UNIVERSITÀ DEGLI STUDI DI MILANO

FACOLTÀ DI SCIENZE E TECNOLOGIE

CORSO DI LAUREA IN INFORMATICA

UN ALGORITMO MCMC PARALLELO PER LA COLORAZIONE DI GRAFI

RELATORE: Prof. Giuliano Grossi

CORRELATORE: Prof. Jianyi Lin

TESI DI LAUREA DI: Nicola Tuccari di San Carlo

MATRICOLA: 939944

ANNO ACCADEMICO 2020-21

Indice

Capitolo 1

Introduzione

1.1 Il problema affrontato

Il problema di colorazione del grafo è uno dei problemi più famosi e difficili da risolvere di teoria dei grafi. Esso richiede di assegnare ad ogni nodo di un grafo G un "colore", che sarà rappresentato da un numero intero. La colorazione fornita dovrà essere "propria" e il numero di colori utilizzati dovrà essere minimizzato, poiché il problema sarebbe risolvibile in maniera banale assegnando un colore differente per ogni nodo del grafo. Una colorazione è definita propria se ad ogni coppia di nodi $u, v \in V$ adiacenti, quindi che sono collegati da un arco $e \in E$, è stato assegnato un colore differente, quindi si avrà $c(u) \neq c(v)$. Il numero di colori diversi assegnati durante la colorazione è definito come numero cromatico di G ed è denotato con $\chi(G)$. Un grafo G con $\chi(G) = k$ è chiamato k-cromatico e se $\chi(G) \leq k$, definiremo G k-colorabile.

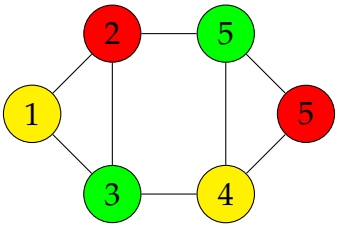


Figura 1.1: Colorazione propria di un grafo

Il problema di colorazione del grafo è molto famoso e studiato per le possibili applicazioni, le più comuni sono l'allocazione di registri, questo poiché permetterebbe di trovare il numero di registri necessari durante un certo slot temporale, schedulazione, problemi su reti sociali come l'identificazione di comunità in reti sociali dinamiche o il miglioramento del matching di amicizie nei Social Network.

Una caratteristica degli algoritmi di colorazione di grafo è il bilanciamento ottenuto, ovvero potrebbero esistere colori che sono troppo utilizzati o colori troppo poco

utilizzati, e nel caso di alcune applicazioni, come la schedulazione, per ottenere un migliore utilizzo delle risorse è necessario che la colorazione sia bilanciata.

Esistono diverse tipologie di algoritmi di colorazione come ad esempio algoritmi in tempo polinomiale, algoritmi esatti, che utilizzano o metodi di forza bruta o programmazione dinamica, questo tipo di algoritmi però nel caso di grafi con un numero elevato di nodi richiedono un tempo di esecuzione troppo elevato. Esistono anche algoritmi di colorazione greedy, questo tipo di algoritmo chiaramente non garantisce di ottenere un numero di colori sufficientemente basso. L'algoritmo che presenteremo è un algoritmo parallelo, questo tipo di algoritmi permette di sfruttare al meglio dispositivi con architetture altamente parallele, quindi con un gran numero di unità per il calcolo.

1.2 Stato dell'arte

1.2.1 I Grafi

Definizione 1. *Un grafo G è definito come una coppia $G = (V, E)$ di insiemi dove $E \subseteq [V]^2$. Gli elementi di V sono definiti come nodi (o vertici) del grafo G, mentre gli elementi all'interno di E sono i suoi archi (o lati).*

Per rappresentare graficamente un grafo è possibile disegnare dei punti, o dei cerchi, e dei segmenti per rappresentare gli archi incidenti.

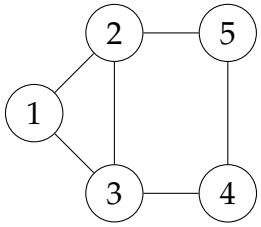


Figura 1.2: Esempio di grafo

Siano $u, v \in V$ se $\exists e = u, v$ si dice che u e v sono tra loro adiacenti e diremo che l'arco e è incidente ai vertici u e v, e che essi rappresentano gli estremi del lato e. Esistono due tipi di grafi: diretti o indiretti, un grafo è definibile diretto se gli archi presenti in E sono composti da una coppia ordinata u, v di elementi con $u, v \in V$, mentre è definibile indiretto se tale coppia non è ordinata.

Sia $G = (V, E)$ un grafo. L'insieme dei vicini di un vertice v in G è denotato come $N(v)$. Il grado $d(v)$ di un nodo v è il numero di archi adiacenti ad esso, o anche detto il numero di vicini; un vertice di grado 0 è definito *isolato*. Il numero $\delta(G) := min(d(v)|v \in V)$ è il grado minimo di G, mentre il numero $\Delta(G) := max(d(v)|v \in V)$ è il grado massimo. Se tutti i vertici di G hanno lo stesso grado k, allora G è k-regolare.

1.2.2 Cammini, Cicli e Connettività

Definizione 2. *Un cammino P di un grafo $G = (V, E)$ di lunghezza $k >= 0$ è un sottografo contenente $k + 1$ vertici distinti $v_0, ..., v_k \in V$ e k archi $e_1, ..., e_k$ tale che $e_i = (v_i - 1, v_i)$ per $i = 1, ..., k$.*

Chiameremo lunghezza del cammino P da u a v il numero di archi presenti in P. Un cammino da u a v si dice minimo se la sua lunghezza è la più piccola possibile tra i possibili cammini da u a v. La distanza tra due vertici è la lunghezza del cammino minimo tra i due.

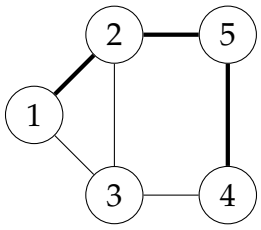


Figura 1.3: Esempio di cammino dal nodo 1 al nodo 4

Definizione 3. *Un ciclo C in G di lunghezza $k >= 3$ viene creato quando un cammino P di lunghezza $k - 1$ può essere esteso includendo l'argo $(v_k - 1, v_0) \in E$.*

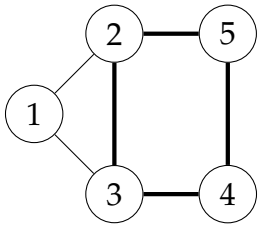


Figura 1.4: Esempio di ciclo

L'ultimo concetto da definire in questo sottoparagrafo è quello della connettività.

Definizione 4. *Sia $G = (V, E)$ un grafo, due vertici u e v si dicono connessi se esiste un cammino in G che li connetta. Se in un grafo G ogni coppia di vertici è connessa allora esso si definisce connesso.*

1.2.3 Rappresentazione dei grafi

Un grafo G può essere rappresentato in termini di matrici o di liste. Sia $G = (V, E)$ un grafo. Sia $V = v_1, v_2, ..., v_n$ l'insieme dei nodi e sia $E = e_1, ..., e_m$ l'insieme degli archi di G. Definiamo per ogni coppia di nodi $vi, vj \in V$

$$a_{ij} = \begin{cases} 1 & \text{se } v_i v_j \in E \\ 0 & \text{se } v_i v_j \in E \end{cases}$$

La matrice $A(G) = (a_{ij})$ è detta matrice di adiacenza del grafo G. A(G) sarà una matrice quadrata nXn e simmetrica.

Un altro modo per rappresentare i grafi sono le liste di adiacenza. Per ogni nodo v in V di un grafo G si definisce una lista che al suo interno conterrà gli indici dei nodi adiacenti a v.

1.2.4 Metodi Monte Carlo basati su Catena di Markov

Per poter introdurre i metodi Metodi Monte Carlo basati su Catena di Markov, o anche detti MCMC, è necessario introdurre il concetto di processo stocastico markoviano. Un processo stocastico markoviano è semplicemente un processo stocastico che soddisfa la proprietà di Markov, essa afferma che la probabilità di transizione di stato dipende solamente dallo stato immediatamente precedente e non da altri precedenti ad esso. I processi markoviani sono comunemente utilizzati nei campi del campionamento e di teoria della probabilità.

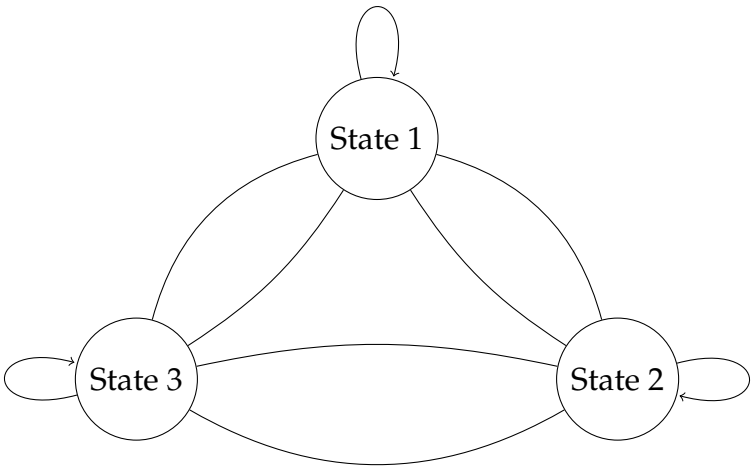


Figura 1.5: Esempio di Processo Markoviano

L'idea principale dei metodi MCMC è quello di costruire una catena di Markov nello spazio degli stati χ la cui distribuzione stazionaria è la distribuzione obiettivo $\pi^*(x)$ di interesse. Quindi viene eseguito un cammino random nello spazio degli stati in modo tale che la frazione di tempo passato in ogni stato x sia proporzionale a π^*. Scegliendo una serie di campioni casuali $x_0, x_1, x_2, \ldots$, dalla catena possiamo effettuare l'integrazione Monte Carlo con la distribuzione $\pi^*(x)$. Un importante teorema per le catene di Markov è il Teorema Ergodico, esso afferma che una catena di Markov è definibile ergodica se possiede le seguenti proprietà:

- Irriducibilità: essa garantisce che esista una sequenza di transizioni con una probabilità maggiore di 0 per il passaggio di qualsiasi coppia di stati della catena di Markov

- Aperiodicità: con questa proprietà si garantisce gli stati non sono partizionati in insiemi tali che tutte le transizioni di stato avvengono da un insieme ad un altro

1.2.5 Algoritmo Metropolis-Hastings

L'algoritmo di Metropolis-Hastings è un famoso metodo MCMC, l'idea è che ad ogni passo proponiamo di spostarci dallo stato corrente x ad un nuovo stato x' con probabilità $Q(x'|x)$, dove Q è chiamata la distribuzione di proposta, anche chiamato kernel. L'utente può utilizzare qualsiasi distribuzione voglia, ma è soggetto ad alcune condizioni che andremo ad introdurre a breve.

Avendo proposto uno spostamento allo stato x', dobbiamo decidere se accettare questa proposta, o di rifiutarla secondo una formula che garantisca che la frazione di tempo passato in ogni stato sia proporzionale a $\pi^*(x)$. Se la proposta è accettata, allora il nuovo stato è x', altrimenti il nuovo stato sarà quello corrente, x.

Se la distribuzione di proposta è simmetrica, quindi $Q(x'|x) = Q(x|x')$, la probabilità di accettazione è data dalla seguente formula

$$A = \min\{1, \frac{\pi^*(x')}{\pi^*(x)}\} \tag{1.1}$$

Quindi se x' è più probabile di x sicuramente la catena accetterà il nuovo stato, ma se fosse meno probabile, potremmo lo stesso accettarlo con una certa probabilità. Quindi invece di muoverci solo negli stati più probabili, occasionalmente permettiamo mosse con stati meno probabili. Se la distribuzione di proposta è asimmetrica, $Q(x'|x) \neq Q(x|x')$, avremo bisogno della correzione di Hastings, che è la seguente:

$$A = \min\{1, \alpha\} \tag{1.2}$$

$$\alpha = \frac{\pi^*(x')Q(x'|x)}{\pi^*(x)Q(x|x')} \tag{1.3}$$

Questa correzione è necessaria per compensare il fatto che la distribuzione di proposta potrebbe favorire certi stati.

Una ragione importante per cui Metropolis-Hastings è un algoritmo utile è che, quando si valuta α, dobbiamo conoscere solo la distribuzione obiettivo fino ad una costante di normalizzazione. Ad esempio supponiamo che $\pi^*(x) = \frac{1}{Z}\tilde{p}(x)$ dove $\tilde{x}$ è una distribuzione non normalizzata e Z è la sua costante di normalizzazione. Quindi avremo

$$\alpha = \frac{\tilde{p}(x')Q(x'|x)}{\tilde{p}(x)Q(x|x')} \tag{1.4}$$

e Z verrà cancellata. Quindi potremo campionare con la distribuzione π^* pure senza conoscere Z.

L'algoritmo di Metropolis-Hastings è un metodo MCMC che ci permette di generare campioni casuali che rispettino distribuzioni di probabilità complesse costruendo una catena di Markov che rispetta le due condizioni indicate in precedenza: irriducibilità e aperiodicità. Definiamo la distribuzione di probabilità $\pi(x)$ come distribuzione obiettivo (o target). Oltre alla distribuzione target avremo bisogno di una distribuzione di proposta $Q(x*|x_t)$, per l'algoritmo di Metropolis è necessario che essa sia una distribuzione di probabilità simmetrica per garantire la convergenza, ma Hastings propose un algoritmo che rilassa questa condizione permettendo qualsiasi tipo di distribuzione.

Di seguito è presente lo pseudocodice dell'algoritmo

Algorithm 1 Algoritmo di Metropolis-Hastings

Initialize x^0;
for $s = 0, 1, 2, \ldots$ *do*
Define $x = x^s$
Sample $x' \sim Q(x'|x)$
$\alpha = \frac{\pi^*(x')A(x'|x)}{\pi^*(x)A(x|x')}$
Compute $A = min(1, \alpha)$
Sample $u \sim U(0, 1)$
Set new sample to
$x^{s+1} = \begin{cases} x' \text{ if } u \leq A \\ x^s \text{ if } u > A \end{cases}$

1.3 Soluzioni in letteratura

Gli algoritmi di colorazione paralleli tipicamente considerano il fatto che qualsiasi insieme indipendente di nodi possono essere colorati in paralleli. Un insieme indipendente è un insieme di nodi tale che nessuna coppia di nodi condivida un arco comune. Quindi questo tipo di algoritmi si basano sul modo con cui vengono scelti gli insiemi indipendenti. La scelta degli insiemi indipendenti inoltre non tiene sempre conto dell'intero grafo G, ad esempio dopo aver scelto il primo insieme indipendente la scelta dei prossimi set indipendenti verrà fatta sul sottografo G' contenente solo i nodi che non hanno ancora un colore assegnato ad un altro insieme indipendente.

