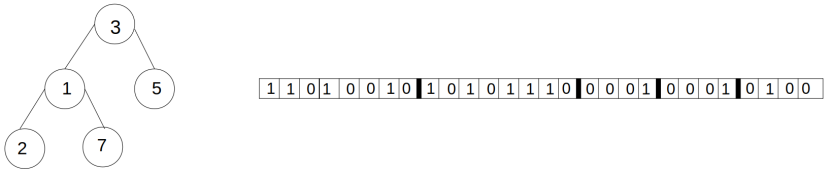


Figura 2.2: un treelet colorato e la sua codifica, mostrata per semplicità solo su 8+8+4+4+4=28 bit

In Figura 2.2 è mostrato un esempio di treelet colorato e della sua codifica

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 stati valutati 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 in un nodo particolare dell'albero. Nella sezione 3.1 verrà definito tale nodo 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 creare la tabella nel metodo venfono usati i segueenti metodi:

- **build**(): si procede alla creazione della tabella mediante l'invocazione al suo interno dei metodi principali:
  - **do_build1**() : colora ogni nodo $v \in V$ di $G$ in maniera casuale o round robin. Popola la tabella per $h = 1$, creando per ogni nodo $v$ un albero colorato con un solo nodo, esattamente $v$, e numero di occorrenze pari a 1 (paragrafo 2.1 algoritmo 1 da linea 2 a linea 5).
  - **do_build**($h$) : al variare di $h = 1, \ldots, k$ vengono popolate le rispettive tabelle, salvando per ogni nodo $v$ i treelet colorati con $h$ nodi e le rispettive occorrenze (paragrafo 2.1 algoritmo 1 da linea 6 a linea 12).

Come detto precedentemente una volta calcolati i k-treelet raggiungibili dai singoli nodi andranno aggregati, secondo equivalenze isomorfiche, per ottenere le occorrenze totali dei treelet nel grafo. Questa operazione viene garantita nel metodo

- **aggregate**(): resituisce gli alberi di dimensione $k$ radicati in ogni nodo aggregati secondo equivalenze di isomorfismo e somma tutte le rispettive occorrenze.

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 una macchina con 4 processori, vengono usati 3 thread, uno in meno rispetto al totale, poichè lavorando in Java era neccessario garantirel'utilizzo di un processore al *Garbage Collector* essendo la mole di dati molto elevata.

Per garantire un funzionamento in parallelo senza conflitti si 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 vengono modificati i precedenti dettagli implementativi ottenendo un miglioramento delle perfomance.

## 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:*

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

*con $f$ 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 indica $\delta(z)$ 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.

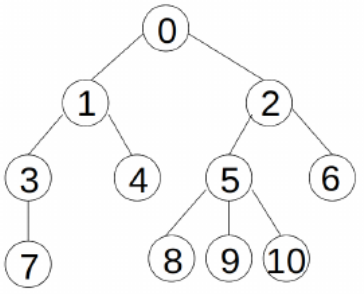


Figura 3.1: Albero $T$ per la ricerca del centroide

**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 :*

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

*Poiché $\lfloor \frac{n}{2} \rfloor = \lfloor \frac{11}{2} \rfloor = 5$, l'unico nodo per cui la disuguaglianza risulta vera è il nodo 2, infatti $5 \leq 5$.*
*Poichè il numero di nodi è dispari certamente questo sarà 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 $T$, si avrá che $f(k) = f(k-1)$. 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: $|T_i| \geq |T_{i+1}| \quad \forall i = 1, \ldots, n-1$.

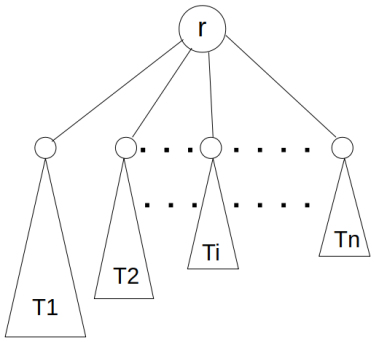


Figura 3.2: Esempio di albero $T$ radicato in $r$, con $k$ nodi, valido come input per l'algoritmo 2

