Alma Mater Studiorum · Università di Bologna

FACOLTÀ DI SCIENZE MATEMATICHE, FISICHE E NATURALI
Corso di Laurea Magistrale in Informatica

# Analisi di grafi su architetture a memoria distribuita

Tesi di Laurea in Algoritmi e Strutture Dati

Relatore:
Dott. Moreno Marzolla

Correlatore:
Dott. Matteo Magnani

Presentata da:
Mattia Lambertini

Sessione III
Anno Accademico 2009 - 10

*Dedico questo lavoro e questi cinque anni di studio li dedico ai giorni vissuti dal 2 maggio 1986 al 7 aprile 2010.*

# Introduzione

Le reti sono da molti anni argomento di ricerca nei più svariati campi: biologia, economia, logistica, telecomunicazioni e scienze sociali. In tutte queste discipline sono presenti sistemi reali che è risultato utile, al fine di ampliare la nostra comprensione, rappresentare come reti ossia come un insieme di nodi connessi tramite archi. Fra i vari campi di ricerca, nonostante le differenze, vi sono forti somiglianze fra le reti e la terminologia utilizzata. Negli ultimi anni grazie ad affascinanti scoperte è aumentata la consapevolezza dell'importante ruolo che giocano le reti nel mondo che ci circonda aumentando così l'interesse verso questo campo di ricerca.

Da un punto di vista informatico, data la dimensione delle reti reali, c'è un forte interesse nel trovare algoritmi che ne permettano l'analisi in tempi contenuti e al contempo mantenendo un ridotto consumo di risorse. Vi sono stati vari sforzi in proposito e questo lavoro di tesi si propone di fornire una panoramica dei possibili approcci ed un'analisi prestazionale delle soluzioni allo stato dell'arte, mostrandone i limiti e i punti di forza. L'analisi si focalizza sul calcolo della Betweenness centrality, ossia un indice che fornisce una stima dell'importanza dei nodi della rete. La scelta di tale metrica è stata guidata dall'importanza che ricopre in tutti i campi precedentemente elecanti. Inoltre, in campo informatico, il calcolo efficiente di tale metrica su reti di grandi dimensioni è tutt'ora un problema aperto ed un'interessante sfida.

Il miglior algoritmo sequenziale attualmente noto per il calcolo della betweenness centrality su grafi di $n$ nodi ed $m$ archi ha un costo computazionale di $O(nm)$ su grafi non pesati, ossia ai cui archi viene associato il peso 1, ed un costo di $O(nm+n^2 \log n)$ su grafi pesati, ossia ai cui archi viene associato un peso positivo arbitrario.
Un approccio sequenziale rende dunque impraticabile l'analisi di reti la cui dimensione supera le centinaia di migliaia di nodi, per problemi legati alla potenza di calcolo. Inoltre vi sono anche problemi dovuti alla quantità di

memoria centrale disponibile sul singolo calcolatore. Per ridurre i tempi d'esecuzione e sfruttare la potenza di calcolo delle sempre più diffuse architetture multi-core e multi-processore sono state proposte diverse soluzioni concorrenti basate sul citato algoritmo sequenziale. Tali proposte, nonostante riducano il tempo di calcolo, sono limitate dall'ancora scarsa quantità di memoria centrale e dalla potenza di calcolo disponibile sulla singola macchina. Queste considerazioni portano a trovare soluzioni al problema distribuendo il calcolo su diversi calcolatori che cooperano nella computazione.

Questo lavoro di tesi tratta il tema delle reti complesse, mostrando i principali modelli di rete complessa quali: il modello *Random*, il modello *Small-World* ed il modello *Scale-free*; si introdurranno alcune metriche usate per descrivere le reti complesse quali la *Degree centrality*, la *Closeness centrality* e la *Betweenness centrality*; si descriveranno i problemi da tenere in considerazione durante la definizione e l'implementazione di algoritmi su grafi; i modelli di calcolo su cui progettare gli algoritmi per risolvere i problemi su grafi; un'analisi prestazionale degli algoritmi proposti per calcolare i valori di Beweenness centrality su grafi di medio-grandi dimensioni. Parte di questo lavoro di tesi è consistito nello sviluppo di *LANA, LArge-scale Network Analyzer*, un software che permette il calcolo e l'analisi di varie metriche di centralità su grafo.

In particolare, il capitolo 1 presenta le principali metriche d'analisi e i diversi modelli di reti complesse. Il capitolo 2 ed il capitolo 3 forniscono una panoramica sui modelli di calcolo parallelo a memoria condivisa, a memoria distribuita e sui problemi legati agli algoritmi su grafi. Il capitolo 4 presenta, nella prima parte, gli strumenti utilizzati per le analisi e le implementazioni degli algoritmi analizzati; nella seconda parte invece vengono mostrati i risultati delle analisi svolte.

# Indice

# Elenco delle figure

# Capitolo 1

# Teoria dei grafi e reti complesse

In questo capitolo si spiegheranno brevemente alcune proprietà delle reti e se ne elencheranno le differenti tipologie in modo da fornire un quadro completo sul dominio del problema.

Nonostante, come già accennato, le reti complesse siano una realtà multidisciplinare, si utilizzerà come punto di riferimento le reti sociali anche se nel proseguo si forniranno esempi che esulano da questo contesto. Nelle reti sociali i nodi rappresentano attori e gli archi rappresentano solitamente la presenza di una relazione fra gli attori. Per esempio, è possibile rappresentare il social network facebook come una rete sociale dove nodi sono gli utenti di facebook e gli archi rappresentano le relazioni di amicizia. Mentre nel social network twitter i nodi possono essere gli utenti di twitter e gli archi possono rappresentare la possibilità degli attori di scambiarsi informazioni.

Si rende noto comunque che verrà fornita un'introduzione non esaustiva in quanto il mio sforzo, in questo lavoro di tesi, è centrato su un punto di vista algoritmico e prestazionale e non sull'ampliare la comprensione delle reti. Per una trattazione esaustiva si rimanda alle pubblicazioni indicate nei paragrafi seguenti.

## 1.1 Preliminari

In questa sezione si forniscono le definizioni e la terminologia base della teoria dei grafi in modo da venire incontro anche ai lettori che si approcciano per la prima volta all'argomento.

Con il termine *grafo* o *rete* si fa riferimento ad un insieme di nodi connessi fra loro mediante archi. I nodi rappresentano le entità del nostro problema mentre gli archi sono le relazioni fra i nodi. Più formalmente si definisce un grafo $G = (V, E)$ dove $V$ denota l'insieme dei nodi ed $E$ l'insieme degli archi. Il numero di nodi $|V|$ viene solitamente indicato con $n$ ed il numero di archi $|E|$ viene indicato con $m$.

Un grafo $G = (V, E)$ può essere definito *orientato* o *non orientato*. Se il grafo $G$ è orientato allora l'insieme $E$ è costituito da coppie ordinate di nodi, $E = \{(s, t) \mid s, t \in V\}$. Gli archi $(e, t) \in E$ sono detti archi orientati e sono caratterizzati da una direzione, da $s$ verso $t$. Mentre se il grafo $G$ non è orientato allora l'arco $\{s, t\} \in E$ è un arco non orientato. Gli archi non orientati sono bidirezionali. Un generico arco $(s, t) \in E$ che connette i nodi $s, t \in V$ è detto *incidente* ad ognuno dei due nodi.

Sia nei grafi orientati che in quelli non orientati, ad ogni arco viene associato un peso $w(s, t)$ strettamente maggiore di zero. I grafi in cui tutti gli archi hanno peso $w(s, t) = 1$ si definiscono *grafi non pesati*. Altrimenti si fa riferimento a *grafi pesati*. Per semplicità, in questo capitolo, si farà riferimento solamente a grafi non pesati salvo altre precisazioni.

In un grafo $G$, un *cammino* dal vertice $s$ al vertice $t$ è definito come una sequenza di archi $(v_i, v_{i+1})$, con $0 \leq i < l$, dove $v_0 = s$ e $v_l = t$. La *lunghezza* di un cammino è definita come la somma dei pesi degli archi del cammino. I cammini di lunghezza minima fra due nodi $s$ e $t$ vengono detti *cammini minimi* da $s$ a $t$. La *distanza* $d(s, t)$ fra il nodo $s$ ed il nodo $t$ equivale alla lunghezza di un cammino minimo fra $s$ e $t$.

In un grafo $G = (V, E)$, si definisce *componente connessa* $C(G)$ un sottoinsieme $W \subseteq V$ tale per cui esiste un cammino tra ogni coppia di nodi $s, t \in W$. Un grafo $G$ è detto *connesso* se esiste una sola componente connessa composta da tutti i nodi del grafo, viene detto *sconnesso* altrimenti.
Si definisce *fortemente connesso* un grafo $G = (V, E)$ in cui esiste un cammino per ogni coppia di nodi $s, t \in V$ dal nodo $v$ al nodo $t$ e viceversa. Le definizioni di grafo connesso e fortemente connesso differiscono solo nel caso in cui il grafo è orientato. Un grafo $G = (V, E)$ è detto *completo* quando tutti i nodi $v \in V$ hanno un arco che li collega a tutti i nodi $u \in V \backslash v$. Un grafo $G = (V, E)$ è detto *completamente sconnesso* se $E = \emptyset$.

In un grafo $G = (V, E)$, dato l'insieme di nodi $V^* \subseteq V$ si definisce *sottografo indotto* $SG = (V^*, E_{V^*})$ come un grafo formato dall'insieme di nodi

$V^*$ e dagli archi che connettono tali nodi. Se in un grafo G=(V,E) esiste una componente connessa che contiene una frazione significativa dei nodi del grafo allora tale componente è detta *componente gigante*.

## 1.2 Proprietà delle reti

In questa sezione si forniscono alcune metriche per l'analisi di grafi [1]. In particolare si forniscono alcune metriche generali per determinare "l'importanza" dei nodi cercando di rimanere il più possibile generali. Il concetto di importanza, che viene spesso utilizzato quando si parla di reti complesse, si basa sull'idea che un nodo (es. attori nel contesto delle reti sociali) è importante se occupa una posizione strategica all'interno della rete.

Come prima metrica viene presentata la *Degree centrality* che fornisce una stima della capacità dei nodi di avere una relazione diretta con gli altri nodi. Per esempio, prendendo in considerazine il grafo di internet a livello rete, dove i nodi sono i dispositivi di rete[1] e gli archi rappresentano le connessioni fra i router, indica la possibilità di comunicare direttamente. La seconda metrica, il *coefficiente di clustering*, stima quanto i nodi adiacenti ad un nodo siano anch'essi in relazione fra loro. Per esempio, nelle reti sociali dove gli archi rappresentano la relazione di amicizia/frequentazione, il coefficiente di clustering fornisce una stima di quanto il gruppo, o comunità, sia chiuso rispetto agli altri nodi nella rete. Infine vengono introdotte altre due metriche, la *Closeness centrality* e la *Betweenness centrality* che invece tengono in considerazione le distanze fra i nodi quindi forniscono una stima che risalta l'importanza dei nodi in base alla loro centralità nelle relazioni fra gli altri nodi. Per esempio, in un grafo che rappresenta una rete ferroviaria, dove i nodi sono le stazioni e gli archi sono i collegamenti fra le varie stazioni, una possibile metrica per stabilire l'importanza delle stazioni può essere quella di contare il numero di tratte di cui fa parte ogni stazione e dividerlo per il numero di tratte presenti nella rete. Le stazioni che risultano essere presenti nel maggior numero di tratte vengono considerate nodi cruciali della rete.

Essendo la teoria dei grafi un tema vastissimo si rimanda per approfondimenti a [2].

### 1.2.1 Degree centrality

La Degree centrality è una delle più semplici metriche utilizzate per descrivere una rete. Tale indice fornisce una stima generale sulla struttura del

[1] Per dispositivi di rete si intende router, switch, hub.

grafo basandosi unicamente sul numero di archi incidenti in ogni nodo. In particolare si distinguono i casi in cui il grafo sia orientato o non orientato. Nei grafi non orientati il grado di un nodo è esattamente il numero di archi in cui è coinvolto.

**Def.** *Dato un grafo non orientato $G = (V, E)$, il grado di un nodo $v \in V$, definito come $\delta(v)$, rappresenta il numero di archi incidenti in $v$. Segue la definizione formale.*

$$\delta(v) \stackrel{def}{=} \sum_{i \in V} e_{v,i}, \ \ per \ \ e_{v,i} = \begin{cases} 1 & se \ \{v, i\} \in E \\ 0 & altrimenti \end{cases}$$

In un grafo orientato invece si distingue fra il numero di archi orientati verso il nodo (archi entranti) e il numero di archi orientati verso i nodi adiacenti (archi uscenti). Si hanno quindi un grado "entrante" e un grado "uscente" detti rispettivamente IN-Degree e OUT-Degree.

**Def.** *Dato un grafo orientato $G = (V, E)$, l'IN-Degree di un nodo $v \in V$, definito come $\delta_{in}(v)$, rappresenta il numero di archi entranti in $v$. Definita formalmente come segue.*

$$\delta_{in}(v) \stackrel{def}{=} \sum_{i \in V} e_{v,i}, \ \ per \ \ e_{v,i} = \begin{cases} 1 & se \ (i, v) \in E \\ 0 & altrimenti \end{cases}$$

**Def.** *Dato un grafo orientato $G = (V, E)$, l'OUT-Degree di un nodo $v \in V$, definito come $\delta_{out}(v)$, rappresenta il numero di archi uscenti da $v$. Definita formalmente come segue.*

$$\delta_{out}(v) \stackrel{def}{=} \sum_{i \in V} e_{v,i}, \ \ per \ \ e_{v,i} = \begin{cases} 1 & se \ (v, i) \in E \\ 0 & altrimenti \end{cases}$$

In un grafo $G = (V, E)$, la Degree centrality $DC(v)$ di un nodo $v \in V$ fornisce quindi una stima dell'importanza del nodo $v$, relativa al dominio del sistema che stiamo analizzando, basandosi sul numero di nodi con cui è in relazione diretta. Per esempio, nel social network twitter, il valore di Degree centrality di un nodo mi fornisce una stima della sua capacità di ottenere e divulgare informazioni direttamente. I nodi con grado maggiore possono essere considerati come punti cruciali della rete e vengono definiti *hub*. Segue la definizione di Degree centrality.

**Def.** *Dato un grafo $G = (V, E)$, si definisce la Degree centrality $DC(v)$ di un nodo $v \in V$ nel seguente modo:*

$$DC(v) \stackrel{def}{=} \frac{\delta(v)}{n-1}$$

*Se il grafo $g$ è orientato allora $\delta(v) = \delta_{out}(v) + \delta_{in}(v)$.*

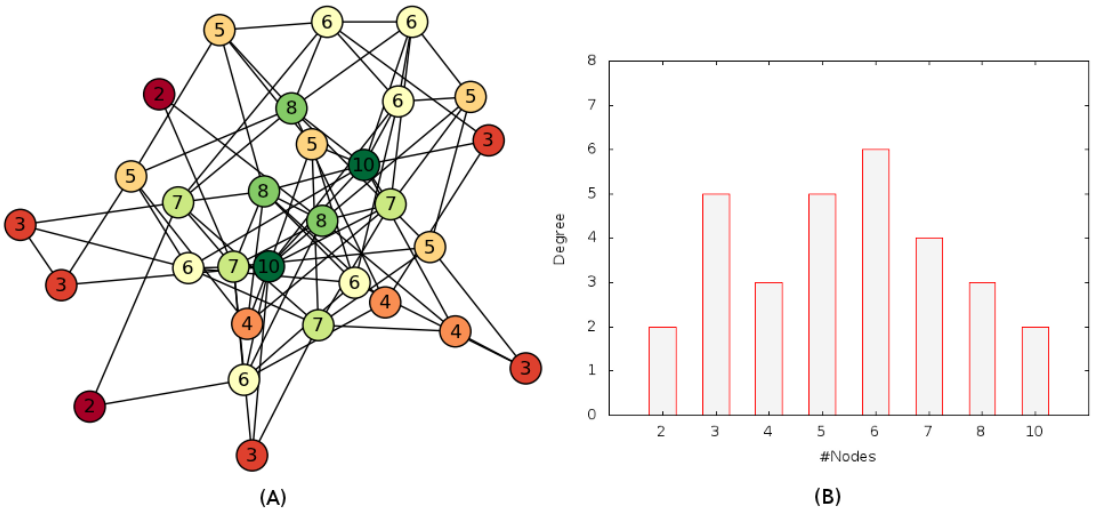


Figura 1.1: L'immagine (A) mostra un grafo di 30 nodi e nell'etichetta di ogni nodo è mostrato il grado. L'immagine (B) mostra la distribuzione del grado del grafo a sinistra.

