Università degli Studi di Roma Tor Vergata
Facoltà di Scienze Matematiche Fisiche e Naturali
Dipartimento di Matematica
Corso di Laurea in Informatica

Tesi di Laurea

Approssimazione efficiente del diametro in reti sociali di grandi dimensioni

Relatore
Prof. Giorgio Gambosi

Candidato
Valerio Capozio

Correlatore
Prof. Marco Bianchi

Anno Accademico 2005-2006

Università degli Studi di Roma Tor Vergata
Facoltà di Scienze Matematiche Fisiche e Naturali
Corso di Laurea in Informatica

Tesi di Laurea

Approssimazione efficiente del diametro in reti sociali di grandi dimensioni

Relatore:
Prof. Giorgio Gambosi .....

Correlatore:
Prof. Marco Bianchi .....

Candidato
Valerio Capozio .....

Anno Accademico 2005-2006

Ai miei genitori
che lo hanno permesso
e a mio fratello
che lo ha sopportato.

Ringraziamenti

Vorrei, in primo luogo, ringraziare il professor Gambosi, che mi ha proposto l'argomento indicandomi la rotta quando, soprattutto all'inizio, non avevo ancora le idee chiare e non sapevo come muovermi.

Ringrazio, inoltre, il professor Bianchi che mi ha seguito nello svolgimento pratico della tesi, mettendo a mia disposizione il suo tempo la sua esperienza.

Desidero ringraziare anche tutti i compagni che hanno, in questi anni, condiviso con me l'iter universitario rendendolo, con la loro presenza, più piacevole e leggero da affrontare.

Vorrei, infine, rivolgere un particolare ringraziamento a Stefano che ha guidato, con infinita pazienza, i miei primi passi nel mondo dell'informatica, permettendo al mio interesse per questa scienza di crescere sempre di più.

Indice

Elenco delle figure

Elenco delle tabelle

Elenco degli Algoritmi

Introduzione

"Nessun uomo è un'isola, completo in sé stesso;
ogni uomo è un pezzo del continente,
una parte del tutto."
John Donne

Modellare i dati raccolti, attraverso l'ausilio di grafi, sta diventando una metodologia molto diffusa e al contempo sta suscitando, all'interno della comunità di data mining, notevoli interessi, come ad esempio tentare di comprendere gli sviluppi di Internet e del World Wide Web. Queste entità, infatti, possono essere modellate come grafi in cui ogni nodo rappresenta un computer diverso, o un dominio Internet, o una pagina web ed ogni arco simula una relazione, esistente tra i vari nodi. Ovviamente, allo stesso modo di Internet, anche gli altri tipi di rete possono essere modellati attraverso l'utilizzo dei grafi. Si può generare, infatti, il grafo delle citazioni scientifiche, oppure quello degli abbonati telefonici, o ancora il grafo estratto dalle mappe genetiche del DNA, ecc.

In questo lavoro di tesi studieremo i grafi ottenuti dalla rete sociale del cinema. Una rete sociale è una particolare tipologia di rete in cui ogni vertice può rappresentare persone, organizzazioni, o altre entità inserite in un contesto sociale. Queste entità, a loro volta, sono legate da relazioni che possono rappresentare, rapporti di collaborazione, conoscenza o influenza. Nel nostro caso, la rete del cinema è composta di nodi rappresentanti ognuno un determinato attore e da archi che legano due vertici distinti se e solo se gli attori,

corrispondenti ai vertici in esame, hanno lavorato insieme in almeno un film. Il grafo così modellato sarà quindi, per la natura della funzione di relazione associata agli archi, un grafo non orientato e privo di pesi sugli archi.

Una caratteristica comune a tutte le tipologie di reti fino ad ora elencate è la dimensione. Tutti questi network presentano, infatti, dimensioni molto elevate, essendo composte da centinaia di migliaia di nodi, anche nei casi delle reti più piccole.

La dimensione della rete risulta essere un problema da non sottovalutare per chi intende compiere studi sui network. Dimensioni elevate, infatti, comportano quasi sempre tempi di elaborazione dei dati ugualmente grandi, se non maggiorati dalla complessità delle operazioni richieste.

L'obiettivo che in questo lavoro di tesi ci siamo proposti di raggiungere è quello di trovare una soluzione efficiente per il calcolo del diametro su reti sociali di grandi dimensioni, nello specifico sulla rete del cinema.

Il problema principale, nella ricerca del diametro risiede proprio nelle dimensioni del network in esame, poichè dovendo trovare il più lungo tra tutti i percorsi minimi presenti nel grafo, si è obbligati a navigare, almeno una volta, l'intera rete. Per tentare di venire a capo di un simile problema abbiamo portato avanti uno studio sistematico, mirato ad ottenere, alla fine del percorso di analisi, la soluzione che ci eravamo prefissi di raggiungere.

Per prima cosa abbiamo cercato di comprendere su cosa avremmo dovuto lavorare. Sono stati effettuati, quindi, degli studi sulle reti sociali, analizzando la loro storia, alcune delle loro più importanti peculiarità e l'evoluzione dei modelli proposti in letteratura per rappresentare questa particolare tipologia di network. In seguito, al fine di trovare una soluzione efficiente, in termini di memoria, ma soprattutto di tempo - che rischia di crescere in maniera esponenziale rispetto al numero di vertici ed archi della rete - sono stati compiuti una serie di studi su algoritmi probabilistici di approssimazione, capaci di fornire una stima accurata della funzione di vicinanza. La funzione di vicinanza permette di investigare il vicinato di un dato nodo e quindi, con diverse tecniche, di estrapolare da questa altre utili informazioni, tra cui il diametro. Gli algoritmi proposti in letteratura che si avvicinavano alle nostre richieste, per la computazione di tale funzione, erano principalmente

riconducibili a tre.

Una volta individuato, quindi, l'algoritmo che meglio si avvicinava alle nostre esigenze, non avendone trovata alcuna implementazione che non fosse in C o C++, ne è stata progettata e sviluppata una versione in Java™, totalmente corrispondente ai nostri bisogni. Una volta in possesso del codice dell'algoritmo sono stati eseguiti i test sui grafi del cinema che si era intenzionati ad esaminare, raggiungendo conclusioni che saranno meglio specificate all'interno del seguente documento.

La struttura della tesi

La struttura fornita alla tesi è stata studiata per guidare il lettore attraverso un percorso logico-conoscitivo che gli permetta di comprendere pienamente i temi trattati. La presenza di capitoli, paragrafi e sottoparagrafi rispetta totalmente questa scelta, permettendo la suddivisione di macroconcetti, a volte troppo complessi nel loro insieme a causa della loro vastità, in trattazioni atomiche e più facilmente comprensibili.

Nello specifico, la struttura della tesi risulta essere così costituita.

Nel Capitolo 1 verranno presentate le reti in generale ed introdotti i formalismi matematici utilizzati, in ambito scientifico, per studiare tali strutture. Dopo questa prima rassegna di nozioni fondamentali, utili per una corretta comprensione del lavoro svolto, si passerà alla presentazione delle reti sociali, e all'esposizione di alcuni dei più famosi studi compiuti, negli anni, su questo tipo di reti. Si introdurranno quindi, alcuni dei principali modelli ideati, nel corso degli anni, per studiare le reti sociali. Infine saranno presentate alcune delle proprietà base delle reti con le loro definizioni.

Nel Capitolo 2 sarà introdotta *la funzione di vicinanza*. Verrà mostrata la sua utilità nell'ambito degli studi relativi alle reti e non solo. Saranno, quindi, mostrati alcuni dei campi applicativi in cui è attualmente utilizzata. In seguito, verranno introdotte le problematiche relative alla sua computazione,

specialmente per reti di grandi dimensioni e attraverso un procedimento di raffinazione della metodologia di calcolo si arriverà ad introdurre la tecnica dell'approssimazione della funzione di vicinanza, mostrando quelli che in letteratura sono i due algoritmi chiave fino ad ora usati in ambito scientifico, il Random Interval, comunemente detto RI, e il BitMap, o BM.

Nel Capitolo 3 sarà introdotto l'algoritmo scelto per lo studio che intendiamo portare avanti in questo lavoro di tesi, *Approximate Neighbourhood Function*, o ANF. Verrà mostrata, anzitutto, l'idea alla base di tale algoritmo, seguita da una sua spiegazione formale. Infine, saranno introdotte e spiegate le differenti versioni proposte in letteratura per tale algoritmo, mostrando, per ognuna di esse le migliorie che apportano alla forma di base, in termini di tempo e spazio richiesti.

Nel Capitolo 4 saranno trattati i tool e le risorse utilizzate per i nostri studi sulle reti. In particolare verrà presentato il framework *WebGraph*, sviluppato presso l'università di Milano. Saranno, inoltre, introdotti e spiegati i dati raccolti attraverso l'*Internet Movie Database* (IMDb) relativi a film ed attori. Verranno descritti i metodi di estrazione utilizzati per ottenere le informazioni con cui, in seguito, saranno generati i grafi su cui andremo ad eseguire i nostri studi. Infine, sarà presentato il package it.unimi.dsi.fastutil, sviluppato sempre presso l'università di Milano, utilizzato per migliorare l'efficienza delle operazioni svolte, all'interno del codice prodotto, con le collezioni di elementi.

Nel Capitolo 5 sarà presentato tutto ciò che è stato sviluppato per questo lavoro di tesi. Si introdurranno i grafi del cinema, spiegando cosa sono e le metodologie utilizzate per l'estrazione delle informazioni e la loro creazione. Verranno presentate l'importanza del diametro nelle reti sociali e le problematiche relative al calcolo di questa misura sulle reti di grandi dimensioni a causa del grande impiego di risorse richieste per un calcolo diretto. Si passerà quindi a descrivere la soluzione da noi proposta per tale problema, introducendo la variante ibrida di ANF, ottenuta dal merge di due delle versioni presentate dagli autori dell'algoritmo. Infine verrà descritta, nella sua interezza, l'applicazione realizzata prestando attenzione ai risultati ottenuti sia in termini di efficienza, sia relativamente allo studio dei grafi del cinema.

Nel Capitolo 6 verranno presentate le conclusioni a cui si è giunti al termine del lavoro e degli studi compiuti sulle reti di grandi dimensioni mostrando i risultati ottenuti. Si elencheranno, inoltre, eventuali sviluppi futuri che potrebbero interessare il lavoro svolto durante la tesi.

Nell'Appendice A è possibile trovare i listati di alcune delle più importanti classi prodotte, durante questo lavoro di tesi, per eseguire i nostri studi sui grafi a disposizioni.

Capitolo 1

Le Reti

"Le cose sono legate da legami invisibili: non puoi cogliere un fiore senza turbare una stella."
Galileo Galilei

In questo capitolo sarà introdotto il concetto di rete, con alcuni accenni storici ai primi studi compiuti sulle reti.

Saranno presentate alcune nozioni base di teoria dei grafi, necessarie per capire quanto esposto nei capitoli successivi.

Sarà poi effettuata una breve panoramica sui diversi modelli di reti esistenti per entrare, quindi, nello specifico delle reti sociali.

Infine saranno presentate le proprietà principali di questa tipologia di reti.

1.1 Introduzione alle reti

Una rete, in inglese network, è un insieme di entità connesse tra loro attraverso una qualche relazione. Da questa definizione è facile comprendere perchè le reti, nella letteratura matematica, sono anche chiamate comunemente grafi. In termini informali un grafo è costituito di:

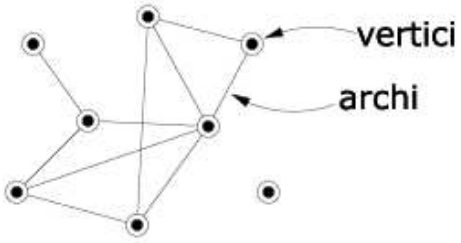

Figura 1.1: Esempio di una piccola rete composta di otto vertici e dieci archi.

oggetti semplici , detti *vertici* o *nodi*.

collegamenti tra i vertici. I collegamenti a loro volta possono essere:

orientati , in questo caso gli archi possono essere percorsi solo nel verso indicato.

non orientati , gli archi possono essere percorsi in qualunque verso.

Per modellare una rete, quindi, sarà sufficiente associare ad ogni entità un nodo e collegarlo con le altre entità con cui è in relazione attraverso degli archi.

Nel mondo sono moltissimi i sistemi che possono essere rappresentati attraverso una rete. Tra gli esempi più famosi abbiamo Internet, il World Wide Web, le reti sociali di conoscenza, le reti neurali, le catene alimentari, le mappe genetiche e molti altri ancora. Come abbiamo appena visto, reti e grafi sono fortemente legati tra loro da similitudini strutturali, ed è per questo che gli studi compiuti in ambito matematico, relativamente alla teoria dei grafi, sono considerati alla base della moderna teoria delle reti. La soluzione di Eulero al problema dei ponti di Königsberg[1] è spesso citata come il primo studio nella teoria dei grafi [3].

[1]Risolvere il problema di Königsberg significava trovare una strada intorno alla città che permettesse di attraversare ciascun ponte soltanto una volta. Nel 1736 Eulero diede vita alla teoria dei grafi, modellando il problema attraverso nodi e archi e dimostrando che, sul grafo di Königsberg, non era possibile trovare un percorso simile a meno di non inserire un nuovo ponte, come poi fu fatto nel 1875.

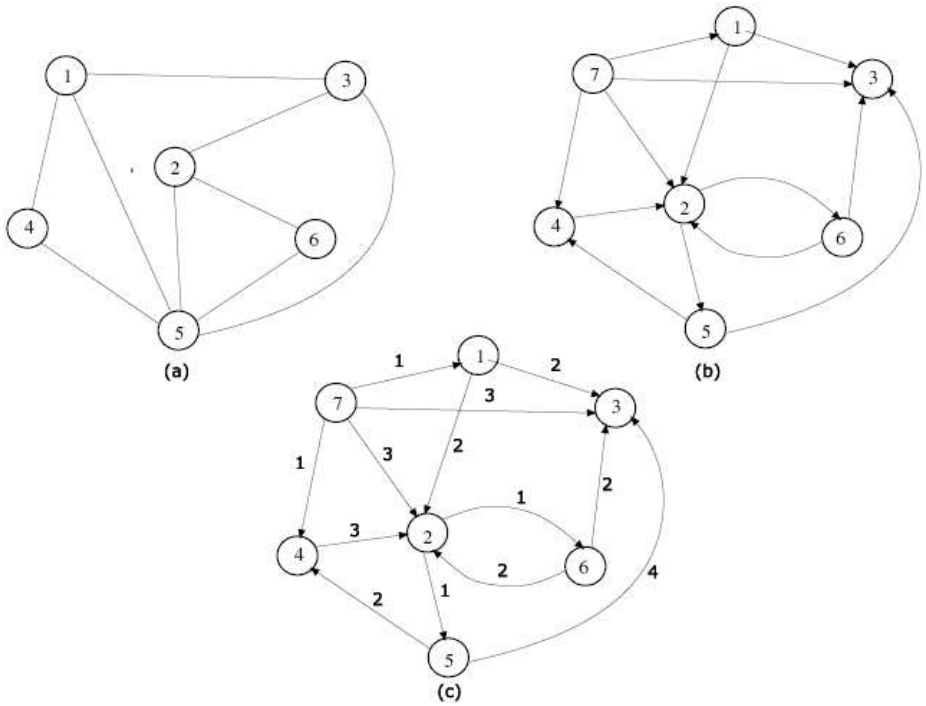

Figura 1.2: Esempio di vari tipi di grafo. (a) Grafo non orientato. (b) Grafo orientato. (c) Grafo pesato.

A partire dal ventesimo secolo la teoria dei grafi è stata ampiamente sviluppata. Le reti sono state studiate in maniera estesa anche grazie all'interesse, per questo tipo di struttura, di differenti discipline quali la sociologia, l'antropologia, la geografia, la matematica, le scienze informatiche e molte altre ancora.

1.2 Teoria dei grafi

Come detto nel paragrafo 1.1, una rete può essere modellata attraverso un insieme di vertici legati da archi. Questa, però, è solo una delle più semplici rappresentazioni di una rete. Esistono, infatti, diversi modi per rappresentare un network e alcuni di questi possono risultare piuttosto complessi. In

questo caso, però, la maggiore complessità non è considerata uno svantaggio, poichè permette di modellare un più alto numero di situazioni, grazie al supporto offertoci dalla teoria dei grafi. Per poter utilizzare i grafi, però, è prima necessario fissare alcune nozioni ed introdurre alcuni formalismi che risulteranno essenziali per la comprensione delle asserzioni che andremo a sviluppare nel prosieguo del lavoro.

Formalmente un grafo G può essere definito nel seguente modo:

Definizione 1 *Un grafo G è una coppia (V, E) tale che:*

- *V è un insieme finito e non vuoto di vertici o nodi.*
- *$\lfloor E \subseteq \{\{u, v\} | u, v \in V\}$, ovvero è un insieme di sottoinsiemi di V aventi cardinalità 2.*

Il grafo appena introdotto è detto **grafo non orientato**. In questo particolare tipo di grafo gli archi non hanno un verso e quindi possono essere percorsi in qualsivoglia direzione. Vale quindi la relazione $(i, j) = (j, i)$[2].

Un **grafo orientato**, invece, è così definito:

Definizione 2 *Un grafo orientato G è una coppia (V, E) tale che:*

- *V è un insieme finito e non vuoto di vertici o nodi.*
- *$E \subseteq V \times V = \{\langle u, v\rangle | u, v \in V\}$, ovvero è un insieme ordinato di coppie di V.*

Un *arco orientato* è un arco caratterizzato da un verso. In particolare è composto di una testa (rappresentata graficamente, di solito, dalla punta di una freccia), che raggiunge il vertice in entrata, e da una coda, che lascia il vertice in uscita. In questo caso vale la relazione $(i, j) \neq (j, i)$.

Infine si definisce **grafo pesato** una struttura che rispetta la seguente regola.

Definizione 3 *Un grafo pesato(orientato o non) G, è una coppia (V, E) che ad ogni arco ha associato un peso, tramite una funzione peso $w : E \to R$.*

[2]In questo tipo di grafi non è importante l'ordine in cui i vertici appaiono negli archi

Definiti i principali modelli di grafi, mostrati anche in Figura 1.2, è necessario introdurre dei concetti che ci permettano di muoverci agevolmente al loro interno così da poterli utilizzare in maniera efficiente. Per poterlo fare, saranno ora presentate le definizioni di *cammino*, *ciclo*, *grafo connesso* e *componente connessa*.

Definizione 4 *Si definisce* ***cammino****, in inglese* ***path****, di lunghezza p di estremi a e b in un grafo G=(V,E), una sequenza di p+1 vertici*$(u_0, u_1, \ldots, u_p)$ *tale che* $a = u_0, b = u_p$ *e* $(u_{i-1}, U_i) \in E$.

Definizione 5 *Un cammino* $(u_0, \ldots, u_n)$ *si chiama* ***circuito*** *o* ***ciclo****, se* $u_0 = u_n$ *e contiene almeno un vertice. Il ciclo è inoltre definito* ***semplice*** *se i vertici* $u_0, \ldots, u_{n-1}$ *sono a due a due distinti.*[3]

Un grafo privo di cicli è detto **aciclico**.

Definizione 6 *Un grafo G=(V,E) si dice* ***connesso*** *se c'è almeno un cammino congiungente due suoi qualsiasi vertici.*

Un grafo che non rispetta la definizione 6 si dice **sconnesso**

Definizione 7 *Se* $a \in V$*, si chiama* ***componente connessa*** *di a, l'insieme* C_a *formato da tutti i vertici* $x \in V$ *per i quali esiste un cammino che va da a ad x*

Durante l'esplorazione di un grafo può, spesso, essere utile tenere traccia della lunghezza del cammino percorso o comunque avere un'unità di misura per quantificare il numero di archi presenti. Per soddisfare questa nuova necessità saranno ora introdotte le notazioni di: *distanza*, *eccentricità*, *raggio*, *diametro* e *grado*.

Definizione 8 *Dato un grafo G=(V,E), si definisce* ***distanza*** *o* ***geodetica*** *tra due vertici* $u, v \in V$*, il numero di archi che compongono il cammino più corto per andare da u a v.*

[3]Un ciclo di lunghezza pari ad 1 è definito *cappio*.

Definizione 9 *Dato un grafo G=(V,E), si definisce* ***eccentricità*** ε *di un vertice v, con* $v \in G$*, la più grande distanza tra v e ogni altro vertice di G.*

Definizione 10 *Dato un grafo G=(V,E), si definisce* ***raggio*** *del grafo, la minima eccentricità tra tutti i vertici.*

Definizione 11 *Dato un grafo G=(V,E), si definisce* ***diametro*** *del grafo, la più lunga tra tutte le distanze. Analogamente, il diametro può essere definito come l'eccentricità massima tra tutti i vertici del grafo.*

Definizione 12 *Dato un grafo non orientato G=(V,E), si definisce* ***grado****, o* ***degree****, di un vertice* $v \in V$*, il numero di archi che da esso si dipartono. Intuitivamente questa grandezza fornisce una quantificazione dell'importanza del nodo all'interno del grafo.*

Definizione 13 *Dato un grafo orientato G=(V,E), si definisce* ***grado entrante(uscente)****, o* ***in-degree(out-degree)****, di un vertice* $v \in V$*, il numero di archi incidenti in(da) esso. Il* ***grado*** *di un vertice v, in un grafo orientato, è dato dalla somma del suo grado entrante e del suo grado uscente.*

Per rappresentare un grafo, oltre a particolari modelli che saranno introdotti nel prosieguo di questo lavoro, esistono alcune rappresentazioni di base, tra cui *le matrici di adiacenza* e *le liste d'adiacenza.*

Definizione 14 *Sia dato un grafo* $G = (V, E)$ *tale che* $V = \{1, ...n\}$ *si definisce* ***matrice di adiacenza*** *di G, la matrice booleana di ordine* $n \times n$ *i cui elementi sono definiti come segue:*

$$M_G[i,j] = \begin{cases} 1 & se\{i,j\} \in E \\ 0 & altrimenti \end{cases} .$$

Si osservi come, in un grafo G non orientato, la matrice di adiacenza M_G risulti essere simmetrica[4] rispetto alla diagonale principale.

Definizione 15 *Sia dato un grafo* $G = (V, E)$ *tale che* $V = \{1, ...n\}$ *si definisce* ***lista di adiacenza*** *di G, un vettore* L_G *di lunghezza n i cui elementi sono così definiti:*

[4]Una matrice A si definisce simmetrica quando vale la relazione $A = A^T$

$L_G[i]$ *è una lista contente tutti i vertici* j *tali che* $\{i, j\} \in E$.

Si osservi che, per un grafo non orientato, utilizzando le liste di adiacenza, ogni arco viene rappresentato due volte.

1.3 Le reti sociali

Il termine *rete sociale* è stato coniato nel 1954 da J. A. Barnes. Una rete sociale è una struttura composta di individui, organizzazioni o altre entità, inserite in un contesto sociale. Queste entità sono legate tra loro, da relazioni che possono rappresentare: interazione, rapporti di collaborazione, o influenza. È, ovviamente, possibile considerare vari tipi di relazioni: la mutua dichiarazione di amicizia, la vicinanza fisica, un'email inviata da una persona all'altra, l'esistenza di rapporti simbiotici o lavorativi e molto altro ancora.

Come qualsiasi altro network, anche le reti sociali possono essere modellate attraverso un insieme di nodi e di archi, secondo le regole della teoria dei grafi (come riportato nel paragrafo 1.1). I nodi rappresenteranno, quindi, gli attori della rete (individui, organizzazioni, molecole, ecc.), mentre gli archi simuleranno le relazioni che intercorrono tra di essi. L'utilità di una tale rappresentazione risiede nella possibilità di generare un modello visivo sul quale compiere i nostri studi. In questo modo sarà possibile, infatti, valutare in maniera immediata, alcuni aspetti della rete come, ad esempio, la presenza di nodi non connessi.

Fin dai primi anni 50 sono stati affrontati, da sociologi e psicologi, significativi studi sulle reti sociali [23]. Tali studi furono condotti su reti di piccole dimensioni, a causa della scarsa potenza dei mezzi allora a disposizione. Questo particolare, però, non ha inficiato i risultati ottenuti. Si è scoperto, infatti, che molti di quei risultati sono validi anche per reti di grandi dimensioni ed alcuni degli esperimenti eseguiti allora, sono tuttora ripresi e studiati applicandoli alle nuove reti. Internet ad esempio ha dato un nuovo e forte impulso agli studi sulle reti sociali fornendo agli studiosi una rete

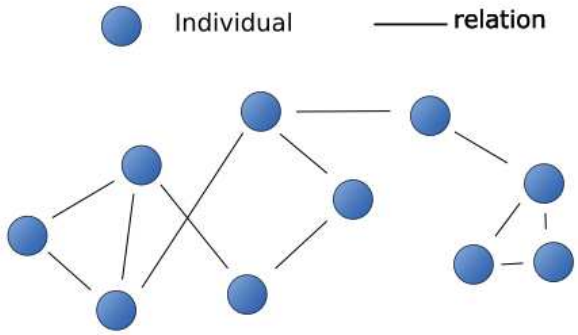

Figura 1.3: Rappresentazione di una rete sociale

di dimensioni impensabili fino a pochi decenni addietro. Uno degli esperimenti certamente più conosciuti, nell'ambito delle reti, è quello condotto da Milgram.