Esistono diverse strategie di colorazione, nel caso si stia cercando una colorazione ottimale, quindi volta all'utilizzo del minor numero di colori possibili, la scelta del colore sarebbe quella del più piccolo colore disponibile; se invece si cerca una colorazione bilanciata la scelta del colore potrebbe essere quella del colore meno utilizzato. Tipicamente si utilizzano delle scelte locali, ad esempio per la scelta del colore di un nodo verranno considerati solo i colori dei nodi vicini e non i vicini dei vicini.

1.3.1 Algoritmo di Luby

Un esempio di algoritmo di colorazione parallelo è stato presentato da Luby. Questo algoritmo si basa sull'algoritmo Maximal Independent Set, questo algoritmo colora il grafo cercando ripetutamente il più grande insieme indipendente del grafo, in seguito viene assegnato un colore all'insieme trovato, genera un sottografo G', ottenuto dalla rimozione dei nodi dell'insieme trovato dal grafo originale e itera l'operazione sul sottografo finché non vengono colorati tutti i nodi. Luby ha presentato un metodo

Algorithm 2 Algoritmo Maximal Indipendent Set

1: $I := U$
2: $V' := V$
3: **while** $(|V'| > 0)$ **do**
4: *Scegli un insieme indipendente I' da V'*
5: $I := I + I'$
6: $X := I' + N(I')$
7: $V' := V' - X$
8: **end while**

Monte Carlo per generare gli insiemi indipendenti in parallelo. L'algoritmo procederà ad assegnare un peso ad ogni nodo e i pesi scelti da Luby erano una permutazione casuale degli interi 1, 2 ... |V|, dove V è l'insieme dei nodi del grafo da colorare. L'insieme indipendente verrà generato in parallelo scegliendo tutti i vertici i cui pesi sono massimi locali, quindi maggiore dei pesi di tutti i vicini. Dopo aver generato l'insieme, a tutti i nodi di esso verrà assegnato un nuovo colore.

1.3.2 Algoritmo di Jones-Plassmann

Un altro esempio di algoritmo di colorazione parallelo è stato descritto da Jones e Plassmann, in questo caso non sarà necessario generare una permutazione casuale per ogni iterazione necessaria per la generazione di un insieme indipendente, ma sarà sufficiente assegnare una singola volta i pesi casuali all'inizio dell'esecuzione dell'algoritmo.

Algorithm 3 Algoritmo Jones-Plassmann

1: $U := V$
2: **while** $(|U| > 0)$ **do**
3: **for all** $v \in U$ **do in parallel**
4: $I := \{v$ tale che $w(v) > w(u) \forall u \in U\}$
5: **for all** $v' \in I$ **do in parallel**
6: $S := \{$colori di tutti i vicini di $v'\}$
7: $c(v) := \{$colore minimo non in $S\}$
8: **end for**
9: **end for**
10: $U := U - I$
11: **end while**

L'algoritmo si comporta in modo simile all'algoritmo presentato da Luby, quindi seleziona per l'insieme i nodi il cui peso è un massimo locale in parallelo. La differenza rispetto all'algoritmo è che all'insieme generato non verrà assegnato a tutti lo stesso colore, ma verrà assegnato il più piccolo colore disponibile, cioè che non è stato assegnato ai vicini.

Capitolo 2

Modello studiato

In questo capitolo descriveremo l'algoritmo parallelo MCMC per la colorazione dei grafi che vogliamo presentare e studiare.

Di seguito tratteremo le notazioni che utilizzeremo in questo capitolo, alcune sono già state introdotte nel capitolo precedente. L'algoritmo verrà applicato su un grafo indiretto $G = (V, E)$ con n nodi $n = |V|$ e un insieme $k = 1, ..., k$ di colori che verranno assegnati ad ogni nodo. Con $\Delta(G)$ rappresentiamo il grado massimo di G ed è sicuro che con almeno $\Delta(G) + 1$ colori sarà possibile ottenere una colorazione propria. Con $N(v)$ denotiamo l'insieme di nodi adiacenti al nodo v e con $c_{N(v)}$ denotiamo l'insieme dei colori assegnati all'insieme dei nodi adiacenti di v, con $\bar{c}_{N(v)}$ rappresentiamo il suo complemento. Indicheremo le cardinalità di questi due insiemi con $h_v(c) = |c_{N(v)}|$ e $\bar{h}_v(c) = |\bar{c}_{N(v)}|$. Data una colorazione c e un arco $uv \in E$ se $c_u = c_v$ allora avremo un conflitto e indicheremo con $\#(c) : [k]^n \to \mathbb{N}$ il numero di conflitti presenti nella colorazione c. Inoltre, indicheremo con le lettere minuscole c, c' e c^* per colorazioni date, mentre utilizzeremo le lettere maiuscole C, C' e C^* per indicare le colorazioni random. Quindi nel caso volessimo indicare la probabilità di ottenere $C' = c'$ data una colorazione $C = c$ lo detoneremo con P(C' = c' | C = c), per abbreviarlo scriveremo P(c' | c).

2.1 Modello

L'obiettivo dell'algoritmo è quello di definire una catena di Markov ergodica di primo ordine, la catena durante la transizione tra i vari stati visiterà una sequenza di diverse k-colorazioni, sia improprie che proprie. La distribuzione stazionaria π della catena dipenderà fortemente dai conflitti presenti nelle colorazioni ottenute nei diversi stati che visiterà durante le transizioni. Come distribuzione obiettivo per la catena di

Markov utilizzeremo la distribuzione di Gibbs:

$$\frac{e^{-\beta \#(c)}}{Z(\beta)}, \quad \text{con} \quad Z(\beta) = \sum_{c' \in k^n} e^{-\beta \#(c')} \tag{2.1}$$

Il parametro β ha un ruolo importante, cioè quello di penalizzare o promuovere le nuove colorazioni, nel caso in cui il numero di conflitti della nuova colorazione diminuisca rispetto alla colorazione precedente allora la nuova colorazione sarà accettata con probabilità molto alta, questo permetterà alla catena di convergere rapidamente verso colorazioni proprie, mentre, al contrario, nel caso in cui il numero di conflitti nella nuova colorazione fosse maggiore rispetto ai conflitti della colorazione precedente essa sarà accettata con probabilità bassa. Per la teoria MCMC sappiamo che la catena convergerà alla distribuzione proposta asintoticamente, e questo avverrà utilizzando l'algoritmo di Metropolis-Hastings, che è stato introdotto nel precedente capitolo. Verrà utilizzata una classica tecnica di "rejection sampling" per il campionamento di una nuova colorazione, questa tecnica si svolge in due fasi, nella prima fase viene generata una colorazione C' utilizzando la distribuzione di probabilità proposta $r(c, c') := P(c'|c)$, nella seconda fase C' verrà accettata o meno con probabilità

$$\alpha(c, c') = \min \left\{ \frac{\pi(c')r(c', c)}{\pi(c)r(c, c')}, 1 \right\} \tag{2.2}$$

quindi la colorazione C verrà mantenuta con probabilità $1 - \alpha(c, c')$.

La probabilità proposta r(c, c') è definita come

$$r(c, c') = \prod_{v \in V} P(c'_v|c) \tag{2.3}$$

questo significa che per ogni nodo $v \in V$ il nuovo colore c_v verrà generato in maniera indipendente rispetto agli altri nodi con una distribuzione di probabilità $P(c'_v|c)$, questo dettaglio è fondamentale poiché l'indipendenza permette di generare i colori per ogni nodo parallelamente.

È importante notare che per il calcolo di $\alpha(c, c')$ oltre a calcolare il valore di $r(c, c')$, che è definita come probabilità forward, è necessario anche calcolare quella che viene definita probabilità backward $r(c', c)$, cioè la probabilità del passaggio di stato dalla nuova colorazione alla colorazione corrente.

2.1.1 Distribuzione

In questo paragrafo descriveremo in dettaglio la distribuzione di probabilità utilizzata per la scelta del colore nell'algoritmo. La scelta del colore per ogni nodo può essere suddivisa in due casi distinti, nel primo caso consideriamo la situazione in cui col

colore c_v del nodo v, nello stato corrente della catena, è presente almeno un conflitto, quindi $c_v \in c_{N(v)}$. Poiché la scelta del nuovo colore deve portare ad ottenere con alta probabilità una colorazione senza conflitti, la scelta dovrà ricadere nei colori liberi, che abbiamo in precedenza indicato con $\bar{c}_{N(c)}$:

$$\eta_v(j,c) = \begin{cases} \frac{1-\epsilon h_v(c)}{k-h_v(c)} & \text{se } j \in \bar{c}_{N(c)} \\ \epsilon & \text{se } j \in c_{N(c)} \end{cases}$$

Dove $\epsilon < 1/k$ è un valore molto piccolo, maggiore di zero, che rappresenta la probabilità di scegliere un colore non libero, e questo è necessario per ampliare lo spazio di ricerca. La distribuzione è definita in modo tale da fornire la stessa probabilità ad ogni colore libero di essere scelto, in modo tale da permettere un buon bilanciamento, che è uno degli obiettivi della colorazione da ottenere. Il secondo caso da tenere in considerazione è quando non sono presenti conflitti per il colore c_v del nodo v, è naturale che in questo caso si voglia mantenere il colore attuale con alta probabilità in modo tale da garantire la convergenza dell'algoritmo, quindi in questo caso definiremo:

$$\zeta_v(j,c) = \begin{cases} 1-\epsilon(n-1) & \text{se } j = c_v \\ \epsilon & \text{se } j \neq c_v \end{cases}$$

dove anche qui avremo epsilon, che è un valore molto piccolo maggiore di zero, che rappresenta la probabilità di scegliere un altro colore diverso da c_v per ampliare lo spazio di ricerca.

Quindi abbiamo presentato i valori necessari per determinare la probabilità r(c, c') e la distribuzione di probabilità proposta per ogni nodo v è:

$$P(c'_v,c) = \begin{cases} \eta_v(c'_v,c) & \text{se } h_v(c) < k \text{ e } c_v \in \bar{c}_{N(c)} \\ \zeta_v(c'_v,c) & \text{altrimenti} \end{cases}$$

2.2 Analisi dell'algoritmo

2.2.1 Analisi di convergenza

L'analisi dell'algoritmo verrà fatta studiando una variante dell'algoritmo dove avremo che $\epsilon = 0$ e una probabilità di accettazione della nuova colorazione pari a $\alpha(C, C') = 1$, in questo modo verrà accettata qualsiasi colorazione ed essendo l'algoritmo randomico non è possibile garantire la riduzione del numero di conflitti ad ogni iterazione.

Lemma 1. *Data la colorazione corrente C, definiamo #W e #W* come il numero di nodi aventi almeno un conflitto rispettivamente nella colorazione corrente e nella colorazione generata*

nell'iterazione successiva dell'algoritmo. Dato il numero di colori $k \geq \Delta(G)$, per ogni intero $r > 1$ varrà la seguente disuguaglianza:

$$\mathbb{E}[\#W^* \mid C] \leq \rho \#W$$

per alcuni $0 < \rho = 1 - (1 - 1/r)^{\Delta(G)} < 1$.

Per la dimostrazione introduciamo l'espressione $\mathbb{1}(cond)$ che equivale ad 1 se la condizione cond è vera, altrimenti 0. Quindi il numero di conflitti in una colorazione C^* è definito come $\#C^* = \sum_{uv \in E} \mathbb{1}(C_u^* = C_v^*)$, mentre il numero di conflitti locali di un nodo v è $\#_v C^* = \sum_{u:uv \in E} \mathbb{1}(C_u^* = C_v^*)$, è possibile definire similarmente anche i conflitti per la colorazione C. Data la colorazione corrente C, partizioneremo i nodi di V in due insiemi, un insieme W dei nodi con almeno un conflitto e $\bar{W} = V \setminus W$, che è l'insieme dei nodi senza conflitti. Considerando il numero di conflitti locali di un dato nodo $v \in W$, cioè $\sum_{uv \in W} C_u^* = C_v^*$, sappiamo che per un dato $u \in \bar{W}$ avremo che $\sum_{u \in \bar{W}: uv \in E} C_u^* = C_v^* = 0$, poichè se u non si trova nell'insieme W manterrà il suo vecchio colore $C_u^* = C_u \neq C_v^*$. Quindi nella la sommatoria $\#_v C^*$ ogni termine $\mathbb{1}(C_u^* = C_v^*)$ è una variabile Bernoulliana con parametro

$$p_{uv} := \frac{|\bar{C}_{\mathcal{N}(u)} \cap \bar{C}_{\mathcal{N}(v)}|}{|\bar{C}_{\mathcal{N}(u)}| \cdot |\bar{C}_{\mathcal{N}(v)}|} \leq \frac{\min\left\{|\bar{C}_{\mathcal{N}(u)}|, |\bar{C}_{\mathcal{N}(v)}|\right\}}{|\bar{C}_{\mathcal{N}(u)}| \cdot |\bar{C}_{\mathcal{N}(v)}|} \leq \frac{1}{r},$$

condizionato dalla colorazione C, questo è possibile scriverlo anche nel seguente modo

$$\mathbb{1}(C_u^* = C_v^*) \mid C \sim \text{Bernoulli}(p_{uv}).$$

Poiché ogni colorazione di C_u^* per ogni vicino u di v è eseguita autonomamente dagli altri vicini, questi termini sono stocasticamente indipendenti. La variabile $\#_v C^*$, essendo la somma di variabili Bernoulliane indipendenti, è definibile come una variabile casuale binomiale di Poisson:

$$\#_v C^* \mid C \sim \text{PoissonBinomial}(\{p_{uv} : uv \in E, u \in W\})$$

Definiamo la variabile Bernoulliana $B_v^* := \mathbb{1}(\#_v C^* > 0)$ verificando se v abbia conflitti nella nuova colorazione e la distribuzione condizionata da C sarà definita come

$$(B_v^* \mid C) = (\mathbb{1}(\#_v C^* > 0) \mid C) \sim \text{Bernoulli}(1 - \Pi_{u \in W: uv \in E}(1 - p_{uv}))$$

se $v \in W$, poichè $\Pi_{u \in W: uv \in E}(1 - p_{uv})$ è la probabilità che v non abbia alcun conflitto coi suoi vicini nella nuova colorazione C^*, mentre chiaramente avremo $B_v^* = 0$ se

$v \notin W$. Il numero di nodi aventi almeno un conflitto nella nuova colorazione $\#W^* = \#\{u \in V : \#_u C^* > 0\} = \sum_{v \in W} B_v^*$ soddisferà la seguente relazione

$$\begin{aligned}
\mathbb{E}[\#W^* \mid C] &= \mathbb{E}\left[\sum_{v \in W} B_v^* \mid C\right] = \sum_{v \in W} \mathbb{E}[B_v^* \mid C] \\
&= \sum_{v \in W} \left[1 - \prod_{u \in W : uv \in E} (1 - p_{uv})\right] \\
&\leq \sum_{v \in W} \left[1 - \prod_{u \in W : uv \in E} \left(1 - \frac{1}{r}\right)\right] \\
&\leq \sum_{v \in W} \left[1 - \left(1 - \frac{1}{r}\right)^{\deg v}\right] \leq \left[1 - \left(1 - \frac{1}{r}\right)^{\Delta(G)}\right] \sum_{v \in W} B_v \\
&= \left[1 - \left(1 - \frac{1}{r}\right)^{\Delta(G)}\right] \#W = \rho \#W
\end{aligned}$$

dove $0 < \rho := 1 - (1 - 1/r)^{\Delta(G)} < 1$. E quindi in questo modo è stato dimostrato il lemma, poichè $\mathbb{E}[\#W^* \mid C] \leq \rho \#W$.