La figura 1.1(A) mostra un grafo non orientato indicando per ogni nodo il valore del grado, mentre la figura 1.1(B) mostra un grafico a istogrammi che mette in relazione i valori $k$ assunti dal grado con il numero di nodi che

mostrano tale grado, ossia la distribuzione del grado $DR(k)$. Segue la definizione formale di distribuzione del grado.

**Def.** *Si definisce distribuzione del grado $DR(k)$ di un grafo non orientato $G = (V, E)$ nel seguente modo:*

$$DR(k) \stackrel{def}{=} |\{v \in V \mid \delta(v) = k\}|$$

Nel grafo in figura 1.1(A) i nodi sono colorati in base ad una scala che aiuta a distinguere l'importanza del nodo secondo la Degree centrality. La scala varia dal colore rosso al colore verde. Il colore rosso viene assegnato ai nodi con bassa importanza mentre il verde ai nodi con alta importanza.

### 1.2.2 Coefficiente di clustering

Il coefficiente di clustering è un indice che mostra quanto i vicini di un generico nodo $v$ siano a loro volta adiacenti. Tale indice è di utilità nell'analizzare la potenziale diffusione d'informazione, rimanendo nel tema delle reti sociali, fra i nodi. Vi sono risultati [3] che mostrano come l'aumentare del coefficiente di clustering diminuisca l'efficienza della rete nel diffondere l'informazione. Nonostante il focus per le reti sociali e la diffusione di informazioni, il coefficiente di clustering viene usato per analizzare vari contesti come la diffussione di malattie o di virus software.

**Def.** *Sia G = (V, E) un grafo connesso e non orientato. Dato un nodo $v \in V$, l'insieme dei nodi adiacenti a v è detto N(v). Si definiscono rispettivamente $n_v$ ed $m_v$ come il numero di nodi in N(v) ed il numero di archi presenti nel sottografo indotto da N(v).*
*Il coefficiente di clustering CC(v) del nodo v è definito nel seguente modo:*

$$CC(v) \stackrel{def}{=} \begin{cases} \dfrac{m_v}{\binom{n_v}{2}} = \dfrac{2m_v}{n_v(n_v-1)} & se\ \delta(v) > 1 \\ non\ definito & altrimenti \end{cases}$$

Nel caso in cui il grafo $G = (V, E)$ sia orientato è necessario tenere in considerazione la differenza fra gli archi in uscita e gli archi in entrata. Il numero massimo di archi presenti nel sottografo indotto dai nodi adiacenti a $v$ diventa $2\binom{n_v}{2} = n_v(n_v - 1)$.

**Def.** *Sia G = (V, E) un grafo connesso e orientato. N(v) l'insieme dei nodi adiacenti a v.* $n_v = |N(v)|$ *il numero di nodi in N(v). Ed* $m_v$ *il numero di archi presenti nel sottografo indotto da N(v).*
*Il coefficiente di clustering CC(v) di un nodo v, con grado* $\delta(v) = \delta_{in}(v) + \delta_{out}(v)$*, è definito nel seguente modo:*

$$CC(v) \stackrel{def}{=} \begin{cases} \frac{m_v}{2\binom{n_v}{2}} = \frac{m_v}{n_v(n_v-1)} & se\ \delta(v) > 1 \\ non\ definito & altrimenti \end{cases}$$

Date le definizioni di coefficiente di clustering per i singoli nodi, si definisce il coefficiente di clustering $CC(G)$ di un grafo $G = (V, E)$ come la media dei coefficienti definiti per ogni nodo. Ossia:

**Def.** *Sia G = (V,E) un grafo connesso. Sia* $V^* \stackrel{def}{=} \{v \in V \mid \delta(v) > 1\}$*. Si definisce coefficiente di clustering di v, CC(G), come segue:*

$$CC(G) \stackrel{def}{=} \frac{1}{|V^*|} \sum_{v \in V^*} CC(v)$$

Per completezza si cita la proposta di modifica del coefficiente di clustering in caso di grafi pesati nel lavoro di Barrat et al [4]. Dato che il coefficiente di clustering indica quanto i nodi del grafo formino gruppi chiusi allora il peso degli archi dovrebbe essere tenuto in considerazione.

### 1.2.3 Closeness Centrality

La closeness centrality fornisce una misura della distanza di un nodo da tutti gli altri nodi. Al contrario della Degree centrality, questa metrica necessità di una visione globale della rete e non è quindi limitata alla visione locale dei singoli nodi.

La closeness centrality $CN(v)$ del nodo $v$ è definita nel seguente modo:

**Def.** *Dato un grafo $G = (V, E)$ (fortemente) connesso e pesato, sia $d(u, v)$ la distanza fra i vertici $u, v \in V$. Si definisce la Closeness centrality $CN(v)$ di un vertice $v \in V$ come:*

$$CN(v) \stackrel{def}{=} \frac{n-1}{\sum_{u \in V \setminus v} d(u, v)}$$

L'utilità di tale metrica è evidente se si pensa che fornisce un'indicazione su quali punti della rete minimizzano la distanza media fra i nodi. Ossia, il valore di closeness di un nodo $v$ stima il grado di vicinanza del nodo $v$ dal resto dei nodi del grafo.

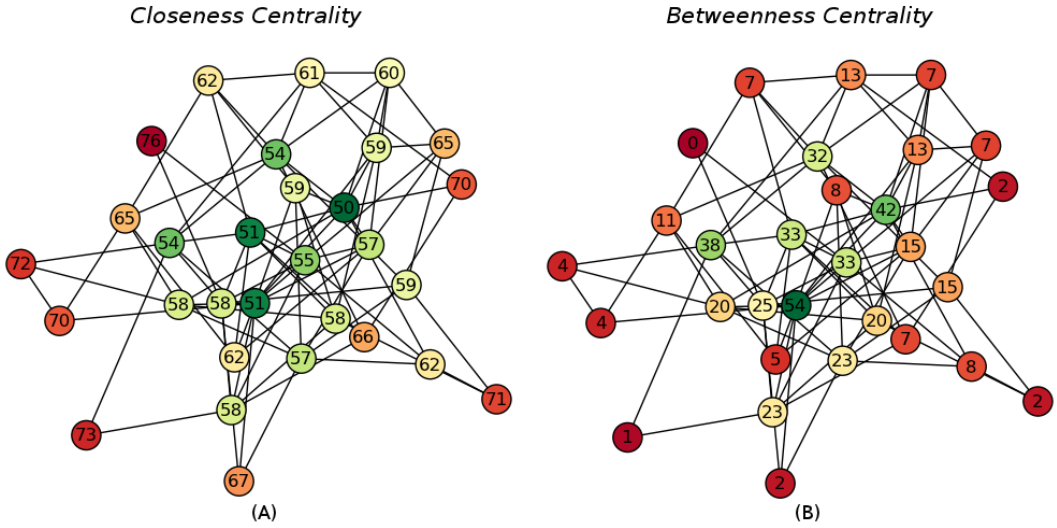


Figura 1.2: Due rappresentazioni dello stesso grafo di 30 nodi. Il grafo (A) mostra la sommatoria delle distanze $d(u, v)$ sull'etichetta di ogni nodo. Il grafo (B) mostra invece i valori di betweenness.

Come esempio viene mostrata la figura 1.2(A) che mostra la Closeness centrality del grafo (A). Nelle etichette dei nodi del grafo 1.2(A) vengono forniti i valori della sommatoria delle distanze $d(u, v)$ in modo da non complicare la figura; i nodi, sempre in figura 1.2(A), sono colorati in base ad una scala che aiuta a distinguere l'importanza del nodo secondo la Closeness centrality. La scala varia dal colore rosso al colore verde. Il colore rosso viene assegnato ai nodi con bassa importanza mentre il verde ai nodi con alta importanza. Per un approfondimento ed una generalizzazione a grafi sconnessi si vedano [5] e [6] .

### 1.2.4 Betweenness Centrality

La Betweenness centrality si basa sull'osservazione che se un nodo fa parte di molti cammini minimi allora è un nodo che riveste una posizione importante nel dominio del sistema che stiamo rappresentando come rete. Per esempio, in una rete sociale come twitter, dove i nodi sono gli utenti di twitter e gli archi sono la possibilità di scambiarsi informazioni, rimuovere un nodo con alto valore di betweenness significa rallentare il flusso di informazioni fra i nodi e in alcuni casi può significare la creazione di componenti sconnesse. Un secondo esempio può essere la rete formata dai raccordi stradali come nodi e dalle strade come archi. È chiaro che in una rete del genere i raccordi che risiedono sulla maggior parte dei cammini minimi occupano una posizione strategica all'interno della rete. La conoscenza di tali punti può essere utile in vari scenari, per esempio ogni volta che si verificano degli esodi verso un unico luogo. I nodi con alto valore di Betweenness possono essere visti anche come i punti deboli della rete in quanto l'eliminazione di tali punti può portare a rendere la rete sconnessa. Prima di definire in maniera formale la Betweenness centrality è necessario definire cosa sia la *dipendenza di coppia* $\delta_{st}(v)$.

**Def.** *Dato un grafo $G = (V, E)$ (fortemente) connesso e pesato, sia $\sigma_{st}$ il numero di cammini minimi fra i vertici $s, t \in V$ e sia $\sigma_{st}(v)$ il numero di cammini minimi fra $s$ e $t$ che passano per il nodo $v$.*
*Si definisce la dipendenza di coppia $\delta_{st}(v)$ come la frazione dei cammini minimi fra $s$ e $t$ che includono $v$. Formalmente:*

$$\delta_{st}(v) \stackrel{def}{=} \frac{\sigma_{st}(v)}{\sigma_{st}}$$

Tramite la dipendenza di coppia è possibile capire l'importanza del nodo $v$ nelle comunicazioni da $s$ a $t$. La betweenness centrality $BC(v)$ di un nodo $v$ in un grafo $G = (V, E)$ consiste nel calcolare la dipendenza di coppia per tutte le possibili coppie di vertici $s, t \in V$.

**Def.** *Dato un grafo $G = (V, E)$ (fortemente) connesso e pesato, la betweenness centrality BC(v) di un nodo $v \in V$ è definita come*

$$BC(v) \stackrel{def}{=} \sum_{s \neq v \neq t \in V} \delta_{st}(v)$$

Come già accennato, nell'esempio precedente, la Betweenness centrality indica quanto controllo possiede un vertice sul flusso di informazioni nelle reti sociali. La Betweenness centrality negli ultimi anni si è rivelata molto utile in diversi campi, per esempio la biologia, le reti di trasporti, il marketing, la logistica.

L'immagine 1.2 mostra, per un grafo di 30 nodi, sia i valori di closeness centrality (A) sia i valori di betweenness centrality (B). Confrontando i due grafi si riesce a capire la differenza fra le due metriche. La Closeness centrality fornisce una stima della distanza media fra un nodo e i restanti nodi del grafo, dove la definizione di distanza dipende dal sistema modellato. La Betweenness centrality fornisce una stima della capacità di un nodo di manipolare l'efficienza della rete. Nelle reti sociali, per esempio, se gli archi rappresentano lo scambio di informazioni, allora la Closeness centrality classifica i nodi in basa alla loro capacità potenziale di far diffondere l'informazione nell'intero grafo minimizzando il numero medio di passaggi. Mentre la Betweenness centrality, nelle reti sociali i cui archi rappresentano lo scambio d'informazioni, classifica i nodi in base alla loro capacità di influire sul flusso di informazioni, per esempio rallentandolo o alterandolo.

## 1.3 Reti complesse

Una rete complessa è una rete che modella un fenomeno reale che coinvolge delle entità che interagiscono fra loro. Le entità coinvolte hanno comportamenti dinamici basati su scelte inviduali e condizionate dalle interazioni con le altre entità. Ciò che le rende complesse è che da queste scelte emergono dei comportamenti globali che è impossibile capire o predire semplicemente osservando il comportamento delle singole entità e delle loro interazioni. Modellare i fenomeni tramite reti complesse ha permesso la scoperta di somiglianze comportamentali fra fenomeni che, apparentemente, non hanno nulla in comune.

In questo paragrafo vengono introdotti, seguendo un filo cronologico, i modelli a cui si fa riferimento quando si tenta di modellare le reti complesse.

### 1.3.1 Erdös-Rényi

Una rete di Erdös-Rényi anche detta rete Random è una tipologia di rete introdotta a fine anni '50 [7].

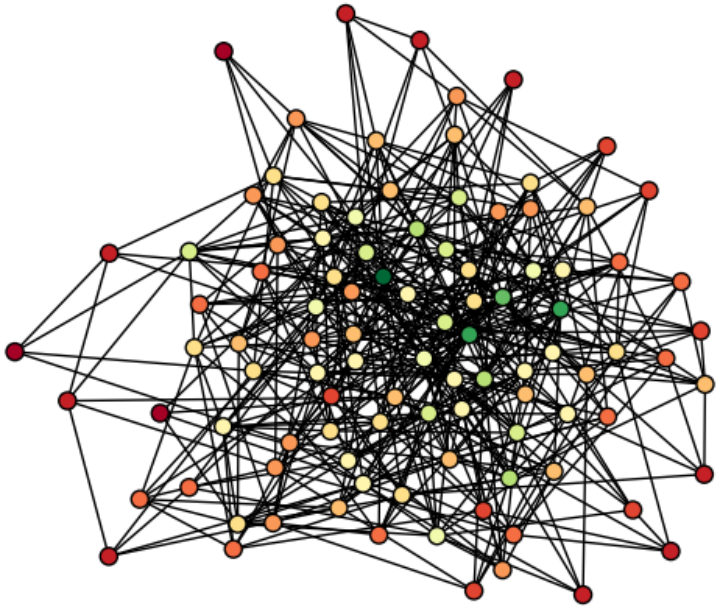

Figura 1.3: L'immagine mostra un grafo generato mediante l'algoritmo di Erdös-Rényi. Il grafo consta di 100 nodi e la probabilità dell'arco è 0.1.

**Def.** *Un **modello Erdös-Rényi** $G_{N,p}$ è un grafo G = (V,E) non orientato con $N = |V|$ vertici in cui ogni coppia di nodi distinti $u, v \in V$ è connessa da un arco con probabilità p.*

Un grafo casuale mostra alcune interessanti caratteristiche:

- La **distribuzione del grado dei nodi** segue una distribuzione binomiale. Infatti la probabilità che un nodo $v$ abbia grado $k$ è data da:

  $$P[\delta(v) = k] = \binom{N-1}{k} p^k (1-p)^{N-1-k}$$

  Si può quindi dimostrare che il grado medio è $\bar{\delta} = \sum_{k=1}^{N-1}(kP[\delta(v) = k]) = Np$. Senza inoltrarmi in ulteriori dettagli è sufficiente notare che all'aumentare della dimensione del grafo il grado medio dei nodi si avvicinerà a $\bar{\delta}$ e la rete assumerà via via una struttura sempre più regolare.

- Il coefficiente di clustering di un grafo $G_{N,p}$ è uguale a $p$.

- La distanza media fra i nodi è bassa.