1.3.1 Sei gradi di separazione

L'esperimento che andremo ad introdurre e spiegare è unanimemente riconosciuto dalla comunità scientifica come uno dei più famosi studi condotti sulle reti. Ideato per ricerche in ambito sociologico, portò alla definizione del concetto dei '*sei gradi di separazione*' e alla divulgazione, anche fuori dall'ambito scientifico, di tale concetto.

L'idea che le persone presenti sul globo possano essere 'facilmente' rintracciate attraverso una corta catene di conoscenze, affonda le sue radici negli scritti di un poeta ungherese di nome *Frigyes Karinthy*, molto conosciuto nei primi del 1900. In uno dei suoi scritti, intitolato Catene[5], Karinthy ipotizzò che fosse possibile unire due persone sconosciute con un massimo di cinque passaggi intermedi. Per quanto insensata potesse apparire a quei tempi, l'intuizione dello scrittore fu la prima apparizione documentata, di quello che in seguito sarebbe stato il concetto dei sei gradi di separazione [3].

[5]Contenuto nel libro 'Ogni cosa è diversa', antologia di quarantadue racconti.

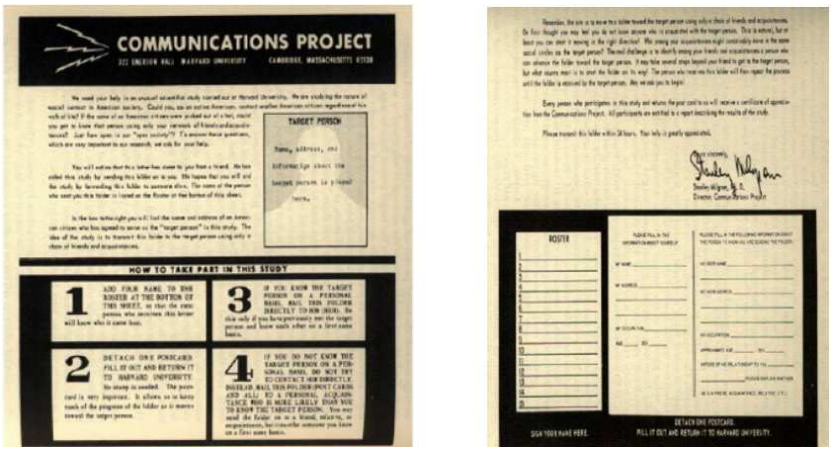

Figura 1.4: Copia originale delle istruzioni allegate da Milgram alle lettere.

Circa trent'anni dopo gli scritti di Karinthy i sei gradi di separazione vennero ripresi da Stanley Milgram, un sociologo di Harvard che, partendo da quest'intuizione, riuscì a dare vita ad un avanzato e originalissimo studio sull'interconnettività delle persone.

L'obiettivo, che Milgram si era prefisso, consisteva nel comprendere quale fosse la "distanza" che intercorreva tra due qualsiasi cittadini degli Stati Uniti. Quindi, come prima cosa, il sociologo selezionò i due destinatari finali. La scelta ricadde sulla moglie di uno studente di teologia, nella città di Sharon, nel Massachusetts, e su di un agente di cambio residente a Boston. Inviò, quindi, centosessanta lettere ad altrettanti abitanti, scelti casualmente, nelle città di Wichita e Omaha. In ogni busta erano riportate delle brevi spiegazioni sugli obiettivi dell'esperimento, una fotografia con il nome e l'indirizzo e poche altre informazioni sul destinatario finale e le istruzioni che il cittadino avrebbe dovuto seguire per far pervenire ai cittadini prescelti la missiva (Figura 1.4). I timori di Milgram sulla possibilità che l'esperimento potesse fallire, furono fugati pochi giorni più tardi, quando tornò indietro la prima lettera[6] Alla fine tornarono indietro ben quarantadue delle centosessanta lettere inviate. Con questi dati Milgram fu in grado di determinare che

[6]La lettera tornò dopo aver compiuto solo due passaggi, si trattò di quella con il percorso più breve in assoluto tra tutti quelli registrati.

per raggiungere il destinatario finale erano necessari, in media, 5,5 passaggi. Una cifra incredibilmente piccola[7] e, sorprendentemente, vicina a quella ipotizzata da Karinthy. Questo fu l'esperimento che portò alla formulazione del famoso concetto dei sei gradi di separazione[8].

1.4 Modelli di reti

Quarant'anni dopo l'originale esperimento di Milgram sarebbe fin troppo facile pensare che i risultati, a cui il sociologo era giunto, fossero ovvi e intuitivi, ma a quel tempo la scoperta dei sei gradi di separazione fu una grande sorpresa. La pubblicazione degli studi di Milgram attirò l'attenzione di molti studiosi, tra cui anche molti matematici, che negli anni a seguire crearono diversi modelli per rappresentare e spiegare i risultati ottenuti dal sociologo di Harvard.

1.4.1 Erdős e Rényi: random graph

In una serie di articoli tra il 1950 e il 1960, Paul Erdős e Alfred Rényi proposero e studiarono uno dei primi modelli teorici di rete, i **grafi randomici**, in inglese **random graph**. Questo minimale modello consiste di n vertici, collegati da archi che sono posti tra una coppia di vertici scelti in maniera casuale. Erdős e Rényi fornirono differenti versioni del loro modello. Il modello utilizzato comunemente è composto di un grafo $G_{n,p}$, in cui ogni possibile arco tra due vertici è presente con identica probabilità p, e assente con probabilità $1 - p$. Tecnicamente, infatti, $G_{n,p}$ è l'insieme dei grafi di n

[7] Le stime, richieste da Milgram prima dell'esperimento, si attestavano su un minimo di 100 passaggi.

[8] Six degree of separation. Sebbene si fondi sull'esperimento di Milgram questa dicitura è stata resa celebre solo alcuni decenni più tardi grazie all'omonima commedia di John Guare.

vertici in cui ogni grafo appare con la probabilità appropriata al suo numero di archi[9].

Sebbene i grafi randomici, introdotti da Erdős e Rényi, siano uno dei più vecchi e studiati modelli di rete, soprattutto per alcune loro proprietà che comportano, in fase di studio, considerevoli vantaggi (ad esempio la presenza di un diametro piccolo), come modello per le reti reali essi lasciano piuttosto a desiderare. In particolare questo modello, differisce dalle reti reali per due motivi principali:

- La loro distribuzione di grado (si veda il sottoparagrafo 1.5.3) segue un'irreale legge di Poisson.
- Non tengono conto del coefficiente di clustering (si veda il sottoparagrafo 1.5.2) ritenendo, erroneamente, tutti i nodi uguali, riducendo così la rete ad un unico gigantesco cluster.

1.4.2 Watts e Strogatz: the rewired ring lattice

Verso la fine del 1990, Duncan Watts e Steve Strogatz definirono un nuovo modello di grafo random per le reti sociali, tale modello era in grado di soddisfare simultaneamente il bisogno di avere un diametro piccolo e un alto coefficiente di clustering. Il loro lavoro ebbe l'effetto di suscitare nuovo interesse verso gli studi per le reti sociali, attirando molti studiosi, tra cui numerosi matematici.

Watts e Strogatz immaginarono di disporre alcune persone su di una griglia circolare k-dimensionale, nel nostro caso sarà sufficiente porre $k = 1$, e che ogni persona, u, posta sulla griglia, fosse collegata con i due nodi a sinistra, a lei immediatamente successivi. In questo modo ogni vertice avrebbe avuto esattamente quattro vicini, collegati l'un l'altro da tre link e la rete risultante avrebbe avuto un elevato coefficiente di clustering. Un

[9]Per un grafo di n vertici e m nodi questa probabilità è: $p^m(1-p)^{M-m}$, dove $M = \frac{1}{2}n(n-1)$.

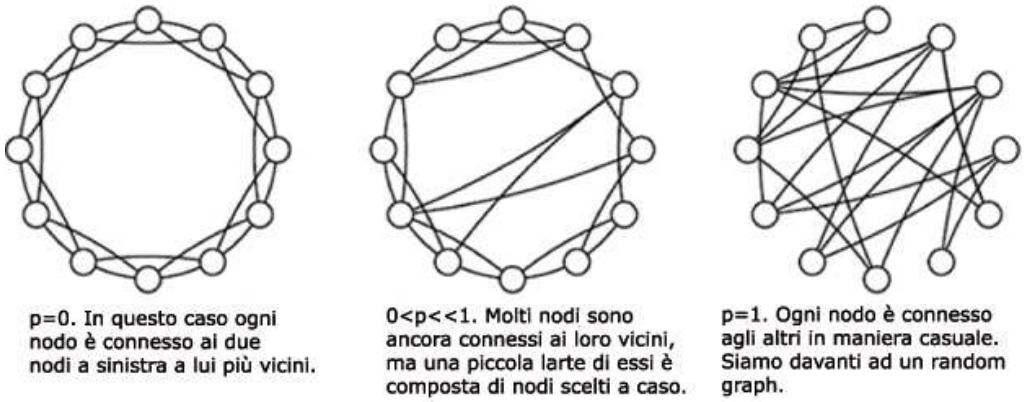

Figura 1.5: Modello di rete sociale ideato da Watts e Strogatz. Nella figura è possibile notare come al variare della probabilità p si modifichi la topologia del grafo.

coefficiente così elevato, però, avrebbe fatto svanire il fenomeno del piccolo mondo (si veda il sottoparagrafo 1.5.1, poichè in una rete simile solo i vicini immediati di un vertice sono a stretto contatto con esso. Per ovviare a questo problema, alcune connessioni nella rete furono alterate in maniera casuale. Con la medesima probabilità p, per ogni arco $\langle u, v\rangle$ nel grafo, questo sarebbe stato "ricollegato" casualmente diventando $\langle u, v'\rangle$ dove v' è scelto in maniera indipendente dall'insieme di tutti i nodi del grafo. Per valori di p pari allo zero ci troveremo nella situazione iniziale, con un alto coefficiente di clustering che impedisce la creazione di piccoli mondi poichè, in questo caso, il diametro del grafo cresce in maniera polinomiale rispetto alla grandezza della popolazione. D'altro canto per valori di $p \approx 1$ si avrà un diametro bassissimo che porterà ad un modello topologicamente e ideologicamente identico a quello dei grafi randomici di Erdős e Rényi.

Il risultato principale, degli studi condotti da Watts e Strogatz, consiste nella scoperta empirica che la rete riesce a mantenere i vantaggi di entrambe le proprietà (diametro piccolo e alto coefficiente di clustering) per valori intermedi della probabilità p.

Per valori di p compresi tra 0 e 1, infatti, l'alto coefficiente di clustering persiste, perchè molti degli archi originali rimangono immutati (la probabilità

p che questi siano ricollegati è piccola), e per due nodi vicini u e v, i nodi chiusi ad u sono, nella maggior parte dei casi, gli stessi che sono chiusi a v. Al contempo, risulta intuitivo come, il piccolo diametro appaia anche per valori relativamente piccoli di p, perchè gli archi inseriti casualmente, formano a tutti gli effetti un grafo randomico, formato da piccoli cluster di nodi vicini, in cui il diametro di ogni singolo cluster risulta piccolo.

1.4.3 Kleinberg: Un piccolo mondo navigabile

Un recente e maggiore sviluppo nella modellazione delle reti sociali è arrivato con il lavoro di Kleinberg che ha avuto il merito di compiere un'importante osservazione sull'esperimento che portò Milgram alla scoperta dello small world. L'esperimento del sociologo di Harvard mostrava due importanti peculiarità delle reti sociali. Non solo, infatti, esistono corte catene di "amici" tra due vertici scelti a caso nella rete, ma i membri della rete sono in grado di costruire cammini brevi utilizzando solo le informazioni 'locali', rappresentate dai nodi a cui sono direttamente connessi. In altre parole piuttosto che vedere l'esperimento di Milgram come un risultato relativo alla grandezza del diametro di una rete sociale, Kleinberg considerò l'esito dell'esperimento come il risultato relativo al successo di un particolare algoritmo di routing applicato al grafo.

Compiuta questa osservazione, Kleinberg dimostrò che il modello di Watts e Strogatz non forniva informazioni sufficienti per la creazione del cammino da intraprendere. Sviluppò, quindi, un estensione del modello di Watts e Strogatz, consistente in una rete con una maglia k-dimensionale, dove ogni vertice era a conoscenza dei suoi immediati vicini, in ogni direzione, e la probabilità dell'esistenza di un collegamento a lunga distanza che va da u a v è proporzionale a $1/d(u,v)^{\alpha}$, per qualche costante $\alpha \geq 0$, dove $d(u,v)$ denota la distanza di Manhattan[10]. In questo modello l'aggiunta degli archi in base

[10]La distanza di Manhattan è una grandezza data dalla somma delle differenze in valore assoluto delle componenti dimensionali delle posizioni di u e v. $\sum_{i=0}^{k} |u_i - v_i|$.

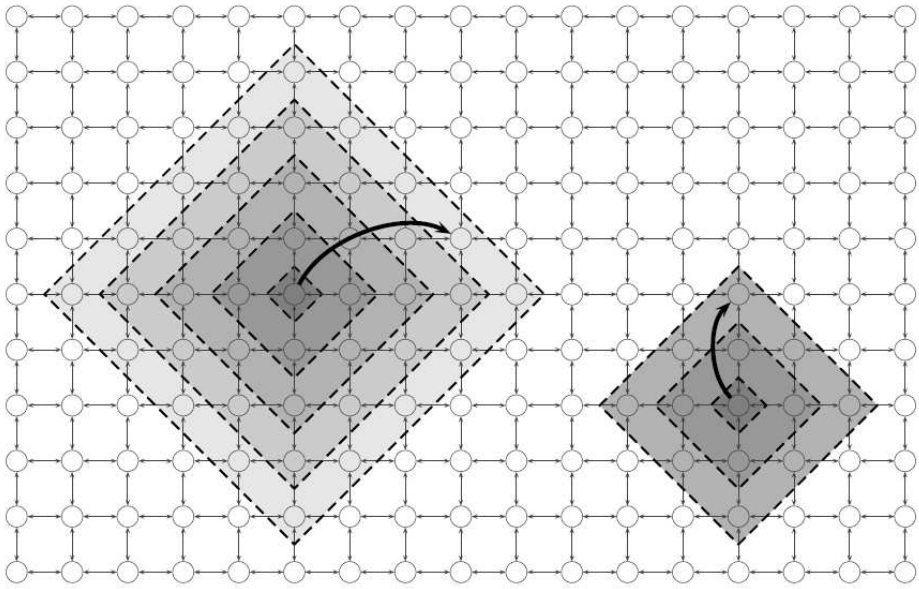

Figura 1.6: Modello di rete sociale ideato da Kleinberg. Nel modello costruito in figura la maglia k-dimensionale è ottenuta assegnando $k = 2$. Le aree più scure indicano una più alta probabilità di trovare un arco.

ad una probabilità p, viene generalizzata dalla presenza della costante α. Porre, infatti, $\alpha = 0$ equivale a scegliere un arco diretto.

Il principale risultato a cui Kleinberg approdò fu la caratterizzazione di un teorema in grado di stabilire esattamente quando un algoritmo, basato su informazioni locali, è in grado di costruire un cammino corto tra i nodi di un grafo.

1.4.4 Barabási e Albert: Reti ad invarianza di scala

Studiando la struttura del Web, attraverso l'utilizzo di un Web crawler[11], il fisico Albert Barabási scoprì, con non poca sorpresa, che questa non presentava una connettività casuale. Notò, invece, la presenza di alcuni nodi con un

[11] Un web crawler, conosciuto anche come web spider, è un programma capace di navigare il World Wide Web in maniera automatizzata e metodica per raccogliere informazioni.

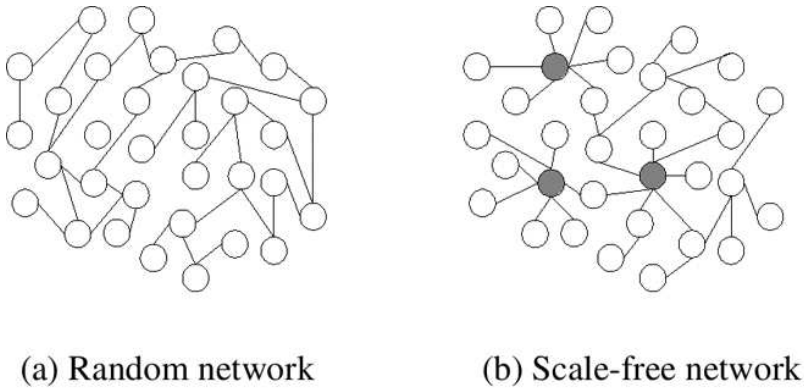

Figura 1.7: (a) Rete randomica. (b) Rete ad invarianza di scala. Nella rete ad invarianza di scala gli hub sono i nodi evidenziati.

numero di archi insolitamente elevato, che vennero denominati *hub* o *connettori*. Studi successivi mostrarono che gli hub non erano un fenomeno isolato, ma rappresentavano una caratteristica importante di sistemi complessi anche molto diversi tra loro, dall'economia alla cellula vivente. L'ammissione dell'esistenza dei connettori ha portato al definitivo abbandono di ogni prospettiva casualista nelle reti suscitando, al contempo, la necessità di un nuovo modello che tenesse conto dell'esistenza di questo particolare tipo di nodi.

Le *reti ad invarianza di scala* o, in inglese, *scale-free networks*[12] sono la classe di reti che presentano una distribuzione di grado di tipo power law (legge di potenza) e che quindi meglio si prestano per studiare la presenza degli hub e i loro effetti.

Il modello maggiormente utilizzato per la generazione di modelli atti a studiare reti ad invarianza di scala è quello proposto dalla coppia di scienziati Barabási ed Albert, nel 1999, denominato *"i ricchi sono sempre più ricchi"*. In questo tipo di modello, ogni nuovo nodo crea un collegamento verso un altro nodo della rete non con una probabilità uniforme, come accadeva invece per i grafi randomici di Erdős e Rényi, ma in maniera proporzionale al grado entrante corrente dei nodi. Secondo questo sistema un nodo con molti archi

[12]Con il termine scale-free si fa riferimento a ogni funzione nella forma $f(x)$ che rimane invariata, a meno di una fattore moltiplicativo, seguendo il ridimensionamento della variabile indipendente x $f(ax) = bf(x)$.

entranti continuerà ad attrarre altri archi rispetto ad una pagina normale e questo è in linea anche con il principio dell' *80/20* formulato da Pareto[13].

Nello specifico Barabási ed Albert, per formulare il loro modello, partirono dalla constatazione che le reti, nel tempo, tendono a modificare la loro dimensione aumentando il numero di nodi e per fare questo i nuovi nodi, utilizzano un *collegamento preferenziale*, sfruttano cioè nodi altamente connessi, gli hub, piuttosto che nodi con un minore numero di archi. La probabilità che un nuovo nodo generi un arco per collegarsi ad un vertice di grado k è pari a:

$$\frac{kp_k}{\sum_k kp_k} = \frac{kp_k}{2m}. \tag{1.1}$$

La sommatoria posta al denominatore è pari al grado medio della rete che è circa $2m$, poichè ci sono m archi per ogni vertice aggiunto, e ogni arco, essendo non orientato, contribuisce due volte al grado dei vertici della rete. Il numero medio di vertici di grado k che acquistano un arco quando un vertici con m archi viene collegato alla rete è pari a $m \times kp_k/2m = \frac{1}{2}kp_k$ ed è indipendente da m. Il numero np_k di vertici con grado k in questo modo viene decrementato da questo stesso insieme, poichè i vertici che ottengono un nuovo arco diventano di grado $k+1$. Anche il numero di vertici di grado k aumenta a causa dell'influenza dei vertici precedentemente di grado $k-1$ che ora hanno acquisito un nuovo arco. Unica eccezione sono i vertici di grado m che hanno esattamente influenza 1. Se definiamo da p_k, n il valore di p_k quando il grafo ha n vertici, allora il cambiamento della rete in np_k per ogni vertice aggiunto è:

$$(n+1)p_{k,n+1} - np_{k,n} = \frac{1}{2}(k-1)p_{k-1,n} - \frac{1}{2}kp_k \tag{1.2}$$

per $k > m$ oppure

[13]La cosiddetta "legge 80/20" è una legge empirica, nota anche con il nome di principio di Pareto. É sintetizzabile nell'affermazione: la maggior parte degli effetti è dovuta ad un numero ristretto di cause. Secondo la legge 80/20, in genere l'80% dei risultati dipende dal 20% delle cause.

$$(n+1)p_{m,n+1} - np_{m,n} = \frac{1}{2}mp_{m,n} \tag{1.3}$$

per $k = m$ e non ci sono vertici con $k < m$.

1.5 Proprietà delle reti

Definiti i principali modelli delle reti sociali ed esposti i formalismi utilizzati per la loro rappresentazione, passeremo ora ad introdurre alcune importanti peculiarità di questo tipo di reti.

1.5.1 Il piccolo mondo

L'osservazione dei dati, prodotti dall'esperimento di Milgram, introdotto nel sottoparagrafo 1.3.1, ha portato a notare come molte coppie di nodi, anche in reti differenti, sembrino essere connesse attraverso un cammino molto corto se confrontato con la grandezza della rete. Questa osservazione è alla base di quello che in letteratura è conosciuto come fenomeno del *piccolo mondo* o, in inglese, *small world*.

Sebbene già prima dell'esperimento di Milgram, circolassero lavori e dissertazioni sullo small world, solo in seguito alle pubblicazioni di Milgram gli altri lavori videro la luce. Oggi il fenomeno del piccolo mondo è stato studiato e verificato direttamente in un largo numero di reti differenti.

Considerato un grafo non orientato, sia ℓ la distanza più corta tra le coppie di vertici della rete:

$$\ell = \frac{1}{\frac{1}{2}n(n+1)} \sum_{i \geq j} d_{ij} \tag{1.4}$$

dove d_{ij} è la distanza (si veda definizione 8 nel paragrafo: 1.2) tra il nodo i

e il nodo j. C'è da notare che l'equazione:1.4 include, nella media, anche la distanza di un nodo da se stesso, anche se quest'ultima è pari a 0. Nonostante sia possibile anche non optare per questa scelta, in questo caso si è deciso di includere anche la distanza di un nodo da se stesso così da moltiplicare ℓ per $(n-1)/(n+1)$ ed apportare una correzione al valore di ℓ dell'ordine di n^{-1}.

La quantità ℓ può essere ricavata, in una rete di n vertici e m archi, in tempo asintoticamente pari a $O(mn)$, applicando l'algoritmo del *breadth first search* (visita in ampiezza).

In Tabella: 1.1 vengono riportati i valori di ℓ, conosciuti in letteratura, per alcune differenti tipologie di reti. Come mostrato, i valori sono, in tutti i casi, piuttosto bassi, ed estremamente più piccoli rispetto al numero di vertici presenti nella rete.

La definizione di ℓ presenta un problema per reti con più componenti. Può accadere, in alcuni casi, che esista un vertice che non sia connesso con nessun altro nodo della rete. In questo caso, secondo le definizioni ereditate dalla teoria dei grafi, la geodetica tra vertici simili assumerebbe valore infinito e di conseguenza anche il valore di ℓ diventerebbe infinito. Per ovviare a questo problema, si definisce ℓ solo sui vertici che sono dotati di almeno un cammino. Un'alternativa, apparentemente migliore, potrebbe essere quella di definire ℓ nel seguente modo:

$$\ell^{-1} = \frac{1}{\frac{1}{2}n(n+1)} \sum_{i \geq j} d_{ij}^{-1} \tag{1.5}$$

Attraverso l'equazione: 1.5 il valore infinito di d_{ij} non contribuirà in alcun modo alla somma. Sebbene quest'approccio risulti, in termini matematici, più corretto è raramente adottato poichè inutile ai fini della pratica.

Il fenomeno del piccolo mondo ha ovvie implicazione per la dinamica dei processi relativi alle reti. Per esempio, se considerassimo la diffusione delle informazioni attraverso una rete, in presenza di piccoli mondi, tale diffusione dovrebbe avvenire con maggiore velocità. Il fenomeno del piccolo mondo è ben evidenziato anche da giochi come il calcolo del numero di Erdős o di Kevin Bacon.