**Algortimo 1:** Insiemi $k$-bilanciati

```
input : Albero T con k nodi, fattore di bilanciamento f(k − 1);
T′, T″ ⟵ due alberi inizialmente senza nodi;
for i = 1, . . . , n do
    if Σ_{j=1}^{i} |T_j| > f(k − 1) then
        T′ ⟵ sottoalbero di di T indotto da {r} ∪ ⋃_{j=1}^{i−1} V(T_j).
        T″ ⟵ sottoalbero di T indotto da V(T) ⧹ T′.
    end
end
return (T′, T″).
```

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

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

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 in precedenza era, dato un albero $T_C$, con $T$ un albero colorato radicato di $k$ nodi i cui colori giacciono in $C$, si procedeva al conteggio delle occorrenze di $T_C$ nel seguente modo

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

con $T'$ e $T''$ due alberi colorati, radicati rispettivamente in $v$ ed $u$ tali che, l'albero $T'$ avesse un numero di nodi pari ad $i$, con $i = 1, \ldots, k-1$, mentre l'albero $T''$ avesse un numero di nodi pari a $k - i$.

In questa nuova versione, invece, $T_C$ viene suddiviso in due alberi $T'$ e $T''$ tali da rispettare il principio delle decomposizioni bilanciate e più nello specifico il teorema 3.1.5.
Innanzitutto si determina se l'albero $T_C$ è radicato nel proprio centroide, nel caso in cui ciò non fosse vero, l'albero $T_c$ non viene conteggiato.
Successivamente vengono individuati due alberi $T'_{C'}$ e $T''_{C''}$ radicati entrambi nella radice $r$ di $T_C$ e colorati in modo tale che $C' \cap C'' = c_r$, con $c_r$ il colore del nodo radice.
I due treelet $T'$ e $T''$ saranno tali che $2 + \left\lfloor \frac{(k-1)}{3} \right\rfloor \leq |T'| \leq \left\lfloor \frac{2}{3}(k-1) \right\rfloor + 1$ e $|T''| = |T| - |T'|$.
Come nel caso precedente, l'algoritmo che si utilizza per il conteggio delle occorrenze dei $k$-treelet colorati in $G$ è dinamico, quindi si procede dai sottoproblemi più piccoli arrivando a quello più grande.
Anche qui per ogni nodo $v$ si inizializza $c(T_{C_0}, v) = 1$, dove T è il treelet di 1 nodo e $C_0 = \{c_v\}$. Questa volta, però, per calcolare le occorrenze dei $k$-treelet radicati in ogni $v \in V$ di $G$ non sarà necessario aver valutato tutte le occorrenze dei treelet fino a $k-1$, ma sarà necessario calcolarli fino a $\left\lfloor \frac{2}{3}(k-1) \right\rfloor + 1$. Tale conteggio verrà eseguito sulla base dell'algoritmo 1 di sezione 2.1.
Per calcolare $\forall v \in V$ di $G$ il numero $c(T_C, v)$ di occorrenze dei $k$-treelet (non indotti)

radicati in $v$ isomorfi a $T$ i cui colori giacciono nell'insieme $C$, invece, si suddivide $T$ in $(T', T'')$ in modo che tale decomposizione risulti bilanciata come descritto in precedenza e si procede al calcolo nel modo seguente

$$c(T_C, v) = \frac{1}{\gamma_T} \sum_{\substack{(T',T'') \ bilanciati \\ C' \cap C'' = c_r}} c(T'_{C'}, v) \cdot c(T''_{C''}, v)$$

con $\gamma_T$ la nuova costante di normalizzazione che è uguale a $\binom{p}{q}$, dove $p$ è il numero di sottoalberi di $T$ isomorfi al sottoalbero radicato nell'ultimo figlio della radice di $T'$ e $q$ il numero di sottoalberi radicati a partire dal primo figlio della radice di $T''$ isomorfi al sottoalbero radicato nell'ultimo figlio della radice di $T'$. Di seguito l'algoritmo formalmente descritto.