- La maggior parte dei nodi fa parte della stessa componente detta *componente gigante*. Inoltre, è interessante notare che la probabilità $p$ è direttamente proporzionale alla dimensione della componente gigante, infatti la componente gigante di un grafo $G_{N,0.005}$ è composta da circa il 25% dei nodi della rete mentre in un grafo $G_{N,0.015}$ è composta da circa il 100% dei nodi. Questa caratteristica è spesso presente anche nelle reti reali.

### 1.3.2 Small world

La nascita del modello **Small World** può essere fatta risalire agli studi di Stanley Milgran, uno psicologo di Harvard. Milgran era interessato a scoprire quale fosse la distanza media, in una rete sociale, di una qualsiasi coppia di persone scelte a caso. Milgran scelse varie persone a caso in diversi stati americani e gli chiese di far arrivare una lettera ad una persona scelta da Milgran a Boston, Massachusetts. L'unico vincolo risiedeva nelle modalità con cui la lettera avrebbe dovuto raggiungere la persona scelta. Infatti era possibile spedire o consegnare il messaggio solo a persone che si conoscevano di persona, incaricandole di fare altrettanto, nel tentativo di raggiungere la persona indicata a Boston. Milgran definì la distanza fra persone come la

lunghezza di questa catena di conoscenti. Sorprendentemente gli esperimenti che condusse rivelarono che la distanza media fra due persone scelte a caso era molto bassa. Questi risultati portarono alla nota frase "sei gradi di separazione" basata sul risultato medio che ottenne Milgran, $5,5$. Per una descrizione approfondita degli esperimenti e dei risultati di milgran si faccia riferimento a [8].

**Def.** *Si definisce una rete small-world come un grafo G=(V,E) dove la distanza media L fra una qualsiasi coppia di nodi $u, v \in V$ scelta a caso cresce proporzionalmente al logaritmo del numero di nodi n nella rete. Ossia:*

$$L \propto \log(n)$$

*Questo significa che la maggior parte dei nodi non è necessariamente adiacente ma per ogni coppia di nodi esiste un cammino relativamente breve che li unisce.*

Il primo modello di rete *small world* fu introdotto nel 1998 da Watts e Strogatz [9]. Tale modello nasce dall'osservazione che molte reti reali mostrano un alto coefficiente di clustering mantenendo però le caratteristiche delle reti Erdös-Rényi.

Nel loro lavoro Watts e Strogatz proposero un algoritmo per generare reti che esibissero tali caratteristiche. Inoltre mostrarono che la maggior parte dei vertici si raggruppa fra loro e che una buona parte di questi vertici ha anche un arco ad un vertice "lontano". Tali archi vengono definiti "archi deboli" (weak links) e giocano un importante ruolo nella rete [10]. L'algoritmo seguente genera un grafo Small-World.

**Algoritmo** *Si consideri un insieme di di n vertici $\{v_1, v_2, ..., v_n\}$ ed un numero intero k. Si scelgano n e k in modo tale che $n \gg k \gg \ln(n) \gg 1$, così da garantire che il grafo sia sparso, ossia che possieda relativamente pochi archi. L'algoritmo consta in due fasi:*

1. *Si dispongano gli n vertici ad anello. Si connetta ogni vertice con i suoi successivi $k/2$ vicini in senso orario ed i $k/2$ vicini in senso anti-orario. Ottenendo un grafo $G = (V, E)$ di n vertici e kn archi.*

2. *Con probabiltà p, si sostituisca ogni arco $\{u, v\}$ con un arco $\{u, w\}$ dove $w \in V \backslash \{u, v\}$.*

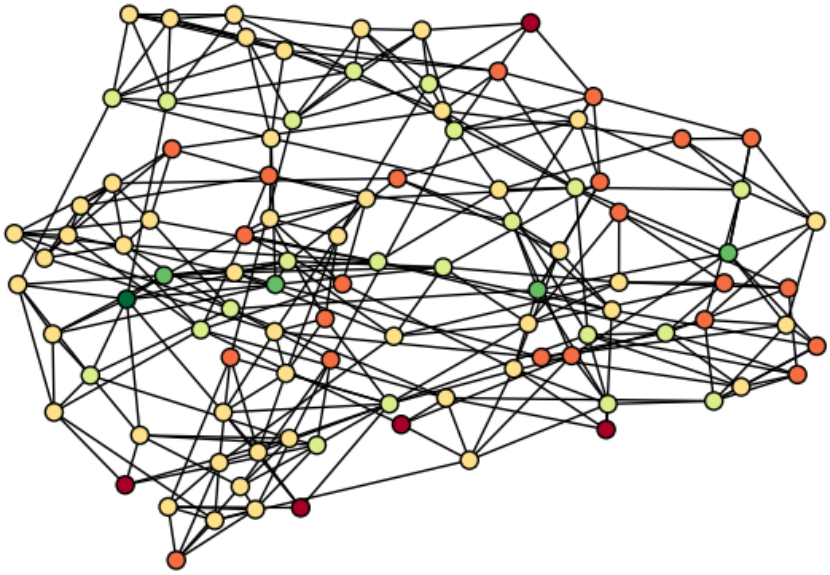

Figura 1.4: L'immagine mostra un grafo generato mediante l'algoritmo proposto da Watts e Strogatz. Il grafo consta di 100 nodi, il numero k è uguale a 7 e la probabilità p è 0.2.

L'algoritmo mostrato, che corrisponde all'algoritmo proposto nel lavoro di Watts e Strogatz, riesce ad unire la maggior parte delle proprietà del modello Erdös-Rényi ottenendo inoltre un alto coefficiente di clustering. Purtroppo il modello proposto da Watts e Strogatz non riesce, al crescere della dimensione della rete, a mantenere una delle caratteristiche principali delle reti small world, ossia una bassa distanza fra i nodi.

### 1.3.3 Reti Scale-free

Nonostante il grande contributo portato dal lavoro di Watts e Strogatz il loro modello fallisce nel tentativo di catturare alcune proprietà di molte reti reali. Nel 1999 il lavoro di Barabàsi e Albert [11] porta alla luce una proprietà fondamentale di molte reti reali: **la distribuzione del grado segue una legge di potenza** (power law).

In un grafo $G(V, E)$, la probabilità che un generico nodo $v \in V$ abbia grado $\delta(v) = k$ è proporzionale a $(\frac{1}{k})^{\alpha}$ con $\alpha > 1$. Ossia:

$$P[\delta(v) = k] \propto \left(\frac{1}{k}\right)^{\alpha}$$

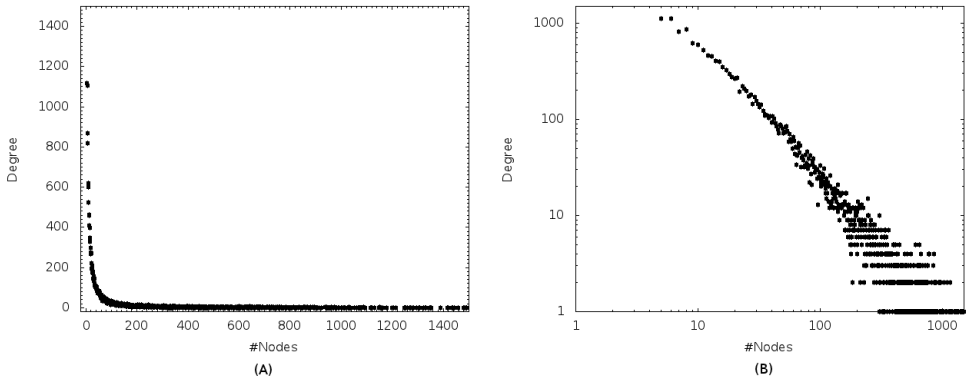


Figura 1.5: I grafici mostrano la distribuzione del grado dei nodi dello stesso grafo scale-free (28.250 nodi e 692.668 archi). Il grafico (A) mostra la distribuzione in scala lineare mentre il grafico (B) in scala logaritmica.

ed $\alpha$ è detto esponente di scala. Nella figura 1.5 viene mostrata la distribuzione del grado per una rete scale free mentre nella figura 1.6 viene mostrata una rete scale-free generata con l'algoritmo proposto da Barabàsi. Come è possibile intuire dai grafici, nelle reti scale free vi sono pochi vertici, gli hub, che hanno un grado elevato mentre la maggior parte possiede un grado basso. Questa osservazione si vede bene osservando i colori dei nodi nel grafo mostrato nella figura 1.6. Infatti i nodi sono colorati in base ad una scala che aiuta a distinguere il grado del nodo. La scala varia dal colore rosso al colore verde. Il colore rosso viene assegnato ai nodi con grado ridotto mentre il verde ai nodi con grado elevato. Il numero di nodi di colore verde è molto basso mentre i restanti nodi, per la maggior parte, tendono al rosso. Per spiegare questo fenomeno vi sono stati vari lavori tra cui [12], [13] che come Barabàsi si basano sul concetto di **preferential attachment**.

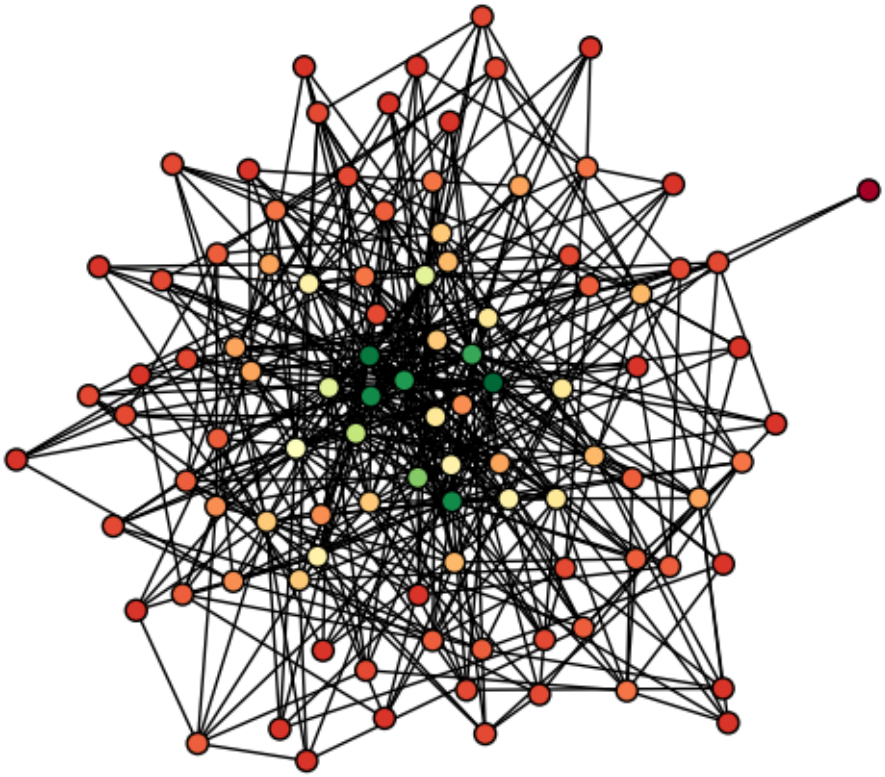

Figura 1.6: L'immagine mostra un grafo generato mediante l'algoritmo proposto da Barabàsi. Il grafo consta di 100 nodi. I nodi sono colorati mediante una scala che ne risalta il grado. Il colore rosso viene assegnato ai nodi con grado ridotto mentre il verde ai nodi con grado elevato.

# Capitolo 2

# Modelli di calcolo

In questo capitolo si illustreranno i modelli di calcolo presi in considerazione in questo lavoro di tesi.

## 2.1 Il modello parallelo a memoria condivisa

Il termine programmazione parallela si riferisce alla scrittura di algoritmi che sfruttano la capacità dei calcolatori di effettuare più operazioni in parallelo. Per esempio sfruttando le architetture multicore e multi-processore.

Lo scopo di questo modello è diminuire il tempo di esecuzione di un determinato algoritmo suddividendo il problema in sottoproblemi e affidando la risoluzione dei sottoproblemi ad unità di calcolo distinte. Generalmente le unità di calcolo condividono lo stesso spazio di indirizzamento e in questi casi si fa riferimento alla programmazione parallela a memoria condivisa. Uno schema di architettura parallela a memoria condivisa è mostrato in figura 2.1.
Nonostante vi siano problemi intrinsecamente sequenziali che non ottengono, per propria natura, miglioramenti dalla parallelizzazione vi sono anche molti problemi che possono essere risolti in un tempo sensibilmente ridotto grazie a tale approccio.

Oltre ai vantaggi vi sono anche diversi problemi introdotti dall'approccio parallelo che non erano presenti nel sequenziale come l'accesso esclusivo a risorse condivise e il bilanciamento del carico di lavoro fra le diverse entità.

Possiamo suddividere le forme di parallelismo in due categorie:

- **Parallelismo sui dati**: In questa forma di parallelismo ogni entità svolge lo stesso compito su porzioni distinte dei dati in ingresso.

- **Parallelismo sui compiti**: Questa forma di parallelismo consiste nel far svolgere compiti diversi alle diverse entità sugli stessi dati.

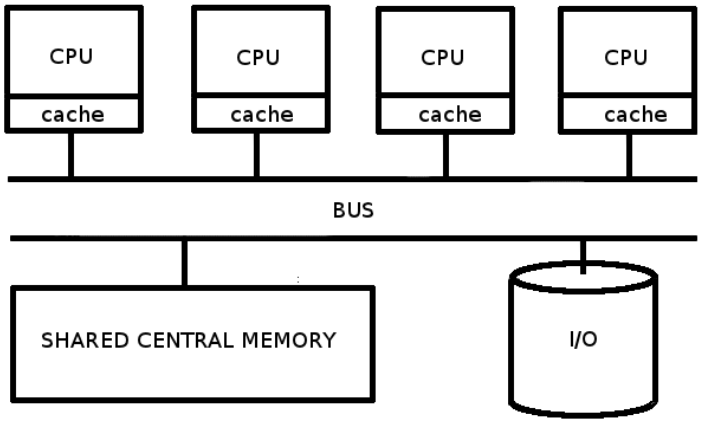


Figura 2.1: Schema di un'architettura parallela a memoria condivisa.

Ovviamente vi sono varie implementazioni per entrambe le forme di parallelismo. In particolare, OpenMP [14] è un esempio di API per la programmazione parallela a memoria condivisa con cui è possibile ottenere entrambe le forme di parallelismo.

Vi è un ulteriore distinzione fra i modelli paralleli, quelli sincroni come il modello PRAM e quelli asincroni come il modello concorrente.
Negli algoritmi paralleli a memoria condivisa, alla classica analisi di complessità degli algoritmi sequenziali viene aggiunta un'ulteriore risorsa oltre a tempo e spazio ed è il numero di processori. Infatti un algoritmo parallelo a memoria condivisa, nelle analisi asintotiche, è considerato efficiente se riduce il costo computazionale di ordini di grandezza sfruttando un numero maggiore di processori. Per esempio, un algoritmo parallelo a memoria condivisa che effettua la sommatoria di una sequenza di n numeri arbitrari in tempo $O(\log n)$ sfruttando $O(\frac{n}{\log n})$ processori. In particolare il modello PRAM viene ampiamente utilizzato per mostrare un'analisi del costo computazionale degli algoritmi paralleli a memoria condivisa. Per approfondimenti si faccia riferimento a [15] e [16].

## 2.2 Il modello distribuito

In generale un sistema si definisce distribuito se è composto da diversi componenti distinti che cooperano nella risoluzione di un problema computazionale. Seguendo questa definizione non è chiaro distinguere fra un sistema distribuito ed un sistema parallelo a memoria condivisa spiegato nella sezione 2.1. Questa difficoltà risiede nelle diverse accezioni che ha assunto il modello distribuito nel corso degli anni. Nonostante l'assenza di una definizione formale di sistema distribuito che ne segni la differenza dai sistemi paralleli, vi sono alcune caratteristiche, comunemente accettate, che un sistema distribuito deve avere. Le caratteristiche di un sistema distribuito sono:

- i nodi di calcolo che compongono il sistema sono unità di calcolo fisicamente distinte;
- i nodi di calcolo cooperano nella risoluzione di un problema computazionale comune;
- le comunicazione fra i nodi di calcolo possono avvenire solo tramite messaggi mediante una rete di comunicazione.