D'altro canto, il fenomeno del piccolo mondo ha anche una chiara esplica-

Rete	**Tipo**	**Nodi n**	**Archi m**	**Grado z**	ℓ	$C^{(1)}$	$C^{(2)}$
film actors	non orientato	449913	25516482	113,43	3,48	0,20	0,78
company directors	non orientato	7673	55392	14,44	4,60	0,59	0,88
math coautorship	non orientato	253339	496489	3,92	7,57	0,15	0,34
physics coauthorship	non orientato	52909	245300	9,27	6,19	0,45	0,56
biology coauthorship	non orientato	1520251	11803064	15,53	4,92	0,088	0,60
telephone call graph	non orientato	47000000	80000000	3,16	—	—	—
email messages	orientato	59912	86300	1,44	4,95	—	0,17
email address books	orientato	16881	57029	3,38	5,22	0,17	0,13
student relationship	non orientato	573	477	1,66	16,01	0,005	0,001
WWW nd.edu	orientato	269504	1497135	5,55	11,27	0,11	0,29
WWW Altavista	orientato	203549046	2130000000	10,46	16,18	—	—
citation network	orientato	783339	6716198	8,57	—	—	—
Roget's Thesaurus	orientato	1022	5103	4,99	4,87	0,13	0,15
word co-occurrence	non orientato	406902	17000000	70,13	—	—	0,44
power grid	non orientato	4941	6594	2,67	18,99	0,10	0,080
train routes	non orientato	587	19603	66,79	2,16	—	0,69
software packages	orientato	1439	1723	1,20	2,42	0,070	0,082
electronic circuits	non orientato	24097	53248	4,34	11,05	0,010	0,030
peer-to-peer network	non orientato	880	1296	1,47	4,28	0,012	0,011
metabolic network	non orientato	765	3686	9,64	2,56	0,090	0,67
protein interactions	non orientato	2115	2240	2,12	6,80	0,072	0,071
marine food web	orientato	135	598	4,43	2,05	0,16	0,23
freshwater food web	orientato	92	997	10,84	1,90	0,20	0,087
neural network	orientato	307	2359	7,68	3,97	0,18	0,28

Tabella 1.1: Statistiche di base per alcune reti conosciute. Le proprietà misurate sono le seguenti: Tipologia di rete, grafo orientato o non orientato, totale dei nodi, totale degli archi, grado, distanza tra i vertici(ℓ), coefficiente di clustering ($C^{(1)}$) ottenuto dall'eq. 1.6, coefficiente di clustering ($C^{(2)}$) ottenuto dall'eq. 1.9,

zione matematica. Se il numero di vertici con distanza r da un tipico vertice centrale cresce in maniera esponenziale rispetto ad r allora il valore di ℓ crescerà con un andamento pari a $\log n$. Negli ultimi anni si è infine giunti alla conclusione che una rete presenta piccoli mondi, se il valore di ℓ cresce in maniera logaritmica (o più lenta) rispetto alla grandezza della rete.

1.5.2 Coefficiente di clustering

Una chiara conseguenza del comportamento dei grafi randomici può essere vista nella proprietà di transitività delle reti, in inglese network transitivity, conosciuta anche come clustering. Attraverso gli studi eseguiti ci si è accorti che in molte reti, se il vertice A è connesso al vertice B e questo a sua volta è connesso con il vertice C, allora con grande probabilità, anche il vertice A sarà collegato al vertice C. Trasferendoci nell'ambito delle reti sociali possiamo riassumere questo concetto con la seguente affermazione: "l'amico di un nostro amico è spesso anche esso un nostro amico". Analizzando la topologia delle reti il concetto di transitività indica la presenza di un alto numero di triangoli[14] all'interno della rete stessa. La transitività puo essere quantificata attraverso la definizione di un **coefficiente di clustering** C definito nel seguente modo:

$$C = \frac{3 \times \text{Numero di triangoli presenti nella rete}}{\text{Numero di triple connesse di vertici}} \tag{1.6}$$

dove con *triple connesse* si intende un singolo vertice con archi che conducono ad un altro paio di vertici.

In effetti, C misura la frazione di triple che con il loro terzo arco riescono a chiudere il triangolo. Il fattore 3 che compare al numeratore dell'equazione: 1.6 è giustificato dal fatto che ogni triangolo fornisce un insieme di tre triple e garantisce che C rimanga nell'intervallo $0 \leq C \leq 1$. In termini semplici C rappresenta la probabilità che due vertici vicini ad un terzo vertice nella

[14]Insieme di tre vertici in cui ognuno dei tre è collegato a tutti gli altri.

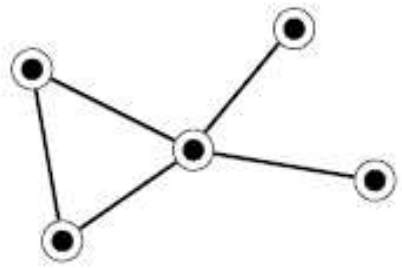

Figura 1.8: Illustrazione della definizione di coefficiente di clustering C, Eq.1.6. Questa rete mostra un triangolo e otto triple connesse. Ha quindi un coefficiente di clustering pari a $\frac{3}{8}$. I singoli vertici hanno un coefficiente, Eq.1.8, pari a 1, 1, $\frac{1}{6}$, 0, 0 per un valore complessivo di C, Eq.1.9, pari a $\frac{1}{6}$.

rete siano vicini anche tra loro. Questo concetto può essere formalizzato attraverso la seguente equazione:

$$C = \frac{6 \times \text{Numero di triangoli presenti nella rete}}{\text{Numero di cammini con lunghezza 2}} \tag{1.7}$$

dove il path di lunghezza 2 si riferisce ad un cammino che parte da uno specifico nodo.

Le definizioni del coefficiente di clustering fornite fino ad ora sono largamente usate in sociologia. In ambito matematico esiste, però, un'ulteriore definizione di questo concetto, derivata dal modello di rete ideato da Watts e Strogatz [24], i quali proposero la seguente equazione:

$$C_i = \frac{\text{Numero di triangoli connessi al vertice i}}{\text{Numero di triple centrato sul vertice i}} \tag{1.8}$$

Per vertici i con grado pari a 0, per cui numeratore e denominatore sono pari a 0, assumiamo $C_i = 0$. In questo modo il coefficiente di clustering per l'intera rete può essere ottenuto dalla media dei singoli C_i

$$C = \frac{1}{n} \sum_i C_i \tag{1.9}$$

L'equazione: 1.9 tende a pesare il contributo dei vertici con un grado basso in maniera molto più pesante[15]. Questo può portare, come accade nell'esempio di Figura: 1.8, ad avere risultati anche molto differenti tra le equazioni: 1.9 e 1.6. Il coefficiente di clustering risulta essere particolarmente importante nei grafi orientati nel caso di archi che puntano in versi opposti. In un grafo orientato probabilità che due vertici, puntino l'uno verso l'altro, è detta *reciprocità*.

1.5.3 Distribuzione di grado

Definiamo p_k come il sottoinsieme di vertici della rete che presentano grado (si veda la definizione:12 nel paragrafo: 1.2) pari a k. Equivalentemente p_k può essere vista come la probabilità che un vertice scelto a caso abbia grado uguale a k. Un grafico di p_k, per ogni rete data, può essere formato da un istogramma composto dei gradi dei singoli vertici della rete. L'istogramma così costruito rappresenterà la distribuzione di grado della rete.

In un grafo randomico, del tipo studiato da Erdős e Rényi, ogni arco è presente o assente con uguale probabilità, e quindi la distribuzione di grado è di tipo binomiale, o di Poisson, nel caso di reti di grandi dimensioni. Dagli studi compiuti è però emerso che le reti reali sono ben diverse dai grafi randomici per quanto riguarda la loro distribuzione. Lontane dall'avere una distribuzione di Poisson, il grafico del grado dei vertici in molte reti reali è rappresentato da una curva altamente inclinata verso destra. Ciò significa che la loro distribuzione ha una lunga coda di valori che sono molto lontani dalla media. Misurare la coda risulta però spesso ingannevole, poichè, sebbene in teoria si sia costruito un grafico dei gradi, nella pratica quasi mai si hanno dati sufficiente per ottenere buone statistiche dalla coda. Per questo motivo gli istogrammi diretti sono utilizzati molto raramente per il calcolo della distribuzione di grado.

[15] I vertici con basso grado hanno un denominatore piccolo nell'equazione.1.9.

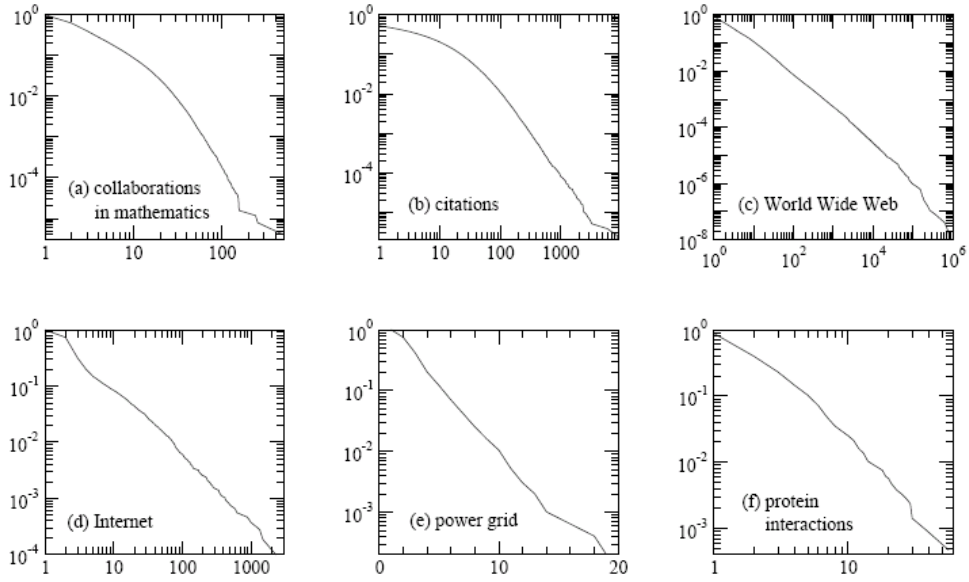

Figura 1.9: Distribuzione di grado cumulativa per sei differenti tipologie di network. L'asse orizzontale rappresenta il vertice di grado k. L'asse verticale mostra invece la probabilità cumulativa della distribuzione di grado, cioè la frazione di vertici che hanno grado maggiore o uguale a k.

Per rappresentare la distribuzione di grado di tali reti si è quindi pensato di costruire un grafico attraverso una funzione di distribuzione così definita:

$$P_k = \sum_{k'=k}^{\infty} p_{k'}. \tag{1.10}$$

L'equazione 1.10 rappresenta la probabilità che il grado dei nodi sia maggiore o uguale a k. Il grafico, costruito in tal modo, avrà il vantaggio di rappresentare tutti i dati originali, senza alcuna approssimazione. In Figura: 1.9 sono mostrate alcune distribuzioni di grado cumulative, riferite a differenti tipologie di reti. Come evidenzia la figura, le distribuzioni sono tutte raffigurate da curve inclinate verso destra e molte di loro risultano essere di tipo power-law lungo le code: $p_k \sim k^{-\alpha}$ per α esponente costante. Si noti che tali distribuzioni rimangono di tipo power-law anche nella distribuzione cumulativa, ma con esponente $\alpha - 1$.

$$P_k \sim \sum_{k'=k}^{\infty} k'_{-\alpha} \sim k^{-(\alpha-1)}. \tag{1.11}$$

Alcune altre reti presentano distribuzioni con code di tipo esponenziale: $p_k \sim e^{-k/\kappa}$. Anche questo tipo di distribuzioni rimangono esponenziali una volta sommate, ma mantengono invariato anche l'esponente.

$$P_k = \sum_{k'=k}^{\infty} p_k \sim \sum_{k'=k}^{\infty} e^{-k'/\kappa} \sim e^{-k'/\kappa}. \tag{1.12}$$

Le equazioni 1.11 e 1.12 rendono le distribuzioni di tipo power-law ed esponenziali particolarmente facili da tracciare sperimentalmente attraverso il grafico delle corrispondenti distribuzione cumulative. Per altri tipi di reti la distribuzione di grado risulta estremamente complessa da calcolare. Per grafi bipartiti, ad esempio, ci sono due distribuzioni di grado da calcolare, una per ogni tipo di vertice. Per grafi orientati ogni vertice possiede un *in-degree* e un *out-degree*, la distribuzione di grado sarà, quindi, una funzione p_{jk} di due variabili, rappresentante il sottoinsieme di vertici che hanno, al contempo, in-degree pari a j e out-degree pari a k.

Le reti con distribuzione di grado power-law sono state al centro di numerosi studi in letteratura. Spesso ci si riferisce loro anche con il termine di reti a invarianza di scala, sebbene sia solo la loro distribuzione di grado ad essere scale-free.

La forma della distribuzione di $P(k)$ implica l'esistenza di alcuni nodi, all'interno della rete, con grado altissimo, i cosiddetti *hub*, presentati nel modello di Barábasi e Albert (si veda il sottoparagrafo 1.4.4).

Capitolo 2

La funzione di vicinanza

"Anche se può sembrare un paradosso,
tutte le scienze esatte sono
dominate dall'approssimazione".
Bertrand Russell

In questo capitolo sarà trattata la funzione di vicinanza e la sua importanza ed utilità nello studio delle reti ed in particolare delle reti sociali.

Sarà, inoltre, compiuta una panoramica sullo stato dell'arte, mostrando alcuni degli algoritmi più conosciuti ed utilizzati in letteratura per il calcolo di tale funzione.

Verrà esposto, quindi, uno studio approfondito di due di questi algoritmi. Precisamente saranno analizzati l'algoritmo *BitMap*, proposto Flajolet e Martin, e il *Random Interval*, proposto da Cohen.

2.1 Introduzione alla funzione di vicinanza

La *funzione di vicinanza*, $N(h)$ in inglese *neighbourhood function*, può essere definita informalmente come il numero di coppie di nodi che si trovano

a distanza h. Lo studio della funzione di vicinanza fornisce informazioni utili per grafi differenti come la struttura di un documento XML, web graph, il grafo dei film, ecc.

- "Qual'è il diametro di un grafo?"
- "Il grafo dei tabulati telefonici americano è simile a quello europeo?"
- "Il grafo delle citazioni Fisiche differisce da quello delle citazioni Matematiche?"

Queste sono solo alcune delle domande a cui la funzione di vicinanza può aiutarci a rispondere, ma per far questo è utile introdurla formalmente. Dato un grafo $G = (V, E)$, si definisce:

Definizione 16 ***Funzione individuale di vicinanza (Individual neighbourhood function)*** *per $u \in V$, il numero di nodi a distanza h, o minore, da u.*

$$S(u, h) = |v : v \in V, d(u, v) \leq h|.$$

Definizione 17 ***Funzione di vicinanza (Neighbourhood function)****, il numero di nodi a distanza h.*

$$N(h) = |(u, v) : u \in V, v \in V, d(u, v) \leq h|, \textit{ oppure}$$

$$N(h) = \textstyle\sum_{u \in V} S(u, h).$$

La funzione di vicinanza può essere utilizzata in diversi campi di studio. Uno di questi campi è sicuramente *l'ottimizzazione del path-query.* Gli l'ottimizzatori di query, OODMS e XML, hanno bisogno, infatti, di poter valutare il costo del cammino di una query e calcolare il costo di espansione di un path-query, sopra h passi, è analogo a voler calcolare la funzione di vicinanza. Un altro campo in cui la funzione di vicinanza è spesso utilizzata è la *comprensione dei dati.*

Molti ricercatori, attualmente, stanno utilizzando la funzione di vicinanza per comprendere Internet e il Web. Esperimenti condotti recentemente sono

atti a studiare la struttura del Web utilizzando $S(u,h)$ con $h = \infty$. La funzione di vicinanza viene utilizzata anche relativamente alle reti sociali, per calcolare la distribuzione delle distanze dei nodi, così da poter studiare il fenomeno dello small world. Infine la funzione di vicinanza è utilizzata anche per compiere studi topologici sui grafi. Tale funzione permette, infatti, di comparare due grafi e misurare le similitudini relative alla connettività dei due schemi. Rende possibile, inoltre, stimare l'importanza di un nodo, all'interno di un grafo, in base alla sua connettività.

Nonostante questa sua grande utilità la funzione di vicinanza su grafi di grandi dimensioni non viene quasi mai viene calcolata espressamente, poichè il processo di computazione risulta estremamente costoso. Per questo motivo sono stati introdotti diversi metodi per fornirne un'approssimazione quanto più esatta.

Per poter essere utilizzata in tutte queste applicazioni la funzione di vicinanza, o una sua approssimazione, deve soddisfare le seguenti proprietà:

- *Garanzia di errore.* Deve essere in grado di stimare $N(h)$ per tutti gli h con un alto livello di accuratezza.
- *Veloce.* Deve crescere linearmente rispetto al numero di nodi e di archi.
- *Bassi requisiti di memorizzazione.* Per grafi con n nodi, la memoria addizionale utilizzata deve essere $O(n)$.
- *Adattarsi alla memoria disponibile.* memoria $O(n)$ potrebbe essere troppo grande per il grafo del Web, ma deve essere in grado di usare $O(n)$ memoria esterna e meno RAM.
- *Parallelizzabile.* Per gestire efficientemente grafi molto grandi deve essere in grado di essere elaborata da più processori
- *Scansione sequenziale degli archi.* Accesso agli archi utilizzando scansioni sequenziali per migliorare le performance e permettere di memorizzare gli archi in memoria secondaria.
- *Stimare* $|S(u,h)|$. In aggiunta alla stima di $N(h)$ alcune volte risulta utile poter calcolare anche $S(u,h)$.

2.2 Stato dell'arte

Calcolare la funzione di vicinanza, $N(h)$, per $h = 0$ o $h = 1$ risulta alquanto banale. Nel primo caso, infatti, $N(h)$ sarà pari alla cardinalità dell'insieme V, poichè ogni nodo a distanza $h = 0$ ha come vicino, solamente se stesso. Allo stesso modo per $h = 1$ si può intuitivamente affermare che $N(h)$ avrà valore pari a $|V| + |E|$ per grafi orientati e $|V| + 2|E|$ per grafi non orientati. Il discorso si complica per valori di $h \geq 2$. In questo caso, infatti, esistono differenti metodi per calcolare la funzione di vicinanza, ma molti di questi richiedono, anche con grafi di piccole dimensioni, tempi di calcolo molto elevati che diventano quindi improponibili con grafi di grandi dimensioni, come ad esempio il Web.

Uno degli approcci più intuitivi per calcolare il valore di $N(h)$ per $h \geq 2$ consiste nel moltiplicare più volte la matrice di adiacenza del grafo (si veda la definizione :14 nel paragrafo: 1.2). Applicando alcuni accorgimenti all'algoritmo di moltiplicazione delle matrici questo metodo impiegherà un tempo asintoticamente pari a $O(n^{2.81})$ per processare un grafo con n nodi. Occuperà, inoltre, uno spazio in memoria pari a $O(n^2)$. Questi valori, come si può facilmente comprendere non sono proponibili per grafi di milioni di nodi come il grafo del web. Per ridurre la quantità di memoria necessaria, solitamente si usa una visita in ampiezza dei nodi del grafo. Con la visita in ampiezza, partendo dal nodo u si può facilmente calcolare $S(u, h)$ per tutti gli h. In questo modo, dopo aver calcolato $S(u, h)$ per tutti gli h, potremo ricavare la funzione di vicinanza attraverso la relazione:

$$N(h) = \sum_{u \in V} |S(u, h)|.$$

Così la memoria necessaria, per il calcolo di $N(h)$, si riduce a $O(n + m)$ per un grafo di n nodi e m archi ed il tempo di calcolo, nel caso peggiore è pari a $O(nm)$. Tali risultati, seppure migliori di quelli ottenuti in precedenza, si dimostrano ancora troppo elevati per grafi di grandi dimensioni.

Il problema di fondo, che porta a tempi e risorse così elevati, risiede nel voler calcolare esplicitamente $S(u, h)$. Si è mostrato come $S(u, i)$ possa essere

computato prendendo ogni arco (u,v) e appendendo ad esso il cammino di lunghezza $i-1$ o minore che parte da v. In questo modo avremo:

$$S(u,0) = \{u : u \in V\}$$

e poi, iterando sull'insieme degli archi per la i-esima volta potremmo calcolare $S(u,i)$:

$$S(u,i) = \{v' : (u,v) \in E \text{ and } v' \in S(v,i-1)\}.$$

Quindi, processando gli archi per h volte, otterremo i valori di $S(0), S(1), ...S(h)$. Questo approccio risulta però totalmente impraticabile, poichè, gestire un insieme di valori così vasto, richiederebbe quantitativi troppo elevati di tempo e memoria. Diversi studi hanno dimostrato che una valida alternativa a tale metodo consiste nell'utilizzare algoritmi basati sul *conteggio approssimativo*, in inglese *approximate counting*, per rimpiazzare l'insieme di operazioni appena descritte. Questi tipi di algoritmi possono generare un'approssimazione della funzione di vicinanza.

Gli algoritmi per il conteggio approssimativo sono utilizzati, per lo più, per approssimare il numero di oggetti distinti all'interno di un multi-insieme.

In letteratura esistono due differenti metodi per effettuare il conteggio approssimativo della funzione di vicinanza. Il primo, comunemente chiamato BitMap (BM), per l'uso, all'interno della sua procedura caratteristica, di un vettore di bit, è stato introdotto da Flajolet e Martin [14]. Il secondo approccio, conosciuto come Random Interval (RI), è stato proposto e sviluppato da Cohen [9].

Entrambe gli algoritmi citati sono atti a risolvere il medesimo problema: dato un insieme X di N elementi e un multi-insieme M di elementi provenienti da X, stimare il numero di elementi distinti in M.

2.3 L'algoritmo BitMap

L'algoritmo *BitMap*, presentato da Flajolet e Martin, si basa su di una procedura caratteristica denominata **COUNT** che ora andremo ad introdur-

re e spiegare.

Assumiamo, prima di tutto, che esista una funzione di hashing, di cui disponiamo, capace di trasformare i record, in interi distribuiti in maniera sufficientemente uniforme su un insieme di stringhe binarie di lunghezza L:

$$\textbf{funzione}\ \text{hash(x:record)}\ \textit{binary string}\ \{0 \dots 2^L - 1\}$$

Per ogni intero non negativo y, rappresenteremo $bit(y,k)$ come il k-esimo bit nella trasposizione binaria di y, cosicchè:

$$\sum_{k \geq 0} bit(y,k) 2^k.$$

Introduciamo, inoltre, $\rho(y)$ che rappresenta la posizione del 1-bit meno significativo nella trasposizione binaria di y. Per convenzione, $\rho(y)$ risulta essere così definita:

$$\begin{array}{lll} \rho(y) & = min_{k \geq 0} \quad bit(y,k) \neq 0 & se \quad y > 0 \\ \rho(y) & = L & se \quad y = 0. \end{array}$$

È possibile notare che, quando i valori di hash(x) sono uniformemente distribuiti, il pattern $0^k 1 \dots$ appare con probabilità pari a 2^{-k-1}. Partendo da tale dato, Flajolet e Martin, svilupparono l'idea di memorizzare le osservazioni sull'occorrenza di tali pattern in un vettore $BITMAP[0 \dots L-1]$.

Se M è il multi-insieme di cui siamo interessati a calcolare la cardinalità, è possibile utilizzare il seguente algoritmo: In questo modo, $BITMAP[i]$

avrà valore pari ad 1 se, al termine dell'esecuzione, all'interno dei valori degli hash ottenuti dai record di M, sarà apparso un pattern nella forma $0^i 1$. È importante notare che *per costruzione* il vettore $BITMAP$ dipende solo dall'insieme dei valori degli hash e non dalla particolare frequenza con cui tali valori possono ripetersi. Per questo motivo è lecito aspettarsi che la probabilità di marcare uno dei pattern, se n è il numero di elementi distinti in M, sia pari a $n/2$ volte per $BITMAP[0]$, pari a $n/4$ volte per $BITMAP[1]$ ecc. Alla fine $BITMAP[i]$ risulterà essere pari 0 se $i \gg log_2 n$, 1 se $i \ll log_2 n$ e compreso tra 0 e 1 per valori di $i \approx log_2 n$.

Algoritmo 1 Procedura: COUNT

```
for i=0 to L-1 do
    BITMAP[i]:=0;
end for
for x in M do
    begin
    index:= ρ(hash(x));
    if BITMAP[index]=0 then
        BITMAP[index]:=1;
    end if
    end.
end for
```

Flajolet e Martin proposero di usare la posizione dello *zero* più a sinistra nel vettore BITMAP come un indicatore di $log_2 n$. Chiamarono questa quantità R e attraverso vari esempi e dimostrazioni arrivarono ad affermare, sotto l'assunzione che i valori degli hash fossero uniformemente distribuiti, che la speranza matematica di R doveva essere limitata da:

$$E(R) \approx log_2 \varphi n, \qquad \varphi^1 = 0.77351\cdots$$

Flajolet e Martin arrivarono a provare anche che, sotto ragionevoli assunzioni probabilistiche, la deviazione standard[2] di R era limitata a:

$$\sigma(R) \approx 1.12$$

Cosicchè una stima basata su $E(R)$ si discosterà dal risultato esatto di un ordine binario di grandezza. Un simile grado di errore risulta, per molte applicazioni, troppo elevato da poter sopportare senza conseguenze. Per rimediare a questa situazione i due studiosi proposero, in un primo momento, di usare un insieme H di m funzioni di hash, dove m è un *design parameter* e computare m differenti vettori BITMAP. In questo modo si sarebbero ottenute m differenti stime $R^1, R^2, \ldots, R^m$ di cui poi sarebbe stato sufficiente

[1] Il *fattore di correzione* φ gioca un ruolo piuttosto importante nella progettazione della versione finale dell'algoritmo.