**Algortimo 2:** Fase di costruzione

```
0 input : Grafo G = (V, E), dimensione del treelet k ,insieme [k] di colori;
for h = 1 to ⌊2/3(k − 1)⌋ + 1 do
    Si applica l'algoritmo 1
end
for v ∈ V do
    foreach T : |T| = k do
        Si cerca c il centroide di T;
        if c != v then
            break;
        end
        foreach T' : 2 + ⌊(k−1)/3⌋ ≤ |T'| ≤ ⌊2/3(k − 1)⌋ + 1 do
            foreach T'' : |T''| = |T| − |T'| do
                c(T_C, v) = 1/γ_T Σ_{(T',T'') bilanciati, C'∩C''=c_r} c(T'_{C'}, v) · c(T''_{C''}, v)
            end
        end
    end
end
```

Come in algortimo 1 le occorrenze di $T$ sono calcolate con un approccio top-down, però, anche in questo caso nella tesi l'algoritmo 3 é stato sviluppato bottom-up. Perciò per calcolare le occorrenze dei $k$-treelet in $G$, $\forall v \in V$ di $G$ si prendono tutte

le possibili coppie di alberi colorati $|T'_{C'}|$ e $|T''_{C''}|$ radicate in $v$ tali che: $2 + \left\lfloor \frac{(k-1)}{3} \right\rfloor \leq |T'| \leq \left\lfloor \frac{2}{3}(k-1) \right\rfloor + 1$ e $|T''| = k - |T'|$. Prima di poterle unire a creare l'albero $T_C$ bisogna fare le opportune verifiche, ossia:

- bisogna verificare che la coppia $(T', T'')$ sia una decomposizione bilanciata per $T$. Pertanto dovrà risultare che $|T'| - 1$, si va a guardare solo i sottoalberi radicati nei figli della radice, sia $|T'| - 1 \leq \left\lfloor \frac{2}{3}(k-1) \right\rfloor$ e che aggiungendo a tale quantità la cardinalità del sottoalbero radicato nel primo figlio di $T''$, chiamata $t''$, si abbia che $|T'| + t'' - 1 \geq \left\lfloor \frac{2}{3}(k-1) \right\rfloor$.

- Anche in questo caso bisognerà garantire l'ordinamento sulla struttura dell'albero. Pertanto non sarà possibile unire $T'$ e $T''$ se la dimensione del sottoalbero radicato nell'ultimo figlio della radice di $T'$ è minore del sottoalbero radicato nel primo figlio della radice di $T''$.

- Bisogna garantire che sia rispettato il vincolo sui colori $C' \cap C'' = c_v$, ossia i due insiemi condividono esclusivamente in colore della radice $v$, che è la stessa per entrambi.

- bisogna verificare che il nodo $v$ sia il centroide dell'albero $T$ risultante dall'unione di $T'$ e $T''$.

Queste condizioni sono necessarie affinchè l'unione tra gli alberi $T'_{C'}$ e $T''_{C''}$ produca un albero valido $T_C$.

Come si nota, anche in questo caso, i conteggi inizialmente vengono fatti su ogni $v \in V$, ma poichè in questa tesi interessano le occorrenze dei diversi $k$-treelet su tutto $G$ sará necessario aggregare gli alberi radicati su ogni $v$ di $G$, unendoli a seconda della propria struttura e sommando le rispettive occorrenze.

## 3.3 Dettagli implementativi. Modifica e aggiunte alle rappresentazioni

Anche in questa nuova versione gli oggetti principali manipolati restano i treelet colorati e le occorrenze associate.

Ogni treelet colorato $T_C = (T, C)$ continua ad avere una rappresentazione unica, memorizzata usando interi da 64 bit. I bit hanno lo stesso ordinamento della precedente versione. Quello che cambia, però, è la suddivisione dei bit:

- i bit da 0-3 contengono un fattore di "uguaglianza" nominato *equal_ rooted_ tree*. Tale numero è pari ad 1 se l'albero ha un solo centroide o due centroidi tali che, gli alberi ottenuti radicando $T$ in ognuno di essi risultino diversi. Mentre è pari a 2 se l'albero ha due centroidi tali che, gli alberi ottenuti radicando $T$ in ognuno di essi risultino uguali.

- i bit da 4-7 contengono il numero di sottoalberi in $T''$ radicati a partire dal primo figlio della radice isomorfi al sottoalbero radicato nell'ultimo figlio della radice di $T'$.

- i bit da 8-11 contengono il numero $q$ necessario nel calcolo binomiale usato in fase di normalizzazione. La somma di questa quantità con il numero contenuto nei bit precedenti restituisce l'altro fattore $p$.