Lo schema dell'architettura di un sistema distribuito è mostrato in figura 2.2.

Data la particolarità degli algoritmi analizzati in questo lavoro di tesi che sono pensati per essere eseguiti in cluster chiusi[1] viene assunto che l'architettura della rete e dei nodi di calcolo sia conosciuta a priori.

Nel capitolo 3 verranno mostrati due approcci distribuiti alla risoluzione di problemi su grafo.

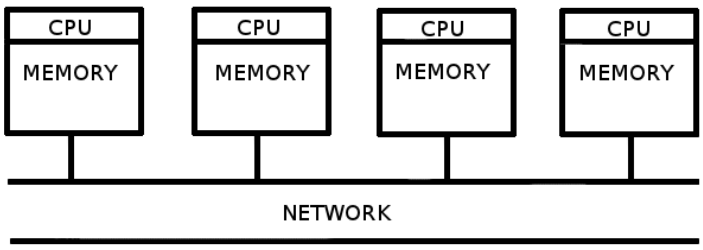


Figura 2.2: Schema di un'architettura parallela a memoria distribuita.

[1]Tutti i calcolatori del cluster sono gestiti da un'unica organizzazione.

Il modello distribuito di riferimento si appoggia sulle librerie di Message Passing [17] ed è il più comune modello distribuito per quanto riguarda il supercalcolo.

Gli algoritmi per i modelli di calcolo distribuiti sono detti algoritmi distribuiti. La metrica di maggiore interesse negli algoritmi distribuiti non è il costo computazionale ma il numero di messaggi, anch'esso spesso valutato asintoticamente.

Vi sono inoltre modelli distribuiti come l'UPC [18], che forniscono un'astrazione sulla memoria distribuita fornendo al programmatore un'illusione di memoria globale condivisa fra i vari nodi fisici. Tale astrazione può essere fornita anche via hardware. Per una panoramica più approfondita si può fare riferimento a [19].

# Capitolo 3

# Algoritmi paralleli su grafi

In questo capitolo si illustreranno i principali problemi che si presentano nella definizione e implementazione di algoritmi paralleli su grafi. Verranno illustrati i problemi inerenti agli algoritmi su grafi ponendo l'enfasi sul calcolo parallelo e distribuito.

## 3.1 Algoritmi su grafo

I problemi su grafi possiedono delle caratteristiche intrinseche che rendono difficile la parallelizzazione del calcolo. In particolare vi sono quattro problemi principali:

- **Il calcolo è guidato dai dati**. Buona parte degli algoritmi su grafo calcola proprietà sui nodi. Per calcolare queste proprietà è necessaria la conoscenza di dati relativi ad altri nodi. Dato che la topologia del grafo non è nota a priori, non è possibile stabilire a priori quali nodi serviranno nel corso dell'esecuzione dell'algoritmo ad un determinato processore. Questo rende molto difficile ottenere del parallelismo sui dati.

- **I problemi non sono strutturati**. Un altro punto che rende difficile ottenere del parallelismo sui dati è la mancanza di una struttura regolare nei grafi. L'irregolarità nella struttura del grafo rende molto difficile il partizionamento dei dati. Quindi la scalabilità dell'algoritmo può essere compromessa da un pessimo bilanciamento del carico di lavoro dei diversi processori.

- **Bassa località**. Dato che i grafi vengono usati per rappresentare entità e relazioni fra entità e queste relazioni possono essere irregolari e non strutturate si ha che il calcolo e l'accesso ai dati tende ad avere bassa località.
- **Alto tasso di accesso ai dati rispetto al calcolo**: I problemi su grafi spesso tendono a focalizzare l'attenzione sull'esplorazione del grafo piuttosto che all'effettuare calcoli e questo porta ad un alto tasso di letture in memoria che unito alla bassa località porta ad avere un tempo d'esecuzione dominato dagli accessi in memoria. In particolare, nel caso di partizione dei dati su diversi processori si traduce in tempo di comunicazione.

Oltre ai problemi legati alla natura delle reti bisogna considerare anche i problemi legati alla tecnologia disponibile. Scegliere di mantenere lo spazio di memoria condiviso semplifica la programmazione in quanto non vi è la necessità di comunicare ed è più facile fornire ai processi una visione globale coerente. Al tempo stesso, in uno scenario del genere, i vari processi potrebbero avere la necessità di accedere simultaneamente alle stesse locazioni di memoria, sia in lettura che in scrittura, e questo riduce notevolmente le prestazioni. D'altra parte, come abbiamo visto, non è sempre possibile suddividere i dati in modo che i processi non interferiscano l'uno con l'altro. Per esempio, il calcolo di vari cammini minimi, come avviene per gli indici di centralità, può essere facilmente parallelizzato sui compiti ma l'impossibilità di sapere a priori quali dati serviranno a quale processo rende impraticabile il parallelismo sui dati.

Questa difficoltà nell'ottenere il parallelismo sui dati è il principale ostacolo alla parallelizzazione sia su architetture a memoria condivisa che distribuita. Nelle due sezioni successive si mostrano i due approcci proposti per il calcolo di algoritmi su grafi.

## 3.2 Replicazione dei dati in ingresso

Un primo approccio consiste nel mantenere una copia completa del grafo nella memoria centrale di ogni nodo coinvolto nel calcolo come schematizzato nell'immagine 3.1. La replicazione dei dati in ingresso ha lo scopo di semplificare il parallelismo sui dati.

Data l'impossibilità di conoscere a priori quali parti del grafo saranno necessarie ad un dato calcolatore, questo primo approccio risolve il problema

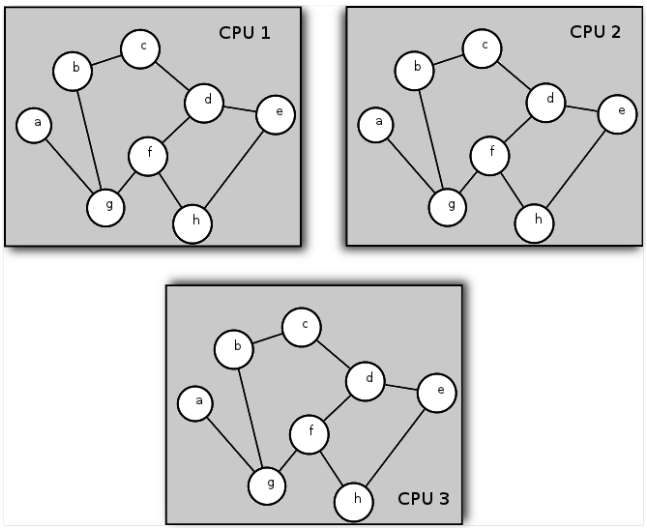


Figura 3.1: L'immagine mostra la replicazione dell'intero grafo per ogni processore coinvolto nel calcolo.

distribuendo l'intero grafo a tutti i nodi di calcolo. La distribuzione dell'intero grafo risulta porta diversi vantaggi in vari problemi su grafi. I problemi che traggono il maggior beneficio dalla replicazione dell'intero grafo sono tutti i problemi in cui la parallelizzazione sui compiti risulta facile[1]. Per esempio, i problemi che riguardano il calcolo di varie visite su grafo da nodi sorgente distinti possono essere efficacemente distribuite su diversi calcolatori. I calcolatori possono effettuare le visite su grafo limitando le comunicazioni solo al termine della computazione grazie alla visione globale del grafo. Per esempio, considerando lo scenario in figura 3.1, il calcolatore 1 può effettuare il calcolo del cammino minimo dal nodo $a$ al nodo $f$ senza effettuare alcuna comunicazione con gli altri calcolatori. Contemporaneamente il calcolatore 3 può effettuare il calcolo del cammino minimo dal nodo $f$ al nodo $e$, anch'egli senza dover effettuare alcuna comunicazione. L'esempio mostra come sia possibile calcolare i cammini minimi fra tutte le coppie senza effettuare comunicazioni oltre a quelle necessarie a suddividere il lavoro. La maggior parte degli indici di centralità ricade in questo tipo di problemi.

I vantaggi della replicazione dell'intero grafo sono la visione globale da parte di tutti i calcolatori coinvolti e di conseguenza la drastica riduzione delle

[1]Si intende che sia facile trovare un algoritmo che divida i compiti fra i vari calcolatori. Dividere i compiti in modo efficiente raramente risulta facile.

comunicazioni. Nell'esempio di prima, le fasi di comunicazione si riducono alla distribuzione del grafo ed alla distribuzione del risultato. Gli svantaggi sono l'alta necessità di memoria centrale richiesta ed inoltre, riprendendo l'esempio dei cammini minimi, è doveroso notare che avere tutte le strutture dati replicate senza scambiarsi informazioni durante la fase di calcolo può portare vari calcolatori ad effettuare calcoli già effettuati da altri calcolatori.

A dimostrazione dell'efficacia della distribuzione dell'intero grafo fornirò, nei prossimi capitoli, un'analisi prestazionale dell'algoritmo che calcola la Betweenness centrality.

## 3.3 Partizione dei dati in ingresso

Negli algoritmi che lavorano su grafi, partizionare i dati in ingresso significa dividere l'insieme dei nodi del grafo in diversi sottoinsiemi e assegnare ad ogni calcolatore uno di questi sottoinsiemi come mostrato in figura 3.2. Un approccio del genere tenta di ridurre l'utilizzo di memoria sui singoli calcolatori rendendo così possibile l'analisi di grafi di dimensioni superiori a quelle memorizzabili in memoria centrale di una singola macchina.

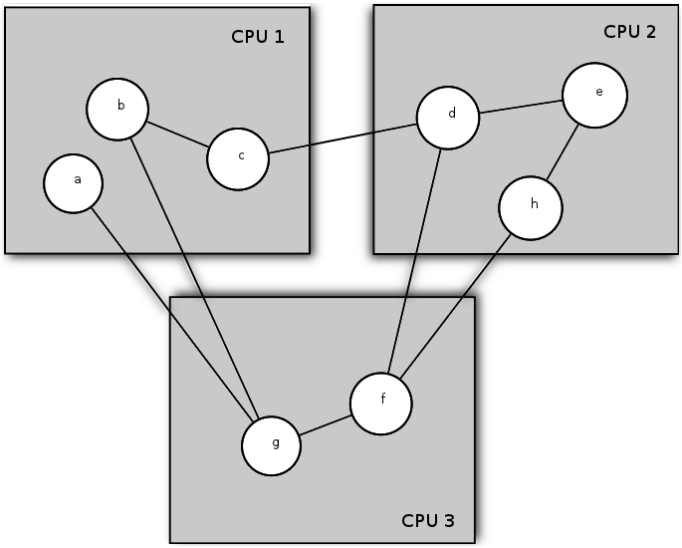


Figura 3.2: Partizione del grafo su diverse macchine. Ogni macchina possiede una parte del grafo.

Confrontando la figura 3.1, che mostra la replicazione dell'intero grafo nella memoria di ogni calcolatore coinvolto, con la figura 3.2 dove lo stesso

grafo viene partizionato e distribuito fra i calcolatori, si notano facilmente le differenze fra i due approcci. Sebbene il secondo approccio offra notevoli risparmi sull'utilizzo di memoria centrale, comporta anche un notevole aumento delle comunicazioni. Infatti ogni qualvolta un calcolatore necessiti di informazioni relative a parti del grafo non presenti nella propria memoria è costretto a inviare messaggi agli altri calcolatori.
Nei sistemi distribuiti riuscire a mantenere il numero di messaggi inviati il più basso possibile è uno dei principali scopi. L'invio di un messaggio per ottenere dati e la conseguente impossibilità di portare avanti il calcolo aumenta considerevolmente il tempo d'esecuzione ed in alcuni casi il tempo impiegato a comunicare diventa maggiore del tempo impiegato a svolgere effettivamente il calcolo interessato. Basti pensare che la velocità della memoria centrale può arrivare sino a circa 400Gb/s teorici mentre il miglior canale di comunicazione via rete disponibile raggiunge circa i 100Gb/s teorici. Nella maggior parte dei casi, salvo centri di supercalcolo, la velocità del canale di comunicazione spazia dai 50Mb/s al 1Gb/s mentre la memoria centrale mantiene velocità nell'ordine delle centinaia di Gigabit per secondo.

Uno dei modelli più usati nel calcolo distribuito è il **Bulk Synchronous Parallel** (BSP). Il calcolo parallelo nel modello BSP è organizzato in macropassi, ognuno di questi consta di una fase di calcolo seguita da una fase di comunicazione. Durante la fase di calcolo tutti i processori lavorano esclusivamente sui dati locali e non vi è alcuna forma di comunicazione. Durante la fase di comunicazione tutti i processi si scambiano i dati necessari. I messaggi inviati nella fase di comunicazione verranno ricevuti nella fase di calcolo del macro passo successivo.
Questa modello di calcolo viene applicato con successo in vari calcoli scientifici ma per le analisi su grafo pone vincoli troppo stretti; vedremo più avanti un'applicazione meno restrittiva del BSP.

Infine si rende noto che esistono diversi approcci alla distribuzione dei nodi fra i vari calcolatori negli algoritmi che si basano sulla distribuzione del grafo. Gli approci più usati risultano essere la distribuzione random e la distribuzione a blocchi. In breve se ne elencano le caratteristiche.

- **Distribuzione Random**: la distribuzione random consiste nello scorrere i nodi del grafo e scegliere mediante una certa probabilità $p$ a quale unità di calcolo assegnare il nodo. Il valore della probabilità $p$ è scelto arbitrariamente dal programmatore.

- **Distribuzione a blocchi**: I nodi del grafo vengono suddivisi in un numero di insiemi (blocchi) pari al numero di unità di calcolo nel sistema. I blocchi possono essere tutti della stessa dimensione oppure di dimensioni differenti.

# Capitolo 4

# Analisi di algoritmi paralleli su grafi

In questo capitolo si fornirà una panoramica sugli algoritmi che calcolano la Betweenness centrality e sulle implementazioni disponibili. In particolare si offrirà, una panoramica degli strumenti e delle librerie utilizzate per effettuare l'analisi e l'esecuzione degli algiritmi; una panoramica sugli algoritmi disponibili per il calcolo della Betweenness centrality; un'analisi prestazionale delle implementazioni degli algoritmi che calcolano la Betweenness centrality.

## 4.1 Strumenti e librerie

In questa sezione si fornisce una panoramica sugli strumenti e le librerie utilizzate durante lo svolgimento della tesi. Dalla libreria per il calcolo parallelo a memoria condivisa SNAP, alla libreria per il calcolo parallelo a memoria distribuita PBGL sino allo strumento di analisi prestazionale Scalasca. Infine si introduce LANA, un'applicazione basata sulla libreria PBGL, di cui ho contribuito all'implementazione durante questo lavoro di tesi. LANA consente l'esecuzione di algoritmi paralleli di analisi su grafi.

### 4.1.1 SNAP

Il framework SNAP, Small-world Network Analysis and Partitioning [20], sviluppato presso il Berkeley Lab, in California, è una piattaforma open-source per l'analisi di reti complesse che offre vari algoritmi di analisi delle reti quali quelli relativi agli indici di centralità. L'intero framework è scritto in

linguaggio C ed è progettato per sfruttare le architetture parallele multi-core e multi-processore a memoria condivisa mediante l'utilizzo dei thread posix e di OpenMP. Questo framework può essere utilizzato sia come strumento finito per l'analisi, dove l'utente finale può semplicemente limitarsi ad utilizzarlo a "scatola chiusa", sia come una ricca libreria di strutture dati e algoritmi sfruttabili per scrivere i propri programmi. Nel paragrafo 4.2.3 verrà fornita un'analisi prestazionale dell'implementazione della betweenness centrality in questo framework.