[2] La deviazione standard, o scarto quadratico medio, è un indice di dispersione derivato direttamente dalla varianza, che ha la stessa unità di misura dei valori osservati. La deviazione standard misura la dispersione dei dati intorno al valore atteso.

considerare semplicemente la media:

$$A = \frac{R^{\langle 1 \rangle} + R^{\langle 2 \rangle} + \cdots + R^{\langle m \rangle}}{m}$$

Quando n elementi distinti sono presenti nel file, la variabile casuale A avrà speranza e deviazioni standard che soddisfano le seguenti quantità:

$$E(A) \approx log_2 \varphi n; \qquad \sigma(A) \approx \sigma_\infty / \sqrt{m}$$

Algoritmi simili, in cui si fa uso di una media diretta, offrono elevate probabilità di ottenere buoni risultati, ma presentano uno svantaggio per quanto concerne il tempo di calcolo, reso molto alto dal numero di funzioni di hash. In questo modo, infatti, il costo della CPU, per scansionare ogni elemento, deve essere moltiplicato di un fattore m. Per ovviare a questo incremento eccessivo delle prestazioni richieste, invece della media diretta, Flajolet e Martin, hanno utilizzato una *media stocastica*. L'idea consiste nell'utilizzare la funzione di hash per distribuire ogni record in uno degli m lotti, computando $\alpha = h(x) \bmod m$. Sarà così sufficiente aggiornare solo il corrispondente vettore BITMAP di indirizzo α con il "resto" dell'informazione contenuta in $h(x)$, vale a dire $h(x) \quad div \quad m \equiv \lfloor h(x)/m \rfloor$. Infine, si determineranno, come in precedenza, gli $R^{\langle j \rangle}$ e si procederà con il calcolo della media.

L'algoritmo che implementa il procedimento appena esplicato è chiamato *Conteggio Probabilistico, con Media Stocastica*, in inglese *Probabilistic Counting with Stochastic Averaging*, o PCSA (si veda l'algoritmo: 2 per una sua possibile implementazione). Per misurare l'accuratezza dell'algoritmo ideato, Flajolet e Martin introdussero alcune definizioni così da ottenere riscontri precisi sui risultati ottenuti.

Definizione 18 *Si definisce* ***Errore standard****, il quoziente tra la deviazione standard di una stima di n ottenuta con il PCSA e il valore reale di n.*

Questa quantità fornì ai due studiosi una precisa indicazione riguardo l'esattezza relativa prevista di un algoritmo atto a fornire una stima di n.

Definizione 19 *Si definisce, con* ***bias*** *di un algoritmo, il rapporto tra la stima di n e l'esatto valore di n, per n di grandi dimensioni.*

m	**bias**	**Errore Standard (%)**
2	1.1662	61.0
4	1.0792	40.9
8	1.0386	28.2
16	1.0191	19.6
32	1.0095	13.8
64	1.0047	9.7
128	1.0023	6.8
256	1.0011	4.8
512	1.0005	3.4
1024	1.0003	2.4

Tabella 2.1: Bias ed Errore standard del PCSA per alcuni valori di m, con m numero di vettori BITMAP usati.

Errore standard e bias dell'algoritmo PCSA sono riportati, per alcuni valori del design parameter nella tabella: 2.1.

Nell'abito degli studi relativi alla funzione di approssimazione l'algoritmo BitMap, introdotto da Flajolet e Martin, viene rivisitato ed utilizzato nel seguente modo. Dato un parametro utente r, si approssima il numero di elementi di M utilizzando $\log n + r$ bits[3]. Per ogni elemento in X si assegna casualmente uno dei $\log n + r$ bits usando una distribuzione esponenziale (circa metà dei nodi sono assegnati al bit 0, un quarto al bit 1 ecc.). Inizialmente il vettore BITMAP sarà vuoto, poi, per ogni elemento $x \in M$, si controllerà il bit assegnato ad x e si setterà tale bit all'interno del vettore. Una volta processati tutti gli elementi di M, si individuerà il bit 0, b, con ordine più basso all'interno del vettore e si stimerà $O(2^b)$. Per ridurre la varianza nella stima, è possibile ripetere la procedura per k volte e quindi calcolare la media.

$$\overline{b} = \frac{b_1 + b_2 + \cdots + b_k}{k}.$$

In questo caso, avendo utilizzato quella che Flajolet e Martin chiamano *media*

[3]dove r assume solitamente valori compresi tra 5 e 7.

Algoritmo 2 Probabilistic Counting with Stochastic Averaging (PCSA)

program PCSA;

Const nmap = 64; ▷ Con nmap=64 si ha un'accuratezza di circa il 10% nmap rappresenta la variabile m
$\varphi = 0.77351$;
maxlength=32; ▷ maxlength == L = 32 offre stringhe di lunghezza di 10^8

var M:multiset di dati di tipo : *records*;
x : *records*;
hashedx, index, α, R, S, Ξ : *integer*;
BITMAP : array[0..nmap-1,0..maxlength-1] of *integer*;

function *getelement(var z:records)*; ▷ Legge un elemento x di tipo record da M
function *hash(x:records)integer*; ▷ Crea l'hash di un record x in un intero di range $0 \ldots 2^{maxlength} - 1$
function *ρ(y:integer)integer*; ▷ Ritorna la posizione del primo 1-bit in y.

begin
while not eof(M) **do** ▷ Procedura COUNT adattata al PCSA
    getelement(x);
    hashedx := hash(x);
    α := hashedx mod nmap;
    index := ρ(hashedx div nmap);
    if BITMAP[α,index] == 0 **then**
        BITMAP[α, index] := 1;
    end if
end while
S := 0;
for i := 0 **to** nmap-1 **do**
    R := 0;
    while (BITMAP[i,R] == 1)and(R¡maxlength) **do**
        R := R+1;
        S := S+R;
    end while
end for
Ξ := trunc(nmap/φ*2**(S/nmap)); ▷ Risultato Ξ del PCSA che offre una stima di n
end.

stocastica, la stima esatta sarà data da:

$$\frac{2^b}{0.7731 * bias}$$

dove la bias può essere approssimata come $1 + 0.31/k$.

Un osservazione importante da compiere riguarda il caso di più multi-insiemi, ad esempioM_1 e M_2. Qualora si utilizzasse l'algoritmo BitMap per computare i vettori bm_1 e bm_2 (bm_1 OR bm_2) si otterrebbe il medesimo risultato di quello generato dal multi-insieme $M_1 \bigcup M_2$. Questo perchè, per come sono costruiti, il "bitwise-OR" può essere utilizzato come operatore di unione sui vettori BITMAP che rappresentano un multi-insieme.

2.4 L'algoritmo Random Interval

L'algoritmo *Random Interval*, presentato da Edith Cohen [9], risulta essere, per la sua progettazione, piuttosto particolare. Introdotto per fornire una procedura atta a calcolare la chiusura transitiva di un grafo orientato in tempo lineare, rispetto al numero degli archi, $= O(m)$[4], venne poi impiegato in numerose altre applicazioni tra cui la stima della grandezza dei vicini in un grafo diretto.

La peculiarità di questo algoritmo risiede nella sua duttilità che ne ha permesso l'utilizzo in svariati campi, grazie all'uso di quello che lo stesso Cohen ha definito l'*estimation framework*. L'algoritmo del Random Interval, infatti, non consiste solamente in una procedura da applicare al problema, ma fornisce un'intelaiatura, facilmente adattabile, che ne permette la modellazione in base al problema da risolvere.

[4]In precedenza l'algoritmo più efficiente impiegava $O(min(mn, n^{2.38}))$.

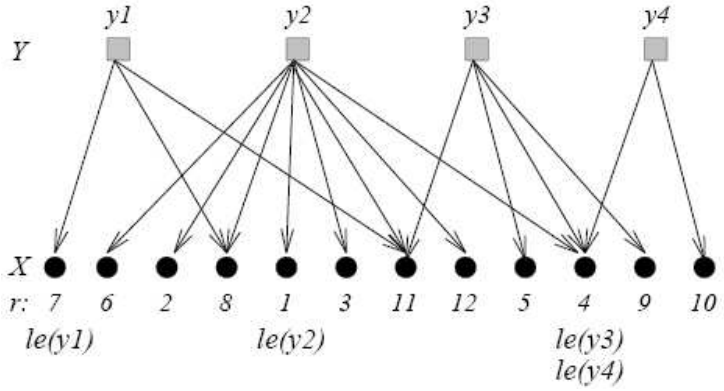

Figura 2.1: Esempio di insiemi X e Y, di un ranking r, e di $le : Y \rightarrow X$.

2.4.1 RI: Estimation framework

Siano X e Y due insieme e sia $S :\rightarrow 2^X$ una funzione in grado di mappare gli elementi di Y con sottoinsiemi di X. Sia, inoltre, $w : X \rightarrow R_+$ il peso, intero e non negativo, associato agli elementi di X. L'obiettivo è quello di fornire una stima di:

$$w(S(y)) = \sum_{x \in S(y)} w(x) \text{ per ogni } y \in Y.$$

Anche assumendo che gli elementi di X e Y ed i pesi siano dati, risulta troppo costoso calcolare direttamente $w(S(y))$ per tutti gli $y \in Y$. La seguente procedura Elemento-minimo, in inglese Least-Element, (LE) è fornita come oracolo. Quando LE è presente con ranking $r : X \rightarrow \{1, \ldots, |X|\}$ degli elementi di X, restituisce un mapping del tipo:

$$le : Y \rightarrow X | \forall y \in Y, \quad le(y) \in S(y) \text{ e } r(le(y)) = min_{x \in S(y)} r(x).$$

Quindi, per ogni elemento $y \in Y$, LE calcola l'elemento minimo $S(y)$ rispetto al ranking r. La figura 2.1 mostra un esempio di quanto appena introdotto.

L'estimation framework produce n iterazioni, dove n è determinato in base all'accuratezza desiderata per la stima. L'i-esima $(1 \leq i \leq n)$ iterazione è definita come segue:

- Si seleziona una chiave $R_i : X \to R_+$, indipendentemente per ogni valore di $x \in X$. La distribuzione con cui $R_i(x)$ è scelto è determinata da $w(x)$
- Si applica la funzione LE su X con ranking indotto dall'ordinamento delle chiavi R_i

Da quanto detto finora risultano immediate le seguenti asserzioni.

Proposizione 1 *Per tutti gli $y \in Y$ e gli $x \in S(y)$, la distribuzione della chiave minima $R(le(y))$ dipende unicamente da $w(S(y))$.*

Proposizione 2 *Per tutti gli $y \in Y$ e gli $x \in S(y)$, $Prob\{x = le(y)\} = \frac{w(x)}{w(S(y))}$. In grafi non orientati, $le(y)$ ha una distribuzione uniforme su $S(y)$.*

Per ogni elemento $y \in Y$, è possibile valutare $w(S(y))$ applicando un estimatore ai valori $R_i(le_i(y)) \quad (1 \leq i \leq n)$. Gli estimatori saranno basati su:

1. una media di n esempi
$$\hat{s}(y) \equiv \frac{n}{\sum_{1 \leq i \leq n} R_i(le_i(y))}, \frac{n-1}{\sum_{1 \leq i \leq n} R_i(le_i(y))},$$
2. Sia $le(y)$ il $\lfloor n(1-1/e) \rfloor$ più piccolo nella sequenza $R_i(le_i(y)) \quad (1 \leq i \leq n)$.
$$\hat{s}(y) \equiv \frac{1}{\tilde{le}(y)}$$

Un modo alternativo, per ottenere i medesimi risultati in termini di convergenza asintotica, consiste nell'utilizzare una distribuzione uniforme dove la chiave di $x \in X$ è data da un distribuzione $U^{(w(x))}$ con funzione di densità pari a $w(x)(1-t)^{w(x)-1)}$ e funzione di distribuzione $1-(1-t)^{w(x)}(0 \leq t \leq 1)$. Per reti dove il peso è unitario U^1 la distribuzione è uniforme nell'intervallo [0,1]. Nel estimation framework si richiede che $w(S(y)) \geq 1 \forall \quad y \in Y$. Quindi la media degli estimatori sarà:
$$\hat{s}(y) \equiv \frac{n}{\sum_{1 \leq i \leq n} R_i(le_i(y))} - 1$$

e l'estimatore prescelto sarà:

$$\hat{s}(y) \equiv \frac{1}{\tilde{le}(y)} - 1$$

Calcolare le stime $\hat{s}(y)$ per ogni $y \in Y$ equivale a produrre n, ordinate, chiavi casuali, assegnarle ed effettuare, infine, n chiamate alla funzione LE. Assumiamo per semplicità di calcolo e di analisi che le chiavi selezionate siano numeri reali. Una semplice considerazione ci mostra che è sufficiente usare $O(\log \varepsilon^{-1})$ bit significativi, dove ε è la tolleranza verso l'errore relativo. Studi sulla complessità hanno mostrato che una lista di numeri indipendenti e identicamente distribuiti può essere ordinata in tempo lineare (ad esempio utilizzando l'hashing). Quindi, per pesi unitari, la chiave ordinata è selezionata in tempo $O(|X| \log \varepsilon^{-1})$. Le chiavi possono anche essere ordinate in tempo lineare quando i pesi $w(x)$ sono forniti già ordinati.

2.4.2 RI: Calcolo dei vicini

Il calcolo di tutti i vicini, presenti in un grafo, equivale a voler calcolare l'insieme di tutti i cammini più corti presenti all'interno del medesimo grafo. L'algoritmo Random Interval, introdotto da Cohen, è in grado di stimare la dimensione di tutti i vicini in un grafo orientato con pesi non negativi sugli archi.

L'algoritmo restituisce in output, per ogni nodo $v \in V$, una lista di valutazione che contiene anche la grandezza stimata di tutti i vicini. La lista di valutazione è una lista di coppie $(D(v))_i$ e $(S(v))_i$ dove $(i = 0, \ldots, n(v))$ tali che:

- $D(v)_i$ e $S(v)_i$ sono sempre crescenti, $0 = D(v)_0 < D(v)_1 < \ldots < D(v)_{n(v)+1} \equiv \infty$ e $1 < S(v)_0 < S(v)_1 < \ldots < S(v)_{n(v)} \equiv n$.
- Per tutte le coppie $(v, d) \in V \times R_+$ varrà la relazione $\hat{n}(v, d) = S(v)_j$ dove $D(v)_j \leq d < D(v)_{j+1}$.

In virtù dei vincoli di costruzione della lista, appena forniti, risulta lapalissiano affermare che, una volta ottenuta la lista di valutazione, per stimare la grandezza dei vicini di una coppia (v,d) si impiegherà $O(\log n(v))$ tempo, grazie all'utilizzo della ricerca binaria.

Applicando l'estimation framework, presentato nel paragrafo 2.4.1, al grafo che si intende studiare, avremo l'insieme X corrispondente all'insieme dei vertici V, l'insieme Y rappresentato dalla collezione di tutte le coppie $(v, d) \in V \times R_+$, mentre S mapperà ogni coppia (v, d) a $N(v, d) \subset V$. Per un dato ranking dei nodi, *le* assegna ogni coppia (v, d) al minimo nodo con rank corrispondente in $N(v, d)$. Per ogni chiave R e $v \in V$, il valore $R(le(v, d))$ è una funzione costante non crescente di d. È possibile rappresentare *le* per tutti i valori $d \in R_+$ come una *lista dell'elemento minimo* di intervalli identificati. La lista dell'elemento minimo di un nodo $v \in V$ è una lista di coppie $(a_v(i), u_v(i)) \in R_+ \times V (1 \leq i \leq \ell_v)$ tali che:

- $a_v(1) > \ldots > a_v(\ell_v) = 0 \quad (a_v(0) \equiv \infty)$.
- $\forall 1 \leq i \leq \ell_v$ e $a_v(i-1) > d \geq a_v(i), le(v, d) = u_v(i)$ dove $u_v(i)$ è il nodo di rank minore in $N(v, d)$.

La figura 2.2 mostra un esempio di quanto fino ad ora introdotto.

Proposizione 3 *Se il ranking $r : V$ dei nodi è una permutazione random allora:*

1. *l'algoritmo impiegherà $O(m \log n + n \log^2 n)$ tempo (per archi di lunghezza unitaria il tempo diventa $O(m \log n)$).*
2. *la grandezza attesa per ogni lista dell'elemento minimo è $O(\log n)$.*

Effettuiamo k iterazioni di selezione delle chiavi e di calcolo delle liste dell'elemento minimo. Siano le_i il mapping dell'elemento minimo, R_i le chiavi assegnate, e $(a_v^i(j), u_v^i(j))$ per $(1 \leq j \leq \ell_v^i)$ le liste dell'elemento minimo all'i-esima iterazione. Il valore della media basata sugli estimatori per $N(v, d)$ è determinata dalla somma $s_v(d) = \sum_{i=1}^{k} R_i(le_i(v, d))$. Alla medesima maniera, il valore della selezione basata sugli estimatori è determinata da $m_v(d)$ che

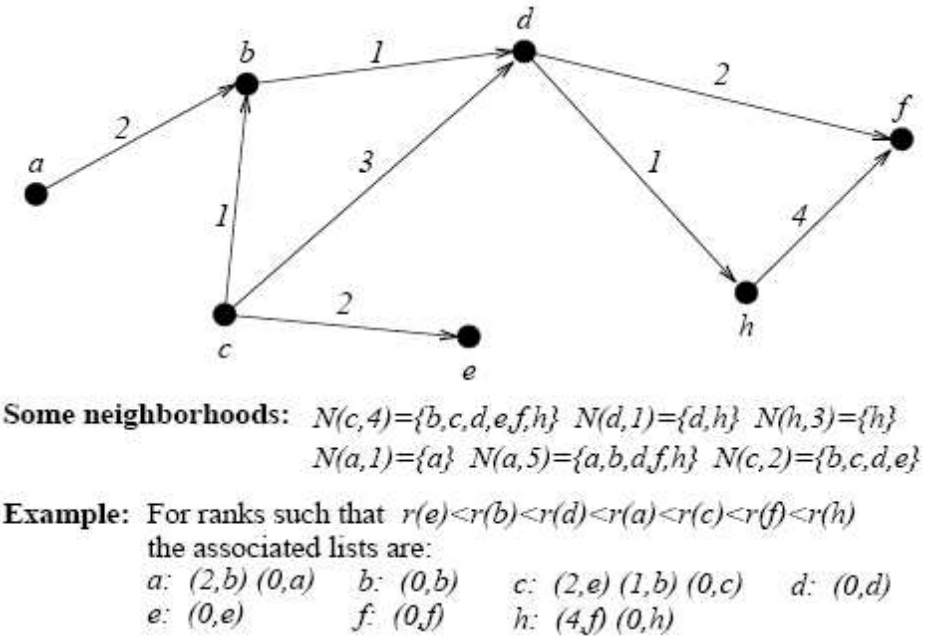

Figura 2.2: Esempio di grafo pesato, con ranking e liste associate.

risulta essere il più piccolo $\lfloor k(1-1/e) \rfloor$ degli $le_1(v,d), \ldots, le_k(v,d)$. Risulta facile verificare come le funzioni $s_v(d)em_v(d)$ siano:

- Funzioni costanti a tratti di d.
- Funzioni non crescenti (cioè, $\forall i$ e v la chiave minima $R_i(le_i(v,d))$ non cresce con d).
- Funzioni che hanno $\sum_i \ell_v^i$ punti di rottura $\{a_v^i(j) | 1 \leq i \leq k, 1 \leq j \leq \ell_v^i\}$.

Risulta altresì facile osservare come un intervallo di rappresentazione di s_v e $m_v(d)$ (quindi la lista stimata di v) possa essere computato in tempo, asintoticamente pari a $O(\sum_i \ell_v^i \log k)$ effettuando, dapprima, il merge delle k liste ordinate $a_v i(j) \quad (1 \leq i \leq k)$ ed effettuando $O(1)$ operazione per ogni punto di rottura della lista unita.

La grandezza attesa delle liste dell'elemento minimo è $O(\log n)$. Quindi, le liste stimate avranno un numero di punti di rottura atteso dell ordine di

$O(k \log n)$. Si noti che per una accuratezza ε, le liste stimate possono essere ridotte a grandezza $O(\varepsilon^{-1} logn)$. Per ogni coppia (v, d) la stima $\hat{n}(v, d)$ può essere computata in $O(\log k + \log \log n)$ tempo utilizzando la ricerca binaria sulla lista stimata di v.

Per quanto concerne il calcolo delle liste dell'elemento minimo, Cohen propose un algoritmo basato su una modifica dell'algoritmo di Dijkstra. Assumendo che i nodi $v_1, \ldots, v_n$ siano ordinati con chiavi in ordine crescente, si indichi con e_{ij} l'arco che va da v_i a v_j e sia $D : E \to R_+$ la lunghezza degli archi. Partendo da queste premesse Cohen formulò l'algoritmo 3.

La proposizione 4 mostra la correttezza dell'algoritmo 3.

Proposizione 4 *Un nodo v_k è posizionato nell'heap alla i-esima iterazione se e solo se:*

$$dist(v_i, v_k) < dist(v_j, v_k) \forall j < i.$$

Se v_k è posizionato nell'heap durante la i-esima iterazione, allora la coppia $(dist(v_i, v_k), v_i)$ è posta nella lista di v_k e il valore di d_k è aggiornato, diventando $dist(v_i, v_k)$.

Per provare quanto appena affermato nelle proposizione 4 è possibile procedere nel seguente modo. Notiamo prima di tutto che la proposizione risulta verificata all'inizio della i-esima iterazione se $\forall 1 \leq k \leq n, d_k = min_{j<1} dist(v_j, v_k)$. Consideriamo, quindi, i v_k tali che $dist(v_i, v_k) < dist(v_j, v_k)$ $\forall j < i$ e mostriamo che v_k è posto nell'heap e prima della fine dell'iterazione ha etichetta $dist(v_i, v_k)$. Si continua quindi per induzione sul numero di archi nel cammino più corto da v_i a $dist(v_j, v_p)$ mostrando che se $dist(v_i, v_k) \geq dist(v_j, v_k)$ si arriverebbe ad una contraddizione. Sia v_q il vicino all'ultimo nodo nel cammino. L'ipotesi dell'induzione asserisce che v_q è posto nell'heap, e sarebbe rimosso nel caso un cui la sua etichetta fosse $dist(v_i, v_k)$. Perciò quando i vicini di q sono scansionati, k è posto nell'heap con etichetta $dist(v_i, v_k)$ o, nel caso in cui fossero già sull'heap, l'etichetta sarebbe aggiornata a $dist(v_i, v_k)$. A questo punto la verifica dell'induzione risulta essere immediata.

Algoritmo 3 Procedura: least-element lists

Invertire la direzione degli archi del grafo.

for i := 1 **to** n **do**
  $d_i \leftarrow \infty$
end for
for i := 1 **to** n **do**
  inizializza la lista di v_i a vuoto.
end for

for i := 1 **to** n **do** ▷ modifica all'algoritmo di Dijkstra
  Inizia con una struttura heap vuota, quindi posiziona v_i nel heap con etichetta 0.
  while heap != $\emptyset$ **do**
    Rimuovi il nodo v_k con etichetta minima dall'heap. ▷ Sia d l'etichetta di v_k.
    Posiziona la coppia (d, v_i) nella lista di v_k.
    Sia $d_k \leftarrow d$.
    for vicino v_j di v_i **in**
      if do thenv_j è nell'heap
        aggiorna la sua etichetta alla più piccola delle etichette correnti.
        $d + D(e_{kj})$.
      else
        if $d + D(e_{kj}) < d_j$ **then**
          inserisci v_j nell'heap con etichetta $d + D(e_{kj})$.
        end if
      end if
    end for
  end while
end for

Capitolo 3

Approximate Neighbourhood Function

"È peculiare di una mente bene istruita,
rimanere soddisfatta con il grado di precisione
che la natura gli permette, e non cercare l'esattezza,
dove solo l'approssimazione è consentita."
Aristotele

In questo capitolo sarà introdotto l'algoritmo scelto per lo studio che intendiamo portare avanti in questo lavoro di tesi. Tale algoritmo è conosciuto in letteratura con il nome di *Approximate Neighbourhood Function*, o più semplicemente ANF.

L'algoritmo verrà presentato con un approccio iterativo incrementale atto a delineare e raffinare, ad ogni iterazione, la struttura dello stesso, fino a giungere ad una sua versione definitiva.

Si è optato per un approccio di tal genere così da mostrare, a partire dalla definizione base dell'algoritmo, tutte le sue varianti spiegando, per ognuna di esse, i miglioramenti apportati.

3.1 Introduzione ad ANF

L'algoritmo *Approximate Neighbourhood Function*, meglio conosciuto attraverso il suo acronimo, ANF, è stato ideato da C. Palmer, B. Gibbons e C. Faloutsos per approssimare la funzione di vicinanza, da loro utilizzata in alcuni studi atti ad esperire la struttura del web. Il motivo che ha portato, i tre ricercatori, a sviluppare un nuovo algoritmo, piuttosto che utilizzare quelli già presenti in letteratura, è da ricercare nella necessità di ridurre ulteriormente i costi relativi a tempo e memoria impiegati nell'approssimazione della funzione di vicinanza. Basandosi e riprendendo alcune delle idee emerse negli algoritmi BitMap e Random Interval (trattati rispettivamente nei paragrafi: 2.3 e 2.4), Palmer, Gibbons e Faloutsos arrivarono alla definizione di ANF che si mostrò, nel calcolo dell'approssimazione della funzione di vicinanza, relativa al grafo del web, ben 700 volte più veloce della computazione esatta, mantenendo, al contempo, un elevato livello di accuratezza nella stima.