Theorem 1. *Il numero di conflitti #C converge con probabilità a 0,* $\lim_{t \to \infty} \mathbb{P}(\#(C^t) < \delta) = 1 \; \forall \delta > 0$.

Per la dimostrazione utilizzeremo il risultato ottenuto dal Lemma 1. Utilizzando la proprietà di monotonia del valore atteso avremo

$$\mathbb{E}[\mathbb{E}[\#W^* \mid C]] \leq \rho \mathbb{E}[\#W].$$

Per la legge delle aspettative iterate($\mathbb{E}[\mathbb{E}[\#W^* \mid C]] = \mathbb{E}[\#W^*]$) possiamo definire la disuguaglianza

$$\mathbb{E}[\#W^*] \leq \rho \mathbb{E}[\#W].$$

Considerata la sequenza di colorazioni successive C^t, $t = 0, 1, 2, ...$ generate dall'algoritmo, è possibile garantire $\mathbb{E}[\#W^t] \leq \rho^t \mathbb{E}[\#W^0]$ per ogni t. Applicando il limite $\lim_{t \to \infty}$ ad entrambi i membri della disequazione, otterremo la convergenza a 0 del valore atteso del numero di nodi con almeno un conflitto:

$$\lim_{t \to \infty} \mathbb{E}[\#W^t] = \lim_{t \to \infty} \rho^t \mathbb{E}[\#W^0] = 0.$$

È ben risaputo che la convergenza in aspettativa implica anche la convergenza in probabilità quindi

$$\lim_{t \to \infty} \mathbb{P}(\#W^t < \delta) = 1 \qquad \text{per ogni } \delta > 0.$$

Quindi poichè il numero di conflitti $\#C^t$ è limitato superiormente $\frac{1}{2}\Delta(G)\#W^t$, esso converge in probabilità:

$$\lim_{t\to\infty} \mathbb{P}(\#C^t < \delta) = 1 \qquad \text{for any } \delta > 0.$$

2.2.2 Analisi qualitativa del bilanciamento

In questa sezione proponiamo una possibile analisi qualitativa della colorazione, nonostante essa sia difficile da provare rigorosamente poiché la distribuzione presentata in questo modello evolve durante le svariate transizioni tra gli stati della catena prima di raggiungere la colorazione propria. Proveremo ad analizzare i motivi per cui a prescindere dalla topologia del grafo proposto l'algoritmo riesca a produrre colorazioni quasi uniformi.

Per l'analisi utilizzeremo la Distribuzione Multinomiale di Poisson, che indicheremo con PMD, essa è la distribuzione della somma di n variabili casuali e indipendenti, dove l'insieme $\mathcal{B}_k = \{e_1, \ldots, e_k\}$ è la base canonica per $\mathbb{R}^k$, che rappresenta i k colori forniti all'algoritmo.

Ad ogni passo t dell'algoritmo, ogni nodo $v \in V$ sceglie casualmente un nuovo colore utilizzando la distribuzione $P(c_v, c)$, definita nelle precedenti sezioni. Definiamo l'insieme di colori disponibili $F_v^{(t)} \subseteq \mathcal{B}_k$ nel passo t per il nodo v, il cui colore è definito come $C_v^{(t)} \in \mathcal{B}_k$, e definiamo la distribuzione del nuovo colore di v in $\mathcal{B}_k$ come $p_v^{(t)} = \left(p_{v,1}^{(t)}, \ldots, p_{v,k}^{(t)}\right)$. A causa della distribuzione $P(c'_v, c)$ avremo che la distribuzione per il nuovo colore del nodo v nel caso di presenza di conflitti sarà

$$p_{v,j}^{(t)} \approx \begin{cases} 1/|F_v^{(t)}|, & \text{se } e_j \in F_v^{(t)} \\ 0, & \text{se } e_j \notin F_v^{(t)} \end{cases}, \tag{2.4}$$

mentre nel caso di assenza di conflitti

$$p_{v,j}^{(t)} \approx \begin{cases} 1, & \text{se } e_j = C_v^{(t)} \\ 0, & \text{altrimenti} \end{cases}. \tag{2.5}$$

Questo significa che il processo di assegnamento del nuovo colore per il nodo v nella prossima iterazione t+1 $C_v^{(t+1)} \in \mathcal{B}_k$ può essere espressa da una distribuzione multinoulli con parametro $p_v^{(t)}$, quindi,

$$C_v^{(t+1)} \sim \text{Multinoulli}(p_v^{(t)}).$$

Sia $X^{(t)} = \sum_{v\in V} C_v^{(t)}$ il numero di occorrenze per ogni colore della colorazione del grafo al tempo t. Per ogni $j \in [k]$, $X_j^{(t)}$ rappresenta la frequenza del colore e_j nella

colorazione $C^{(t)}$. Quindi il vettore casuale $X^{(t)}$ è la somma delle variabili multinoulli indipendenti, non identicamente distribuite che seguono una (n,k)-PMD, dove n è il numero di nodi e k il numero di colori. Definiamo la funzione di massa di probabilità di $X^{(t)}$ $\xi(x) = \mathbb{P}(X^{(t)} = x)$, dove x è un vettore di dimensione k formato da elementi interi non negativi, la cui sommatoria al più sarà n. Per ottenere un buon bilanciamento sarà necessario assegnare un'alta probabilità $\xi(x)$ ai vettori x con valori equalizzati.

Analizziamo adesso i diversi passi dell'algoritmo. Durante il primo passo possiamo assumere che verrà assegnato un colore randomico ad ogni nodo con probabilità uniforme. Di conseguenza l'insieme di colori disponibili per v $F_v^{(1)}$ si presenterà con probabilità uniforme tra tutti i sottoinsiemi di dimensione $\left|F_v^{(1)}\right|$. Consideriamo un colore e_j, le probabilità che esso venga scelto dopo n round indipendenti in un nodo v dobbiamo tenere conto degli insiemi di probabilità $P_j = \left\{p_{v,j}^{(1)} : v \in V\right\}$. Per n molto grande e considerando l'ipotesi di indipendenza, la probabilità che per ogni nodo v venga scelto un dato colore e_j $p_{v,j}^{(1)}$ è mediamente la stessa a causa dell'uniformità di $F_v^{(1)}$. Quindi mediamente ogni ogni nodo la probabilità $\bar{P}_j$ dovrebbe essere mediamente $1/k$ per un grafo di grande dimensione n.

Supponendo che al tempo $t-1$, la colorazione $C^{(t-1)}$ sia abbastanza bilanciata per quanto riguarda le frequenze dei colori possiamo concludere che anche la colorazione $C^{(t)}$ dovrebbe ottenere un buon bilanciamento. Questo dovrebbe accadere principalmente per due motivi:

- Alcuni dei colori dovrebbero essere già stati fissati dato che sono stati scelti randomicamente nei passi precedenti, questo dovrebbe contribuire alla distribuzione uniforme dei colori per i nodi nel vicinato che devono fare una nuova scelta

- Tutti i nodi che devono selezionare un nuovo colore a causa di conflitti si troveranno nella condizione descritta in precedenza quindi dovrebbero ripetere la generazione indipendente del colore.

Capitolo 3

Analisi dei Parametri

Il modello analizzato utilizza diversi parametri:

- Il grafo stesso che a sua volta contiene il parametro del numero di nodi e di archi, nel caso in cui il grafo fosse generato randomicamente è necessario considerare la probabilità p con cui viene assegnato un arco
- Il numero di colori k
- Il valore ϵ che abbiamo introdotto nel precedente capitolo, definita nella distribuzione proposta
- Il valore β, definita nella distribuzione di Gibbs, che è la distribuzione obiettivo del modello

3.1 Grafo e il numero di colori k

Il grafo da colorare presenta dei parametri che saranno fondamentali nella scelta degli altri parametri che andremo a discutere e che influenzeranno i risultati ottenuti. In particolare il numero di archi e il grado medio sono dei parametri importanti sia per quanto riguarda i tempi di esecuzione sia per l'eventuale scelta del valore di numero di colori k.

Per la scelta di k sappiamo che scegliendo $k = \Delta(G) + 1$ allora sicuramente sarà possibile ottenere una colorazione propria. Ovviamente la scelta di questo valore può essere valida in grafi molto densi, mentre invece invece potrebbe essere possibile utilizzare un valore di k più piccolo trovando facilmente una colorazione propria, ad esempio un valore di k valido in un grafo di dimensione n generato randomicamente con una densità p potrebbe essere $k = \lceil np \rceil$.

3.2 I parametri ϵ e β

La scelta del parametro ϵ è dipendente sia dal numero di nodi presenti nel grafo da colorare, sia dal numero di colori scelto per l'esecuzione. La dipendenza dal numero di colori scelto è chiara poichè se $\epsilon < 1/K$ non fosse sufficientemente piccolo durante il calcolo della probabilità $\eta_v(j, c)$ potrebbero esserci casi in cui $\epsilon h_v(c) \approx 1$, elevando di molto la probabilità di scegliere colori già occupati, nonostante vi possano essere disponibili colori liberi.

Un esempio molto semplice è nel caso di $\epsilon = 0.001$ e un numero di colori $k = 1000$, nel caso in cui ci fosse solo un colore libero e 999 occupati avremo $\epsilon h_v(c) = 0.999$ e quindi il colore libero sarà scelto con probabilità pari ad 0.1% mentre la probabilità di scelta di un colore occupato pari a 99.9%, quindi è necessario che il valore di epsilon scelto sia di diversi ordini di grandezza più basso.

La scelta del parametro β concorre a definire il valore del fattore $e^{-\beta(\#c'-\#c)}$ che è legato ai valori del rapporto tra le probabilità backward e forward $\frac{r(c',c)}{r(c,c')}$, fattori che compaiono entrambi nel ratio di Hastings (2.2). Quindi la scelta del parametro β è critico e di seguito analizziamo una possibile scelta che si dimostra molto efficace a livello sperimentale. In base alla definizione delle probabilità proposte, viste come produttoria di probabilità sui singoli nodi, le probabilità backward e forward possono anche essere rimaneggiate come segue:

$$r(c', c) = \prod_{v \in V} \mathbb{P}(c_v \mid c') = \epsilon^m \rho^r \prod_{w \in W} \frac{1 - \epsilon h_w}{k - h_w}$$

e

$$r(c, c') = \prod_{v \in V} \mathbb{P}(c'_v \mid c) = \epsilon^{m'} \rho^{r'} \prod_{u \in U} \frac{1 - \epsilon h_u}{k - h_u}$$

dove W e U sono oppotuni sottoinsimei di nodi. Questo perchè durante il calcolo delle probabilità forward e backward ϵ compare con molteplicità m o m', rispettivamente per la distribuzione forward e backward. Allo stesso modo il fattore $\rho = 1 - \epsilon(n-1)$ avrà molteplicità r o r', rispettivamente. Poiché ρ è molto vicino ad 1 per ϵ è molto piccolo, possiamo trascuralo introducento un'approssimazione che preserva il risultato.

Con questa riscrittura e tenendo conto delle approssimazioni introdotte possiamo riscrivere la probabilità di accettazione del criterio di Hastings come segue:

$$A = \frac{\pi(c')r(c', c)}{\pi(c)r(c, c')} = e^{-\beta\Delta_\#} \epsilon^{\Delta_m} \rho^{\Delta_r} \frac{B}{C} \approx e^{-\beta\Delta_\# + \Delta_m \log \epsilon} \frac{B}{C}$$

con $\Delta_\# = \#c' - \#c$, $\Delta_m = m - m'$, $\Delta_r = r - r'$, $B = \prod_{w \in W} \frac{1-\epsilon h_w}{k-h_w}$ e $C = \prod_{u \in U} \frac{1-\epsilon h_u}{k-h_u}$.

Poiché nei grafi ER in particolare la differenza di conflitti $\Delta_\#$ ad ogni passaggio di

colorazione e la differenza del numero di volte che viene scelto un colore non libero Δ_m è dello stesso ordine di grandezza, trova un senso la posizione $\beta = -\log \epsilon$ che dà luogo a una formula come quella che segue

$$A = e^{\log \epsilon(\Delta_m + \Delta_\#)} \frac{B}{C} = \epsilon^{\Delta_m + \Delta_\#} \frac{B}{C}$$

e applicando il logaritmo ad entrambe le parti

$$\log A = (\Delta_m + \Delta_\#) \log \epsilon + \log B - \log C$$

E poiché l'algoritmo preserva l'ordine il criterio definito da Hastings diventa

$$\log \alpha = \min\{\log A, 0\}$$

Quindi nel caso in cui $(\Delta_m + \Delta_\#) \log \epsilon \geq \log B - \log C$ il valore scelto per $\log \alpha$ sarà pari a 0, quindi il caso in cui la colorazione viene accettata sempre.