### 4.1.2 Parallel Boost Graph Library

La libreria PBGL [21] è una libreria open-source, scritta in linguaggio C++, che fornisce varie strutture e algoritmi paralleli su grafo. Questa libreria si basa sulla propria versione sequenziale, la Boost Graph Library. A differenza della versione sequenziale, che è un progetto famoso supportato da un'ampia comunità di sviluppatori ed utilizzatori, la versione parallela è un progetto relativamente giovane e di ristretta applicazione, non possiede una vasta comunità e risulta essere scarsamente documentata.

L'architettura della PBGL si basa sul paradigma di programmazione generica ossia tentando il più possibile di astrarre l'implementazione degli algoritmi paralleli dall'architettura sottostante. Quindi un qualsiasi algoritmo parallelo implementato avvalendosi della PBGL può essere eseguito sfruttando strutture dati o mezzi di comunicazione differenti a patto che la struttura e il canale soddisfino certi requisiti. La programmazione generica non è comunque sempre applicabile efficacemente, vi sono casi infatti in cui per una scrittura ottimizzata dell'algoritmo è necessario fare assunzioni sull'architettura sottostante. Questi tipi di ottimizzazione non sono facili da ottenere in quanto non in linea con l'approccio di programmazione adottato nella PBGL.

La PBGL utilizza il modello di calcolo BSP visto nel paragrafo 3.3 ma rilassandone i vincoli e aggiungendo la possibilità di inviare messaggi asincroni durante la fase di calcolo. Nel paragrafo 4.2.3 viene fornita un'analisi prestazionale di due distinte versioni della betwenness centrality.

### 4.1.3 Scalasca

Scalasca [22] è uno strumento software rilasciato sotto licenza BSD e sviluppato dal centro di supercalcolo *Forschungszentrum Jülich* e dal laboratorio per la programmazione parallela del *German Research School for Simulation Sciences*. Lo scopo di scalasca è di analizzare il comportamento di program-

mi paralleli durante la loro esecuzione. Le analisi identificano i potenziali colli di bottiglia per quanto concerne la comunicazione e la sincronizzazione, ed offre alcuni indizi per identificarne la causa. Scalasca è stato progettato per analizzare principalmente applicazioni scientifiche basate su OpenMP ed MPI o una combinazione dei due approcci in applicazioni scritte nei linguaggi C, C++ e Fortran. Le architetture su cui è possibile eseguire Scalasca spaziano dai grandi cluster dei centri di supercalcolo sino a cluster più comuni creati utilizzando componenti di largo mercato.

Il processo di analisi tramite Scalasca consta di tre fasi:

**Instrumentazione del programma** Per permettere a Scalasca di effettuare le analisi è necessario che vengano indicati nel codice del programma da analizzare i punti a cui si è interessati. Vi sono tre modalità principali per indicare i punti d'interesse:

- *Automatica*: In questa modalità non è necessario indicare nulla, Scalasca inserisce automaticamente nel sorgente le linee di codice necessarie.
- *Semi-automatica*: Questa modalità consta nell'inserire direttive al preprocessore prima e dopo della zona del programma che si intende analizzare con Scalasca. È possibile analizzare diverse zone contemporaneamente.
- *Manuale*: In questa modalità è necessario inserire direttamente le chiamate di funzione alle librerie fornite da Scalasca. Tra le tre modalità è la più intrusiva.

Una volta deciso quale modalità adottare è sufficiente effettuare una fase di pre-processamento dei sorgenti, per quanto riguarda le prime due fasi, per poi compilare come di consueto. Queste due fasi vengono svolte automaticamente da Scalasca.

**Esecuzione del programma instrumentato** Finita la fase di instrumentazione del programma è sufficiente eseguire Scalasca passando come parametro il comando che esegue il programma come di consueto. In questa fase verranno raccolti i dati d'interesse da Scalasca e salvati su disco.

**Visualizzazione e analisi dei dati** Terminata la raccolta dei dati vi è la fase di analisi da parte di Scalasca in modo da evidenziare in un file di resoconto i possibili problemi del programma. Al termine di questa fase è possibile visualizzare, tramite interfaccia grafica, un resoconto dettagliato di

tutte le chiamate a funzione effettuate nelle zone analizzate e i relativi indici prestazionali.

In questo lavoro Scalasca è stato utilizzato per verificare quali parti dell'algoritmo richiedono maggior tempo e se vi fossero o meno evidenti problemi nell'implementazione degli algoritmi.

### 4.1.4 LANA

LANA, Large-scale Network Analyzer, è un software che implementa diversi algoritmi paralleli che calcolano proprietà su grafi. Gli algoritmi finora implementati si appoggiano sulla libreria PBGL. Inoltre fornisce la possibilità di generare grafi sintetici e di caricare grafi da file. Essendo l'analisi prestazionale il nostro principale interesse, il codice contiene anche le direttive al preprocessore necessarie per le analisi mediante scalasca e alcune funzioni per il tracciamento delle prestazioni.

Il mio contributo nello sviluppo di LANA riguarda l'integrazione degli algoritmi che calcolano la Betweenness centrality forniti dalla PBGL. L'integrazione degli algoritmi ha comportato anche l'implementazione/integrazione degli strumenti necessari all'analisi prestazionale di tali algoritmi.

Il codice sorgente è liberamente scaricabile, sotto licenza GPLv2, all'indirizzo `http://sigsna.trac.cs.unibo.it/`.

## 4.2 Analisi della Betweenness Centrality

In questa sezione si mostra l'analisi svolta sulle implementazioni degli algoritmi di Betweenness centrality disponibili. L'analisi effettuata si pone l'obbiettivo di mostrare le differenze fra i vari approcci e di evidenziarne le limitazioni ed i punti di forza.

In primo luogo verranno presentati gli algoritmi per il calcolo della Betweenness centrality disponibili, dalla nota versione sequenziale alla versione distribuita con grafo partizionato. In secondo luogo si mostreranno le analisi svolte ed i risultati ottenuti.

### 4.2.1 Brandes Betweenness centrality

L'algoritmo di Brandes [23] mostra come calcolare i valori di centralità di tutti i vertici di un grafo nello stesso tempo asintotico di un calcolo SSSP[1] su $n$ vertici.

Per spiegare l'algoritmo di Brandes è necessario introdurre alcune nuove notazioni. Dato un grafo $G = (V, E)$ non orientato e connesso, si definisce la *dipendenza*, $\delta_{s*}(v)$, di un vertice sorgente $s \in V$ su un vertice $v \in V$ come:

$$\delta_{s*}(v) \stackrel{def}{=} \sum_{t \neq s \neq v \in V} \delta_{st}(v)$$

Data tale definizione, il valore di Betweenness centrality $BC(v)$ del nodo $v$ può essere espresso come:

$$BC(v) \stackrel{def}{=} \sum_{s \neq v \in V} \delta_{s*}(v)$$

Inoltre, si denota l'*insieme dei predecessori* $P_s(v)$ di un vertice $v \in V$ sui cammini minimi che hanno come nodo sorgente il nodo $s$ come:

$$P_s(v) \stackrel{def}{=} \{u \in V \mid (u, v) \in E,\ d(s, v) = d(s, u) + w(u, v)\}$$

Quindi l'insieme dei predecessori corrisponde all'insieme dei nodi $u \in V$ che appartengono ad almeno un cammino minino da $s$ a $v$ e sono adiacenti a $v$.
Brandes ha dimostrato che la *dipendenza* $\delta_{s*}(v)$ soddisfa la relazione di ricorrenza mostrata nel teorema seguente. Sfruttare la relazione di ricorrenza è l'idea chiave dell'algoritmo.

**Teorema** *In un grafo G=(V,E) non orientato, la dipendenza* $\delta_{s*}(v)$ *del nodo* $s \in V$ *su ogni altro nodo* $v \in V$ *obbedisce a:*

$$\delta_{s*}(v) = \sum_{w, v \in P_s(w)} \frac{\sigma_{sv}}{\sigma_{sw}}(1 + \delta_{s*}(w))$$

Date queste notazioni, l'algoritmo consiste nei passi seguenti.

[1] Acronimo di Single Source Shortest Path. Gli SSSP sono una tipologia di problemi su grafo che riguardano il calcolo dei cammini minimi da un dato vertice $s \in V$ verso tutti gli altri vertici $t \in V$ per cui esiste un cammino.

1. Vengono calcolate $n$ SSSP, una per ognuno dei vertici $s \in V$. Durante il calcolo delle SSSP vengono calcolati e memorizzati i sottoinsiemi dei predecessori $P_s(v)$. Gli algoritmi utilizzati per il calcolo dei cammini minimi sono varianti dell'algoritmo *BFS* [15] nel caso di grafi non pesati e l'algoritmo di *Dijkstra* [24] nel caso di grafi pesati.

2. Si calcolano, per ogni nodo $s \in V$, le dipendenze $\delta_{s*}(v)$ per tutti i nodi $v \in V$. Tale calcolo è possibile grazie alle informazioni ottenute al punto 1.

3. Si calcola il valore di betweenness per ogni nodo $v \in V$ effettuando la sommatoria delle dipendenze $\delta_{s*}(v)$.

Per maggiori dettagli sull'algoritmo di Brandes e per la dimostrazione del teorema si faccia riferimento a [23].

### 4.2.2 Betweenness centrality approssimata

Per grafi di grandi dimensioni, il calcolo dei valori esatti di Betweenness centrality è troppo costoso in quanto comporta il calcolo di $n$ SSSP. Ognuno di questi $n$ SSSP contribuisce di una unità al risultato finale, ossia di una delle $n$ dipendenze $\delta_{s*}(v)$. Si definiscono *pivot* i vertici per cui viene risolta l'SSSP. *Brandes* e *Pich* [25] hanno proposto un'approssimazione dei valori di betweenness ottenuta da un piccolo insieme di pivot.

Nel loro lavoro mostrano diversi approcci alla scelta dei pivot mostrando i risultati per svariati tipi di grafo. Fra i vari approcci proposti la scelta completamente casuale dei pivot risulta essere quello che fornisce una migliore approssimazione e una maggiore coerenza fra i risultati sui vari grafi.

*Bader, Kintali, Madduri* e *Mihail* [26] hanno proposto un algoritmo di approssimazione basato su una tecnica di campionamento adattiva. L'approssimazione consiste nel calcolare le SSSP solo per un sottoinsieme dei vertici $v \in V$, esattamente come nell'algoritmo descritto precedentemente. La differenza risiede nella parola "adattivo". Infatti in questo algoritmo il numero di pivot (campioni) viene scelto in base all'informazione ottenuta dai campioni precedenti e varia durante l'esecuzione dell'algoritmo. Gli esperimenti su varie tipologie di grafi mostrano ottimi risultati riducendo il calcolo di un fattore di circa 20.

### 4.2.3 Betweenness centrality parallela

In questa sezione si illustrano gli algoritmi che calcolano la Betweenness centrality sfruttando i diversi modelli spiegati nel capitolo 2 e si mostra l'analisi svolta sulle implementazioni di tali algoritmi. L'analisi svolta vuole verificare se gli approcci distribuiti al calcolo della Betweenness centrality finora proposti e sviluppati risultino essere vantaggiosi rispetto ad un approccio concorrente a memoria condivisa. Si vuole inoltre verificare se il calcolo di un'approssimazione della Betweenness centrality risulti una valida alternativa al calcolo dei valori esatti e quali vantaggi comporta.

Per ottenere una stima prestazionale degli algoritmi analizzati in questa sessione sono stati tracciati due grafici. Il primo grafico mostra lo *speedup* dell'algoritmo, ossia quanto viene ridotto il tempo d'esecuzione nella versione concorrente in proporzione al tempo sequenziale. Lo speedup $S_p$ di un algoritmo parallelo che impiega tempo $T_p$ con $p$ processori, è definito dalla seguente formula:

$$S_p \stackrel{def}{=} \frac{T_1}{T_p}$$

Il secondo grafico, che chiameremo *tempo-processori*, mostra il tempo d'esecuzione in relazione al numero di processori. I due grafici sono ovviamente correlati ma tentano di focalizzare l'attenzione su due cose distinte. Il grafico dello speedup fornisce un'indicazione di quanto sia parallelizzabile un algoritmo. Tale informazione è molto utile quando è necessario quantificare il vantaggio, date le risorse attuali, dell'aggiungere risorse. Per esempio, nei centri di supercalcolo, la potenza di calcolo viene venduta a blocchi di tempo. Il tempo acquistato viene scalato in base al tempo d'esecuzione ed alle risorse utilizzate[2]. Per esempio se l'algoritmo impiega 5 minuti, utilizzando 2 processori il costo di tale esperimento risulterebbe di $5 \cdot 2 = 10$ minuti. Se lo stesso algoritmo impiegasse 4 minuti con 4 processori il costo dell'esperimento diventerebbe di $4 \cdot 4 = 16$ minuti. Quindi il grafico dello speedup può essere un valido strumento di scelta nel trade-off fra tempo d'esecuzione e quantità di risorse utilizzate. Il grafico tempo-processori fornisce una stima del tempo necessario ad eseguire l'algoritmo su un dato input avendo a disposizione una data potenza di calcolo. Per concludere l'introduzione alle analisi si fornisce la definizione di *efficienza* $E_p$ di un algoritmo parallelo eseguito su $p$ processi.

[2]Es. numero di cpu, quantità di memoria centrale.

**Def.** *Si definisce l'efficienza $E_p$ di un algoritmo parallelo eseguito su p processi come un valore tra zero ed uno che quantifica quanto sforzo è impiegato nella comunicazione o nella coordinazione fra le unità di calcolo coinvolte. L'efficienza $E_p$ è definita come*

$$E_p \stackrel{def}{=} \frac{S_p}{p}$$

### Betweenness Centrality su architettura a memoria condivisa

L'algoritmo di betweenness centrality concorrente su architettura a memoria condivisa è fornito dal framework SNAP. Le analisi si riferiscono all'utilizzo dell'algoritmo tramite l'interfaccia a riga di comando fornita da SNAP e sono state svolte presso il centro di supercalcolo CINECA[3] sul cluster IBM-SP6. La tabella 4.1 elenca le caratteristiche del cluster IBM-SP6.

| | |
|---|---|
| **Modello** | IBM Power6 575 |
| **Nodi** | 168 |
| **CPU** | IBM Power6 4.7 GHz |
| **CPU per nodo** | 32 |
| **RAM per nodo** | 128GB |
| **Rete** | Infiniband 4x DDR |
| **Sistema operativo** | AIX 6 |
| **Compilatore C/C++** | Xlc 11.01 |

Tabella 4.1: Caratteristiche del cluster SP6 presso il centro di supercalcolo CINECA.

Il cluster del CINECA è stato scelto per effettuare le analisi in quanto i nodi di calcolo dispongono di 32 cpu a memoria condivisa e quindi ideali per verificare le prestazioni di algoritmi parallelli a memoria condivisa, contrariamente al cluster del dipartimento di Scienze dell'Informazione dell'Università di Bologna i cui nodi di calcolo dispongono di una sola cpu Dual Core. Le analisi svolte riguardano le prestazioni dell'algoritmo per due grafi distinti,

[3]Il CINECA è un Consorzio Interuniversitario senza scopo di lucro formato da 47 Università italiane. Ha sede a Casalecchio di Reno, Bologna. `http://www.cineca.it`

uno di piccole dimensioni, 30.000 nodi, ed uno di medie dimensioni, 224.288. Le analisi di grafi di dimensioni maggiori non sono state effettuate a causa delle limitazioni sulle risorse imposte dal centro di supercalcolo. Infatti ci è stato impedito di lasciare in esecuzione processi per un periodo di tempo superiore alle 120 ore[4]. Tale limite ha impedito la terminazione corretta dell'algoritmo che calcola la Betweenness centrality fornito dal framework SNAP su grafi di grandi dimensioni (milioni di nodi), infatti l'algoritmo non riusciva a terminare correttamente (entro il limite di tempo) nemmeno su grafi di 500.000 nodi.