Dato un grafo $G = (V, E)$, si assuma che $V = \{0, 1, \ldots, n-1\}$ e che E contenga m archi orientati. Nel caso di un grado non orientato sarà sufficiente rappresentare gli archi con coppie di archi orientati, estendendo E ad un insieme di cardinalità $2m$. L'obiettivo che i tre ricercatori si proposero di raggiungere, attraverso l'utilizzo di ANF, era quello di ottenere un'approssimazione della funzione $S(u, h)$ e $N(h)$ (si vedano le definizioni 16 e 17 nel paragrafo 2.1). L'approssimazione avrebbe dovuto essere fatta nel modo più accurato possibile, ma, al contempo, in maniera da far risiedere il grafo sul disco, così da massimizzare le prestazioni. Per raggiungere l'obiettivo desiderato, Palmer, Gibbons e Faloutsos, procedettero per piccoli passi, modificando e raffinando ogni volta i risultati avuti dalle prove eseguite, fino ad ottenere ciò che si erano prefissati.

3.2 ANF: versione base

L'idea, alla base dell'algoritmo ANF, è quella di raggruppare i nodi raggiungibili da x con h passi trovando, inoltre, quei nodi vicini al nodo di partenza che possono essere raggiunti in $h-1$ passi. Più formalmente, sia $\mathcal{M}(x,h)$ l'insieme dei nodi con distanza h da x. Chiaramente, $\mathcal{M}(x,0)=\{x\}$, poichè il solo nodo con distanza 0 da x è solo se stesso. Per calcolare $\mathcal{M}(x,h)$ è utile notare come x possa raggiungere, in h passi, i nodi che si trovano anche a distanza $h-1$, o inferiore. Inoltre, x è anche in grado di raggiungere i nodi posti a $\mathcal{M}(y,h-1)$ se esiste un arco che colleghi x ad y. Cercando di tradurre in codice quanto fino ad ora esposto, i tre ricercatori arrivarono a stendere questa rudimentale procedura.

$\mathcal{M}(x,0)=\{x\}$ for all $x \in V$
For each distance h **DO**
$\mathcal{M}(x,h)=\mathcal{M}(x,h-1)$ for all $x \in V$
EndFor
For each edge(x,y) **DO**
$\mathcal{M}(x,h)=\mathcal{M}(x,h) \cup \mathcal{M}(y,h-1)$
EndFor

In questo modo è possibile iterare sull'insieme degli archi, anzichè eseguire una visita del grafo. Grazie a questo escamotage è possibile computare in maniera efficiente il numero di elementi distinti presenti in $\mathcal{M}(x,h)$. Un'alternativa è quella di utilizzare una struttura dati di tipo dizionario, ad esempio un B-tree, per rappresentare l'insieme $\mathcal{M}(x,h)$. In questo caso, però, ci sarebbe bisogno di tempi e spazi asintoticamente pari a $O(n^2 \log n)$ e questo, specialmente per grafi di grandi dimensioni, risulta estremamente proibitivo. Un approccio spesso utilizzato è quello di impiegare i bit per marcare i membri dell'insieme. In questo modo ad ogni nodo è assegnato uno degli n bit e un insieme è rappresentato da una stringa di bit di lunghezza n. Per aggiungere un nodo all'insieme, sarà sufficiente marcare il suo bit. Sfortunatamente anche questo approccio non risulta applicabile per grafi di grandi dimensioni, poichè richiede un uso di memoria asintoticamente pari a $O(n^2)$. Per ovviare a questo problema, Palmer, Gibbons e Faloutsos hanno ripreso

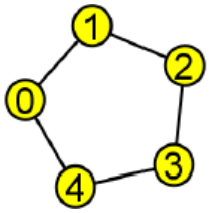

Figura 3.1: Grafo ciclico non orientato di soli cinque nodi.

l'idea utilizzata da Flajolet e Martin nell'algoritmo BitMap (si veda paragrafo 2.3), dove per approssimare la grandezza dell'insieme di bit si utilizzava una stringa di bit più piccola[1].

D'ora in avanti, per fare riferimento alla stringa di bit che approssima $\mathcal{M}(x, h)$, si utilizzerà la seguente notazione $M(x, h)$.

Con questo nuovo approccio, invece di assegnare ad ogni nodo il proprio bit, faremo in modo di dare a circa metà dei nodi il bit 0, ad un quarto di essi il bit 1 e così via (assegnando un nodo all'i-esimo bit con probabilità pari a $1/2^{i+1}$).

Per stimare la grandezza dell'insieme partendo dalla piccola stringa di bit, gli ideatori di ANF partirono dalla seguente intuizione. Se ci si aspetta che il 25% dei nodi sia assegnato al bit 1, ma nessuno di questi nodi è stato ancora trovato (il bit 1 non è settato), allora è probabile che l'insieme in esame sia composto di soli quattro nodi. In questo modo è possibile affermare che l'approssimazione della grandezza dell'insieme $\mathcal{M}(x, h)$ sia proporzionale a 2^b, dove b è il bit meno significativo non settato in $M(x, h)$. Ovviamente una singola approssimazione non è molto robusta in termini statistici, per questo Palmer, Gibbons e Faloutsos eseguirono k parallele approssimazioni di $\mathcal{M}(x, h)$ con una stringa di bit di lunghezza $k(\log n + r)$. L'algoritmo 4 mostra lo pseudocodice completo per implementare la versione di ANF basata sulla scansione degli archi.

Allo scopo di chiarire la concatenazione dei dei bitmask ed illustrarne il calcolo introduciamo il seguente esempio. Si consideri in input un grafo non

[1] $\log n + r$ per qualche costante r piccola, solitamente compresa tra 5 e 7.

Algoritmo 4 ANF: procedura base

//Setta $\mathcal{M}(x,0) = \{x\}$
foreach nodo x **DO**
$M(x,0)$ = concatenazione di k bitmask, ognuno con un insieme di bit $(P(bit \quad i) = 0.5^{i+1})$
end for

foreach distanza h partendo da 1 **DO**
foreach nodo x **DO**
$M(x,h) = M(x,h-1)$
▷ Aggiorna **M(x,0)** aggiungendo un passo
end for
foreach edge (x,y) **DO**
$M(x,h) = (M(x,h)$ BITWISE-OR $M(y,h-1))$
▷ Calcola la stima per il valore corrente di h
end for
foreach nodo x **DO**
Stima individuale $\hat{S}(x,h) = 2^b/0.77351$ dove b è la posizione media dei bit-0 meno significativi nei k bitmask.
Stima di $\hat{N}(h) = \sum_{allx} \hat{S}(x,h)$.
end for
end for

x	$M(x,0)$	$M(x,1)$	$\hat{S}(x,1)$	$M(x,0)$	$\hat{S}(x,2)$
0	100 100 001	110 110 101	4.1	110 111 101	5.2
1	010 100 100	110 101 101	3.25	110 111 101	5.2
2	100 001 100	110 101 100	3.25	110 111 101	5.2
3	100 100 100	100 111 100	4.1	110 111 101	5.2
4	100 010 100	100 110 101	3.25	110 111 101	5.2

Tabella 3.1: Dati di esempio per un grafo ciclico non orientato di cinque nodi (si veda la figura: 3.1), su cui è stata applicata la versione base di ANF.

orientato, ciclico, di soli cinque nodi come quello mostrato in figura 3.1 e si utilizzino per la computazione i parametri $k = 3$ e $r = 0$.

Il primo ciclo FOR all'interno dell'algoritmo 4 si occupa di generare la tabella dei bitmask casuali $M(x, 0)$. Quindi, come spiegato in precedenza, utilizzando una distribuzione esponenziale, settiamo, in maniera casuale, un bit in ognuno dei tre bitmask concatenati. (Nella tabella 3.1 lo 0-bit da considerare è quello più a sinistra per ognuno dei bitmask di dimensione 3.)In seguito, ad ogni iterazione si utilizza l'operazione di OR per combinare i nodi che il vertice in esame potrebbe raggiungere in $h - 1$ passi, più quelli che i suoi diretti vicini possono raggiungere in $h - 1$ passi. Per fare un esempio pratico, $M(2, 1)$ è data da $M(1, 1)$ **OR** $M(2, 1)$ **OR** $M(3, 1)$ poichè i nodi 3 e 1 sono vicini del nodo 2. La stima quindi di $S(2, 1)$, ad esempio, è ottenuta dalla media della posizione del minimo 0-bit (2,1,1 = 4/3) e quindi $\hat{S}(2, 1) = 2^{4/3}/.0.77359 = 3.25$.

L'algoritmo 4 utilizza un eccessivo quantitativo di memoria e non esegue la stima della forma più generale della funzione di vicinanza. Per porre rimedio a tali mancanze Palmer, Gibbons e Faloutsos applicarono alla versione base di ANF le seguenti migliorie.

- Notando che $M(x, h)$ usa $M(y, h - 1)$ e mai $M(y, h - 2)$ modificarono l'algoritmo introducendo $Mcur(x)$ per memorizzare $M(x, h)$ e $Mlast(y)$ per tenere traccia di $M(y, h - 1)$ durante l'iterazione h, eliminando, in questo modo, le chiamate ricorsive alla funzione $M(x, h)$.

- L'inserimento dell'insieme dei nodi iniziali S cambia la stima da somma di $x \in V$ a somma di $x \in S$. In termini implementativi, ciò può essere ottenuto estendendo $Mcur$ per mantenere un bit marcato così da indicarne l'appartenenza ad S.

- L'inserimento dell'insieme dei nodi finali modifica il caso $h = 0$. Ora, infatti, $M(x, 0)$ è uguale a $\{\emptyset\}$ se $x \notin C$, poichè non può raggiungere nodi in C in zero passi. Così ai nodi non in C sono inizialmente assegnati bitmask composti di soli zeri.

L'algoritmo 4, modificato come appena indicato, è conosciuto in letteratura come ANF-0.

Algoritmo 5 ANF - 0

```
//Setta M(x, 0) = {x}
foreach nodo x DO
    if x ∈ C then
        Mcur(x) = concatenazione di k bitmask, ognuno con un insieme di
bit (P(bit  i) = 0.5^{i+1})
    else
        Mcur(x) = concatenazione di k bitmask, ognuno con un insieme di
bit inizializzato a 0.
    end if
end for

foreach distanza h partendo da 1 DO
    foreach nodo x DO
        Mlast(x) = Mcur(x)
                                ▷ Aggiorna M(x, 0) aggiungendo un passo
    end for
    foreach edge (x,y) DO
        Mcur(x) = (Mcur(x) BITWISE-OR Mlast(y))
                                ▷ Calcola la stima per il valore corrente di h
    end for
    foreach nodo x DO
        Stima individuale Ŝ+(x, h, C) = 2^b/0.77351 dove b è la posizione
media dei bit-0 meno significativi nei k bitmask.
        Stima di N̂+(h, S, C) = Σ_{x∈S} Ŝ+(x, h, C).
    end for
end for
```

L'algoritmo ANF-0 incontra quasi tutti i requisiti che un'approssimazione della funzione di vicinanza dovrebbe avere (si veda il paragrafo 2.1). In particolare ANF-0 soddisfa le seguenti proprietà:

- **Garanzia di errore:** ogni $S^+(x, h, C)$ è probabilmente stimato con un basso errore.
- **Veloce:** tempo di esecuzione asintoticamente pari a $O((n + m)d)$. È lecito aspettarsi piccola tale grandezza poichè d è tipicamente $\ll 1$.

- **Bassi requisiti di memorizzazione:** Solo la memoria addizionale per $Mcur$ e $Mlast$.
- **Parallelizzabile:**È possibile partizionare i nodi tra i processori e quindi ogni processore può indipendentemente $Mcur$ per ogni x nel proprio insieme. La sincronizzazione è la sola cosa necessaria alla fine di ogni iterazione.
- **Scansione sequenziale degli archi:** Si
- **Stimare $S(u, h)$:** Si con una accuratezza dimostrata.

L'unica proprietà non rispettata, tra quelle elencate nel paragrafo 2.1, è l'adattamento alla memoria disponibile, infatti, nonostante le migliorie apportate, questa versione di ANF, fa ancora un largo uso di memoria volatile.

3.3 ANF: computazione su disco

Per rendere ANF totalmente conforme alle proprietà esposte nel paragrafo 2.1, Palmer Gibbons e Faloutsos apportarono ulteriori modifiche alla versione ANF-0. Con l'introduzione delle variabili $Mcur$ e $Mlast$ ANF non ha più la necessità di accedere agli archi in maniera casuale, perchè leggendo l'arco (x, y) legge e scrive $Mcur(x)$ e legge solamente $Mlast(y)$, in questo modo solo ad $Mcur$ ed $Mlast$ si accede casualmente. Nel caso in cui le tabelle siano più grandi della memoria a disposizione, però, il processo di swapping rischia di influire pesantemente, ed in maniera negativa, sulle performance. Riflettendo su questo i tre ricercatori, optarono per inserire una fase di preprocessing per rendere più praticabili gli accessi. L'idea di base consiste nel rompere i larghi bitmask $Mcur$ e $Mlast$ in due pezzi b_1 e b_2 di uguale grandezza. Una volta partizionati gli archi in $b_1 \times b_2$ bucket è possibile far girare la seguente procedura per aggiornare $Mcur$: Il costo dell'algoritmo è identico a quello della versione ANF-0 più il costo delle operazioni di I/O. Se $Mcur$ e $Mlast$ sono lunghe N byte allora il costo dell'I/O richiesto per modificare

Algoritmo 6 ANF: Procedura aggiornamento $Mcur$

```
foreach bucket i di Mcur DO
    Leggi bucket i di Mcur
    foreach bucket j di Mlast DO
        Leggi bucket j di Mlast
        foreach edge (x,y) in bucket(i,j) DO
            Mcur(x) = Mcur(x) OR Mlast(y))

        end for
        Scrivi bucket i di Mcur
    end for
end for
```

i bitmask sarà pari a $2N$ per leggere e memorizzare ogni bucket di $Mcur$ e $b_1 N$ per leggere $Mlast$ una volta per ogni bucket di di $Mcur$. Quindi, il costo totale delle operazioni di I/O è pari a $(b_1+2)N$. Per rendere più piccola questa quantità è quindi opportuno selezionare b_1 in modo che risulti minimo. Venne introdotta, infine, una fase di prefetching che permettesse di eseguire la computazione e le operazioni di I/O in parallelo, per non inficiare, a causa delle molte operazioni di I/O richieste, le performance dell'algoritmo che in questo modo riesce a soddisfare tutti i requisiti richiesti ad una funzione di vicinanza o ad una sua approssimazione, come mostrato nel paragrafo 2.1.

Algoritmo 7 ANF: External version.

Scegliere il numero di bucket b_1 e b_2 e partizionarvi gli archi all'interno
foreach nodo x **DO**
    if $x \in C$ **then** $Mcur(x)$ = concatenazione di k bitmask, ognuno con un insieme di bit ($P(bit \quad i) = 0.5^{i+1}$)
    end if
    if $x \in S$ **then** marca($Mcur(x)$)
    end if
    if buffer corrente è pieno **then** switch buffer ed esegui I/O
    end if
    Scorri ogni buffer che deve essere scritto.
end for
foreach distanza h **DO**
    Carica i dati per il primo bucket di $Mcur$ e $Mlast$. Precarica il prossimo bucket di $Mcur$ e $Mlast$
    foreach edge (x,y) **DO**
        if $Mcur(y)$ non è in memoria **then** Scorriamo e lo carichiamo, se necessario attendiamo. Inizia prefetching del prossimo buffer
        end if
        if $Mlast(x)$ non è in memoria **then**Lo carichiamo e attendiamo se necessario. Inizia prefetching del prossimo buffer
        end if
        $Mcur(x) = (Mcur(x)$ OR $Mlast(y))$
    end for
    est = 0
    Carica i dati per il primo bucket di $Mcur$
    foreach nodo x **DO**
        if $Mcur(y)$ non è in memoria **then** Lo carichiamo, se necessario attendiamo che sia avviabile. Inizia prefetching del prossimo buffer
        end if
        $Mlast(x) = Mcur(x)$
        if x è l'ultimo elemento nel bucket di $Mlast$ **then** Scorriamo asincronicamente il buffer. Continuiamo il processo nel doppio buffer
        end if
        if $Mcur(x)$ è marcato **then** $\hat{S}^+(x, h, C) = 2^b/0.77351$
            $est+ = \hat{S}^+(x, h, C)$ dove b è la posizione media dei bit-0 meno significativi nei k bitmask.
        end if
        $\hat{N}^+(h, S, C) = est$
    end for
end for

Capitolo 4

Tool e risorse

"La vera abilità sta nell'utilizzare
tutti i mezzi conosciuti e a disposizione;
L'arte, l'ingegno, consistono nell'operare
malgrado le difficoltà e trovare poco
o niente di impossibile."
Napoleone Bonaparte

In questo capitolo verranno presentati i tool e le risorse utilizzate per lo sviluppo di questo lavoro.

In particolare verrà presentato l'Internet Movie Database (IMDb), si inizierà con una piccola introduzione storica, per poi passare all'analisi dei metodi atti ad estrapolare le informazioni che questa gigantesca basi di dati cinematografica online mette a disposizione dei propri utenti. Le interrogazioni necessarie per il reperimento dei dati dal DB remoto avvengono, infatti, grazie all'ausilio di script che lo stesso IMDb offre in maniera totalmente gratuita.

Verrà descritto il framework WebGraph, sviluppato presso l'università di Milano, e da noi utilizzato per la generazione dei grafi compressi. Si effettuerà una panoramica del metodo di compressione utilizzato nell'implementazione del framework al fine di poter comprendere in maniera più approfondita le

modalità con cui, tale strumento, permette di ridurre le dimensione dei grafi ottenuti in input.

Infine, sarà introdotto il package it.unimi.dsi.fastutil, sviluppato sempre presso l'università di Milano, utilizzato, all'interno del codice prodotto, per migliorare l'efficienza delle operazioni svolte con le collezioni di elementi.

4.1 IMDb

L'Internet Movie Database, più conosciuto attraverso il suo acronimo IMDb, è la più grande basi dati online di informazioni relative a film, attori, registi e tutto ciò che riguarda il mondo dei film e del cinema in generale.

L'archivio fu creato nel 1990 sulla base di una raccolta di script, sviluppati da Col Needham. Tali script erano in grado di consentire la ricerca, all'interno delle FAQ che apparivano su un gruppo di discussione di Usenet.

Nel 1993 venne creata un'interfaccia centralizzata per l'interrogazione dell'archivio tramite l'utilizzo della posta elettronica.

Nel 1994 il sistema venne esteso consentendo anche l'immissione di informazioni, da parti degli utenti, all'interno dell'archivio, sempre tramite l'utilizzo della posta elettronica. In seguito, l'archivio venne trasferito sul Web appoggiandosi, inizialmente, ad una rete di server mirror e usufruendo di donazioni di banda provenienti da svariate organizzazioni.

Solo nel 1996 l'intero progetto venne acquisito dalla Internet Movie Database Ltd. che nel 1998 venne assorbita da Amazon.com, attuale proprietario del sistema.

Le dimensioni estremamente elevate e la completezza delle informazioni, fornite da IMDb hanno contribuito a rendere questo database cinematografico una fonte di dati privilegiata all'interno di parecchi studi accademici. IMDb mette a disposizione, infatti, di chiunque sia interessato ad utilizzare tali informazioni, senza fini di lucro, la totalità dei dati raccolti nella propria basi di dati.

4.1.1 IMDb reperimento dei dati

IMDb consente ai propri utenti di accedere alle informazioni, contenute all'interno della propria base di dati, anche al di fuori dell'infrastruttura offerta dal sito. Per fare questo, IMDb mette a disposizione diversi set di programmi, capaci di interfacciarsi con i differenti sistemi operativi esistenti. Offre, inoltre, un insieme di file testuali, sufficientemente aggiornati, che possono essere utilizzati per ricostruire, in locale, l'intera basi dati.

Per quanto riguarda i programmi a disposizione degli utenti, questi offrono tutti un vasto set di funzioni atte ad interfacciarsi con l'insieme delle informazioni ed interrogarle, così da poter estrarre i dati richiesti. Tra i vari programmi, messi a disposizione, particolare interesse suscita una tra le diverse interfacce offerte per i sistemi Linux/Unix. Tale front-end basato su di un'interfaccia a riga di comando, privo veste grafica, è composto degli stessi script scritti, nel 1990, da Col Needham.

4.1.2 Utilizzo di IMDb

All'interno del lavoro svolto IMDb è stato utilizzato in maniera alquanto limitata, eppure il suo apporto ai fini degli studi compiuti è stato essenziale. Nonostante, infatti, l'Internet Movie Database sia stato consultato solamente in una fase iniziale, per reperire i dati con i quali, in seguito, si sarebbero costruiti i grafi è giusto ricordare che, senza le informazioni forniteci da tale base di dati, non avremmo potuto costruire il dataset dei grafi su cui eseguire i nostri studi.

Per estrapolare le informazioni abbiamo scaricato una delle tante interfacce a riga di comando che lo stesso IMDb mette a disposizione degli utenti e con questa abbiamo iniziato ad interrogare il database attraverso l'ausilio del codice Java™. Una volta ottenuta risposta e selezionata la parte delle informazioni a cui eravamo interessati, tra la totalità dei risultati forniti dagli script, abbiamo salvato il tutto in file di memorizzazione tempora-

nea, formattati in uno pseudo-xml, così da renderne più facile, in seguito, il parsing.

4.2 WebGraph

WebGraph è un framework atto a studiare i grafi del web e non solo. Punto di forza di questo framework e la possibilità di gestire in maniera piuttosto semplice grafi di grandi dimensioni, grazie all'ausilio di moderne tecniche di compressione che vengono egregiamente sfruttate ed implementate dal tool per la creazione di rappresentazioni compresse dei grafi originari.

Precisamente il framework WebGraph è composto di:

1. Un insieme predefinito di codici, detti ζ *codes*, particolarmente adatti per la memorizzazioni di grafi del web, o, più in generale, di interi con una distribuzione di tipo power law appartenente ad un determinato range di esponente.

2. Algoritmi per grafi compressi che sfruttano il gap di compressione la referenziazione e gli ζ *codes* per fornire un alto grado di compressione e algoritmi per accedere ai grafi compressi senza effettuarne la decompressione, grazie a tecniche di tipo lazy che rimandano quest'operazione fino a quando non risulta strettamente necessaria [4].

3. Un'implementazione, completa e documentata, di tutti gli algoritmi appena citati, realizzata in Java, contenuta nel package it.unimi.dsi.webgraph. Il package contiene un insieme di API chiaramente definite che permettono la modifica o la ricompressione dei grafi.

4. Insiemi di dati per grafi di grandi dimensioni (bilioni di archi). Questi dati sono stati ottenuti da risorse pubbliche, come WebBase, oppure mediante l'utilizzo di strumenti come UbiCrawler.

Gli algoritmi di compressione utilizzati nel framework WebGraph sono basati sia su una pesante manipolazione dei bit, sia su sofisticate tecniche

di tipo lazy. Inoltre, per manipolare grafi di simili dimensioni, l'efficienza è una caratteristica essenziale. Per visitare un grafo con bilioni di archi non si può prescindere da tempi d'accesso dell'ordine dei microsecondi. Nonostante WebGraph sia stato totalmente implementato in Java™, secondo i canoni della programmazione OO, partendo, quindi, da un alto livello d'astrazione, le performance offerte non sembrano risentirne.

4.2.1 Il formato dei grafi compressi

Nel framework WebGraph il file di un grafo è rappresentato da una sequenza di liste di successori, per ognuno dei nodi del grafo. La lista del nodo x di un grafo può essere vista come una sequenza di numeri naturali:

1. Il grado uscente dei nodi. Se pari a 0 allora la lista relativa al nodo finisce.
2. La parte di riferimento composta di:
   (a) Interi non negativi, il riferimento. Se il riferimento è r, la lista dei successori sarà specificata come una versione modificata della lista dei successori del nodo $x - r$; se $r = 0$ allora la lista dei successori sarà specificata esplicitamente.
   (b) Se r è diverso da 0:
   - È un numero naturale b, detto *block count*.
   - È una sequenza di numeri naturali b $B_1, B_2, \ldots, B_b$, detti *copy-block list*, dove solo il primo numero può essere 0.
3. La parte extra, atta a specificare ulteriori informazioni contenute nella lista dei successori, questa è composta di:
   (a) Un intero i, detto *interval count*.
   (b) Una sequenza di i coppie, dove il primo componente è l'estremo sinistro dell'intervallo e il secondo componente è la lunghezza dell'intervallo, cioè il numero di interi in esso contenuto.

(c) La lista delle rimanenze, contenente tutti i successori non specificati nei metodi precedenti.

I dati appena riportati possono essere interpretati nel seguente modo:

1. La parte di riferimento, se presente (cioè se il riferimento r è strettamente positivo), specifica quale parte della lista dei successori del nodo $x - r$ deve essere copiata.

   I successori del nodo $x - r$ che dovrebbero essere copiati sono descritti nella rispettiva copy-block list. Per la precisione si dovrebbe copiare il primo B_1 scartare il successivo B_2, copiare B_3 ecc. Gli elementi rimanenti della lista dei successori saranno copiati se b è pari e scartati se dispari.

2. La parte extra specifica successori addizionali (o tutti se la parte di riferimento risulta assente). La parte extra risulta assente se il numeri di successori che devono essere copiati, in accordo a quanto specificato nella parte di riferimento, coincide con il grado uscente di x.