Scegliendo $\beta = -\log \epsilon$ il sistema non dovrà più dipendere da un secondo parametro, e la scelta di ϵ dipende principalmente dai valori del numero di colori k e dal grafo, vediamo di seguito alcuni risultati.

Nella prossima sezione andremo a riportare alcuni dati ottenuti durante le iterazioni dell'algoritmo, oltre ai dati già citati in precedenza avremo

$$\log Gibbs = \log \frac{\pi(c')}{\pi(c)} = \log e^{-\beta_{\Delta\#}} = -\beta\Delta_\#$$

Nella tabella 3.1 riportiamo le iterazioni dell'algoritmo MCMC per un grafo di 50k nodi e densità 1%, con $\epsilon = 10^{-7}$. Nei casi in cui viene accettata la nuova colorazione il valore di $\log A$ è particolarmente grande: questo perché il valore $\log Gibbs$ è positivo poiché il valore $-\beta\Delta_\#$ è positivo in quanto moltiplicazione di due numeri negativi, quindi la scelta $\beta = -\log \epsilon$ si è rivelata adatta. Avendo qui scelto $\epsilon = 10^{-7}$ il sistema è rimasto bloccato su una colorazione per diverse iterazioni, questo lo si intuisce dal fatto che il valore $\log A$ è negativo e la probabilità che venga accettato è molto bassa, dovuto probabilmente al fatto che il valore di ϵ non è sufficientemente piccolo e quindi con una probabilità troppo alta sono stati scelti colori non liberi.

Tabella 3.1: Risultati dei valori studiati su grafo di dimensione 50k e densità 1% con $\epsilon = 10^{-7}$

	Iter. 1	Iter. 2	Iter. 3	Iter. 4	Iter. 5	Iter. 6
$\log A$	368321.9	146909.6	27210.9	1207.6	-26.6	-261.8
$\log \gamma$	-184947.8	-100028.8	-22400.5	-1154.8	-26.6	-54.5
$\log Gibbs$	553269.7	246938.4	49611.5	2362.4	0	-207.2
Δr	-673	-176	-11	-1	-1	-6
Δm	17199	9395	2119	109	2	2
$\#(c^*)$	14426	2510	116	2	2	2
$\#(c)$	41124	14426	2510	116	2	2
$\Delta_\#$	-26698	-11916	-2444	-114	0	0

Nella tabella 3.2 riportiamo le iterazioni dell'algoritmo MCMC per un grafo di 50k nodi e densità 1%, con $\epsilon = 10^{-8}$. In questo caso possiamo notare che diversi valori come Δr, Δm e $\Delta_\#$ non siano cambiati particolarmente, mentre sia $\log \gamma$ che $\log Gibbs$ hanno valori inferiori rispetto al caso precedente. È diminuito il numero di iterazioni per ottenere una colorazione valida, in questo caso 4.

Tabella 3.2: Risultati dei valori studiati su grafo di dimensione 50k e densità 1% con $\epsilon = 10^{-8}$

	Iter. 1	Iter. 2	Iter. 3
$\log A$	262025	99809.8	18294.1
$\log \gamma$	-223212.6	-100028.8	-120759.4
$\log Gibbs$	485237.6	220569.2	48188.5
Δr	-652	-208	-12
Δm	17105	9311	2322
$\#(c^*)$	14698	2724	108
$\#(c)$	41040	14698	272
$\Delta_\#$	-26342	-11974	-164

Provando con un grafo più grande con 100k elementi con densità 1% le iterazioni necessarie per terminare con $\epsilon = 10^{-7}$ sono state 64, per lo stesso motivo notato in precedenza nel caso di grafo con 50k nodi.

Nelle tabelle 3.3 e 3.4possiamo notare che con $\epsilon = 10^{-8}$ e $\epsilon = 10^{-9}$ il numero di iterazioni è lo stesso e i vari valori si comportano come ci aspettiamo. Il numero di iterazione è basso in entrambi i casi, solo 5, possiamo quindi notare che anche in questo caso la scelta di β funziona ed il valore di ϵ è adatto.

Tabella 3.3: Risultati dei valori studiati su grafo di dimensione 100k e densità 1% con $\epsilon = 10^{-8}$

	Iter. 1	Iter. 2	Iter. 3	Iter. 4
$\log A$	594960.4	238991.6	43560.6	1666
$\log \gamma$	-421161.1	-245324.9	-62468.8	-3123.4
$\log Gibbs$	1016121.6	484316.5	106029.4	4789.4
Δr	-903	-271	-16	-4
Δm	33950	19893	5092	253
$\#(c^*)$	32316	6024	268	8
$\#(c)$	87478	32316	6024	268
$\Delta_\#$	-55162	-26292	-5756	-260

Tabella 3.4: Risultati dei valori studiati su grafo di dimensione 100k e densità 1% con $\epsilon = 10^{-9}$

	Iter. 1	Iter. 2	Iter. 3	Iter. 4
$\log A$	644298.5	279715.5	5829.3	4272.6
$\log \gamma$	-560800.8	-377916	-133989.9	-15414.54
$\log Gibbs$	1205099	657632.1	192519.1	19687
Δr	-308	-136	-13	0
Δm	32716	22040	7818	899
$\#(c^*)$	41998	10264	974	24
$\#(c)$	100150	41998	10264	974
$\Delta_\#$	-58152	-31734	-9290	-950

Adesso mostriamo i risultati su grafi più grandi, con 500k nodi. Possiamo vedere i valori ottenuti durante le iterazioni con $\epsilon = 10^{-8}$ e $\epsilon = 10^{-9}$, rispettivamente in 3.5 e in 3.6 , possiamo notare che nel primo caso il numero di iterazioni per terminare è superiore. I valori ottenuti per ogni iterazione sono in linea con quelli che abbiamo notato in precedenza anche con questo tipo di grafi di grande dimensione.

Tabella 3.5: Risultati dei valori studiati su grafo di dimensione 500k e densità 0.1% con $\epsilon = 10^{-8}$

	Iter. 1	Iter. 2	Iter. 3	Iter. 4	Iter. 6	Iter. 7	Iter. 8
$\log A$	3283683	1435738	303838.1	18342.7	51.4	-49.9	-7.9
$\log \gamma$	-2201812	-1459440	-480624.9	-40640.3	-353.9	-49.9	-44.8
$\log Gibbs$	5485495	2895178	784463.1	58983	405.2	0	36.8
Δr	-427	-166	-3	-1	-2	-2	-1
Δm	166589	110434	36384	3076	26	3	3
$\#(c^*)$	202984	45814	3228	26	4	4	2
$\#(c)$	500774	202984	45814	3228	26	4	4
$\Delta_\#$	-297760	-157170	-42586	-3202	-22	0	-2

Tabella 3.6: Risultati dei valori studiati su grafo di dimensione 500k e densità 0.1% con $\epsilon = 10^{-9}$

	Iter. 1	Iter. 2	Iter. 3	Iter. 4
$\log A$	3564082.2	1538040	318147.7	18377.56
$\log \gamma$	-2585464	-1710415	-563751.6	-46942.2
$\log Gibbs$	6149546	3248455	881899.3	65319.7
Δr	-419	-176	-20	0
Δm	166594	110219	36337	3026
$\#(c^*)$	202480	45726	3170	18
$\#(c)$	292746	202480	45726	3170
$\Delta_\#$	-296746	-156754	-42556	-3152

Capitolo 4

Risultati Sperimentali

In questo capitolo riportiamo i risultati sperimentali ottenuti sia su grafi generati randomicamente, utilizzando il modello di Erdős–Rényi, sia su alcuni esempi di reti sociali. L'obiettivo di questi test è quello di valutare la qualità del bilanciamento ottenuto dall'algoritmo presentato e i suoi tempi di esecuzione. Per valutare la qualità del bilanciamento è necessario introdurre una misura che rappresenta la deviazione della colorazione $c \in k^n$ da una colorazione perfettamente bilanciata, dove ogni classe di elementi avrà una dimensione $n_j = n/k$, per ogni $j \in k$. L'indice per misurare la qualità del bilanciamento sarà chiamato indice di sbilanciamento ed è definito come

$$\Gamma_{n,k}(c) = (\frac{1}{k}\sum |n_j - \frac{n}{k}|^2)^{\frac{1}{2}} \tag{4.1}$$

Nel caso di un bilanciamento perfetto avremo $\Gamma_{n,k}(c) = 0$.

4.1 Risultati su Grafi ER

Per ottenere una visione più ampia l'algoritmo è stato eseguito su diversi insiemi di grafi generati randomicamente, su grafi contenenti da 25k fino a 500k vertici e con diverse densità. Per il numero di colori scelti sono state fatte le prove su due diversi casi.

Il primo caso è il numero di colori k definito come $k = \lceil np \rceil$, questo valore sarebbe il grado previsto per ogni vertice e in questo caso l'indice di sbilanciamento avrebbe un valore pari a

$$\Gamma_{n,k}(c) = (\frac{1}{np}\sum |n_j - \frac{n}{k}|^2)^{\frac{1}{2}}. \tag{4.2}$$

Il secondo valore di k scelto invece è quello pari al valore del grado massimo, maggiore quindi di $\lceil np \rceil$.

L'esecuzione è stata effettuata in due modalità, nella prima è stato eseguito l'algoritmo in maniera classica, definendo il valore di β e calcolando la probabilità di accettazione definito dall'algoritmo di Metropolis-Hastings, nel secondo caso questa il criterio Metropolis-Hastings viene ignorato e si decide di accettare ogni colorazione generata.

I risultati sui grafi ER sono stati eseguiti su 3 diverse densità di grafo: 0.1%, 0.5% e 1%. I risultati ottenuti sono riportati nei prossimi sottoparagrafi.

4.1.1 Grafi ER con densità 0.1%

Nella figura 4.1 sono stati riportati i tempi di esecuzione negli esperimenti effettuati con $k = \lceil np \rceil$, per ciascuna dimensione del grafo il test è stato ripetuto almeno 10 volte. Possiamo notare che ignorando il calcolo delle probabilità e, di conseguenza, accettando ogni colorazione generata mediamente i tempi di esecuzione diminuiscono del 45.8%. Questa diminuzione è dipendente dalla dimensione del grafo, lo si può notare in particolar modo confrontando la diminuzione in percentuale dei tempi di esecuzione sui grafi di dimensione 50k, 68.89%, e nei grafi di dimensione 500k, 33.43%, in generale nei casi di grafi di dimensioni pari o superiori a 200mila la diminuzione dei tempi di esecuzione è compresa tra il 30% e il 35%.

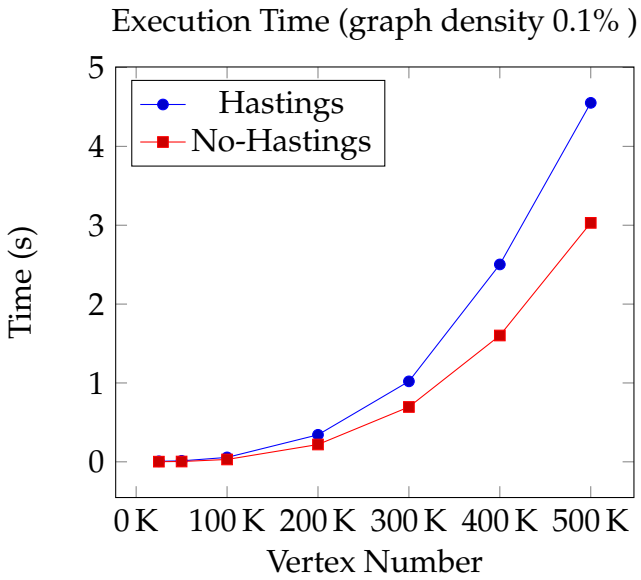


Figura 4.1: Tempi di esecuzione su grafi generati randomicamente con $k = \lceil np \rceil$, Metropolis-Hastings(linea blu), No-Hastings(linea rossa)

Nella figura 4.2 sono stati riportati i valori dell'indice di sbilanciamento su grafi ER con una densità p pari a 0.1% e $k = \lceil np \rceil$. Possiamo notare che il valore dell'indice di sbilanciamento nelle diverse dimensioni del grafo assume un valore medio di 21.45, con un valore di deviazione standard di 0.76.

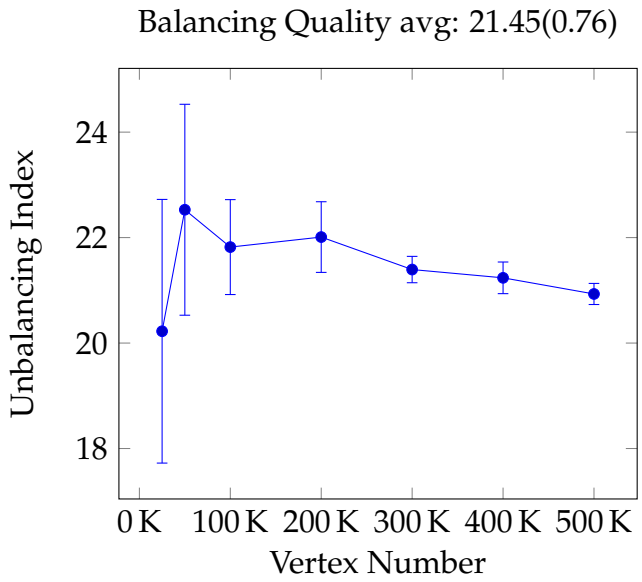


Figura 4.2: Indice di sbilanciamento su grafi generati randomicamente con $k = \lceil np \rceil$

Dopo aver mostrato i seguenti risultati possiamo analizzare i risultati ottenuti nel caso in cui k sia pari al grado massimo del grafo e confrontarlo con i risultati ottenuti per k = np. I risultati mostrati in figura 4.3 ci mostrano che i tempi di esecuzione sono leggermente inferiori nel caso in cui k è pari al grado massimo, questo è probabilmente dovuto al fatto che il numero di round medi è più basso nel caso $k = \Delta(G) + 1$, pari a 5.06, mentre nel caso $k = \lceil np \rceil$ il numero medio di round è pari a 5.76. I tempi di esecuzione diminuiscono in media tra le varie dimensioni dei grafi del 8.7%, con un picco di 26.1% di diminuzione nei grafi di dimensione 25k e un minimo di 1.4% nel caso di grafo di dimensione 300k.