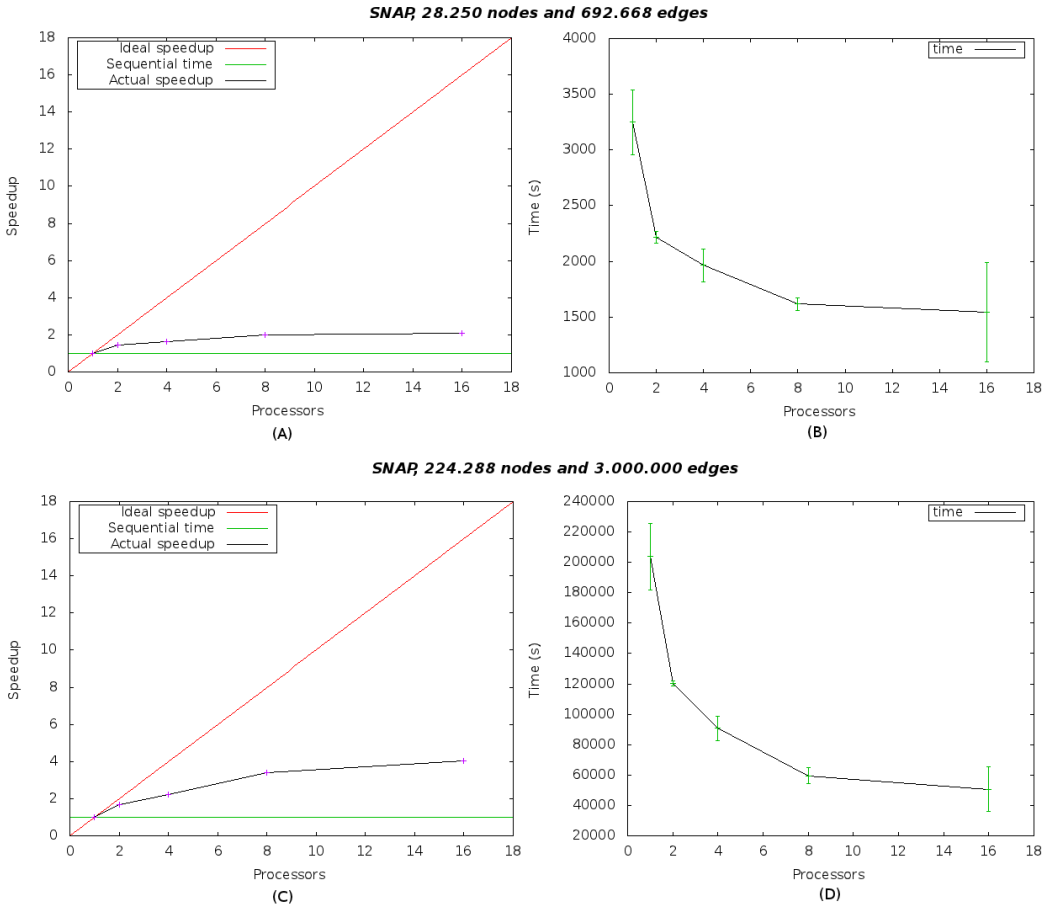


Figura 4.1: L'immagine mostra lo speedup e il tempo ottenuti sul cluster IBM-SP6 del CINECA dall'algoritmo che calcola la betweenness centrality nella libreria SNAP

[4]Il tempo totale di esecuzione consentito varia a seconda delle risorse utilizzate.

Il *primo esperimento* riguarda il calcolo dei valori esatti di Betweenness su un grafo scale-free di 28.250 nodi e 692.668 archi. I grafici in figura 4.1 mostrano i tempi e lo speedup ottenuti dall'algoritmo che calcola la Betweenness centrality implementato in SNAP. In particolare i grafici 4.1(A) e 4.1(B) mostrano i risultati ottenuti nel primo esperimento; nelle ascisse viene mostrato il numero di processori/processi mentre sulle ordinate sono mostrati: lo speedup nel grafico (A) e il tempo in secondi nel grafico (B). Nella tabella 4.2 viene mostrata l'efficienza dell'algoritmo in questo esperimento. L'algoritmo

| $p$ | $E_p$ grafo piccolo | $E_p$ grafo medio |
|---|---|---|
| 1 | 1.00 | 1.00 |
| 2 | 0.73 | 0.84 |
| 4 | 0.41 | 0.56 |
| 8 | 0.25 | 0.42 |
| 16 | 0.13 | 0.25 |

Tabella 4.2: Efficienza dell'algoritmo di Betweenness centrality implementato in SNAP.

mostra cenni di scalabilità solo quando i processori diventano due, ottenenendo un'efficienza pari a 0.73. Ma già con 4 processori non vi è alcun beneficio sostanziale, infatti l'efficienza si abbassa a 0.41 quindi è più il tempo speso a coordinarsi che quello di calcolo utile ai fini della soluzione del problema.

Il *secondo esperimento* riguarda il calcolo dei valori esatti di Betweenness su un grafo Random con 224.288 nodi e 3 milioni di archi. I grafici 4.1(C) e 4.1(D) mostrano i risultati ottenuti in questo secondo esperimento; nelle ascisse viene mostrato il numero di processori/processi mentre sulle ordinate sono mostrati: lo speedup nel grafico (C) e il tempo in secondi nel grafico (D). I grafici mostrano che lo speedup è rilevante solamente con 2 processori, le restanti prove non mostrano miglioramenti significanti. La tabella dell'efficienza 4.2 mostra infatti che già con 4 processori l'efficienza si accosta a 0.56.

Confrontando i risultati ottenuti dall'algoritmo nei due esperimenti, ossia osservando il grafico dello speedup, in figura 4.1 e la tabella 4.2 che mostra l'efficienza dell'algoritmo, si nota un miglioramento delle prestazioni a parità di processori all'aumentare della dimensione del grafo. Tale miglioramento può essere spiegato in quanto, in problemi come i cammini su grafo, aumentare i dati, a parità di processi, significa diminuire la probabilità di contesa per

dato, dove per contesa per dato si intendono i tentavi di accesso contemporaneo alla stessa locazione di memoria. Questo riduce il tempo sprecato nella coordinazione che è la principale causa di perdita di tempo negli algoritmi concorrenti.

In tabella 4.3 viene mostrato l'utilizzo di memoria centrale dell'algoritmo fornito da SNAP per il calcolo della Betweenness centrality al variare della dimensione del grafo. Come mostrano i valori in tabella 4.3, il calcolo su grafi con milioni di nodi risulta proibitivo per calcolatori come quelli disponinili a largo mercato.

In conclusione la versione concorrente dell'algoritmo che calcola i valori di Betweenness centrality implementato in SNAP fornisce una riduzione sostanziale dei tempi solamente con un numero di processi uguale a 2. Si è mostrato che aumentando la dimensione del grafo è possibile utilizzare, con profitto, un numero maggiore di processi.

| **Nodi** | **Archi** | **Memoria (KB)** |
|---|---|---|
| 10.000 | 162.312 | 4.014 |
| 100.000 | 934.023 | 28.098 |
| 500.000 | 5.501.204 | 160.560 |
| 1.000.000 | 24.016.203 | 602.100 |
| 3.000.000 | 71.231.001 | 1.143.990 |
| 5.000.000 | 125.213.032 | 1.906.650 |

Tabella 4.3: Consumo di memoria centrale dell'algoritmo di Betweenness centrality fornito da SNAP al variare della dimensione del grafo.

Vista la disponibilità di calcolatori con varie decine di core si riuscirebbe ad ottenere buoni risultati con grafi di grandi dimensioni (milioni di nodi) ma la limitazione al tempo di calcolo, imposta dal CINECA, ne limita fortemente l'applicabilità. Le macchine acquistabili su largo mercato difficilmente superano gli 8 core di calcolo e la decina di giga byte di memoria nei casi più favorevoli; in media si ha a disposizione macchine con due o quattro core di calcolo e due o quattro giga byte di memoria centrale. Quindi un approccio concorrente, con i calcolatori a disposizione su largo mercato, non risulta applicabile per l'analisi di grafi di decine o centinaia di milioni di nodi.

### Betweenness Centrality su architettura a memoria distribuita

L'analisi degli algoritmi di betwenness centrality disponibili in LANA è stata effettuata nel cluster del dipartimento di Scienze dell'Informazione dell'Università di Bologna. Le caratteristiche del cluster sono indicate nella tabella 4.4. Purtroppo siamo stati impossibilitati ad utilizzare le infrastrutture del centro di supercalcolo CINECA in quanto LANA si appoggia sulla libreria PBGL ver. 1.45 e la compilazione della libreria PBGL sul compilatore nativo del sistema operativo AIX, Xlc++ ver. 10.1.0.8, falliva impedendo così la compilazione di LANA.

| | |
|---|---|
| **Nodi** | 61 |
| **CPU** | Core 2 Duo E7500 2.93GHz |
| **CPU per nodo** | 1 |
| **RAM per nodo** | 2GB |
| **Rete** | 100Mb Ethernet half duplex |
| **Sistema operativo** | Linux 2.6.28-19 |
| **Compilatore C/C++** | Gcc 4.3.3 |

Tabella 4.4: Caratteristiche del cluster del dipartimento di scienze dell'informazione, Università di Bologna.

In questa sezione si analizzano gli algoritmi di Betweenness centrality distribuiti implementati nella PBGL e utilizzabili mediante il software LANA.

L'analisi mostrerà l'applicabilità e le prestazioni degli algoritmi che calcolano la Betweenness centrality disponibili in LANA sui mezzi di calcolo a nostra disposizione, ossia sul cluster illustrato in tabella 4.4.

**Replicazione del grafo** In questo paragrafo si analizza l'algoritmo che calcola i valori di Betweenness centrality disponibile in LANA e basato sull'approccio di replica dell'intero grafo. L'algoritmo è stato proposto da *Yang* e *Leonardi* [27] e implementato nella libreria PBGL. L'algoritmo consta di due fasi:

1. Ogni calcolatore coinvolto calcola in sequenziale i cammini minimi su un sottoinsieme dei nodi del grafo. I cammini minimi sono calcolati come nell'algoritmo di Brandes.

2. I processori si scambiano le informazioni necessarie affinchè siano in grado di completare il calcolo dei valori di betwenness. Al termine ogni processore possiede una struttura dati con i valori di Betweenness centrality.

La complessità dell'algoritmo, su un grafo di $n$ nodi, risulta essere $O(n \log n)$ per ogni SSSP (Algoritmo 1 righe 13-14). La seconda fase (Algoritmo 1 riga 23) che calcola i valori di betweenness costa $O(n \log n)$.

```
1  INPUT: Grafo G=(V,E)
2
3  BC := mappa dei valori di betweenness
4
5  # Inizializza le strutture dati
6  init(BC)
7
8  # Barriera per sincronizzare il calcolo
9  barrier()
10
11 # Calcola i SSSP per i nodi assegnati al processo
12 # ossia i valori di dipendenza
13 foreach s in my_nodes:
14   do_sequential_brandes_SSSP(G, s, BC[s])
15
16 # Barriera per sincronizzare il calcolo
17 barrier()
18
19 # I processori si scambiano fra di loro la BC
20 send_all(BC)
21 # Viene effettuata la sommatoria dei valori di dipendenza
22 # dei nodi
23 sum_all_dependency_values(BC)
```

**Algoritmo 1.** Algoritmo in LANA che calcola la betweenness centrality su grafo replicato.

Nella tabella 4.5 viene mostrata l'efficienza dell'algoritmo in questo esperimento.

Da un'analisi dello speedup e dell'efficienza le prestazioni dell'algoritmo risultano ottime. Dai grafici in figura 4.2, che mostrano le prestazioni e lo

| $p$ | $E_p$ Erdös-Rényi | $E_p$ Small-World |
|---|---|---|
| 1 | 1.00 | 1.00 |
| 2 | 0.996 | 0.997 |
| 3 | 0.994 | 0.996 |
| 4 | 0.996 | 0.996 |
| 5 | 0.982 | 0.996 |
| 6 | 0.949 | 0.986 |
| 7 | 0.967 | 0.996 |
| 8 | 0.967 | 0.996 |

Tabella 4.5: Efficienza dell'algoritmo di Betweenness centrality con grafo replicato implementato nella libreria PBGL e disponibile in LANA.

speedup dell'algoritmo su due grafi di 30.000 nodi, indicano che lo speedup ottenuto dall'algoritmo si accosta allo speedup ideale per entrambi i tipi di grafo. La tabella 4.5 mostra che per entrambi i grafi l'efficienza rimane costantemente al di sopra del 94%. Nei grafici non vengono mostrati gli intervalli di confidenza in quanto la varianza dei risultati delle prove è trascurabile.

Questo risultato non è sorprendente data la nulla necessità di comunicare durante la fase di calcolo dei cammini minimi. Inoltre, replicando il grafo su ogni calcolatore si eliminano anche tutti i problemi di coordinazione incontrati nella versione concorrente. Le ottime prestazioni ottenute sul cluster dell'università indicano che questo approccio risulta essere vincente anche in cluster connessi tramite una rete mediocre. Purtroppo, esattamente come la versione concorrente, le limitazioni imposte dalla quantità di memoria centrale rimangono invariate e confrontando la tabella 4.3 che indica l'utilizzo di memoria della versione concorrente implementata in SNAP e la tabella 4.7 che mostra l'utilizzo di memoria di LANA, che utilizza gli algoritmi della PBGL, si nota che gli algoritmi della libreria PBGL richiedono un quantitativo di memoria superiore.

**Confronto fra SNAP e PBGL** In questo paragrafo si mostra il confronto prestazionale fra i tempi dell'algoritmo concorrente che calcola la Betweenness centrality implementato in SNAP e l'algoritmo distribuito, ba-

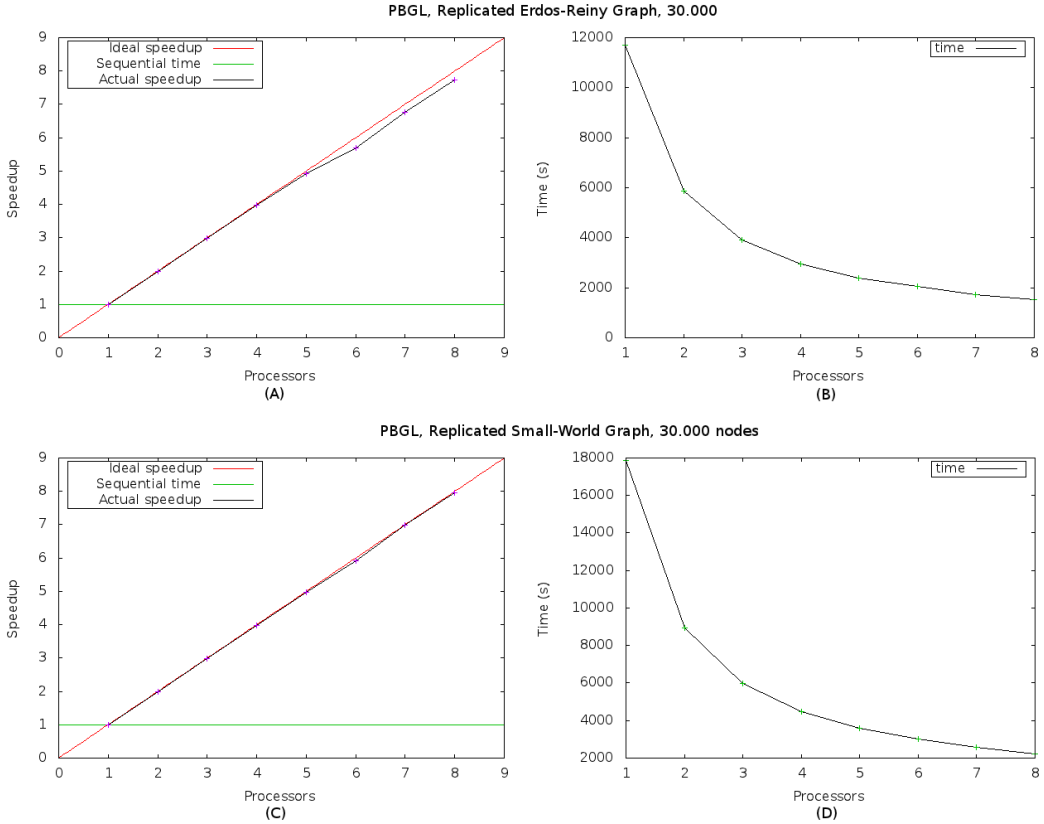


Figura 4.2: L'immagine mostra lo speedup e il tempo ottenuti sul cluster del dipartimento di scienze dell'informazione dall'algoritmo che calcola la betweenness centrality nella libreria PBGL. Il grafo contiene circa 30 mila nodi. I grafici in alto riguardano un grafo Erdös-Rényi e quelli in basso un grafo Small world.

sato sulla replica del grafo, che calcola i valori di Betweenness centrality implementato nella PBGL. La prova è stata effettuata sul cluster del dipartimento di Scienze dell'informazione le cui caratteristiche sono mostrate nella tabella 4.4. La prova consiste nel calcolare i valori esatti di Betweenness centrality su grafo scale-free di 28.250 nodi e 692.668 archi.