- i bit da 12-63 restano invariati alla versione precedente.

La struttura dell'albero è codificata esattamente come la versione precedente e resta un ordinamento non crescente sull'ordine dei sottoalberi radicati nei figli della radice dell'albero, garantendo così anche un ordinamento totale sui treelet.

Alle precedenti operazioni sugli alberi se ne aggiungono delle nuove che sono:

- **balance_merge**$(T', T'')$ : fa l'unione di due alberi $T'$ e $T''$ in maniera bilanciata. All'interno del metodo viene garantito che tutte le condizioni necessarie affinchè l'unione avvenga, descritte nel paragrafo 3.3, siano verificate.

- **normalization_factor_balanced**$(T)$ : restituisce il fattore di normalizzazione $\gamma_T$ dell'unione bilanciata.

Un'altra differenza rispetto la versione precedente riguarda la costruzione della tabelle. Infatti, seguendo quanto descritto in algoritmo 3, verranno costruite tutte le

entrate da 1 a $\left\lfloor \frac{2}{3}(k-1) \right\rfloor + 1$ come veniva fatto nella versione precedente. Mentre non verranno costruite le $h$ tabelle tali che: $\left\lfloor \frac{2}{3}(k-1) \right\rfloor + 1 < h < k$, ma si procederà direttamente alla costruzione della tabella contenente i $k$-treelet applicando il metodo **do_build_balanced**() salvando $\forall v \in V$ di $G$ i $k$-treelet bilanciati raggiungibili con le rispettive occorrenze. Anche in questa versione i treelet colorati raggiunti $\forall v \in V$ verranno aggregati e i loro conteggi sommati ed anche in questa versione vengono sfruttati i *thread* per permettere dei calcoli più veloci.

# CAPITOLO 4

# DATI

# CAPITOLO 5

# CONCLUSIONI

Si può concludere questa tesi riassumendo brevemente quanto visto fino ad ora. Innanzitutto è stato definita la fase di costruzione dell'algoritmo del Color Coding (capitolo 2).

È stato mostrato come tale algoritmo può essere sfruttato per la ricerca e il conteggio di $k$-treelet colorati presenti in un grafo $G = (V, E)$. Si è visto come, in questa tesi, sia stato sviluppato con un approccio bottom-up piuttosto che top-down.

Sono state discusse le scelte implementative sviluppate per rappresentare i treelet colorati e i conteggi associati.

Sucessivamente si è provveduto a introdurre un'ottimizzazione a quanto detto prima.

Per fare ciò è stato necessario introdurre alcune nozioni fondamentali come ad esempio la nozione di centroide e di decomposizioni bilanciate. Una volta fatto ciò si è modificato l'algoritmo del color coding, sulla base delle nozioni date, definendo così un nuovo algoritmo nel capitolo 3.

Anche per questo nuovo algoritmo si è discusso sulla scelta di valutare i $k$-treelet con un approccio bottom-up, piuttosto che top-down e si sono analizzate le scelte implementative aggiunte rispetto alla precedente versione.

Alla fine sono stati messi a confronto i risultati ottenuti con l'esecuzione di entrambe le implementazioni, discutendo il guadagno, in termini di tempo, che si è avuto usando il secondo algoritmo.

## 5.1 Sviluppi futuri

Come detto inizialmente, in questa tesi è stata sviluppata e ottimizzata solo una piccola parte di quello che è realmente l'algoritmo del color coding.
Infatti si è limitata la ricerca ai $k$-treelet (non indotti) di un grafo $G$.

Estensione naturale a tale ricerca è quella che concerne la ricerca dei graphlet, indotti in $G$.
Pertanto un possibile sviluppo sarà quello di aggiungere all'algoritmo ottimizzato una fase di campionamento per la ricerca dei graphlet indotti in $G$, allo scopo di garantire così dei tempi di esecuzione più rapidi.

# BIBLIOGRAFIA


[1] 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.

[2] 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.

[3] 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.

[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] 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.

# RINGRAZIAMENTI

Indipendente dall'ambito tutti abbiamo delle aspirazioni nella vita e, per arrivarci, dobbiamo fissare degli obiettivi da raggiungere cercando poi di superarli.