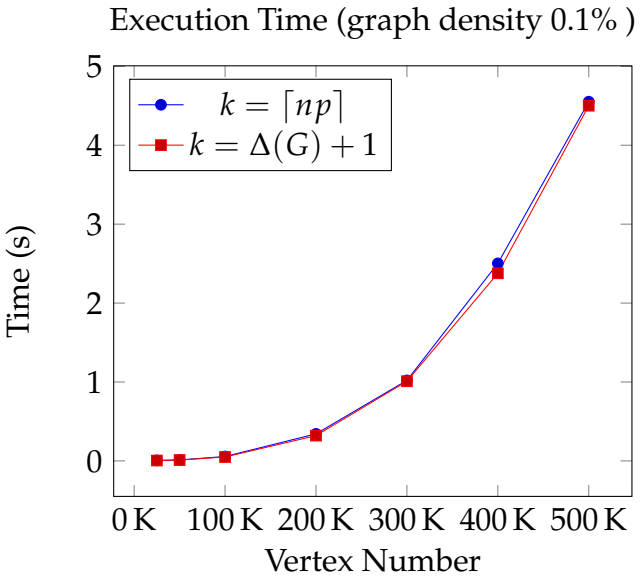


Figura 4.3: Confronto tempi di esecuzione per $k = \lceil np \rceil$(linea blu) e $k = \Delta(G) + 1$(linea rossa)

Per quanto riguarda l'indice di sbilanciamento, i quali dati sono riportati in figura 4.4, possiamo notare che nel caso di $k = \Delta(G) + 1$ il valore medio è pari a 29.27, un valore decisamente più alto rispetto al caso di $k = \lceil np \rceil$, con una deviazione standard pari a 6.62, questo perchè come si può notare nel caso di grafi più piccoli il valore dell'indice di sbilanciamento è più basso.

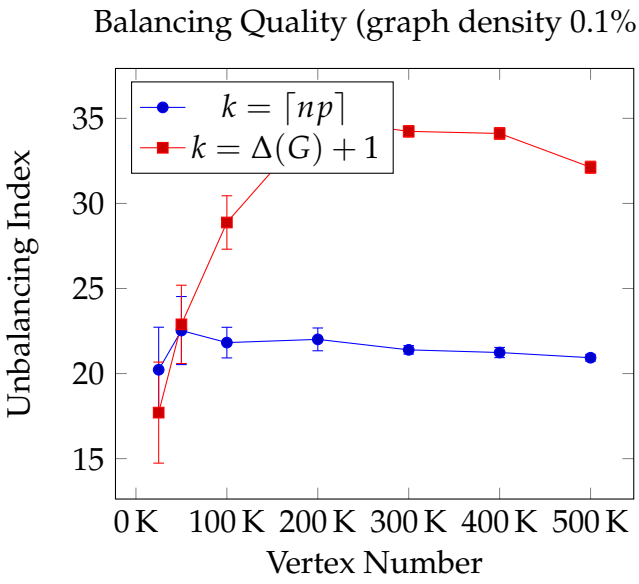


Figura 4.4: Confronto indice di Sbilanciamento per $k = \lceil np \rceil$(linea blu) e $k = \Delta(G) + 1$(linea rossa)

4.1.2 Grafi ER con densità 0.5%

In questo sottoparagrafo andremo ad analizzare i risultati ottenuti nel caso in cui la densità del grafo fosse pari a 0.5%. Nella figura 4.5 possiamo notare il confronto tra i tempi di esecuzione nei due casi in cui il calcolo della probabilità venga effettuato o ignorato, in questo caso la riduzione di tempo di esecuzione in percentuale è del 46.77% ed è nuovamente dipendente dalla dimensione del grafo, dove la minima è rappresentata dal caso 200k con una riduzione del 34.49%.

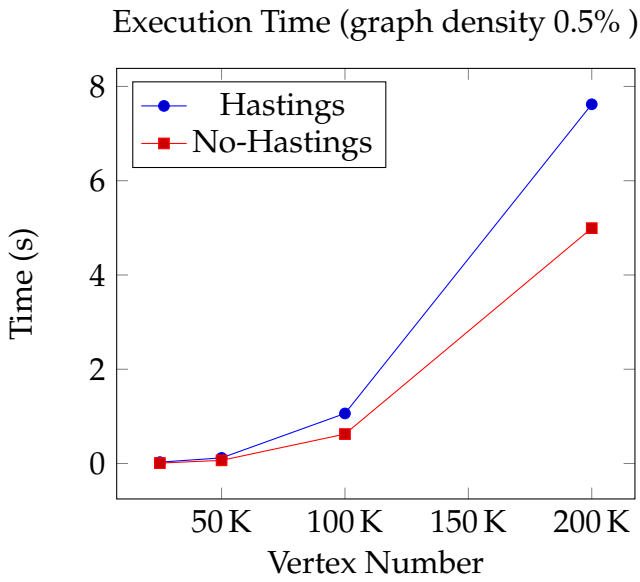


Figura 4.5: Tempi di esecuzione su grafi generati randomicamente con $k = \lceil np \rceil$, Metropolis-Hastings(linea blu), No-Hastings(linea rossa)

Riguardo gli indici di sbilanciamento, in figura 4.6, possiamo notare che il valore dell'indice di sbilanciamento nelle diverse dimensioni del grafo assume un valore medio di 9.36, con un valore di deviazione standard pari a 0.1, questi valori sono decisamente inferiori a quelli del caso di densità 0.1%, quindi possiamo notare che l'algoritmo fornisca soluzioni di bilanciamento migliori su grafi con densità più alta.

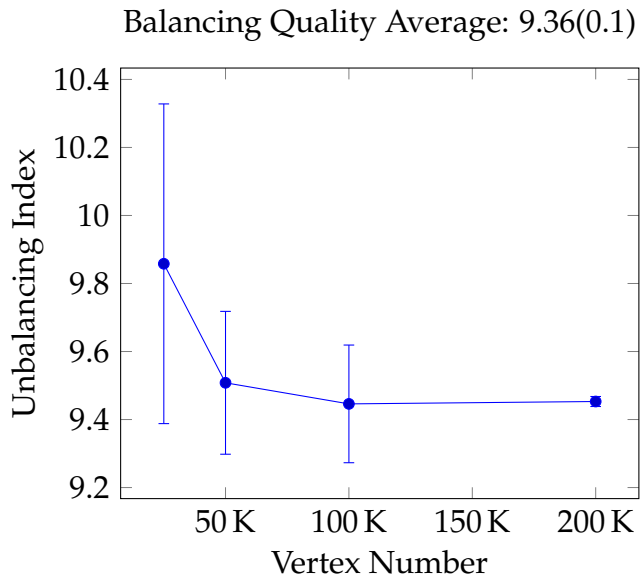


Figura 4.6: Indice di Sbilanciamento per $k = \lceil np \rceil$

Proseguiamo col confronto tra i due valori di k in esame, i tempi di esecuzione sono riportati in figura 4.7. Come nel caso precedente i tempi di esecuzione sono inferiori nel caso in cui k è pari al grado massimo, in questo caso però il tempo medio

è diminuito del 4.89%, una percentuale minore rispetto al caso precedente con grafi meno densi.

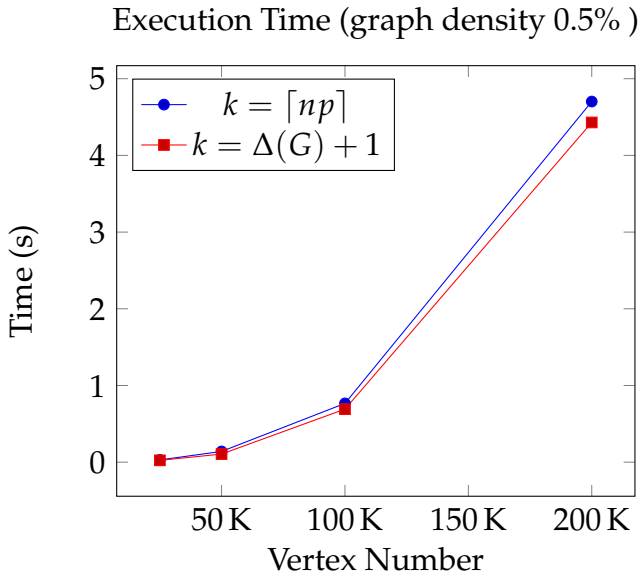


Figura 4.7: Confronto tempi di esecuzione per $k = \lceil np \rceil$(linea blu) e $k = \Delta(G) + 1$(linea rossa)

In figura 4.8 sono riportati i dati e il confronto dell'indice di sbilanciamento, per il caso k = $\Delta(G) + 1$ l'indice di sbilanciamento è mediamente più alto per tutte le dimensioni di grafo testate, ed assume un valore medio pari a 9.99, con una deviazione standard di 0.37.

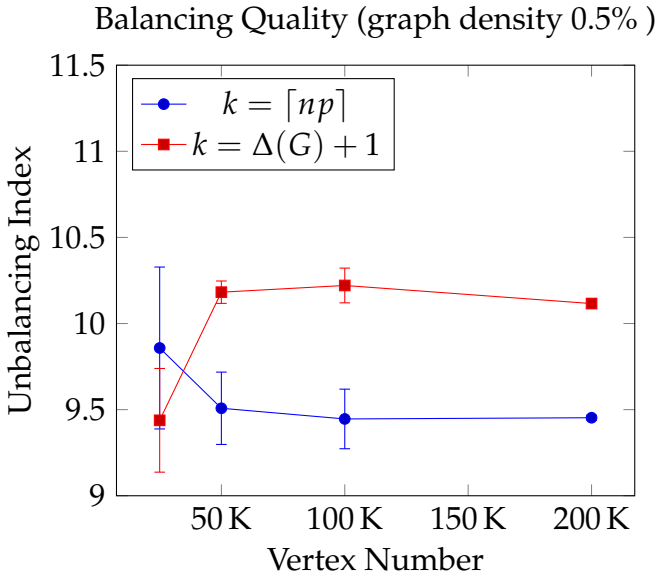


Figura 4.8: Confronto indice di Sbilanciamento per $k = \lceil np \rceil$(linea blu) e $k = \Delta(G) + 1$(linea rossa)

4.1.3 Grafi ER con densità 1%

Questo è l'ultima densità che abbiamo preso in esame nelle nostre prove. Nella figura 4.9 possiamo notare il confronto tra i tempi di esecuzione nei due casi in cui il calcolo della probabilità venga effettuato o ignorato, in questo caso la riduzione di tempo di esecuzione in percentuale è del 42.08%.

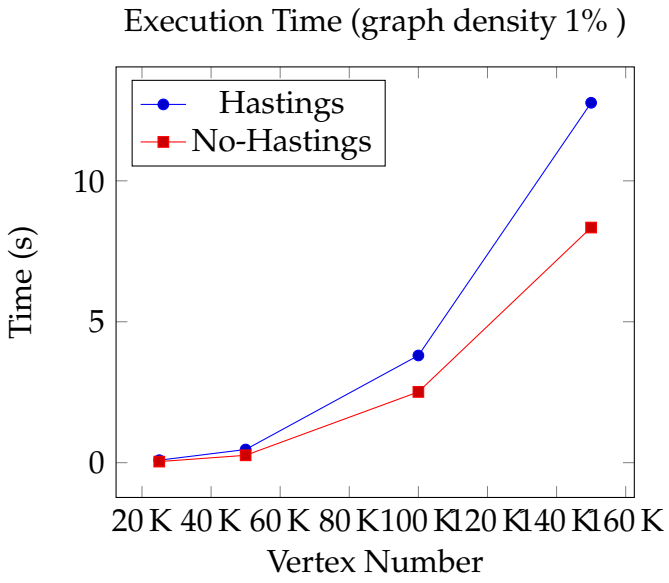


Figura 4.9: Tempi di esecuzione su grafi generati randomicamente con $k = \lceil np \rceil$, Metropolis-Hastings(linea blu), No-Hastings(linea rossa)

Riguardo gli indici di sbilanciamento, in figura 4.10 possiamo notare che il valore dell'indice di sbilanciamento nelle diverse dimensioni del grafo assume un valore medio di 6.67, con deviazione standard pari a 0.04, con l'aumento della densità possiamo notare che è stato ottenuto un ulteriore risultato migliore per quanto riguarda l'indice di sbilanciamento.

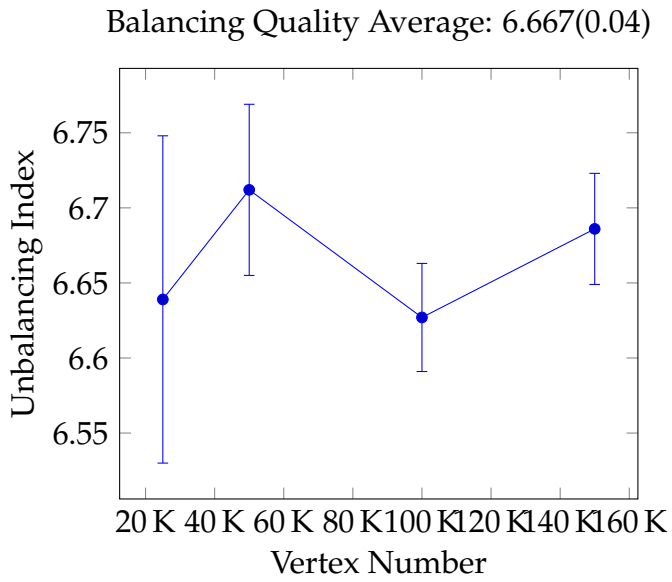


Figura 4.10: Indice di Sbilanciamento per $k = \lceil np \rceil$

I tempi di esecuzione, riportati in figura , sono molto simili nel caso in cui k è pari al grado massimo, in alcuni casi sono superiori e in altri inferiori, mediamente c'è un aumento del tempo necessario pari a 0.1%, quindi abbiamo un caso diverso rispetto alle precedenti densità.