   La lista dei successori contenuta nella parte extra è fornita in due modi:

   - Alcuni di loro sono specificati come appartenenti agli intervalli: l'interval count indica il numero di intervalli, e questi sono a loro volta liste di coppie (estremo sinistro, lunghezza).
   - I rimanenti, composti dai successori scartati.

Il solo dato sempre presente è il grado uscente d. Se $d = 0$ non sono necessari altri dati per memorizzare la lista di adiacenza.

Risulta chiaro, da quanto finora esposto, che gli interi sopra riportati, non possono essere scritti in 32bits, o in un formato fissato, perchè dipendono dalla distribuzione che assumono, vengono quindi codificati in base alla media di interi, usando un variabile numero di bit per intero.

Il formato finale del WebGraph può essere riassunto nel seguente modo:

$$d[r[bB_1 \dots B_b]_{r>0}\overbrace{[iE_1L_1E_2L_2 \dots E_iL_i}^{L_{min<\infty}} R_1 \dots R_k]_{\beta<d}]_{d>0}$$

Dato	**Significato**
d	Grado uscente
r	Riferimento
b	Block count
i	Interval count
β	Successori copiati
$B_1 \dots B_b$	Blocchi
$E_1 \dots E_i$	Estremo sinistro
$L_1 \dots L_i$	Lunghezza intervallo
$R_1 \dots R_k$	Rimanenti

Tabella 4.1: Legenda dati presenti nel formato finale del WebGraph.

Dove i vari simboli sono esplicati nella tabella 4.1 e il valore di β rappresenta il numero di successori che sono stati copiati dalla lista dei riferimenti e può essere calcolato nel seguente modo:

$$\beta = \begin{cases} 0 & \text{se } r = 0 \\ \sum_{1 \leq h \leq b, hdispari} B_h & \text{se } r = 0 \text{ e b dispari} \\ |S(x-r)| - \sum_{1 \leq h \leq b, hpari} B_h & \text{se } r = 0 \text{ e b pari} \end{cases}$$

I grafi, memorizzati attraverso l'ausilio del framework WebGraph, sono oggetti immutabili. Il framework fornisce, infatti, metodi per accedere alla lista dei successori, ma non metodi per modificare tale struttura. Sebbene come decisione progettuale possa risultare piuttosto strana, questa viene però giustificata dalla natura dei grafi del web che sono usualmente computati da alcuni dati di partenza e quindi memorizzati per potervi accedere in maniera continua e spesso intensiva.

4.2.2 Iterazioni di tipo lazy

Dalla descrizione del formato di compressione adottato dal framework WebGraph, introdotta nel sottoparagrafo 4.2.1 , dovrebbe essere facilmente comprensibile che ricostruire la lista dei successori non è un'operazione così sciocca come potrebbe apparire ad una prima analisi. In primo luogo i successori provengono da svariate fonti: intervalli, copy-block e rimanenze, inoltre, i riferimenti potrebbero essere calcolati con una procedura ricorsiva.

Una soluzione banale al problema, potrebbe essere quella di decodificare, ricorsivamente, ogni lista dei successori dalla computazione della lista dei riferimenti. Questo tipo di approccio presenta però svariati svantaggi. In primo luogo, richiede un pesante accesso alla memoria, potenzialmente pari al numero di successori presenti in tutta la lista dei riferimenti. In secondo luogo, richiede la creazione di un largo numero di strutture dati temporanee da costruire, per poi essere subito distrutte dal garbage collector. Per tutte queste ragioni, WebGraph enumera i successori utilizzando *iteratori lazy*.

Nello specifico, il framework genera, in maniera ricorsiva, una serie di iteratori lazy sulle varie liste riferimento, inglobando nella costruzione quelli richiesti. Ogni iteratore, al momento della costruzione, carica tutti i dati sull'insieme dei rimanenti (solitamente più piccoli della lista completa). In seguito, ogni volta che viene richiesto ad un iteratore di alto livello di produrre un successore, questo controlla se è in grado di soddisfare la richiesta utilizzando le informazioni a sua disposizione, se così non è, inoltra a sua volta la richiesta all'iteratore del nodo referenziato.

È da notare che nessuna lista viene mai realmente espansa nella memoria. Durante l'iterazione, infatti, il solo stato mantenuto, dalla pila ricorsiva degli iteratori, è relativo agli intervalli e ai blocchi. Ciò permette di iterare sopra liste di successori molto lunghe indipendentemente dalla memoria avviabile, poichè, in questo modo, gli accessi alla memoria risultano drasticamente ridotti.

4.2.3 Utilizzo del WebGraph

All'interno del lavoro svolto, il framework WebGraph è stato impiegato per costruire e memorizzare i grafi ottenuti da IMDb in formato compresso e quindi utilizzare questo nuovo formato, molto più performante, attraverso le API fornite dal framework stesso.

La scelta di usufruire di WebGraph ha comportato un lavoro iniziale piuttosto lungo ed oneroso dovuto alla generazione dei grafi compressi, partendo dalle strutture estratte a seguito delle interrogazioni effettuate su IMDb.

In alcuni casi - per i grafi degli ultimi 30 anni - la generazione di un singolo file ha richiesto anche un intero giorno di computazione. Questa pesante mole di lavoro è però giustificata dai benefici che se ne trarranno in seguito. Si deve notare, inoltre, che un lavoro simile è necessario compierlo solo una volta, poichè dopo essere stati creati e memorizzati in appositi file, i grafi saranno sempre a disposizione dell'utente e pronti per essere utilizzati in qualsiasi momento.

Un ulteriore motivazione, che ha indirizzato le nostre scelte verso l'utilizzo di WebGraph, è data dalla riflessione che una rappresentazione diretta dei grafi, ottenuti dalle informazioni estratte da IMDb, sarebbe stata difficilmente utilizzabile a causa della dimensione dei file.

Rappresentazioni testuali canoniche dei grafi avrebbero generato, infatti, per i network più grandi, file di dimensioni molto superiori ai 40 megabyte, mentre con l'ausilio del framework WebGraph questo non avviene. Anche se in apparenza può sembrare poco rilevante, viste le dimensioni raggiunte dai moderni mezzi di memorizzazione di massa, la dimensione dei file da caricare in memoria risulta particolarmente importante. File di dimensioni minori, uniti ad altre caratteristiche proprie del framework WebGraph, per il caricamento veloce dei grafi, permettono di spostare in memoria file di grafi composti di milioni di archi nel giro di pochi millisecondi diminuendo, rispetto al medesimo file in formato testuale, la memoria richiesta.

La buona progettazione del framework e la potenza delle API a sua disposizione rendono, inoltre, particolarmente semplice, una volta che il grafo è caricato in memoria, navigare tra i diversi nodi e compiere le operazioni de-

siderate, lasciando all'utente una vasta possibilità di scelte implementative, anche grazie alle numerose classi a disposizione che coprono quasi totalmente le possibili richieste.

Integrare il framework WebGraph all'interno dell'applicazione prodotta per questo lavoro di tesi, così da sfruttarne pienamente le potenzialità, non è risultato affatto proibitivo ed ha prodotto risultati più che soddisfacenti, fornendo tempi di caricamento, accesso e risposta ai dati piuttosto brevi.

4.3 Il package fastutil

Il package it.unimi.dsi.fastutil, sviluppato anche esso presso l'università di Milano, estende il Java™Collections Framework fornendo tipi specifici per mappe, insiemi, liste e code con priorità ed implementando le interfacce specificate all'interno del package java.util. Il package fastutil include anche una serie di API per la gestione delle operazioni di I/O su file binari e di testo.

Uno dei punti di forza di questo package è quello di offrire una grande collezione di classi specializzate, generate a partire da una versione parametrizzata. Le classi risultano in questo modo molto più compatte e molto più veloci rispetto a quelle generali fornite dalle API di Java. Le classi del package fastutil aggiungono, inoltre, ai metodi standard altre funzionalità come iteratori bidirezionali, non avviabili normalmente nelle classi di default.

Il codice compilato è contenuto all'interno di un file compresso di tipo jar liberamente scaricabile ed utilizzabile sotto licenza GNU Lesser General Public License.

4.3.1 Utilizzo del package fastutil

Il package fastutil è stato utilizzato, all'interno del lavoro, per velocizzare la memorizzazione e l'accesso alle differenti strutture dati, richieste dal

programma per funzionare. A parità di funzionalità, l'utilizzo delle strutture fornite dal suddetto package, ha comportato un sensibile miglioramento nelle prestazioni dell'intera applicazione.

Capitolo 5

Calcolo efficiente del diametro

"Make it work.
Make it right.
Make it fast."
Kent Beck

In questo capitolo sarà presentato tutto ciò che è stato sviluppato per questo lavoro di tesi.

Si introdurranno i grafi storici della rete del cinema, spiegando cosa rappresentano, come sono stati ottenuti, a partire dalle informazioni estratte attraverso gli script di IMDb, e le metodologie utilizzate per la loro creazione.

Verra presentata l'importanza del diametro nelle reti sociali e le problematiche relative al calcolo di questa misura sulle reti di grandi dimensioni. Si passerà, quindi, a descrivere la soluzione da noi proposta per tale problema, introducendo la variante ibrida di ANF, ottenuta dal merge di due delle versioni presentate dagli autori dell'algoritmo.

In seguito verrà descritta, nella sua interezza, l'applicazione realizzata. In particolare si discuteranno le scelte progettuali e implementative che sono state prese, spiegando per ognuna di esse le ragioni che hanno condotto a tali scelte.

Infine, saranno presentati i risultati ottenuti al termine del lavoro svolto, sia in termini di efficienza di calcolo, sia relativamente allo studio dei grafi del cinema, con la presentazione dei diametri dei singoli grafi e della variazione della loro distribuzione negli anni.

5.1 I grafi del cinema

Per gli studi sulle reti sociali di grandi dimensioni, si è pensato di utilizzare la rete sociale del cinema. In questo particolare tipo di network, modellato con l'ausilio delle nozioni della teoria dei grafi, esposte nel paragrafo 1.2, l'insieme dei vertici V rappresenta la totalità degli attori, mentre l'insieme degli archi E rappresenta relazioni di collaborazione tra gli attori all'interno dei film girati nei diversi anni.

Il grafo $G(V, E)$ per un dato anno sarà, quindi, composto di tanti nodi quanti sono gli attori che dall'anno d'inizio, fino all'anno in esame, hanno recitato in almeno un film. Inoltre, vi sarà un arco, non pesato, tra un vertice ed un altro, se e solo se, i due attori in questione hanno recitato insieme in almeno un film.

Data l'entità transitiva della relazione appena esposta è pleonastico specificare che gli archi del grafo in esame saranno non orientati.

I grafi del cinema da noi ottenuti sono stati creati a partire dall'anno 1890 attraverso le informazioni estratte da IMDb, tramite gli script introdotti nel sottoparagrafo 4.1.1. Come logicamente era da attendersi i grafi partono da piccolissime dimensioni per crescere in maniera molto veloce con il passare degli anni. A riprova di quanto appena asserito si può notare come il grafo del cinema del 1890 consti di soli 4 nodi, mentre allontanandoci di soli quattro anni, per analizzare il grafo del 1894, questo sia già composto di ben 52 nodi.

Relativamente alla crescita dei grafi c'è da tenere in considerazione, inoltre, che con il passare degli anni il fenomeno cinematografico si è andato ad affermare sempre con maggiore decisione e quindi le produzioni di film, con conseguente impiego di attori, sono aumentate anche del doppio tra un anno

e il successivo. Un ulteriore parametro che influenza la crescita della rete sociale del cinema è l'aumento, con il passare degli anni, dei budget messi a disposizione per le varie produzioni. Nei primi anni, non era insolito che per contenere le spese un attore ricoprisse più ruoli all'interno dello stesso film, così come avviene anche in teatro per le produzioni low-cost, limitando, in tal modo, la crescita del grafo. Viceversa, con l'aumento delle disponibilità finanziarie, specialmente negli ultimi anni, si è potuto assistere ad un aumento vertiginoso dei cast dei film.

A quanto presentato fino ad ora si deve aggiungere la presenza di particolari eccezioni, quali sono i film colossal capaci, con la loro sola presenza, di innalzare di molto il numero di nodi del proprio grafo di appartenenza, rispetto alla media.

I fattori appena descritti dovrebbero far comprendere come, la rete sociale del cinema sia elaborata e complessa e, proprio per questo, estremamente interessante da studiare.

5.1.1 La generazione dei grafi

La modellazione delle informazioni estrapolate dall'Internet Movies Database ha seguito diverse fasi atte a modificare i dati, in maniera tale da poter giungere infine, grazie all'ausilio del framework WebGraph, alla generazione dei grafi compressi su cui svolgere i nostri studi.

Come primo passo, attraverso la creazione di alcune classi Java, si è proceduto all'interrogazione degli script di IMDb e all'estrapolazione delle informazioni che sono state, in seguito, salvate su dei file testuali in uno pseudo-xml. Ovviamente, di tutte le informazioni restituite per ogni film, dallo script fornito da IMDb, sono state memorizzate solo quelle che avevano interesse per i nostri studi. In particolare i nomi dei film girati nell'anno preso in considerazione e l'insieme degli attori che hanno partecipato ad ognuno di essi.

In seguito, le informazioni così formattate, sono state serializzate. Questo

Struttura	Significato
4	Numero totale di nodi
2	Grado uscente nodo 0
1	Vicino del nodo 0
2	Vicino del nodo 0
2	Grado uscente nodo 1
0	Vicino del nodo 1
1	Vicino del nodo 1
2	Grado uscente nodo 2
0	Vicino del nodo 2
1	Vicino del nodo 2
0	Grado uscente nodo 3

Tabella 5.1: Struttura esemplificativa di un grafo del cinema, rappresentato nel file di interi. Quello in tabella corrisponde al file del 1890 dove si ha una clique di 3 vertici {0,1,2} e un nodo sconnesso {3}.

passaggio, all'apparenza superfluo, è stato necessario per poter lavorare in maniera distribuita e permettere la generazione dei grafi, di annate diverse, su computer differenti nello stesso momento. Lo scopo di queste elaborazioni parallele è quello, ovvio, di velocizzare il processo di creazione dei grafi. Per ogni anno sono stati generati due diversi file serializzati, una mappa degli attori e un insieme degli attori.

Nella seconda fase, si passa, finalmente, alla generazione di un file di interi che sarà dato in pasto al metodo *store* della classe *BVGraph*, contenuta nel framework WebGraph, per la generazione dei grafi compressi. All'interno del file di interi è contenuta la struttura del grafo in esame, dove il primo intero rappresenta il numero di nodi, il secondo intero il grado uscente del primo vertice, seguono poi, per un numero di righe pari al grado uscente, gli indici dei successori del vertice in esame e così via fino all'ultimo vertice come mostrato in tabella 5.1.

Una volta generato il file di interi è, infine, possibile richiamare il framework WebGraph e creare un nuovo grafo compresso per l'anno considerato.

A parità di anno, un grafo nel formato compresso, occupa meno della

metà dello spazio su disco rispetto alla versione del file di interi.

5.2 Problemi nel calcolo del diametro

Il diametro di un grafo, come riportato più formalmente nella definizione 11 nel paragrafo 1.2, è la più grande distanza che intercorre tra due vertici $u, v \in G$.

I metodi per computare questa particolare grandezza possono essere classificati in due categorie:

- I metodi di manipolazione delle matrici delle distanze
- I metodi che iterano una singola procedura per la ricerca del cammino più breve, come l'algoritmo di Dijkstra.

Diversi algoritmi, atti a studiare le reti, utilizzano questa misura come limite superiore od inferiore, poichè il diametro, in molti grafi, può essere considerato una funzione del numero di nodi, ma, in generale, può assumere ogni valore e non è possibile conoscerne la dimensione in anticipo. L'importanza del diametro, come misura di studio, è dovuta alle osservazioni che tale parametro permette di compiere sulla forma o sulla struttura di un grafo.

Essendo il diametro la distanza massima tra due vertici, tra tutte le distanze presenti nel grafo, è possibile calcolarlo semplicemente computando le N^2 distanze ($N(N-1)/2$) e, quindi, prendere quella massima. È possibile fare ciò manipolando una matrice $N \times N$.

L'idea è di mantenere, per ogni coppia di vertici (i, j), un valore che tenga memoria del fatto che non sono conosciuti altri cammini minimi, oltre a quello segnalato, tra il vertice i e il vertice j, quindi basterà aggiornare questo valore mentre si naviga il grafo. Infine, il più grande elemento nella matrice rappresenterà il diametro.

Questi metodi sono molto semplici, ma, sfortunatamente, non efficienti. Richiedono, infatti lo spazio necessario a memorizzare la matrice $N \times N$ delle

distanze, ma questo per grafi di grandi dimensioni è impossibile da realizzare. Infatti il metodo migliore, di questa categoria, consente di performare le operazione con un tempo di calcolo, asintoticamente pari a $O(N^3)$.

Per quanto riguarda la seconda categoria metodologica, invece di computare esplicitamente tutte le $N \times N$ distanze e memorizzarle, si considerano solo le distanze tra coppie selezionate nel grafo. Partendo dalla considerazione che ogni cammino minimo può essere visto come un lower bound asintotico del diametro, è possibile applicare continuamente un algoritmo per il calcolo del cammino minimo da una singola fonte, fornendo questo lower bound fino ad ottenere il valore del diametro, che sarà dato dal più grande cammino minimo calcolato. Ciò avverrà una volta processate tutte le fonti, oppure quando tale lower bound arriverà ad essere pari all'upper bound del diametro. Un upper bound può essere calcolato a partire dalla dimensione del cammino più corto sullo spanning tree ritornato dall'algoritmo di Dijkstra. Nel caso peggiore, che si verifica quando due bound non si incontrano e tutte le fonti devono essere processate, il metodo impiega un tempo asintoticamente compreso tra $O(N^2)$ e $O(N^3)$.

Tutte queste problematiche, accresciute dalle dimensioni non certo ridotte dei grafi del cinema, contribuiscono a rendere praticamente impossibile, a causa dell'elevata inefficienza, il calcolo diretto del diametro all'interno delle reti di grandi dimensioni. Sulla base di questi riflessioni si è quindi scelto di non optare per un metodo diretto, ma di puntare sul calcolo di una stima, il più accurata possibile, di questa grandezza.

5.3 La nostra versione di ANF

Sulla base delle considerazioni effettuate nel paragrafo 5.2, relativamente all'opportunità di effettuare, invece di un calcolo diretto, una stima della dimensione del diametro dei grafi del cinema, si è optato, dopo aver ricercato soluzioni già note in letteratura, per l'utilizzo dell'algoritmo ANF, presentato nel Capitolo 3. Ovviamente, per usufruire al meglio di tale algoritmo, non

esistendo in Java™alcuna implementazione dello stesso, pur mantenendo sempre intatti i principi base dalla riusabilità, tipica della programmazione ad oggetti, si è deciso di apportare alcune modifiche alle diverse versioni proposte dagli autori dell'algoritmo, arrivando ad una variante 'ibrida', ottenuta dal merge delle componenti delle 3 versioni proposte che meglio si adattavano con la nostra situazione. Tali scelte, che tra breve andremo ad analizzare nello specifico, non inficiano in alcun modo i risultati raggiunti da Palmer, Gibbons e Faloutsos, ma anzi se possibile li migliorano grazie all'ausilio fornito dai tool utilizzati(si veda il Capitolo 4) che insieme ad ANF contribuiscono ad abbassare, in maniera significativa, i tempi di calcolo.

5.3.1 Versione ibrida di ANF: motivazioni

Una volta scelto di utilizzare l'algoritmo ANF, il passo successivo è stato quello di scegliere quale, tra le versioni proposte, si adattasse meglio alle nostre esigenze. Nello specifico c'è da ricordare che, differentemente da quanto ipotizzato dai creatori di ANF, i dati in nostro possesso rappresentano grafi particolari, poichè compressi attraverso l'ausilio del framework WebGraph, quindi meno pesanti da caricare in memoria. Inoltre, lavorando in un ambito ad oggetti, per accedere alle informazioni contenute in uno dei grafi compressi è necessario che questo venga istanziato. Per questo anche ad una prima analisi è risultato, quantomeno opinabile, pensare di implementare la versione basata su lettura da file, poichè, per realizzare un'applicazione simile, si sarebbe dovuto riversare il contenuto del grafo, dopo averlo caricato in memoria, in un nuovo file per poter poi partire con la computazione. In questo modo, si sarebbero perse, oltre ad una considerevole quantità di tempo per la fase di prefetching, anche tutta una serie di agevolazioni che il framework WebGraph mette a disposizione per navigare all'interno dei grafi compressi con esso generati.

Alla luce di quanto appena asserito, si è quindi deciso di focalizzare l'attenzione solo sulle prime due versioni di ANF, riportate rispettivamente negli

algoritmi 4 e 5.

Ad un'attenta analisi si può notare come, nell'algoritmo 5 si faccia uso della funzione generale di vicinanza, con l'introduzione di un insieme di nodi iniziali e un insieme di nodi finali.

Algoritmo 8 ANF: Versione ibrida

```
//Setta M(x, 0) = {x}
foreach nodo x DO
    Mcur(x) = concatenazione di k bitmask, ognuno con un insieme di bit
(P(bit  i) = 0.5^{i+1})
end for

foreach distanza h partendo da 1 DO
    foreach nodo x DO
        Mlast(x) = Mcur(x)
        Aggiorna M(x, 0) aggiungendo un passo
    end for

    foreach edge (x,y) DO
        Mcur(x) = (Mcur(x) BITWISE-OR Mlast(y))
        Calcola la stima per il valore corrente di h
    end for

    foreach nodo x DO
        Stima individuale Ŝ(x, h) = 2^b/0.77351
        dove b è la posizione media dei bit-0 meno significativi nei k
bitmask.
        Stima di N̂(h) = Σ_{x∈S} Ŝ(x).
    end for

end for
```

Ai fini dei nostri studi, tali modifiche apportate alla versione 4 risultano non solo superflue, ma anche deleterie, poichè ci costringono ad una fase di prefetching inutile e costosa e ad un ulteriore controllo, durante la creazione dei bitmask, per stabilire se il nodo in esame appartiene o meno all'insieme S, senza contare l'ulteriore spazio richiesto per la memorizzazione degli insiemi S e C.

La versione 4, di contro, presenta diverse ed inefficienti chiamate ricorsive per effettuare il calcolo della funzione $\mathcal{M}(x, h)$, mentre nella versione 5 gli autori hanno ovviato a questa pecca, ricorrendo all'utilizzo di due variabili di appoggio, $Mcur$ ed $Mlast$.

Analizzando l'insieme di tutte queste constatazioni, e la validità delle perplessità suscitate, si è deciso di generare una versione ibrida di ANF. Partendo quindi dalla versione 4, abbiamo modificato la struttura dell'algoritmo rimpiazzando tutte le chiamate ricorsive alla funzione $\mathcal{M}(x, h)$ con l'utilizzo delle variabili temporanee $Mcur$ ed $Mlast$, evitando il calcolo della funzione generale di vicinanza e quindi l'introduzione degli insiemi S e C e le relative modifiche strutturali che questi due insiemi comportano.

La totalità delle modifiche apportate può essere meglio studiata e compresa attraverso il listato dello pseudocodice dell'algoritmo 8.

5.3.2 ANF per calcolare il diametro

Una volta descritto l'algoritmo e definite le scelte progettuali che hanno guidato l'implementazione dello stesso, è opportuno definire le modalità che abbiamo individuato per ricavare, attraverso l'ausilio di ANF, una stima del diametro di un grafo, sia quest'ultimo piccolo o composto di diverse centinaia di migliaia di nodi.

ANF, come dice il nome stesso, e come ampiamente documentato nel Capitolo 3, è un algoritmo in grado di calcolare una stima, molto accurata, della funzione di vicinanza, individuale e totale, con tempi di calcolo molto bassi. Partendo dal calcolo di queste due funzioni, ANF può essere utilizzato in molti altri modi per studiare i grafi e la loro topologia effettuando, in maniera non diretta, numerose misurazioni.

Per calcolare il diametro di un grafo, attraverso l'ausilio di ANF, abbiamo sfruttato la funzione di vicinanza individuale $S(u, h)$ che, come spiegato più formalmente nel paragrafo 2.1, rappresenta il numero di nodi a distanza minore o uguale di h da u.

Riflettendo su questo formalismo e su alcune delle definizioni fornite dalla teoria dei grafi si è arrivati a formulare il seguente ragionamento. Se la distanza tra due vertici, altro non è che il più piccolo cammino presente tra questi, l'eccentricità di un nodo, a sua volta, altro non è che la massima fra tutte le distanze raggiungibili dal nodo in esame, e il diametro può, quindi, essere visto come il valore massimo tra tutte le eccentricità[1]. Rapportando il ragionamento appena esposto nell'ambito dell'algoritmo ANF, è stato sufficiente calcolare il valore massimo, raggiungibile da h, per tutti i nodi della rete, ottenendo, in questo modo, la dimensione del diametro del network in esame.

5.4 L'applicazione prodotta

L'applicazione realizzata per questo lavoro di tesi, col fine di compiere studi sulla rete del cinema, è stata implementata con l'ausilio della tecnologia Java™.