Gli obiettivi ci aiutano a capire dove vogliamo andare e che strada prendere quando ci tocca fare delle scelte. Per questo è molto importante averli sempre davanti a noi senza distogliere lo sguardo.

Io oggi ho raggiunto uno dei miei obiettivi e superato uno dei traguardi importanti della vita.
Ma per farlo non posso non ringraziare tutti quelli che hanno camminato insieme a me in questo percorso.

Voglio, innanzitutto, ringraziare il mio relatore il prof. Stefano Leucci. Nonostante la pandemia, con le tante chiamate su teams, mi ha permesso di scoprire un mondo nuovo ed affascinante, continuamente in evoluzione. Grazie.

Voglio ringraziare *Alessia* , *Claudia* e *Giada* che mi hanno supportato anche in questo viaggio. Ammetto di non essere una persona facile, cambio idea mille volte, ho sempre qualche nuova cosa assurda che mi passa per la testa, ma voi non mi avete mai giudicato, ci siete sempre state a supportarmi, sopportarmi o cazziarmi se occorreva. Grazie

Ringrazio *Roberta* ed *Alessandro*, che nonostante i mille impegni reciproci ci so-

no sempre stati, per una chiacchiera, una risata o solo per i dieci minuti di un caffè. Grazie

Ringrazio *Annaluna* e *Angela*. La prima è il mio grillo parlante, quando sto per sbagliare immagino di sentire lei che mi dice "non farlo" oppure "ci può stare" e mi comporto di conseguenza. La seconda è la follia pura, l'allegria e le mille chiacchiere fatte insieme. Grazie

Ringrazio *Alessandro*, il mio amore. Grazie a te ho capito quanto valgo e quello che posso dare. Sei la mia ancora nei momenti bui e il mio conforto nei giorni grigi. Non possiamo lavorare insieme questo è sicuro, ma possiamo continuare a supportarci reciprocamente lungo il percorso che ci si sta formando davanti. Insieme a te ringrazio anche la tua splendida famiglia, che mi ha accolto a braccia aperte dal primo giorno. Grazie

*Laura* e *Miriana*, solo una cosa vi accomuna, la mia iniziale antipatia nei vostri confronti, ma si sa le apparenze ingannano. Grazie a voi ho imparato, sicuramente, che non bisogna fermarsi alle apparenze e non giudicare mai il libro dalla copertina, perchè dietro c'è spesso molto molto di più. Grazie

Grazie a te *Erika*, la mia compagna di viaggio dal primo giorno, la mia spalla destra e altre volte sinistra. Non pensavo di trovare una grande amicizia così in questo viaggio, in realtá continuavo ad essere convinta di non averne neanche bisogno, invece mi sbagliavo. Adesso continuiamo il nostro percorso insieme e arriviamo più lontano che possiamo. Grazie.

Un ringraziamento speciale va alla mia famiglia, ai miei fratelli, *Giorgia*, *Federico* e *Salvatore*, ai loro fidanzati, *Crystal* e *Lorenzo*,ma soprattutto a loro due *Mamma e Papà*, le mie rocce. È solo grazie a voi, ai vostri sacrifici ed ai vostri insegnamenti che non mi è mai mancato niente nella vita. È solo grazie a voi e al vostro supporto quando vi ho detto che volevo riprendere a studiare che posso dire di essere qui oggi.

Spesso vi dò per scontati, ma poichè nulla è scontato in questa vita e gli ultimi mesi ne sono stati la dimostrazione, sappiate che il mio ringraziamento più grande va a voi. Ancora mille volte grazie.

L'ultimo ringraziamento, ma non per importanza va a me. Devo ringraziarmi per aver deciso di stravolgermi la vita, fare un salto nel vuoto e darmi una possibilità. Una possibilità di ottenere il futuro che desidero, la possibilità di fare quello che voglio e di diventare ciò che voglio essere. Alla Giulia del futuro dico non perdere l'ambizione che hai, continua a guardare il mondo che ti circonda con curiosità e non smettere di sognare e di volere sempre di più. Grazie a me.

Dimenticavo di ringraziare i miei Nonni, non sono riuscita a portarvi in questo giorno con me fisicamente, ma vi porto tutti nel cuore. Mi mancate ogni giorno, ma so che ci siete.

Grazie.