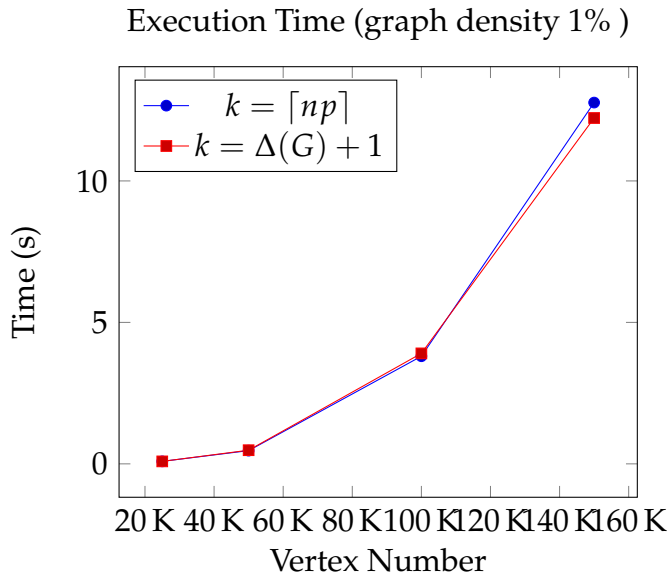


Figura 4.11: Confronto tempi di esecuzione per $k = \lceil np \rceil$(linea blu) e $k = \Delta(G) + 1$(linea rossa)

In figura 4.12 sono riportati i confronti dell'indice di sbilanciamento, nel caso $k = \Delta(G) + 1$ è leggermente più alto, ed assume un valore medio di 6.69, con una deviazione standard pari a 0.2 .

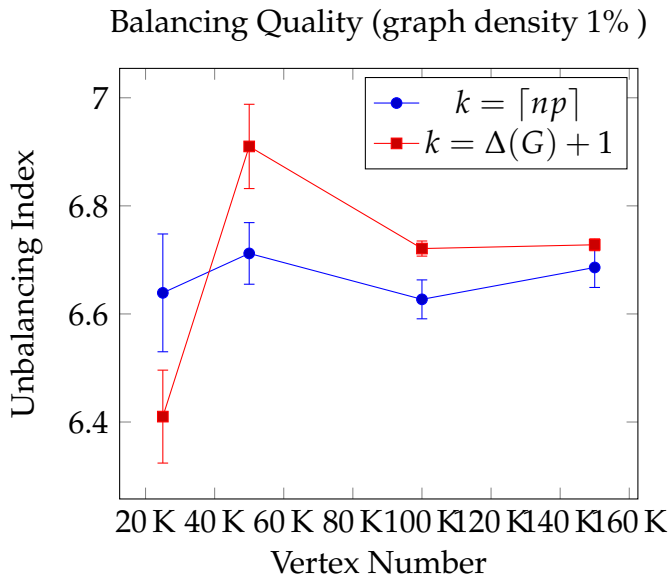


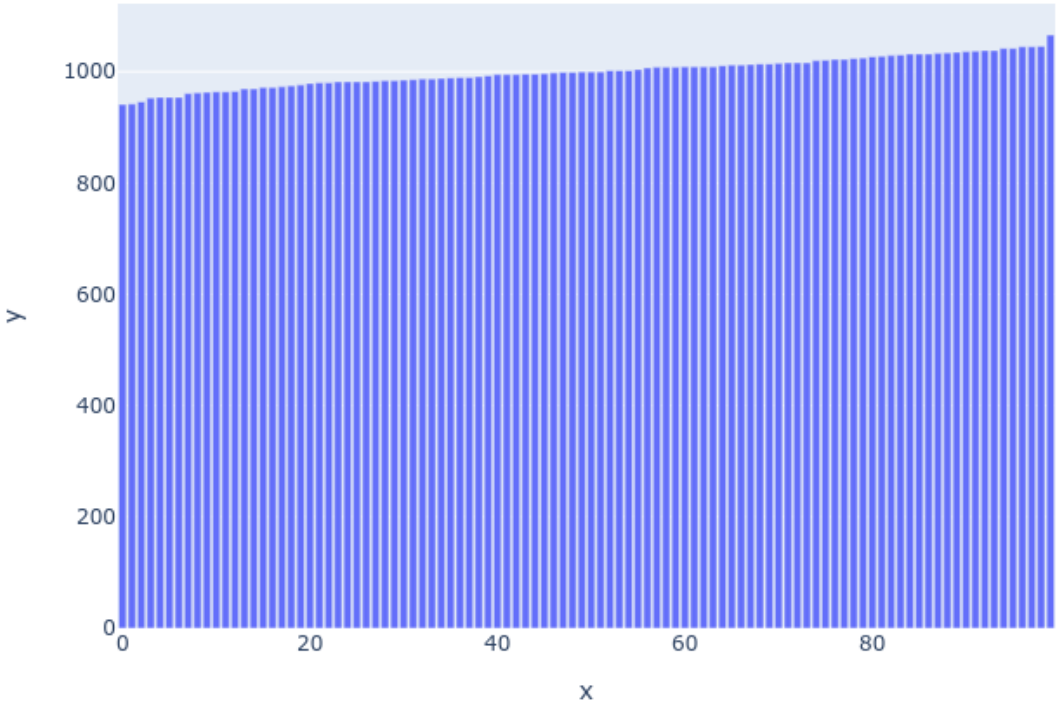
Figura 4.12: Confronto indice di Sbilanciamento per $k = \lceil np \rceil$(linea blu) e $k = \Delta(G) + 1$(linea rossa)

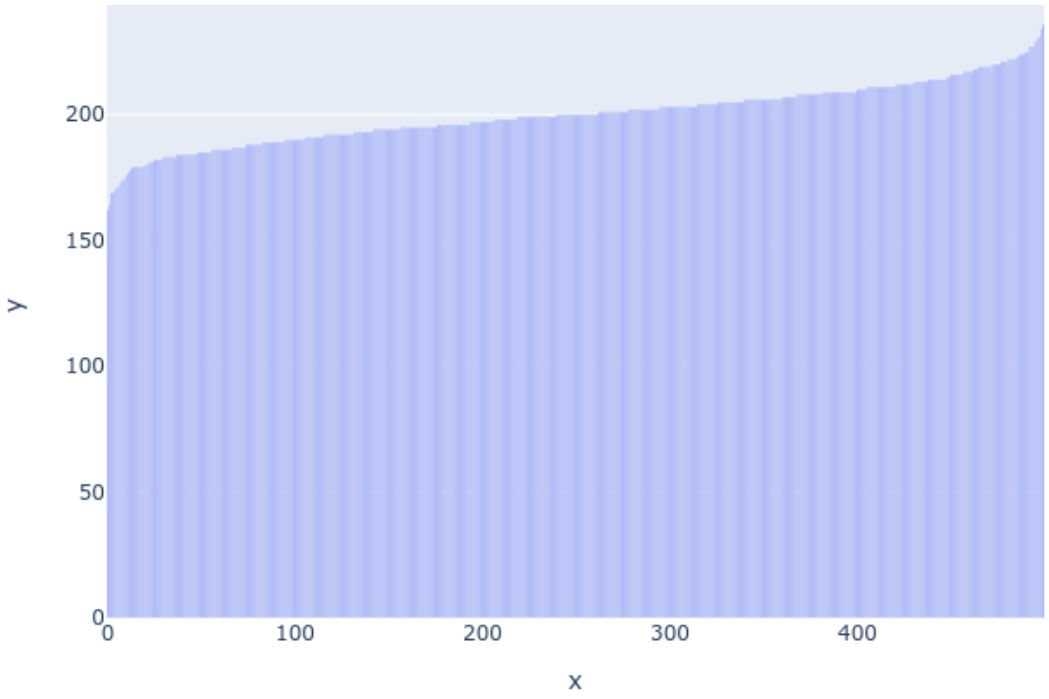
4.1.4 Confronto e qualità del bilanciamento

Dai risultati ottenuti è possibile affermare che all'aumentare della densità nei grafi ER la qualità del bilanciamento è sempre aumentata, con un indice di sbilanciamento più piccolo nel caso di grafi di densità 1%, pari mediamente a 6.67, rispetto al caso di densità di 0.1%, ben più grande e pari a 21.45%. Questo risultato è stato riscontrato anche nel caso in cui $k = \Delta(G) + 1$.

I valori del bilanciamento dell'algoritmo studiato sono decisamente positivi, soprattutto rispetto ad altri algoritmi di colorazione parallela, come l'algoritmo di Luby, di seguito proponiamo alcuni istogrammi della distribuzione dei colori del grafo dopo aver applicato l'algoritmo.

Nella figura 4.13 abbiamo riportato diversi istogrammi della distribuzione di colori su diverse densità. Questi istogrammi ci permettono di visualizzare il bilanciamento ottenuto su diverse distribuzioni, che insieme ai valori dell'indice di bilanciamento ci permettono di verificare la bontà del bilanciamento ottenuto.


1000
800
600
400
200
0
y
0
20
40
60
80
x


200
150
100
50
0
y
0
100
200
300
400
x

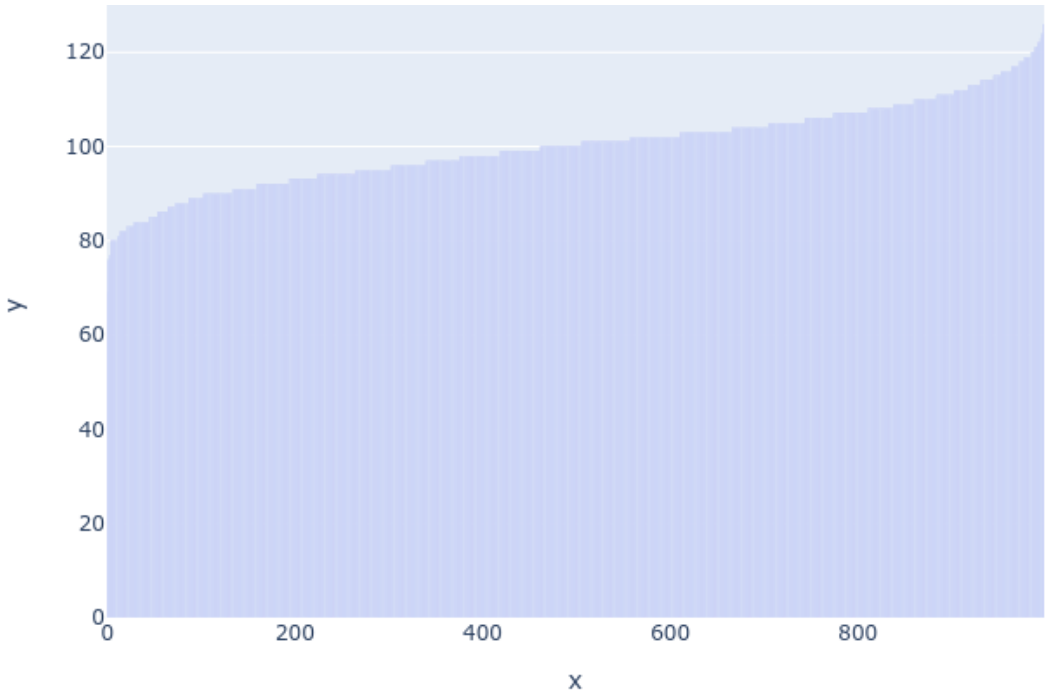


Figura 4.13: Istogrammi della distribuzione dei colori: Esempio 100k con densità 0.1%, 0.5%, 1%

In figura 4.14 abbiamo presentato gli istogrammi ottenuti con un'implementazione dell'algoritmo di Jones-Plassmann e una sua versione bilanciata per fare un confronto tra i bilanciamenti ottenuti. I valori dell'indice di sbilanciamento sono peggiori rispetto a quelli ottenuti col modello MCMC e questo lo possiamo notare anche visivamente dagli istogrammi.

Inoltre il numero di colori utilizzati è decisamente superiore e i tempi di esecuzione nel caso 100k con densità 0.1% sono tipicamente 0.08s, superiori quindi ai valori riscontrati nella colorazione MCMC.

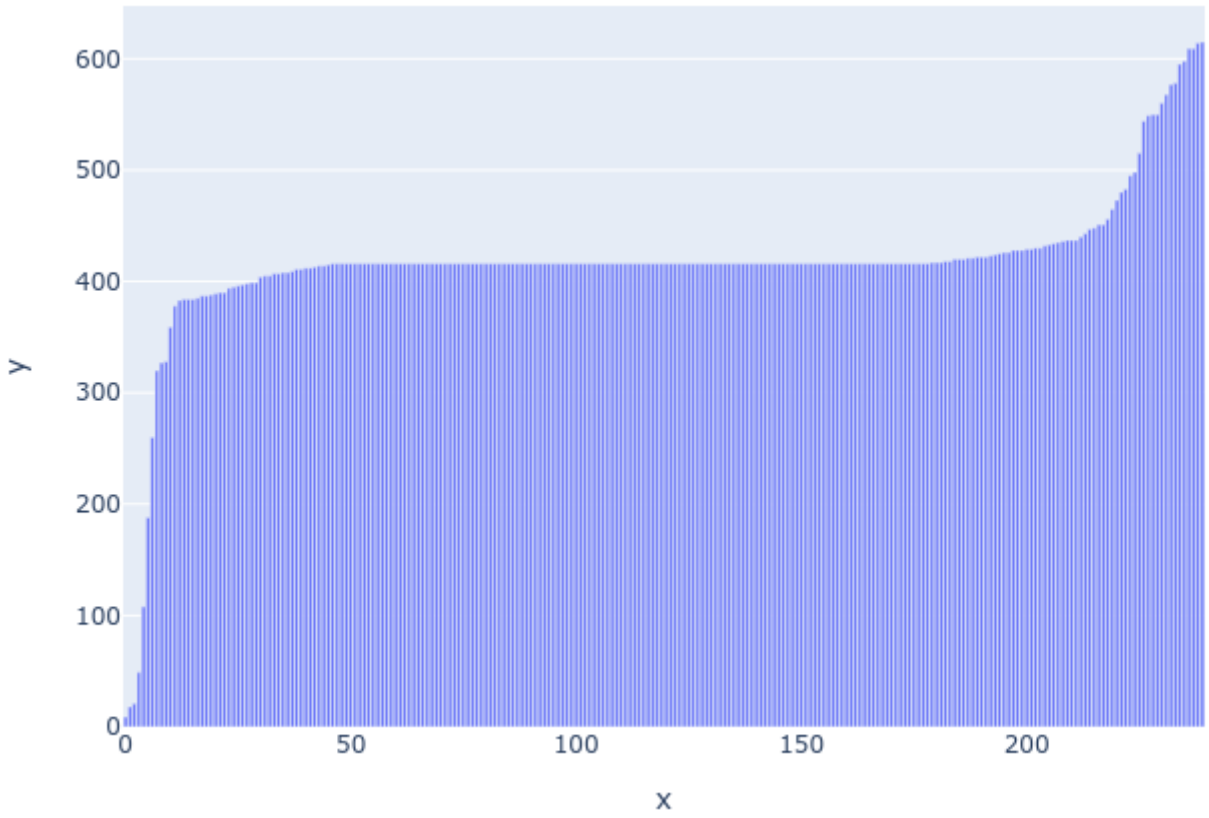


Figura 4.14: Istogrammi della distribuzione dei colori: Esempio 100k con densità 0.1%, implementazione Jones-Plassman

Per questo motivo abbiamo provato a fare un confronto del bilanciamento con una versione dell'algoritmo Jones-Plassmann bilanciato, gli istogrammi mostrati in figura 4.15 ci dimostrano anche visivamente che il bilanciamento ottenuto è confrontabile ma il numero di colori utilizzato rimane comunque superiore e i tempi di esecuzione aumentano ulteriormente.