Concettualmente l'applicazione prodotta può essere immaginata come divisa in due parti distinte, questo soprattutto grazie al lavoro di progettazione compiuto, prima della stesura del codice. In questo modo è stato possibile organizzare l'applicazione secondo una logica piuttosto semplice, ma non per questo meno efficiente. Da una parte avremo, infatti, il **nucleo** dell'applicazione, mentre dall'altra un insieme di classi atte ad eseguire le operazioni di interesse per i nostri studi, tale insieme sarà identificato, d'ora in poi con il nome di **centro statistico**. Il principale beneficio di una simile struttura consiste nella possibilità di espandere, o modificare totalmente, l'insieme delle operazioni a disposizione dell'applicazione senza dover cambiare, in maniera massiccia, il codice. In questo modo, inoltre, si è evitato ad un eventuale utente esterno di dover utilizzare, e quindi studiare, il framework WebGraph. Tale framework viene infatti utilizzato nel nucleo dell'applicazione che non

[1] Per le definizioni formali di distanza, eccentricità e diametro si rimanda al paragrafo 1.2.

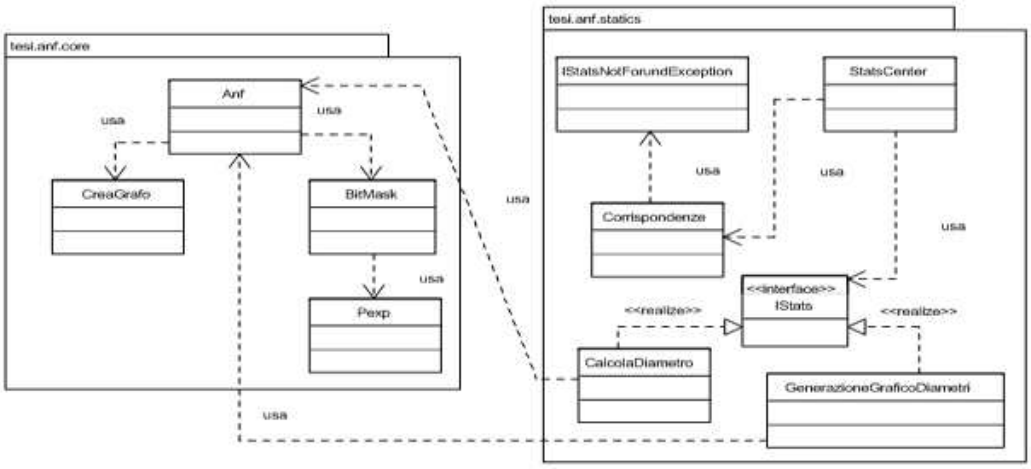

Figura 5.1: Modello UML della struttura relativa ai package tesi.anf.core e tesi.anf.statics.

ha motivo di essere modificato. Al contrario, all'interno del centro statistico dell'applicazione, dove sono racchiuse tutte le operazioni di interesse per gli studi sui grafi, tale framework non è minimamente utilizzato, grazie ad un insieme di oculate scelte progettuali ed implementative tendenti proprio a disaccoppiare le parti.

Alla luce delle argomentazioni appena esposte, il codice dell'applicazione è stato diviso in due package distinti, proprio per rimarcare la differenza di significato che intercorre tra alcune classi e le altre.

I package creati, mostrati in figura 5.1, sono i seguenti:

- tesi.anf.core. In questo package trovano posto le classi Anf, BitMask, Pexp e CreaGrafo.

- tesi.anf.statics. In questo package trovano posto le classi StatsCenter, Corrispondenze, IStats, CalcolaDiametro, GeneraGraficoDiametri e IStatsNotFoundException.

5.4.1 Il Core dell'applicazione

La prima parte, rappresentante il nucleo vero e proprio su cui tutto si appoggia per funzionare, è costituita dalla classe **Anf** e da tutte le altre classi utilizzate da questa per funzionare. La classe Anf, come si può facilmente intuire dal nome, è quella che implementa l'algoritmo 8 presentato nel sottoparagrafo 5.3.1 e permette, quindi, il calcolo delle funzioni di vicinanza locale, $S(u, h)$, e di vicinanza totale, $N(h)$. Non costituisce un'iperbole affermare che questa classe è, per la nostra applicazione, il cuore pulsante senza il quale nulla sarebbe possibile. A supporto della classe Anf sono state create alcune classi atte ad implementare, e quindi simulare, alcune parti particolarmente complesse dell'algoritmo.

In particolare è stata creata una classe **BitMask** per avere la possibilità di istanziare oggetti, capaci di rappresentare l'insieme dei BitMap utilizzati dall'algoritmo ANF per mantenere ed aggiornare la lista dei nodi raggiungibili, a partire da un determinato vertice, in un fissato numero di passi. Tale classe, quindi istanziata attraverso la classe Anf sarà utilizzata per la creazione e la modifica delle variabili $Mcur$ ed $Mlast$ introdotte nell'algoritmo in loco delle chiamate ricorsive alla funzione $\mathcal{M}(x, h)$.

Un'ulteriore classe, creata per fornire supporto diretto alla classe Anf e facente, quindi, sempre parte del nucleo dell'applicazione, è la classe **Pexp**. Il compito di questa classe è quello di fornire in maniera casuale valori ottenuti con una distribuzione esponenziale del tipo $\frac{1}{2}^{i+1}$, dove i rappresenta la posizione del bit che dovrà essere aggiornato. In questo modo, sarà lecito aspettarsi che la metà dei valori siano assegnati al bit in posizione 1, un quarto al bit in posizione 2, un ottavo al bin in posizione 3 e così via come richiesto dall'algoritmo ANF. Tale classe verrà quindi fatta interagire con la classe BitMask per determinare l'aggiornamento dei bit memorizzati in quest'ultima.

Per concludere l'insieme delle classi contenute all'interno di quello che abbiamo definito il nucleo dell'applicazione, si deve introdurre la classe **CreaGrafo**. Come si può facilmente evincere dal nome assegnatole, lo scopo di questo oggetto è quello di creare, a partire dal file di un grafo, in forma-

Figura 5.2: Modello UML della struttura relativa alle classi contenute nel package tesi.anf.core.

to testuale[2], l'insieme dei file in cui verrà memorizzato il grafo in formato compresso, in seguito all'elaborazione dell'input da parte del framework WebGraph.

Per migliorare le prestazioni, diminuendo i tempi d'accesso e di risposta dei singoli componenti, all'interno delle classi *Anf* e *BitMask*, in particolare, si è fatto un uso intensivo degli oggetti e dei metodi forniti dal package it.unimi.dsi.fastutil.

5.4.2 Il centro statistiche dell'applicazione

La seconda parte dell'applicazione, che costituisce il centro statistiche, è costituita anch'essa da diverse classi, fortemente operanti tra di loro, ma grazie all'utilizzo della *reflection* e di alcune tecniche di buona progettazione, quasi totalmente indipendenti le une dalle altre.

Nello specifico, è possibile trovare in questo insieme la classe **StatsCenter** che ha il compito di inizializzare la configurazione del package ed istanziare, attraverso i metodi della classe Class di Java™, oggetti di tipo IStats che saranno poi specificati a runtime dalle scelte dell'utente.

Un'altra classe importante, utilizzata da StatsCenter per determinare a runtime il tipo di oggetto IStats da istanziare è la classe **Corrispondenze** che ha il compito di leggere il file di configurazione, fornito a corredo dell'applicazione, ed creare una lista degli oggetti di tipo IStats presenti nell'applicazione così che possano essere richiamati a runtime per eseguire le operazioni richieste.

L'interfaccia **IStats** è utilizzata per la creazione di oggetti di tipo Stats, lo scopo principale di questa interfaccia è quello di fornire un supertipo comune alle classi delle operazioni così da poter disaccoppiare il codice delle stesse e rendere facilmente espandibile il codice dell'applicazione.

Grazie all'ausilio dell'interfaccia *IStats* e delle classi *StatsCenter* e *Corrispondenze* sarà possibile creare ed inserire all'interno dell'applicazione nuove

[2]Per la formattazione del file testuale di input si veda la tabella 5.1 all'interno del sottoparagrafo 5.1.1.

operazioni statistiche con grande facilità. Un eventuale sviluppatore, interessato ad espandere il set di operazioni disponibili, infatti, non dovrà fare altro che creare una nuova classe che svolga l'operazione desiderata. Una volta codificata tale classe, non dovrà fare altro che aggiungerne il nome all'interno del file di configurazione. Unico vincolo da rispettare, affinchè tutto funzioni, è che la classe implementi l'interfaccia IStats definita dall'applicazione.

Qualora l'applicazione tenti di eseguire un'operazione di tipo IStats senza riuscire a trovarla, verrà sollevata un'eccezione di tipo **IStatsNotFoundException** definita appositamente all'interno del software prodotto.

A corredo dell'applicazione sono già state realizzate due classi che implementano l'interfaccia IStats e che, quindi, permettono di eseguire operazioni sui grafi.

In particolare, visti gli studi che ci eravamo proposti di svolgere durante questo lavoro di tesi, è stata implementata una classe **CalcolaDiametro**, che ha lo scopo, come si può facilmente intuire dal nome, di calcolare il diametro, per un dato grafo. Ovviamente il calcolo viene fatto utilizzando l'algoritmo ANF e quindi sfruttando le classi dell'applicazione presenti nel nucleo della stessa. Come si può notare esaminando il codice, la classe è composta di un solo metodo, dovuto all'implementazione dell'interfaccia IStats. Tale metodo, con l'impiego di pochissime istruzioni, è in grado di restituire ciò che si era interessati ad ottenere.

Un'altra classe di operazioni, già presente all'interno dell'applicazione, è la classe **GeneraGraficoDiametri**. Lo scopo di una classe simile è quello di calcolare il valore dei singoli diametri, all'interno di un dato insieme di grafi, e quindi generare un'immagine, contenente il grafico ottenuto ponendo in ascissa gli anni e sulle ordinate i valori raggiunti. Ovviamente anche questa classe, per il calcolo dei diametri sfrutta l'algoritmo ANF implementato attraverso le classi del nucleo del software stesso.

L'insieme dei grafi da prendere in esame viene calcolato a partire dal grafo del 1890 fino ad arrivare all'anno passato dall'utente attraverso uno dei parametri del metodo[3]. Per calcolare la distribuzione dei diametri, la classe non fa altro che calcolare il diametro del grafo di ogni singolo anno e

[3]Gli estremi sono inclusi nel calcolo dell'insieme.

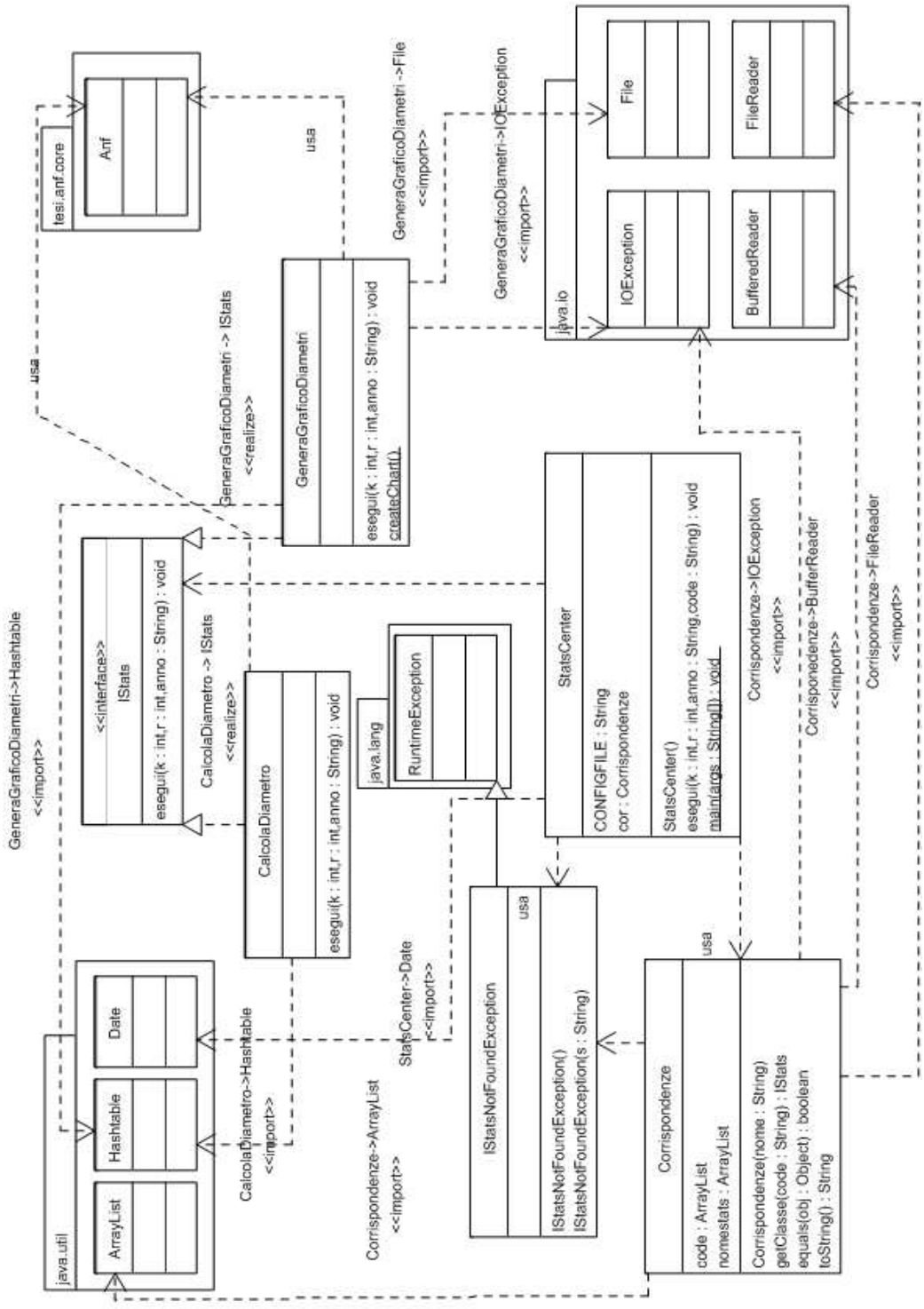

Figura 5.3: Modello UML della struttura relativa alle classi contenute nel package tesi.anf.statics.

quindi memorizzarlo, per poi utilizzare tali dati come valori delle ordinate all'interno del grafico.

Per la generazione dei grafici ci si è avvalsi dell'utilizzo del package org.jfree che permette di realizzare differenti tipologie di grafici a partire da un determinato insieme di dati.

5.5 Risultati

L'applicazione prodotta è stata testata su grafi di differenti dimensioni mostrando tempi di risposta più che soddisfacenti senza che questo andasse ad influire negativamente sull'accuratezza delle stime restituite in output dall'algoritmo.

In particolare, l'applicazione è stata testata su grafi formati da centinaia di migliaia di nodi, collegati tra di loro attraverso almeno un milione di archi. Tali test hanno evidenziato la capacità, da parte del software prodotto, di fornire stime del diametro estremamente accurate, in tempi estremamente bassi (meno di cinque minuti). Una caratteristica molto importante è che tali tempi tendo a salire molto lentamente, rispetto alla crescita di nodi ed archi.

Per quanto riguarda lo studio del network del cinema, si è notato, dopo un assestamento iniziale dei valori, che la rete presenta, nonostante le dimensioni elevate, diametri estremamente piccoli che sembrano decrescere ancora, anche se molto lentamente, con l'aumentare delle dimensioni della rete, proprio come ci attendevamo.

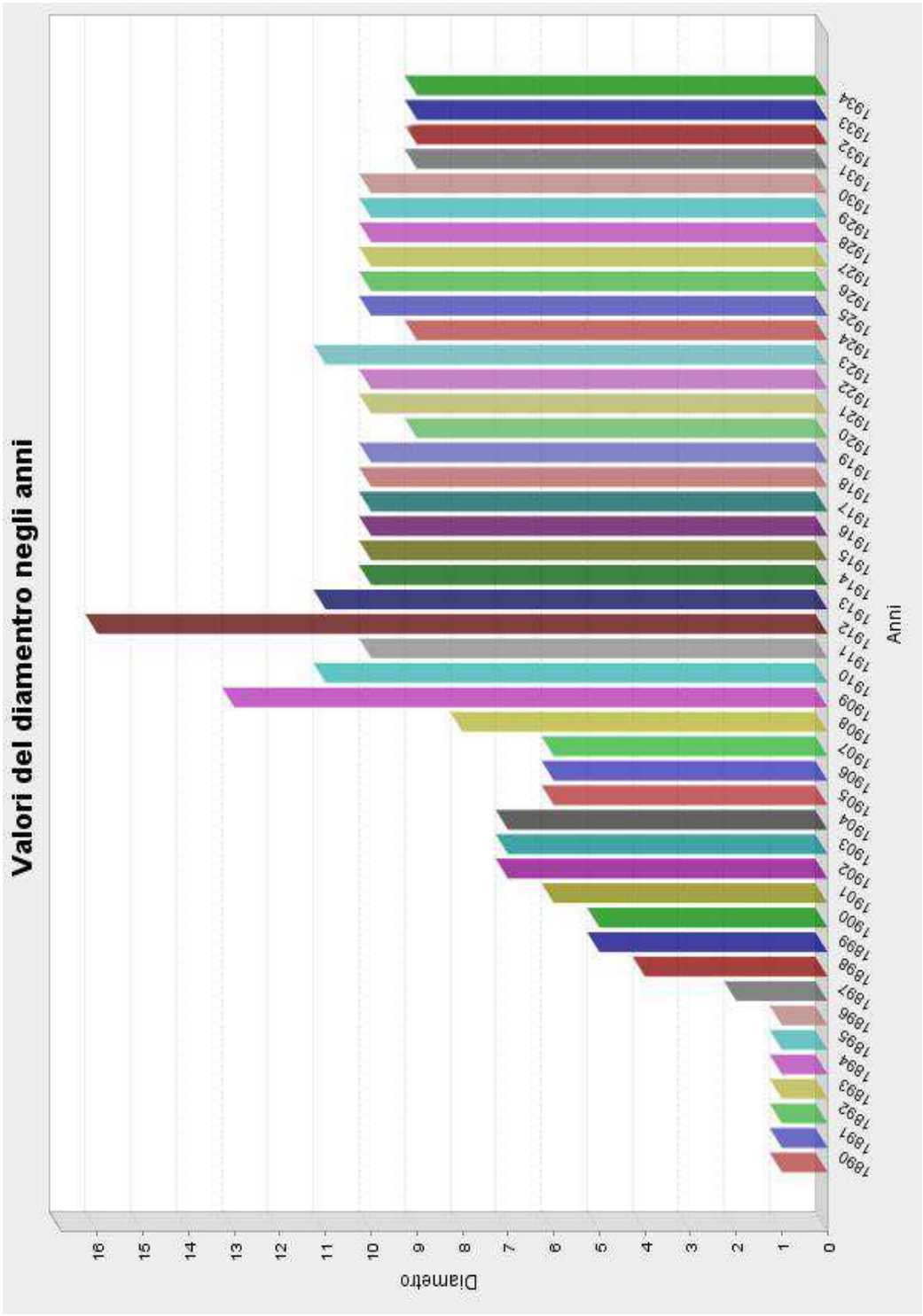

Figura 5.4: Grafico dei valori dei diametri della rete del cinema dal 1890 al 1934

Capitolo 6

Conclusioni e sviluppi futuri

"Ciò che non ha termine non ha figura alcuna."
Leonardo Da Vinci

Verranno ora presentate le conclusioni a cui il lavoro di tesi ci ha condotto, sia dal punto di vista teorico che pratico. Saranno quindi riassunti i risultati ottenuti, in termini di dati e di software prodotto. Infine verrà eseguita una valutazione, degli eventuali sviluppi futuri che l'applicazione sviluppata potrebbe subire anche a seguito del termine del lavoro svolto in funzione della tesi.

6.1 Conclusioni

Il percorso compiuto, per la compilazione di questo lavoro di tesi, ci ha spinto a svolgere un approfondito studio sulle reti sociali di grandi dimensioni. Sono stati analizzati, infatti, i modelli più importanti proposti in letteratura, per questa particolare tipologia di network, quindi sono state studiate alcune delle proprietà peculiari delle reti sociali per arrivare, infine, ad affrontare le tipiche problematiche che si riscontrano negli studi sui network di vaste dimensioni.

In particolare, per quest'ultimo punto si è tentato, in seguito ad un'accurata fase di ricerca per documentarsi sui metodi già presenti in letteratura, di trovare una soluzione efficiente, soprattutto in termini di tempo, per il calcolo del diametro. Sono stati quindi eseguiti studi sulla funzione di vicinanza e sugli algoritmi noti, in grado di fornire una stima, il più accurata possibile, di tale funzione. È stato quindi scelto l'algoritmo ANF, che meglio rispondeva alle nostre esigenze, ed è stato compiuto su di esso uno studio atto a comprenderne appieno i meccanismi e le sottigliezze tecniche. Si è quindi arrivati alla definizione di una variante ibrida di ANF, che si adatta in toto alle nostre esigenze, nata dal merge dei punti di forza di due delle versioni proposte dagli autori dell'algoritmo. Infine, si è giunti alla progettazione e all'implementazione, grazie all'ausilio della tecnologia Java™, di un'applicazione per lo studio dei grafi, incentrata sull'algoritmo ANF e capace di calcolare, in tempi brevi, il diametro del grafo in esame e, se richiesto, generare il grafico dei valori del diametro su di un insieme di grafi distinti, definiti dall'utente.

È importante notare come l'implementazione, da noi fornita, dell'algoritmo ANF sia l'unica esistente, scritta in Java™.

Dell'algoritmo Approximate Neighbourhood Function è possibile trovare, infatti, solamente due versioni codificate.

La prima, scritta in C dallo stesso Palmer[1] implementa la versione 7 che utilizza la memorizzazione sui file.

La seconda implementazione, scritta in C++, è inserita, come nel nostro caso, all'interno di un'applicazione più grande e codifica la versione 5 dell'algoritmo ANF limitandosi, però, al solo calcolo della funzione di vicinanza individuale. Tale applicazione, fortemente basata sulle librerie grafiche LEDA, è stata sviluppata da Jean Botev per una ricerca commissionata dal gruppo di Algoritmi e Strutture Dati della facoltà di Informatica, presso l'Università di Trier in Germania.

Alla luce di quanto affermato fino ad ora è lecito affermare, quindi, che

[1]Robert Palmer è uno degli inventori dell'algoritmo ANF. Per maggiori informazioni si consulti il Capitolo 3.

il lavoro svolto si distingue, oltre che per la validità dei risultati raggiunti, anche per l'originalità e l'importanza del software prodotto.

6.2 Sviluppi futuri

Per quanto concerne gli sviluppi futuri dell'applicazione, questi potranno essere molteplici e variegati. L'algoritmo ANF, infatti, permette di calcolare una stima della funzione di vicinanza individuale e totale, due funzioni molto utili negli studi sulle reti, perchè capaci di fornire differenti informazioni. Tali stime, infatti, possono essere utilizzate per eseguire diverse tipologie di analisi sui grafi, quali la ricerca del raggio, dell'eccentricità, ecc. Possono, inoltre, essere utilizzate per individuare eventuali cluster nel network.

L'utilizzo di ANF permette quindi di eseguire, oltre ad immediati studi, relativi ai cammini e alla topologia dei network in esame, anche analisi sulla morfologia dei grafi. In quest'ottica un possibile sviluppo futuro potrebbe consistere nella generazione di un nuovo set di grafi, provenienti da una rete diversa da quella del cinema, e quindi nella comparazione dei due network alla ricerca delle eventuali similitudini, ma anche delle differenze, che le due reti potrebbero presentare. Si potrebbe, altresì pensare di implementare classi per il calcolo del coefficiente di clustering.

Le grandi possibilità di studio offerte da ANF sono comunque ben supportate dall'applicazione prodotta. Grazie ad un'oculata progettazione e all'utilizzo di alcune tecniche di buona programmazione atte a disaccoppiare le differenti classi, nonchè alla potenza del linguaggio Java™, sarà possibile inserire facilmente, all'interno del nucleo originale dell'applicazione, nuovi moduli per il calcolo di nuove statistiche o misurazioni.

Infine, un'ultima miglioria apportabile all'applicazione, non in termini di prestazioni, ma solo da un profilo puramente estetico, potrebbe consistere nel tentare di fornire una funzionale e al contempo minimale veste grafica al package tesi.anf.statics, così da renderne più piacevole l'utilizzo.

Il motivo frenante, che ha portato a non implementare tale interfaccia,

è stata l'oculata progettazione che sarebbe servita per poter definire una veste grafica completa, ma il più leggera possibile, così da non influire troppo sulla memoria della macchina virtuale Java™ già fortemente utilizzata, togliendo risorse preziose ai componenti che rappresentano il vero fulcro dell'applicazione.