Il grafico in figura 4.3 mostra il confronto fra le due implementazioni. Il grafico mostra le prestazioni dell'algoritmo distribuito implementato in PBGL al variare del numero di macchine mentre la riga rossa parallela all'asse delle ascisse rappresenta le prestazioni dell'algoritmo implementato in SNAP eseguito concorrentemente sui due core del processore. Come mostra il grafico, all'algoritmo implementato nella PBGL sono necessari dieci calcolatori per ottenere dei tempi paragonabili ai tempi ottenuti da SNAP su

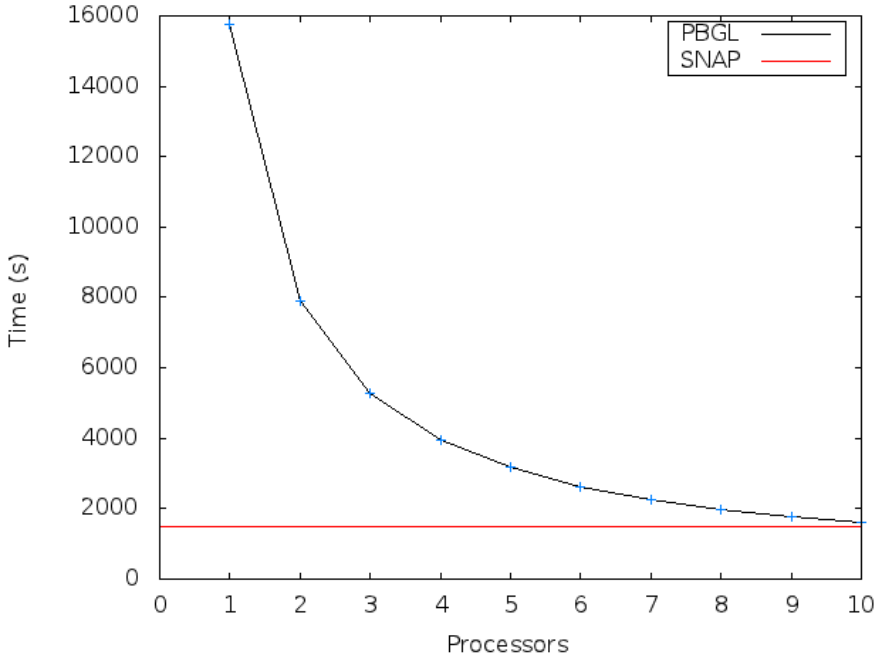


Figura 4.3: Il grafico mostra le prestazioni dell'algoritmo di Betweenness centrality distribuito implementato in PBGL al variare del numero di unità di calcolo confrontato al tempo impiegato dall'algoritmo di Betweenness centrality in SNAP eseguito concorrentemente su due core.

un singolo calcolatore utilizzando entrambi i core a disposizione concorrentemente. Questi risultati si possono spiegare considerando che nell'algoritmo in PBGL, distribuendo l'intero grafo, ogni calcolatore calcola le SSSP in maniera completamente indipendente dagli altri calcolatori, quindi ogni calcolatore deve calcolare per proprio conto tutti i cammini minimi necessari a terminare la computazione. Nel caso di SNAP questo non accade in quanto la memoria è condivisa fra i due processi e quindi ognuno dei processi può avvalersi dei calcoli effettuati dall'altro.

**Distribuzione del grafo.** Il secondo algoritmo distribuito per il calcolo dei valori di Betweenness centrality implementato in LANA si prefigge l'obbiettivo di ridurre il quantitativo di memoria centrale necessario distribuendo il grafo fra i vari calcolatori coinvolti. Ovviamente partizionare il grafo e distribuirlo fra i vari calcolatori implica un aumento considerevole delle comunicazioni nella fase di calcolo dei cammini minimi.

L'algoritmo proposto da *Edmonds*, *Hoefler* e *Lumsdaine* [28] ed implementato nella PBGL dagli autori stessi consta di tre fasi:

1. Si calcolano sul grafo $G = (V, E)$, come in Brandes, i cammini minimi ottenendo un sottografo $G' = (V, E')$ con $E'$ insieme di tutti gli archi $(u, v) \in E$ che risiedono su almeno un cammino minimo. In questa fase vengono calcolati l'insieme dei predecessori $P_s(v)$ e *l'insieme dei successori* $S_s(v)$. L'insieme dei successori è definito come:

$$S_s(v) \stackrel{def}{=} \{u \in V \mid (v, u) \in E,\ d(s, u) = d(s, v) + w(v, u)\}$$

2. Viene calcolato $\sigma_s(v)$ per ogni nodo $v$. Tale calcolo viene effettuato mediante una BFS sincronizzata a livelli, che significa che nessun vertice al livello $i+1$ viene scoperto finchè non sono stati scoperti tutti i vertici a livello $i$. Per fare ciò viene utilizzato l'insieme dei successori $S_s(v)$.

3. Vengono calcolate le dipendenze $\delta_{s*}(v)$ e i valori di betweenness BC per tutti i vertici come nell'algoritmo di Brandes.

```
1  INPUT: Grafo G=(V,E)
2
3  BC := mappa dei valori di betweenness
4
5  # Inizializza le strutture dati
6  init(BC)
7
8  foreach s in V:
9    # Calcola i cammini minimi ed i predecessori
10   P[s] = shortest_paths(G, s, BC[s])
11   # Calcola i successori dai predecessori
12   S[s] = transpose(P[s])
13   # Conta dei cammini minimi
14   (SIGMA[s], Q) = path_counts(S[s], s)
15   # Aggiornamento della BC
16   update_centrality(BC, Q, SIGMA)
```

**Algoritmo 2.** Algoritmo in PBGL che calcola la betweenness centrality su grafo distribuito. Per maggiori dettagli si veda [28].

Per il calcolo degli SSSP viene utilizzato l'algoritmo Δ-stepping [29]. L'algoritmo Δ-stepping calcola i cammini minimi da un nodo sorgente $s$ mantenendo un vettore di "Bucket" $B$, ed ogni Bucket è composto da un insieme di nodi e dal loro limite superiore della distanza da $s$, compreso in un certo range Δ. Ossia $B[i] = \{v \in V \mid tent[v] \in [i\Delta, (i+1)\Delta]\}$ dove $tent[v]$ è un limite superiore alla distanza del nodo $v$ dal nodo sorgente $s$. Durante ogni fase, l'algoritmo rimuove tutti i nodi dal primo Bucket non vuoto e per ogni nodo rilassa tutti gli archi uscenti di peso massimo Δ, che è un numero reale positivo che dovrebbe essere scelto in modo da fornire un buon trad-off fra il numero di "riconsiderazioni" dei nodi e il numero di Bucket. L'algoritmo Δ-stepping rientra nella categoria di algoritmi su grafo detti *Label-correcting*. A differenza degli algoritmi impiegati nell'algoritmo di Brandes che fanno parte della categoria di algoritmi detti *label-setting*. La scelta è motivata dalla miglior parallelizzabilità degli algoritmi label-correcting. Per una spiegazione delle differenze fra gli algoritmi label-setting e label-correcting si veda l'appendice A.

Purtroppo gli esperimenti per valutare le prestazioni dell'algoritmo di betweenness centrality con grafo distribuito non hanno dato esiti positivi. Nella tabella 4.6 vengono mostrati i risultati delle prove su un grafo Erdös-Rényi di 30.000 nodi. Anche limitando il calcolo degli SSSP solo su un sottoinsieme dei nodi, in modo da ottenere un'approssimazione della Betweenness centrality, non vi sono stati segni di scalabilità. Come si nota dalla tabella 4.6, il tempo d'esecuzione aumenta all'aumentare dei calcolatori. Tali risultati negativi sono dovuti all'effetto delle comunicazioni di rete. Purtroppo l'unico cluster a nostra disposizione, tabella 4.4, è risultato inadatto a livello di rete (100Mb Half Duplex). Un'analisi tramite Scalasca ha confermato i nostri dubbi mostrando che il 90% del tempo d'esecuzione viene speso nella comunicazione per il calcolo dei cammini minimi.

La tabella 4.7 mostra l'utilizzo di memoria del software LANA al variare della dimensione del grafo. La terza colonna mostra l'utilizzo di memoria per l'algoritmo che partiziona il grafo mentre la quarta colonna mostra l'utilizzo di memoria dell'algoritmo che replica l'intero grafo. Entrambi i valori mostrano l'utilizzo di memoria relativo all'esecuzione dell'algoritmo su una sola macchina. Il maggior utilizzo di memoria nel caso del grafo partizionato è dovuto alle strutture dati necessarie a tenere traccia della distribuzioine dei nodi fra i vari calcolatori. La dimensione di tali strutture dati è direttamente proporzionale alla dimensione del grafo. La tabella mostra un caso limite in cui vi è un solo calcolatore che mantiene strutture dati di cui in realtà non ha necessità. Partizionando il grafo su diversi calcolatori l'utilizzo di memoria per calcolatore risulta minore dell'occupazione in memoria dell'intero grafo.

| **Nodi di calcolo** | **Tempo d'esecuzione(s)** | **Errore standard** (s) |
|---|---|---|
| 1 | 11685.0 | 110.07 |
| 2 | 14256.0 | 178.10 |
| 3 | 18324.3 | 156.32 |
| 4 | 25982.6 | 181.58 |

Tabella 4.6: Prestazioni dell'algoritmo che calcola la Betweenness centrality distribuendo il grafo. Le prove sono state effettuate sul cluster del Dipartimento di Scienze dell'Informazione.

| **Nodi** | **Archi** | **Grafo partizionato (KB)** | **Grafo replicato (KB)** |
|---|---|---|---|
| 10.000 | 162.312 | 53.292 | 44.410 |
| 100.000 | 934.023 | 118.428 | 90.005 |
| 500.000 | 5.501.204 | 414.499 | 278.306 |
| 1.000.000 | 24.016.203 | 846.763 | 491.478 |
| 3.000.000 | 71.231.001 | 2.374.491 | 1.367.848 |
| 5.000.000 | 125.213.032 | 3.582.461 | 2.238.298 |

Tabella 4.7: Consumo di memoria centrale del software LANA al variare della dimensione del grafo.

**Approssimazione dei valori di Betweenness** L'algoritmo, fornito in LANA, che calcola la Betweenness centrality su un grafo distribuito permette di calcolare un'approssimazione dei valori di Betweenness centrality anzichè i valori esatti. Tale approssimazione viene calcolata mediante il primo algoritmo presentato nel capitolo 4.2.2, che riduce il numero di SSSP da calcolare considerando come nodi sorgente solamente un sottoinsieme arbitrario dei nodi del grafo.

La motivazione che porta a calcolare un'approssimazione è data dalla ancora non sufficiente potenza di calcolo che rende impraticabile il calcolo dei valori esatti di Betweenness centrality su grafi di grandi dimensioni (decine/centinaia di milioni di nodi) in tempi ragionevoli. Calcolare solo un sottoinsieme degli SSSP riduce al tempo stesso sia i calcoli da effettuare sia le comunicazioni, nel caso con grafo distribuito, necessarie a portare a termine il calcolo. L'analisi che segue è volta a verificare l'accuratezza

dell'approssimazione fornita dall'algoritmo implementato nella PBGL.

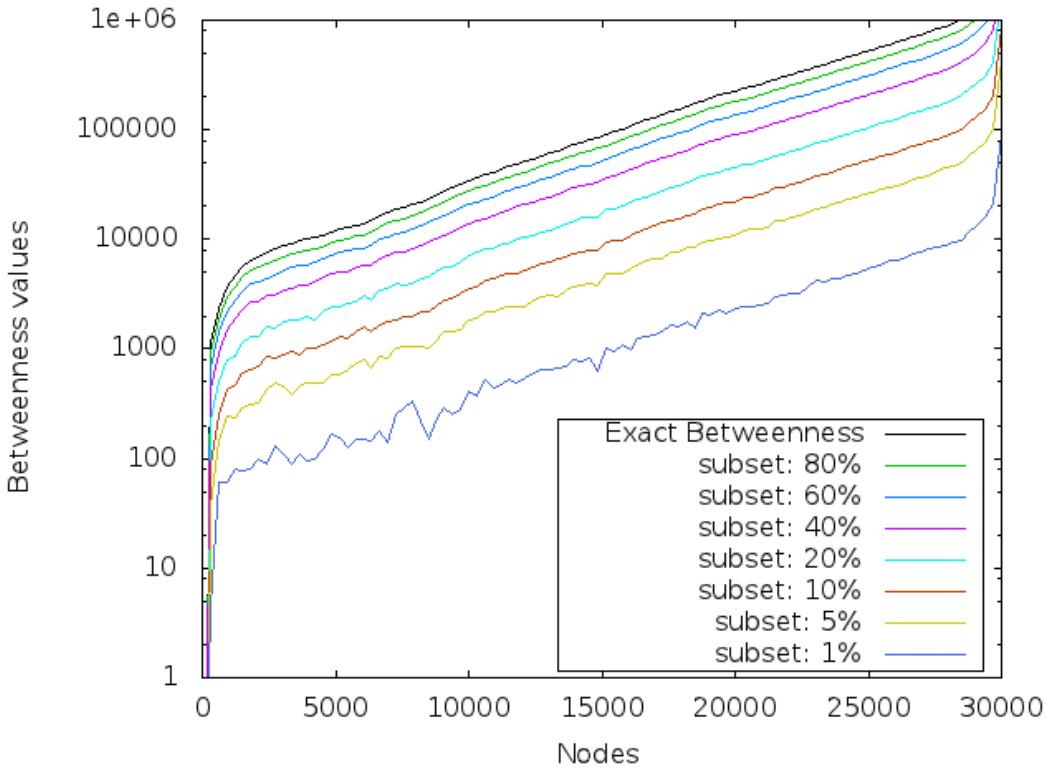


Figura 4.4: L'immagine mostra le approssimazioni dei valori di betweenness su un grafo small world di 30.000 nodi. I nodi sono ordinati in base al valore esatto di betweenness. L'ordine forma una classifica ordinata in maniera crescente sull'asse delle ascisse.

Nel grafico rappresentato nell'immagine 4.4 vengono mostrati i risultati dell'approssimazione della betweenness distribuita fornita in LANA. Nel grafico 4.4 vi sono i valori approssimati di diverse prove. Ognuna delle prove è stata effettuata variando la cardinalità del sottoinsieme dei nodi su cui calcolare gli SSSP. Ognuna delle prove è stata ripetuta varie volte cambiando i nodi che compongono l'insieme su cui calcolare gli SSSP ed il grafico mostra la media dei risultati ottenuti. I nodi sulle ascisse vengono ordinati in maniera crescente in base al valore esatto di betweenness centrality, tale ordinamento forma una classifica dei nodi sulle ascisse. La classifica sulle ascisse inizia dall'origine con il nodo con valore di Betweenness più basso e termina a 30.000 con il nodo con valore di Betweenness più alto.