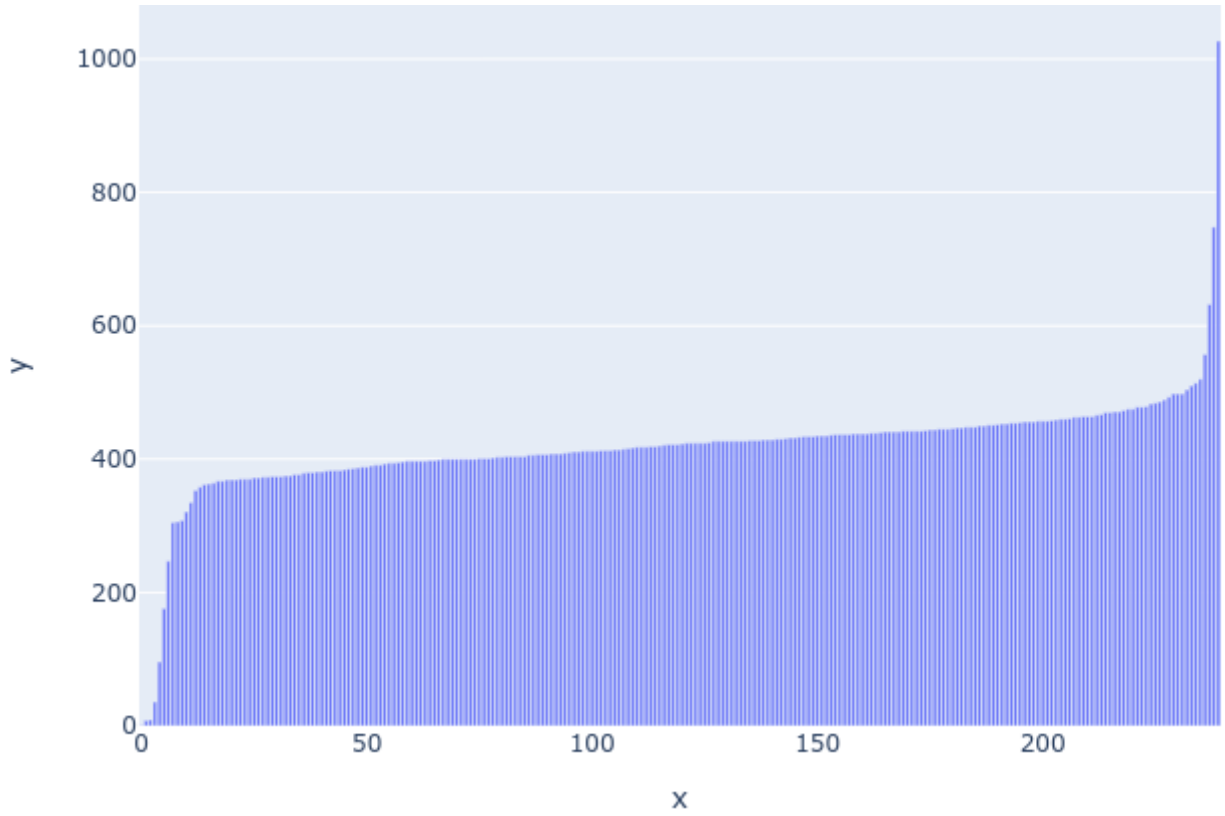


Figura 4.15: Istogrammi della distribuzione dei colori: Esempio 100k con densità 0.1%, implementazione Jones-Plassman

In figura 4.16 riportiamo il confronto tra i tempi di esecuzione per MCMC, Jones-Plassmann e la sua versione bilanciata, possiamo notare che MCMC otterrà tempi di esecuzione minori rispetto alla variante bilanciata.

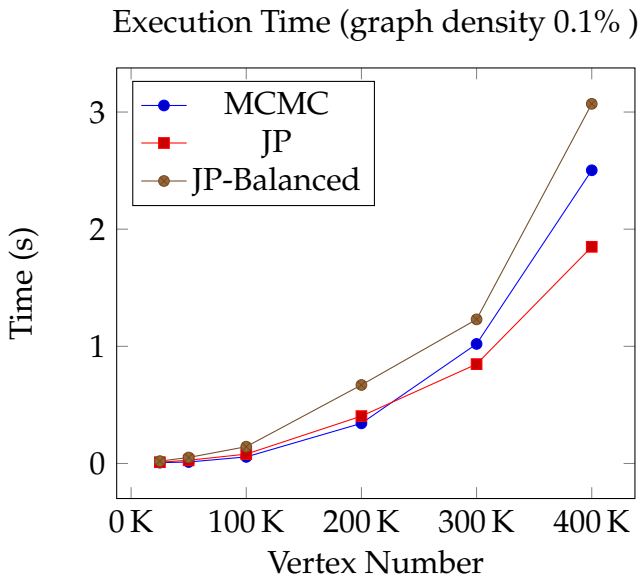


Figura 4.16: Confronto tempi di esecuzione per MCMC, Jones-Plassmann e Jones-Plassmann Bilanciato)

4.1.5 Diminuzione dei conflitti nel tempo

In questo paragrafo analizzeremo il comportamento dell'algoritmo nel corso delle diverse iterazioni dell'algoritmo. Nella tabella 4.1 sono stati riportati i dati medi del comportamento dell'algoritmo per quanto riguarda la diminuzione del numero di conflitti nel caso di grafi ER con densità 0.1%. Possiamo notare che in tutti i casi il numero di conflitti diminuisce maggiormente durante la prima iterazione dell'algoritmo, e che almeno l'85% dei conflitti iniziali vengono rimossi dopo la seconda iterazione.

Tabella 4.1: Diminuzioni conflitti medi in grafi ER con densità 0.1%.

Dim.	Avg.Round	#(c)	#(c) R1	#(c) R2	#(c) R3	#(c) R4	#(c) R5
25k	5.7	24978.4	11775 (52.8%)	3767 (32.1%)	637.8 (12.5%)	50.6 (2.3%)	3.6 (0.2%)
50k	5.11	49907.2	21987 (55.9%)	5919.4 (32.2%)	699.4 (10.5%)	24.2 (1.4%)	0.2 (0.04%)
100k	5.2	100059.8	41891.4 (58.1%)	10316.8 (31.6%)	911.6 (9.3%)	17.6 (0.9%)	0.4 (0.02%)
200k	5	200269	82258.2 (58.9%)	19317 (31.4%)	1546.4 (8.9%)	20.4 (0.8%)	0 (0.01%)
300k	5	300023	200389.6 (33.2%)	41795.8 (52.9%)	4233.23 (12.5%)	36.85 (1.4%)	0 (0.01%)
400k	6.6	399943.8	162051.8 (59.5%)	36996.8 (31.2%)	2693.2 (8.6%)	25 (0.7%)	0.2 (0%)
500k	5.667	499747.11	202374.4 (59.5%)	45603.78 (31.4%)	3282.89 (8.4%)	30.67 (0.6%)	1.14 (0%)

Nella tabella 4.2 abbiamo riportato un'analisi simile nel caso di grafi con densità 0.5%. Possiamo notare che i risultati ottenuti sono simili al caso precedente.

Tabella 4.2: Diminuzioni conflitti medi in grafi ER con densità 0.5%.

Dim.	Avg.Round	#(c)	#(c) R1	#(c) R2	#(c) R3	#(c) R4	#(c) R5
25k	4.8	24972.8	10373 (58.5%)	2460.4 (31.7%)	213.4 (9%)	2.6 (0.8%)	0 (0.01%)
50k	4.7	49746.2	20309.4 (59.2%)	4683.8 (431.4%)	355 (8.7%)	3.4 (0.7%)	0.1 (0%)
100k	5	99954	40685.6 (59.3%)	9339.2 (0.314%)	682.8 (8.7%)	5.6 (0.7%)	0 (0%)
200k	6.1	199812.8	80657.2 (59.6%)	18160.8 (31.3%)	1245.4 (8.4%)	7.8 (0.6%)	2.9 (0%)

4.2 Risultati dovuti a Freezing dei nodi

In questa sezione andremo a mostrare alcuni esempi sul comportamento del sistema utilizzando una tecnica di freezing dei nodi, cioè bloccando una certa percentuale di

nodi casuale del grafo ad ogni iterazione, lasciandoli dunque nello stesso stato precedente. Il test è stato eseguito su diverse percentuali di freezing e di seguito mostriamo i risultati di tempi di esecuzione e di indice di sbilanciamento.

Nelle figure 4.17 e 4.18 sono rappresentati i risultati con diversi valori di freezing per il tempo di esecuzione e indice di sbilanciamento nel caso di grafo di dimensione 100k e densità 0.1%, con percentuali di freezing superiori a 0 possiamo notare che i valori dell'indice di sbilanciamento è più basso, con il caso migliore nel caso di freezing al 95%. I tempi di esecuzione sono migliori nei casi di freezing tra 5% e 30%, e questo è accompagnato anche, come detto in precedenza, da un indice di sbilanciamento più basso.

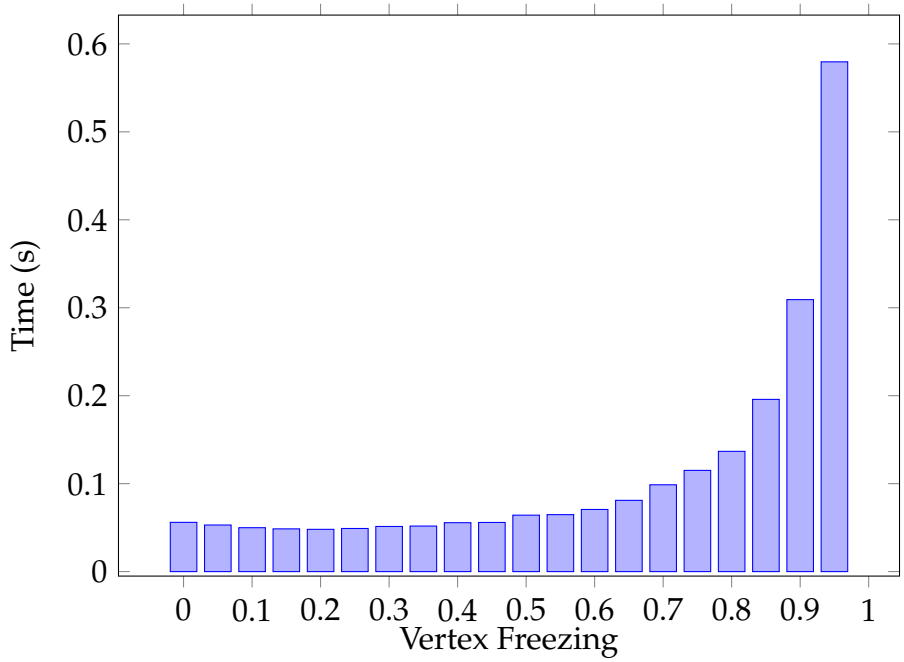


Figura 4.17: Tempi di esecuzione su diversi valori di freezing, Grafo(n=100k, p=0.01%)

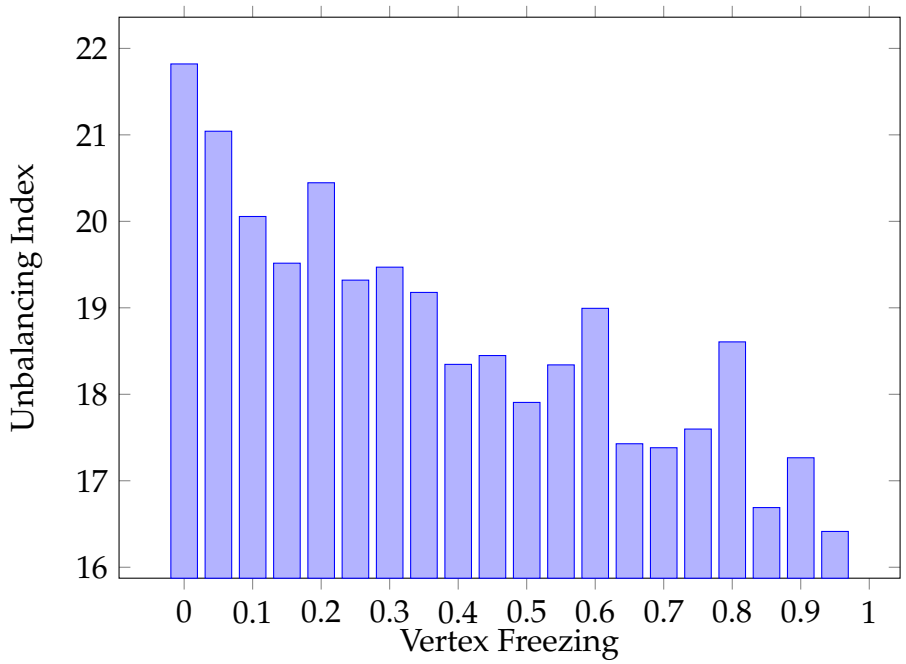


Figura 4.18: Indice di sbilanciamento su diversi valori di freezing, Grafo(n=100k, p=0.01%)

Nelle figure 4.19 e 4.20 abbiamo i risultati ottenuti utilizzando diverse percentuali di freezing per tempo di esecuzione e indice di sbilanciamento nel caso di un grafo di dimensione 100k e con densità 0.5%, possiamo notare che all'aumentare della percentuale tipicamente l'indice di sbilanciamento si abbassa, con i valori più bassi trovati nei casi di freezing al 90% e 95%, un miglioramento di circa 18.8% rispetto al caso in cui non venga applicato alcun freezing, però i tempi di esecuzione aumentano di molto, nel caso 90% circa triplicano, mentre per un freezing pari al 95% saranno circa 6 volte più grandi. Possiamo notare che i tempi di esecuzione si abbassano nei casi in cui la percentuale sia tra 5% e 50%, e si ottengono anche valori di indice di sbilanciamento migliori, ad esempio nel caso di percentuale pari a 40% avremo i tempi di esecuzione diminuiti del 29.4% e indice di sbilanciamento diminuito dell'11.5%.

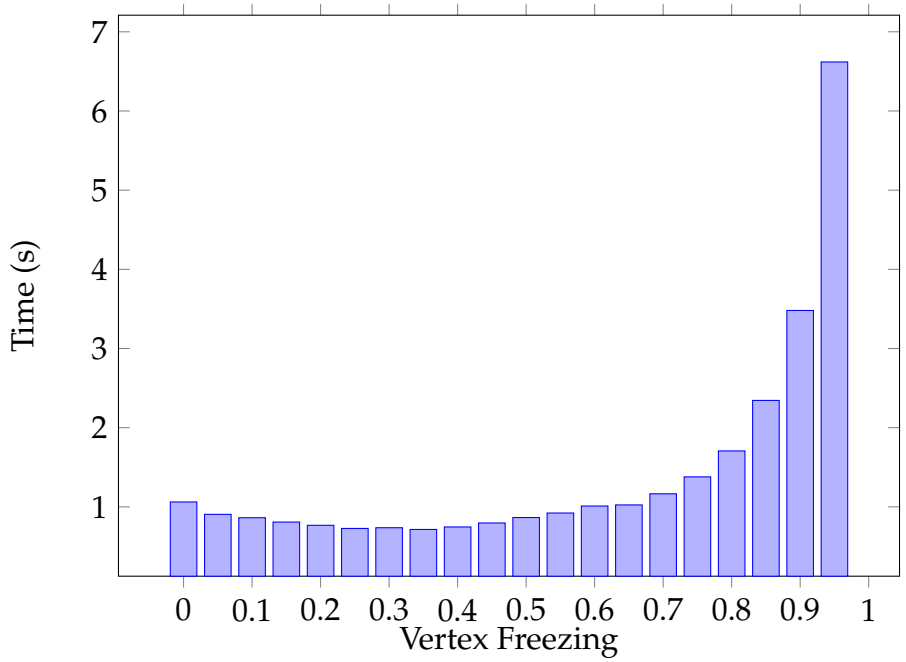


Figura 4.19: Tempi di esecuzione su diversi valori di freezing, Grafo(n=100k, p=0.05%)

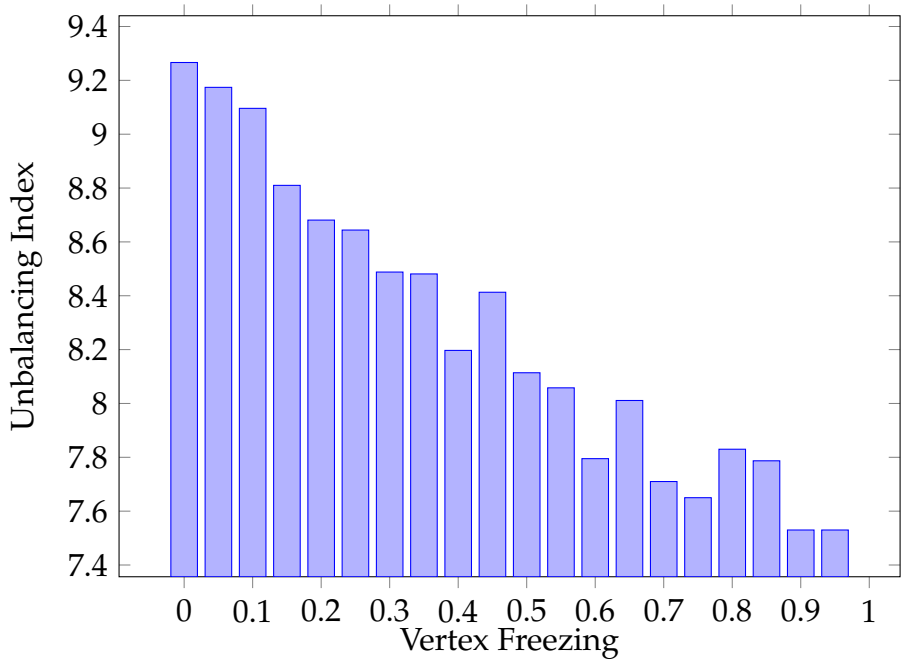


Figura 4.20: Indice di sbilanciamento su diversi valori di freezing, Grafo(n=100k, p=0.05%)

Nelle figure 4.21 e 4.22 abbiamo un ulteriore esempio dei risultati del freezing su grafi da 100k nodi, in questo caso con una densità più alta pari a 1%. Possiamo notare che i risultati non differiscano particolarmente dal caso precedente, i tempi di esecuzione chiaramente aumentano molto nei casi di freezing al 90% e 95%. Questa volta l'indice di sbilanciamento è sempre decrescente all'aumentare del freezing. Possiamo

notare nuovamente che nei casi 35% e 40% si presentano tempi di esecuzione minori, quindi questi valori potrebbero rappresentare una buona scelta per l'esecuzione dell'algoritmo.

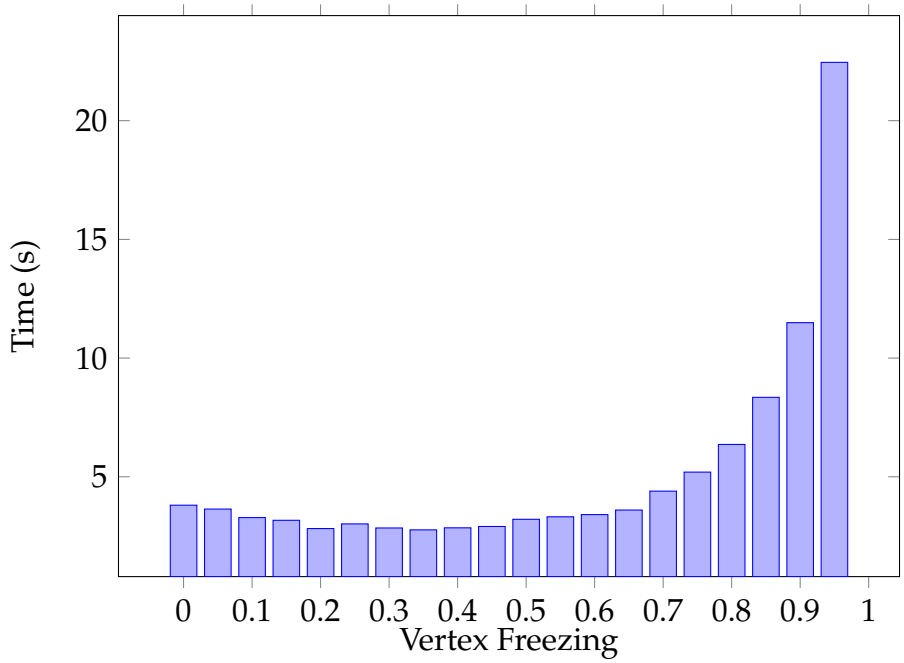


Figura 4.21: Tempi di esecuzione su diversi valori di freezing, Grafo(n=100k, p=1%)

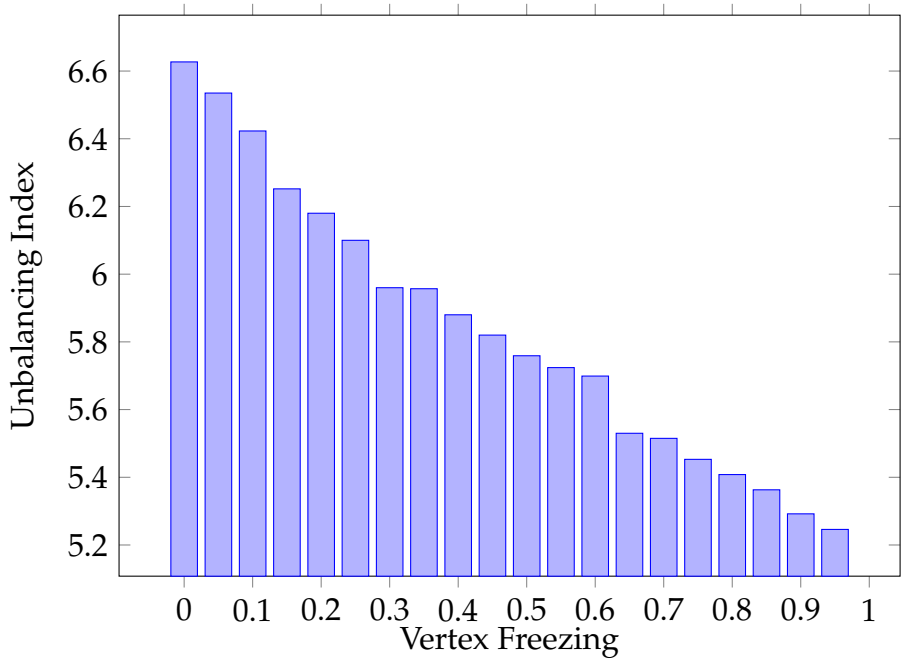


Figura 4.22: Indice di sbilanciamento su diversi valori di freezing, Grafo(n=100k, p=1%)

4.3 Reti Sociali

In questo ultimo paragrafo dei risultati andremo a mostrare i risultati ottenuti con l'algoritmo nel caso in cui i grafi presi in esame sono alcuni esempi di reti sociali, cioè grafi che rappresentano relazioni tra individui.

Le reti sociali analizzate sono le seguenti: sc-shipsec5, sc-pwtk, ca-coauthors-dblp, ca-hollywood-2009. Queste reti sociali sono state prese da Network Repository, e sono alcuni dei dataset utilizzati per gli esperimenti su algoritmi su grafi.

Nella tabella 4.3 sono riportati i risultati dell'algoritmo utilizzando come numero di colori $k = \Delta(G) + 1$, questi risultati ci mostrano che per quanto riguarda il bilanciamento si sono ottenuti risultati migliori sui dataset più grandi. I tempi di esecuzione sono chiaramente maggiori sui dataset di dimensione maggiore, questo è dovuto sia al numero di colori utilizzati, sia al numero di nodi e archi.

Tabella 4.3: Risultati su Reti Sociali con numero di colori pari al grado massimo

Grafo	Nodi	Archi	$\Delta(G)$	Grado Med.	Tempo(s)	Ind. Sbil.
sc-shipsec5	179.1K	4.4M	75	24	0.021	66.4
sc-pwtk	218K	11.4M	179	51	0.031	67.6
ca-coauthors-dblp	540K	30M	3.3K	56	0.935	10.1
ca-hollywood-2009	1.1M	56M	11.5K	105	24.95	7.64

Nella tabella 4.4 invece abbiamo mostrato i risultati ottenuti con il minor numero di colore che ha fornito delle soluzioni, possiamo notare che per i dataset più piccoli l'indice di sbilanciamento ottenuto è peggiore rispetto ad utilizzare $\Delta(G) + 1$ come numero di colori, invece per quanto riguarda i dataset più grandi avviene il contrario, con un'indice di sbilanciamento palesemente peggiore.

Tabella 4.4: Risultati su Reti Sociali con numero di colori minimo.

Grafo	Nodi	Archi	Grado Med.	Colori	Tempo(s)	Ind. Sbil.
sc-shipsec5	179.1K	4.4M	24	40	0.0415	55.8
sc-pwtk	218K	11.4M	51	59	0.0634	53.4
ca-coauthors-dblp	540K	30M	56	410	0.58	36.58
ca-hollywood-2009	1.1M	56M	105	2320	30.22	17.1

4.4 Conclusioni

Gli esperimenti effettuati hanno confermato due fatti importanti. Il primo è che il modello presentato offre un bilanciamento migliore rispetto ad altri algoritmi di colorazione paralleli basati sull'algoritmo di Luby, come l'algoritmo di Jones-Plassmann che

abbiamo mostrato in precedenza, sia per numero di colori utilizzati che per qualità del bilanciamento. Nel caso invece di confronto con un'implementazione di Jones-Plassmann bilanciata i risultati ottenuti sono paragonabili per quanto riguarda la qualità del bilanciamento, ma con un utilizzo di colori decisamente superiore e tempi di esecuzione più lunghi.

Il secondo è che la tecnica di freezing ci permette di ottenere sia tempi di esecuzione migliori, sia un bilanciamento migliore. Nel caso si ricerchi un bilanciamento migliore si può porre una percentuale di freezing molto elevata, come 95%, come abbiamo mostrato nei precedenti paragrafi, naturalmente a discapito di tempi di esecuzioni circa 6 volte superiori, risultato confermato in tutte le prove effettuate. Mentre se si volesse migliorare leggermente i tempi di esecuzione occorrerebbe scegliere una percentuale di freezing compresa tra 5% e 50%. Questa scelta oltre a diminuire leggermente i tempi di esecuzione fornisce anche un indice di sbilanciamento più basso.

Per quanto riguarda gli sviluppi futuri, una possibile variante dell'algoritmo potrebbe essere quella di cambiare l'obiettivo, ovvero non di ricercare una colorazione bilanciata, ma scegliere delle opportune frequenze target per la gamma di colori scelta che influenzino a loro volta la distribuzione di probabilità data nella scelta del colore per ogni nodo. Secondo questo intento, il disegno diventa un modello parallelo con approccio MCMC che premi le colorazioni con cluster aventi cardinalità il più vicino possibile alle frequenze date in input.

Bibliografia

[1] Reinhard Diestel. *Graph Theory*. Springer-Verlag New York, 2000.

[2] R.M.R. Lewis. *A Guide to Graph Colouring: Algorithms and Applications*. Springer, 2016.

[3] Kevin P. Murphy. *Machine Learning A Probabilistic Perspective*. Massachusetts Institute of Technology, 2012.

[4] K. Dincer, C. L. Martin J. R. Allwright, R. Bordawekar, P. D. Co ddington. A comparison of parallel graph coloring algorithms. Technical report, Technical Report, US Nuclear Regulatory Commission, 1995.

[5] Donatello Conte, Giuliano Grossi, Raffaella Lanzarotti, Jianyi Lin, Alessandro Petrini. A parallel mcmc algorithm for the balanced graph coloring problem. *Graph-Based Representations in Pattern Recognition 12th IAPR-TC-15 International Workshop*, pages 161–171, 2021.

[6] Donatello Conte, Giuliano Grossi, Raffaella Lanzarotti, Jianyi Lin, Alessandro Petrini. Analysis of a parallel mcmc algorithm for graph coloring with nearly uniform balancing. *Pattern Recognition Letters*, (149):30–36, 2021.