Le ordinate invece rappresentano il valore di Betweenness calcolato per il nodo. Le funzioni nel grafico mappano il nodo nel corrispettivo valore di

Betweenness, la funzione nera corrisponde alla funzione che mappa i nodi nei valori esatti di Betweenness, le altre forniscono un'approssimazione limitando il numero di SSSP calcolate come spiegato precedentemente.

Il grafo su cui è stato effettuato l'esperimento è un grafo Small-World con 30.000 nodi. Dal grafico si evince che diminuendo la cardinalità del sottoinsieme la classifica tende ad essere più imprecisa. Gli errori sono più evidenti nella parte bassa della classifica mentre nella parte alta della classifica la presenza di errori è quasi nulla. Ovviamente la scelta della cardinalità e quindi della precisione della classifica finale è strettamente legata al dominio del problema. Spesso si è interessati unicamente ai nodi nelle prime posizioni della classifica i quali rappresentano i punti cruciali della rete. Per esempio potrebbero rappresentare i nodi con più potere sul flusso di informazioni in una rete sociale come twitter oppure le stazioni più importanti in una rete ferroviaria.

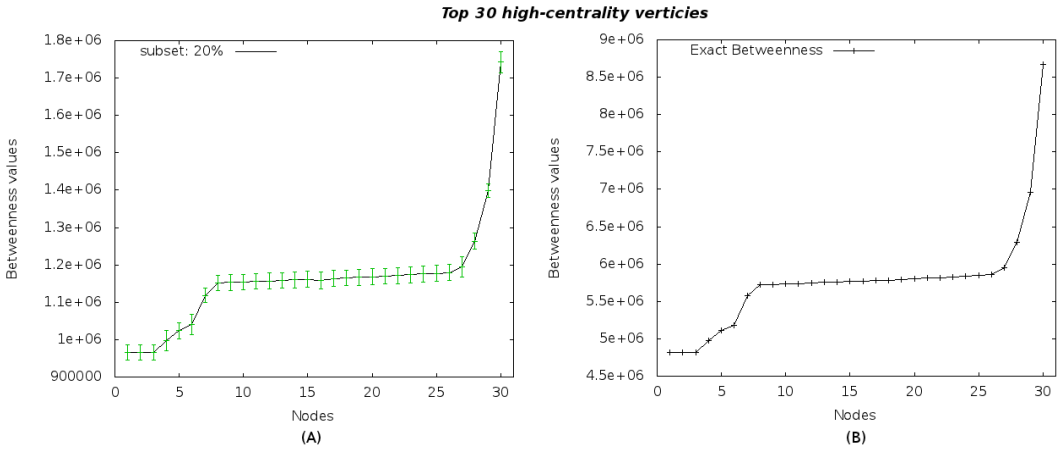


Figura 4.5: L'immagine mostra le approssimazioni dei valori di betweenness dei 30 nodi che hanno i valori di betweenness esatta più alti. I nodi sono ordinati in base al valore esatto di betweenness.

Per verificare la qualità dell'approssimazione dei nodi con indice di centralità maggiore sono stati selezionati i 30 nodi con indice di centralità esatto maggiore. Di questi nodi è stato confrontato il valore di Betweenness esatto con il valore di Betweenness approssimato. Tale confronto è mostrato nei due grafici di figura 4.5. Il grafico 4.5(A) mostra i valori approssimati calcolando gli SSSP solo per un sottoinsieme composto da circa il 20% dei nodi del grafo mentre il grafico 4.5(B) i valori esatti. Sull'asse delle ascisse ci sono i nodi, ordinati per valore di Betweenness crescente, mentre sull'asse delle ordinate ci sono i valori di Betweenness. I valori di Betweenness trovati per i nodi

ovviamente differiscono nella versione esatta rispetto a quella approssimata, questo è dovuto al minor numero di cammini minimi presi in considerazione. Ciononostante la classifica dei nodi è identica come è possibile verificare dal confronto fra la classifica approssimata e la classifica esatta mostrato in figura 4.5. La capacità di ottenere approssimazioni di tale precisione mediante il calcolo degli SSSP per un sottoinsieme dei nodi di circa il 20% porta diversi vantaggi. Il primo vantaggio consiste nel migliorare le prestazioni degli algoritmi che distribuiscono il grafo, riducendo, vista la riduzione degli SSSP da calcolare, il numero di comunicazioni. Il secondo vantaggio riguarda la possibilità di rendere trattabili grafi di dimensioni maggiori rispetto a quelli trattabili, per questioni di tempo d'esecuzione, dagli algoritmi che calcolano i valori esatti di Betweenness centrality.

# Conclusioni

Questo lavoro di tesi ha presentato il tema delle reti complesse mostrandone i modelli e le principali metriche usate nell'analisi di tali sistemi complessi. Sono stati introdotti i modelli di calcolo concorrente e distribuito, fornendo una panoramica sulle difficoltà inerenti alla definizione e all'implementazione di algoritmi che risolvono problemi su grafi, in special modo i problemi legati alla loro parallelizzazione. Sono stati presentati le librerie e gli strumenti utili all'implementazione e all'analisi di algoritmi su grafi paralleli e distribuiti. È stato presentato il software LANA, sviluppato durante questo lavoro di tesi, che fornisce la possibilità di effettuare il calcolo di diverse metriche di centralità su grafo, sia su grafi forniti dall'utente e sia generati direttamente da LANA.
È stata mostrata un'analisi prestazionale degli algoritmi che calcolano la Betweenness centrality sia per il framework SNAP che per il software LANA. Tale analisi offre un parere super partes sui diversi approcci proposti al calcolo della Betweenness centrality su grafi di grandi dimensioni.

In particolare, i risultati ottenuti dalle analisi svolte, riguardanti la Betweenness centrality, hanno confermato i problemi relativi agli algoritmi su grafo ed in particolar modo a quelli inerenti il problema di calcolare un numero di SSSP proporzionale al numero di nodi nel grafo. L'approccio concorrente offerto dal framework SNAP fornisce un valido strumento al calcolo della Betweenness centrality in grafi di medie dimensioni (200 mila nodi). Tuttavia è risultato non applicabile su grafi di dimensioni maggiori data la scarsità di memoria centrale e potenza di calcolo nei calcolatori attualemente disponibili a largo mercato. Inoltre le prestazioni di SNAP su architettture HPC come il cluster del CINECA hanno mostrato la scarsa scalabilità dell'algoritmo all'aumentare del numero di processi.

L'approccio di distribuzione del calcolo e replicazione dell'intero grafo implementato nella PBGL e disponibile in LANA ha mostratato la possibilità di ottenere degli ottimi risultati in termini di speedup all'aumentare delle unità di calcolo anche su cluster composti da calcolatori disponibili su largo mercato. Al tempo stesso ha mostrato delle pessime prestazioni in termi-

ni di tempo paragonate alle prestazioni ottenute dall'algoritmo concorrente implementato in SNAP. In uno scenario in cui si dispone di svariati calcolatori di ridotta capacità di calcolo l'approccio di distribuzione del grafo resta comunque da preferire.

In entrambi i casi, la necessità di grandi quantità di memoria centrale ne limita l'applicabilità a reti di decine/centinaia di milioni di nodi. Purtroppo l'approccio di distribuzione del grafo non ha mostrato segni di applicabilità nel cluster a nostra disposizione. Sarebbero necessarie ulteriori analisi per verificare l'applicabilità di tale approccio.

Il proseguimento naturale di questo lavoro di tesi è quindi la ricerca di algoritmi basati sulla distribuzione del grafo, tentando di sfruttare modelli ibridi, sia concorrenti che distribuiti. Data la qualità delle approssimazioni calcolate dagli algoritmi descritti, non è da escludere che l'ottimizzazione degli algoritmi di approssimazione nei modelli distribuiti ed ibridi rendano possibile l'analisi di grafi di dimensioni fino ad oggi proibitive.

# Appendice A

# Cammini minimi su grafo

Gli algoritmi per il calcolo dei cammini minimi sono generalmente basati su metodi iterativi detti metodi di "etichettamento" (dal termine inglese labeling). In un grafo $G = (v, E)$, nel calcolo di un SSSP dal nodo $s \in V$ per ogni nodo $v \in V$ viene mantenuta una variabile $tent(v)$ detta etichetta. $tent(v)$ è un limite superiore alla distanza del nodo $v$ dal nodo sorgente $s$. Inizialmente $tent(s) = 0$ e $tent(v) = \infty$ per tutti i nodi $v \neq s$.
Il limite superiore $tent(v)$ viene migliorato effettuando una procedura detta *rilassamento*, tale procedura consiste nel riconsiderare il valore di $tent(v)$ per ogni arco $(u, v)$. Ossia $tent(v) = min\{tent(v), tent(u) + w(u, v)\}$.

Gli algoritmi di labeling selezionano ripetutamente un arco $(u, v) \in E$ e verificano se l'arco può essere rilassato. La procedura termina quando non vi sono più archi da rilassare, ossia quando si verifica la seguente condizione

$$tent(v) \leq (tent(u) + w(u, v)) \; \forall \, (u, v) \in E$$

Quindi $d(s, v) = tent(v) \; \forall \; v \in V$. L'etichetta $tent(v)$, se $d(s, v) = tent(v)$, viene detta *stabile* ed il nodo $v$ è detto *fissato*.
Vi sono due categorie di algoritmi di labeling, gli algoritmi *label-correcting* e gli algoritmi *label-setting*. Gli algoritmi di label-setting, come l'algoritmi di Dijkstra [24] e l'algoritmo BFS [15], designano l'etichetta $tent(v)$ di un nodo $v$ come stabile ad ogni iterazione e rilassano gli archi dei soli nodi già marcati come fissati. Quindi l'algoritmo necessita di $n$ iterazioni al massimo, dove $n$ è il numero di nodi nel grafo. Gli algoritmi label-correcting, come l'algoritmo *Bellman-Ford* [30] e l'algoritmo $\Delta$-*stepping* [29] , possono rilassare gli archi di nodi non fissati e un nodo può essere rivalutato più volte. Le etichette negli algoritmi label-correcting sono considerate temporanee fino all'ultimo passo dell'algoritmo, dove vengono marcate tutte come stabili. Gli algoritmi di

label-setting sono caratterizzati dall'aver un costo computazionale significativamente minore, nel caso pessimo, degli algoritmi label-correcting. Analisi prestazionali hanno mostrato però che gli algoritmi label-correcting impiegano meno tempo degli algoritmi label-setting. Inoltre grazie alla possibilità di rilassare diversi archi contemporaneamente, gli algoritmi label-correcting sono facilmente parallelizzabili.

# Ringraziamenti

I miei ringraziamenti vanno a tutti quelli che in questi anni mi hanno saputo ispirare, a chi mi ha incuriosito, a chi mi ha dimostrato che ero in torto e mi ha mostrato altri punti di vista. Ringrazio inoltre la mia famiglia, che nel bene e nel male, mi sopporta. Infine tengo a ringraziare i miei relatori, Moreno Marzolla e Matteo Magnani, per la possibilità offertami di lavorare ad una tesi interessante, i preziosi consigli e l'infinita disponibilità.

# Bibliografia


[1] L. C. Freeman, "Centrality in social networks: Conceptual clarification," *Social Networks*, vol. 1, pp. 215–239, 1979.

[2] M. V. Steen, "Graph Theory and Complex Network, an introduction," 2010.

[3] X. Wu and Z. Liu, "How community structure influences epidemic spread in social networks," *Physica A: Statistical Mechanics and its Applications*, vol. 387, pp. 623–630, Jan. 2008.

[4] A. Barrat, M. Barthelemy, R. Pastor-Satorras, and A. Vespignani, "The architecture of complex weighted networks," *Proceedings of the National Academy of Sciences of the United States of America*, vol. 101, no. 11, p. 3747, 2004.

[5] T. Opsahl, "Closeness centrality in networks with disconnected components," 2010.

[6] T. Opsahl, F. Agneessens, and J. Skvoretz, "Node centrality in weighted networks: Generalizing degree and shortest paths," *Social Networks*, 2010.

[7] P. Erdös and A. Rényi, "On random graphs," *Science*, vol. 6, no. 290, pp. 290–298, 1959.

[8] S. Milgram, "The small world problem," *Psychology Today*, vol. 1, p. 61, 1967.

[9] D. J. Watts and S. H. Strogatz, "Collective dynamics of 'small-world' networks," *Nature*, no. 393, pp. 440–442, 1998.

[10] M. Granovetter, "The strength of weak ties: A network theory revisited," *Sociological Theory*, vol. 1, pp. 201–233, 1983.

[11] A. L. Barabási and R. Albert, "Emergence of scaling in random networks," *Science*, vol. 286, no. 5439, pp. 509–512, 1999.

[12] S. N. Dorogovtsev and J. F. F. Mendes, *Evolution of Networks: From Biological Nets to the Internet and WWW*. New York, NY, USA: Oxford University Press, 2003.

[13] S. Dorogovtsev, J. Mendes, and A. Samukhin, "Structure of growing networks with preferential linking," *Physical Review Letters*, vol. 85, no. 21, pp. 4633–4636, 2000.

[14] L. Dagum and R. Menon, "Openmp: an industry standard api for shared-memory programming. Computational Science & Engineering," *IEEE*, vol. 5, no. 1, pp. 46–55, 1998.

[15] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*. MIT Press, 3rd ed., 2009.

[16] J. JáJá, *An Introduction to Parallel Algorithms*. Addison-Wesley, 1992.

[17] Message Passing Interface Forum, *MPI: A Message-Passing Interface Standard, Version 2.2*. High Performance Computing Center Stuttgart (HLRS), September 2009.

[18] T. El-Ghazawi, W. Carlson, and J. Draper, "UPC Language Specifications V1. 1.1," *October*, vol. 200, no. 3, p. 1.

[19] A. Lumsdaine, D. Gregor, B. Hendrickson, J. Berry, and J. G. Editors, "Challenges in parallel graph processing," *Parallel Processing Letters*, vol. 17, no. 1, pp. 5–20, 2007.

[20] D. Bader and K. Madduri, "SNAP, Small-world Network Analysis and Partitioning: an open-source parallel graph framework for the exploration of large-scale networks," in *Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE International Symposium on*, pp. 1–12, IEEE, 2008.

[21] D. Gregor and A. Lumsdaine, "The Parallel BGL: A generic library for distributed graph computations," *Parallel Object-Oriented Scientific Computing (POOSC)*, 2005.

[22] M. Geimer, F. Wolf, B. J. N. Wylie, E. Ábrahám, D. Becker, and B. Mohr, "The scalasca performance toolset architecture.," vol. 22, pp. 702–719, 2010.

[23] U. Brandes, "A faster algorithm for betweenness centrality," *Journal of Mathematical Sociology*, vol. 25, pp. 163–177, 2001.

[24] E. Dijkstra, "A note on two problems in connexion with graphs," *Numerische mathematik*, vol. 1, no. 1, pp. 269–271, 1959.

[25] U. Brandes and C. Pich, "Centrality estimation in large networks.," *I. J. Bifurcation and Chaos*, vol. 17, no. 7, pp. 2303–2318, 2007.

[26] D. Bader, S. Kintali, K. Madduri, and M. Mihail, "Approximating betweenness centrality," *Algorithms and Models for the Web-Graph*, pp. 124–137, 2007.

[27] Q. Yang and S. Lonardi, "A parallel edge-betweenness clustering tool for Protein-Protein Interaction networks," *International Journal of Data Mining and Bioinformatics*, vol. 1, no. 3, pp. 241–247, 2007.

[28] N. Edmonds, T. Hoefler, and A. Lumsdaine, "A space-efficient parallel algorithm for computing betweenness centrality in distributed memory," *Urbana*, vol. 51, p. 61801.

[29] U. Meyer and P. Sanders, "[Delta]-stepping: a parallelizable shortest path algorithm," *Journal of Algorithms*, vol. 49, no. 1, pp. 114–152, 2003.

[30] R. Bellman, "On a routing problem, Quart," *J. Appl. Math*, vol. 16, no. 1, pp. 87–90, 1958.