Appendice A

Codice

tesi.anf.core.Anf.java

```
package tesi.anf.core;


import it.unimi.dsi.fastutil.doubles.DoubleArrayList;
import it.unimi.dsi.fastutil.doubles.DoubleIterator;
import it.unimi.dsi.fastutil.ints.IntIterator;
import it.unimi.dsi.fastutil.objects.ObjectArrayList;

import it.unimi.dsi.mg4j.util.ProgressMeter;

import it.unimi.dsi.webgraph.ImmutableGraph;
import it.unimi.dsi.webgraph.NodeIterator;

import java.lang.reflect.InvocationTargetException;
import java.util.Hashtable;


/**
 * <p>Title: ANF </p>
 *
 * <p>Description: Lo scopo di questa classe è quello
    di implementare e
 * permettere di eseguire l'algoritmo di
    approssimazione della funzione di
 * vicinanza, N(h), tra i nodi di un grafo, noto in
    letteraturo con il nome di
```

```
 * Approximate Neighbourhood Function, o più
    semplicemente ANF, ideato da Palmer
 * Gibbons e Faloustsos. La classe utilizza i grafi
    ottenuti mediante il
 * pacchetto WebGraph sviluppato presso l'università di
     Milano e reperibile in
 * rete al seguente indirizzo:
 * <a href="http://webgraph.dsi.unimi.it/" target="
    _blank">WebGraph</a> </p>
 *
 * @author Valerio Capozio
 *
 * @version 1.0
 */
public class Anf {
    // Numero di approssimazioni parallele
    private int k;
    // Numero di bit extra per l'approssimazione,
       stabilito dall'utente,
    // solitamente compreso tra 5 e 7.
    private int r;
    // Grafo su cui eseguire le operazioni
    private ImmutableGraph graph;
    // M(x, h)
    private ObjectArrayList Mcur;
    // M(x, h-1)
    private ObjectArrayList Mlast;
    // Costante per memorizzare il valore della bias
    private final double BIAS = 0.77351;

    /**
     * Costruttore della classe Anf, inizializza i
        parametri k ed r
     * @param k int Indica il numero di approssimazioni
         parallele che
     * utilizzerà Anf.
     * @param r int Indica il numero di bit extra da
        usare
     * nell'approssimazione.
     * @param anno String L'anno del grafo da caricare
        e su cui si effettueranno
```

```
     * le operazioni.
     */
    public Anf(int k, int r, String anno) {
        this.graph = caricaGrafo(anno);
        // Iteratore dei nodi
        NodeIterator nodeIterator = this.graph.
            nodeIterator();
        // Nodo corrente
        int nodocur;
        this.k = k;
        this.r = r;
        this.Mcur = new ObjectArrayList();
        this.Mlast = new ObjectArrayList();
        /*
         * Ciclo che inizializza i BitMask di ogni nodo
             calcolando, di fatto,
         * M(x,0)={x}.
         */
        while (nodeIterator.hasNext()) {
            nodocur = nodeIterator.nextInt();
            this.Mcur.add(nodocur, new BitMask(this.k,
                (int) Math.round(Math.log10(this.graph.
                numNodes())) + this.r));
            ((BitMask)this.Mcur.get(nodocur)).
                SetBitMask();
            this.Mlast.add(nodocur, new BitMask(this.k
                ,(int) Math.round(Math.log10(this.graph.
                numNodes())) + this.r));
        }
    }

    /**
     * Metodo per il calcolo della funzione S(x, h)
     * @return double[] Ritorna un array contenente i
        valori della funzione
     * S(x,h) per ogni nodo x presente nel grafo.
     */
    public Hashtable calcolaS() {
        // Variabile per memorizzare i valori della
            funzione di vicinanza
        // individuale S(x,h)
```

```
Hashtable Sh = new Hashtable();
// Iteratore dei nodi
NodeIterator nodeIterator;
// Iteratore dei succesori
IntIterator successori = this.graph.successors
    (0);

int nodocur, succ, nodo = 0, h = 1;
/*
 * Ciclo per il calcolo di tutte le distanze h
    partendo da 1.
 */
while (h <= this.graph.numNodes()) {
    /*
     * Ciclo che memorizza in Mlast(x) il
        valore contenuto in Mcur(x)
     * prima che, quest'ultimo venga aggiornato
        .
     */
    nodeIterator = this.graph.nodeIterator();
    while (nodeIterator.hasNext()) {
        nodocur = nodeIterator.nextInt();
        ((BitMask)this.Mlast.get(nodocur)).
            copiaBitSet((BitMask)this.Mcur.get(
            nodocur));
    }

    /*
     * Ciclo che calcola, per ogni arco (x,y)
        che lega il nodo x agli
     * altri nodi y del grafo, il nuovo Mcur(x)
         secondo la formula
     * Mcur(x) = Mcur(x) OR Mlast(y).
     */
    // Però così ho introdotto un nuovo ciclo
       che impiega O(n x m)
    // invece che O(m) perchè non ho un modo
       per ciclare tutti
    // gli archi(?)
    nodeIterator = this.graph.nodeIterator();
    while (nodeIterator.hasNext()) {
```

```
                nodo = nodeIterator.nextInt();
                successori = this.graph.successors(nodo
                    );
                while (successori.hasNext()) {
                    succ = successori.nextInt();
                    this.Mcur.set(nodo, ((BitMask)this.
                        Mcur.get(nodo)).BitOR((BitMask)
                        this.Mlast.get(succ)));
                }
            }
            /*
             * Controllo se Mcur ed Mlast sono uguali
                allora interrompo il
             * ciclo perchè l è un upperbound troppo
                grande per h
             */
            if (this.Mcur.equals(this.Mlast)) {
                break;
            }
            /*
             * Ciclo per il calcolo della funzione S(x,
                h) su ogni nodo x
             */
            nodeIterator = this.graph.nodeIterator();
            DoubleArrayList S = new DoubleArrayList();
            S.size(graph.numNodes());
            while (nodeIterator.hasNext()) {
                nodocur = nodeIterator.nextInt();
                S.set(nodocur, Math.pow(2, this.
                    calcolaB( (BitMask)this.Mcur.get(
                    nodocur))) / (BIAS * (1 + 0.31 /
                    getK())));
            }
            Sh.put(h, S);
            h++;
        }
        return Sh;
    }

    /**
     * Metodo per il calcolo della funzione di
```

```
     vicinanza N(h).
     * @return double
     */
    public DoubleArrayList calcolaN() {
        // Funzione di vicinanza N(h)
        DoubleArrayList N = new DoubleArrayList();
        // Funzione di vicinanza individuale S(x,h)
        Hashtable S = new Hashtable();
        int h = 0;
        //Richiamo il metodo calcolaS per calcolare
           tutte le funzioni S(u,h).
        S = calcolaS();
        /*
         * Ciclo per il calcolo della funzione N(h),
            data dalla somma di
         * tutti gli S(x,h) per ogni x.
         */
        while (h < S.size()) {
            DoubleIterator doubiter = ((DoubleArrayList
               ) S.get((h + 1))).doubleIterator();
            double sumtemp = 0.0;
            while (doubiter.hasNext()) {
                sumtemp += doubiter.nextDouble();
            }
            N.add(h, sumtemp);
            h++;
        }
        return N;
    }

    /**
     * Metodo di accesso in lettura alla variabile k.
     * @return int Rappresenta il numero di BitSet
        paralleli da processare.
     */
    public int getK() {
        return this.k;
    }

    /**
     * Metodo di accesso in lettura alla variabile r.
```

```
     * @return int Rappresenta il parametro utente per
        l'ulteriore
     * arrotondamento del numero dei bit.
     */
    public int getR() {
        return this.r;
    }

    /**
     * Metodo di accesso in lettura alla variabile
        graph.
     * @return ImmutableGraph Rappresenta il grafo su
        cui ANF eseguirà le
     * operazioni.
     */
    public ImmutableGraph getGraph() {
        return this.graph;
    }

    /**
     * Metodo di accesso in scrittura alla variabile di
         istanza k.
     * @param k int Rappresenta il numero di BitSet
        paralleli da processare.
     */
    public void setK(int k) {
        this.k = k;
    }

    /**
     * Metodo di accesso in scrittura alla variabile di
         istanza r.
     * @param r int Rappresenta il parametro utente per
         l'ulteriore
     * arrotondamento del numero dei bit. (Di solito
        compreso tra 5 e 7)
     */
    public void setR(int r) {
        this.r = r;
    }
```

```
/*
 * Metodo privato per eseguire il caricamento di un
     grafo compresso in
 * memoria, in una variabile di tipo ImmutableGraph
    .
 * @param anno String Rappresenta l'anno del grafo
    da esaminare.
 * @return ImmutableGraph Ritorna il grafo
    compresso dell'anno passato da
 * parametro.
 */
private ImmutableGraph caricaGrafo(String anno) {
    ImmutableGraph graph = null;
    try {
        ProgressMeter pm = new ProgressMeter();
        // Carico attraverso la reflection la
           classe specificata dall'utente.
        Class graphClass = Class.forName( "it.unimi
           .dsi.webgraph.BVGraph", true,
           ClassLoader.getSystemClassLoader());
        // Carico attraverso la reflection il
           metodo load.
        graph = (ImmutableGraph) graphClass.
           getMethod("load", new Class[] {
           CharSequence.class, ProgressMeter.class
           }).invoke(graphClass, new Object[] {
           CreaGrafo.getOutputDir() + anno, pm});

    } catch (ClassNotFoundException ce) {
        System.out.println("Impossibile caricare la
           classe necessaria" + ce.toString());
    } catch (SecurityException ex) {
        System.out.println(ex.toString());
    } catch (NoSuchMethodException ex) {
        System.out.println("Impossibile eseguire il
           metodo richiesto" + ex.toString());
    } catch (InvocationTargetException ex) {
        System.out.println(
                "Impossibile trovare un grafo con
                   il nome specificato:" + anno + "
                   " + ex.toString());
```

```
            System.exit(1);
        } catch (IllegalArgumentException ex) {
            System.out.println("Argomento non
                consentito: " + ex.toString());
        } catch (IllegalAccessException ex) {
            System.out.println("Accesso non consentito:
                " + ex.toString());
        }
        return graph;
    }

    /*
     *  Metodo per il calcolo dell'esponente b da
        utilizzare per il calcolo
     * della funzione S(u,h). b è ottenuto dalla media
        delle posizioni del bit
     * 0 meno significativo all'interno del bitmask
     * @param Mcur BitMask E' il Bitmask su cui
        andranno conteggiate le
     * posizioni degli zeri meno significativi
     * @return double Ritorna la media delle posizioni
        degli zeri meno
     * significativi all'interno del Bitmask passato.
     */
    private double calcolaB(BitMask Mcur) {
        return (double) Mcur.sommaZeri() / this.k;
    }
}
```

tesi.anf.core.BitMask.java

```
package tesi.anf.core;

import it.unimi.dsi.fastutil.objects.ObjectArrayList;
import java.util.BitSet;

/**
 * <p>Title: BitMask </p>
 *
 * <p>Description: Questa classe rappresenta l'insieme
    Bitmask composto di k
 * Bitset, ognuno di lunghezza l. La classe permette di
     creare e aggiornare i
 * BitMask e di eseguire su di essi l'operazione di OR
    .</p>
 *
 * @author Valerio Capozio
 *
 * @version 1.0
 */
public class BitMask {
    //Lunghezza di un singolo bitmask
    private int l;
    //Numero di approssimazioni parallele
    private int k;
    // ObjectArrayList di BitSet
    private ObjectArrayList bitmask;

    /**
     * Costruttore della classe BitMask. Crea un
        BitMask composto di k BitSet
     * ognuno di lunghezza lun.
     * @param k int Numero di BitSet paralleli all'
        interno del BitMask.
     * @param lun int Lunghezza di ognuno dei BitSet.
     */
    public BitMask(int k, int lun) {
        this.k = k;
        this.l = lun;
        this.bitmask = new ObjectArrayList(this.k);
```

```
        for (int i = 0; i < this.k; i++) {
            bitmask.add(i, new BitSet(this.l));
        }
    }

    /**
     * Metodo per il calcolo della funzione OR su due
        BitMask.
     * @param bm BitMask Rappresenta il bitmask con il
        quale eseguire l'OR.
     * @return BitMask Ritorna il valore del BitMask di
         partenza dopo avere
     * eseguito, su di esso, l'operazione di OR con il
        BitMask passato
     * da parametro.
     */
    public BitMask BitOR(BitMask bm) {
        for (int i = 0; i < this.k; i++) {
            this.getBitSet(i).or(bm.getBitSet(i));
            //System.out.println(this.getBitSet(i));
        }
        return this;
    }

    /**
     * Metodo per sommare le posizioni degli zeri meno
        significativi (più a
     * sinistra) all'interno di ognuno dei k BitSet.
     * @return int Ritorna la somma delle singole
        posizioni degli indici dello
     * zero meno significativo per ognuno dei k Bitset.
     */
    public int sommaZeri() {
        int somma = 0;
        boolean flag = true;
        for (int i = 0; i < this.k; i++) {
            for (int k = 0; k < this.l; k++) {
                if (((BitSet) this.bitmask.get(i)).get(
                    k) == false && flag == true) {
                    flag = false;
                    somma += (k);
```

```
                }
            }
            flag = true;
        }
        return somma;
    }

    /**
     * Esegue la copia per valore dei BitSet presenti
        nel bitmask passato da
     * parametro.
     * @param bm BitMask Il BitMask di cui dovranno
        essere copiati i BitSet.
     */
    public void copiaBitSet(BitMask bm) {
        for (int i = 0; i < this.k; i++) {
            this.bitmask.set(i, ((BitSet) bm.bitmask.
                get(i)).clone());
        }
    }

    /**
     * Metodo di accesso in lettura alla variabile di
        istanza bitmask.
     * @param position int Indica la posizione del
        bitset di interesse
     * all'interno dell'array.
     * @return BitSet Ritorna il bitset indicato da
        position.
     */
    public BitSet getBitSet(int position) {
        return (BitSet) this.bitmask.get(position);
    }

    /**
     * Metodo di accesso in lettura alla variabile di
        istanza lunghezza.
     * @return int Rappresenta la lunghezza dei vari
        Bitset presenti
     * nel BitMask.
     */
```

```
public int getL() {
    return this.l;
}

/**
 * Metodo di accesso in lettura alla variabile di
    istanza k.
 * @return int Rappresenta il numero di BitSet
    paralleli all'interno di un
 * Bitmask.
 */
public int getK() {
    return this.k;
}

/**
 * Metodo per settare i valori del BitMask secondo
    la distribuzione
 * esponenziale P(bit[i])=0.5^i+1.
 */
public void SetBitMask() {
    Pexp pos = new Pexp(this.l);
    for (int i = 0; i < this.k; i++) {
        this.modificaBitSet(i, pos.estraiInt());
    }
}

/**
 * Metodo di accesso in scrittura alla variabile di
     istanza l.
 * @param l int Rappresenta la lunghezza dei vari
    BitSet presenti
 * nel Bitmask.
 */
public void setL(int l) {
    this.l = l;
}

/**
 * Metodo di accesso in scrittura alla variabile di
     istanza k.
```

```
     * @param k int Rappresenta il numero di BitSet
        paralleli all'interno di un
     * Bitmask.
     */
    public void setK(int k) {
        this.k = k;
    }

    /**
     * Ritorna una rappresentazione in forma di stringa
         dell'oggetto BitMask.
     * @return String Rappresentazione dell'oggetto
        Bitmask.
     */
    public String toString() {
        String strBitMask = "";
        for (int i = 0; i < this.k; i++) {
            strBitMask += "\t |";
            for (int j = 0; j < this.l; j++) {
                if (((BitSet)this.bitmask.get(i)).get(j
                    )) {
                    strBitMask += 1 + "|";
                } else {
                    strBitMask += 0 + "|";
                }
            }
            strBitMask += "|";
        }
        return strBitMask;
    }

    /**
     * Metodo equals che sovrascrive quello della
        classe Object. Confronta due
     * oggetti di tipo BitMask e restituisce true nel
        caso in cui siano composti
     * dai medesimi valori.
     * @param o Object Oggetto di tipo BitMask.
     * @return boolean True se il BitMask ricevente è
        uguale a quello passato da
     * parametro, false altrimenti.
```

```
     */
    public boolean equals (Object o){
        boolean flag = true;
        BitMask b = (BitMask) o;
        if(b.getK() != this.getK() || b.getL() != this.
           getL())
            flag = false;
        for(int i = 0; i < this.k;  i++)
            if(!((BitSet)this.bitmask.get(i)).equals(b
               .getBitSet(i))))
                flag = false;
        return flag;
    }

    /*
     * Metodo per la modifica di un bit all'interno di
       uno dei BitSet.
     * @param bitsetpos int Indice del BitSet da
       prelevare del BitMask.
     * @param bitposition int Indice del bit da
       modificare nel BitSet.
     */
    private void modificaBitSet(int bitsetpos, int
       bitposition) {
        this.getBitSet(bitsetpos).set(bitposition);
    }
}
```

tesi.anf.core.Pexp.java

```
package tesi.anf.core;

import java.util.Random;

/**
 * <p>Title: Pexp</p>
 *
 * <p>Description: La classe Pexp permette di generare
    in maniera randomica
 * un intero da 0 a n-1 estratto secondo una
    distribuzione esponenziale, del
 * tipo 0.5^(i+1), dove i rappresenta la i-esima
    posizione nella
 * sequenza [0...(n-1)].</p>
 *
 * <p>Copyright: Copyright (c) 2006</p>
 *
 * @author Valerio Capozio
 * @version 1.0
 */
public class Pexp {
    //Rappresenta la lunghezza su cui calcolare la
       probabilità
    private int lun;

    /**
     * Costruttore della classe Pexp
     * @param n int Rappresenta la lunghezza della
        sequenza.
     */
    public Pexp(int n) {
        this.lun = n;
    }

    /**
     * Metodo per l'estrazione dell'intero secondo la
        probabilità 0.5^i+1 con i
     * pari alla i-esima posizione della sequenza. Il
        metodo restituirà 0 per
```

```
     * numeri tra 1 e 0,5 (la metà dei valori), 1 per
        numeri compresi tra
     * 0,5 e 0,25 (un quarto dei valori), 2 per numeri
        tra 0,25 e 0,125
     * (un ottavo dei valori) e così via.
     * @return int Rappresenta il numero corrispondente
         alla posizione ottenuta.
     */
    public int estraiInt(){
        int pos = 0;
        float num = new Random().nextFloat();
        while ((float)(Math.pow(0.5, (pos + 1))) >= num
            ) {
            pos++;
            if(pos == this.getLun()){
                pos = 0;
                num =  new Random().nextFloat();
            }
        }
        return pos;
    }
    /**
     * Metodo di accesso in lettura alla variabile di
        istanza lun.
     * @return int Rappresenta la lunghezza della
        sequenza.
     */
    public int getLun(){
        return this.lun;
    }
    /**
     * Metodo di accesso in scrittura alla variabile di
         istanza lun.
     * @param l int Rappresenta il valore dalla
        lunghezza della sequenza che
     * andrà settato.
     */
    public void setLun(int l){
        this.lun=l;
    }
}
```

tesi.anf.core.CreaGrafo.java

```
package tesi.anf.core;

import it.unimi.dsi.webgraph.BVGraph;
import it.unimi.dsi.webgraph.examples.
    IntegerListImmutableGraph;

import java.io.IOException;
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.DataOutputStream;
import java.io.FileOutputStream;
import java.io.File;
import java.util.Date;

/**
 * <p>Title: CreaGrafo</p>
 *
 * <p>Description: La seguente classe permette di
     generare, partendo da un
 * file di tipo testo, contenente la struttura del
     grafo, un file di interi che
 * verrà in seguito utilizzato per la creazione del
     grafo compresso attraverso
 * il tool WebGraph.<br />
 * Il file di testo, contente la struttura del grafo,
     dovrà avere la seguente
 * formattazione:<br />
 * 5 <---- numero di nodi totoali del grafo<br />
 * 1 <---- grado uscente del nodo 0<br />
 * 1 <---- nodo a cui è collegato il nodo 0<br />
 * 1 <---- grado uscente del nodo 1<br />
 * 2 <---- nodo a cui è collegato il nodo 1<br />
 * 1 <---- grado uscente del nodo 2<br />
 * 3 <---- nodo a cui è collegato il nodo 2 <br />
 * 1 <---- grado uscente del nodo 3<br />
 * 0 <---- nodo a cui è collegato il nodo 3<br />
 * 0 <---- grado uscente del nodo 4<br />
 * <br />
 * Il grafo rappresentato con questa struttura è il
```

```
    seguente:<br />
 * <br />
 * <pre>
 *      0------>1
 *      ^       |
 *      |       |
 *      |       v
 *      3<------4          5
 * </pre>
 </p>
 *
 * <p>Copyright: Copyright (c) 2006</p>
 *
 * @author Valerio Capozio
 * @version 1.0
 */
public class CreaGrafo {
    // Variabile in cui verrà memorizzato il path
       relativo alla posizione in
    // cui andrà salvato il grafo compresso generato.
    private static final String OUTPUTDIR = "./
       resources/webGraphs/";
    // Variabile in cui verrà memorizzato il path
       relativo alla posizione in
    // cui andrà salvato il grafo in formato intero.
    private String intDir;

    /**
     * Costruttore della classe creaGrafo
     * @param intDir String Rappresenta il path della
        posizione in cui andrà
     * salvato il grafo in formato intero.
     */
    public CreaGrafo(String intDir) {
        this.intDir = intDir;
    }

    /**
     * Metodo per la creazione di un grafo compresso,
        mediante la creazione di
     * un grafo in formato intero.
```

```
     * @param nome String Rappresenta il nome che verrà
         assegnato al file del
     * grafo.
     * NOTA: Il nome del file deve essere privo di
        estensione!
     */
    public void creaInt(String nome) {
        DataOutputStream o;
        BufferedReader filebuf;
        String nextStr;

        try {
            o = new DataOutputStream(new
               FileOutputStream(new File(intDir + nome
               + ".int")));
            filebuf = new BufferedReader(new FileReader
               (intDir + nome + ".txt"));
            nextStr = filebuf.readLine(); // legge una
               riga del file

            while (nextStr != null) {
                o.writeInt(Integer.parseInt(nextStr));
                nextStr = filebuf.readLine();
            }
            System.out.println(">>> Inizio generazione
               WebGraph : " + intDir + nome + " (" +
               new Date(System.currentTimeMillis()) + "
               ) <<<");
            // Comando per la creazione del grafo.
            BVGraph.store(IntegerListImmutableGraph.
               loadOffline(intDir + nome + ".int"),
               OUTPUTDIR + nome);
            o.close();
            System.out.println(">>> Termine generazione
               WebGraph : " + OUTPUTDIR + nome + " ("
               + new Date(System.currentTimeMillis()) +
                ") <<<");
            // Chiude il file
            filebuf.close();

        }catch(IOException io){
```

```
            System.out.println("Errore: " + io);
        }
        catch (Exception e) {
            e.printStackTrace();
        }
    }

    /**
     * Metodo di classe per l'accesso alla costante
        outputDir.
     * @return String Contiene il valore della costante
         outputDir.
     */
    public static String getOutputDir() {
        return CreaGrafo.OUTPUTDIR;
    }

    /**
     * Metodo di accesso in lettura alla variabile di
        istanza intDir.
     * @return String Contiene il valore della
        variabile intDir.
     */
    public String getIntDir() {
        return this.intDir;
    }

    /**
     * Metodo di accesso in scrittura alla variabile di
         istanza intDir.
     * @param intDir String Rappresenta il path in cui
        andranno memorizzati i
     * file in formato intero.
     */
    public void setIntDir(String intDir) {
        this.intDir = intDir;
    }
}
```

Bibliografia

[1] R. Albert, H. Jeong, and A. L. Barabási. Diameter of the world-wide web. *Nature*, (401):130–131, 1999.

[2] R. Albert, H. Jeong, and A. L. Barabási. Mean-field theory for scale-free random networks. *Physica A*, (272):173–187, 1999.

[3] A.L. Barabasi. *Link. La scienza delle reti.* Einaudi, 2004.

[4] P. Boldi and S. Vigna. The WebGraph Framework I: Compression Techniques. *Technical Report*, pages 293–303.

[5] P. Boldi and S. Vigna. Webgraph: Things you thought you could not do whit Java™. *In Proc. of the 3rd International Conference on Principles and Practice of Programming in Java*, pages 1–8, 2004.

[6] Ulrik Brandes and Thomas Erlebach. *Network Analysis.* Springer, 2005.

[7] R. Breiger and P. Pattison. K. Carley. *Dynamic Social Network Modeling and Analysis: Workshop Summary and Papers.* National Academies Press, 2003.

[8] N. Christofides. *Graph theory, an algorithmic approach.* Academic Press., 1975.

[9] E. Cohen. Size estimation framework with applications to transitive closure and reachability. *Journal of Computer and System Sciences*, pages 441–453, 1996.

[10] Internet Movie Database. http://www.imdb.com/.

[11] P. Erdős and A. Rényi. On random graphs. *Publicationes Mathematicae*, pages 290–297, 1959.

[12] Michalis Faloutsos, Petros Faloutsos, and Christos Faloutsos. On power-law relationships of the internet topology. *ACM SIGCOMM Computer Communication Review*, 29:251–262, 1999.

[13] Package fastutil. http://fastutil.dsi.unimi.it/.

[14] P. Flajolet and G. N. Martin. Probabilistic counting algorithms for data base applications. *Journal of Computer and System Sciences*, 1985.

[15] Mark S. Granovetter. The strength of weak ties: A network theory revisited. *American Journal of Sociology*, pages 1360–1380, 1973.

[16] D. Jungnickel. *Graphs, network and algorithms.* Springer-Verlag, 1999.

[17] Jon M. Kleinberg. Navigation in a small world. *Nature*, 2000.

[18] David Liben-Nowell. *An Algorithmic Approach to Social Networks.* PhD thesis, Massachusetts Institute of Technology, 2005.

[19] R. J. Lipton and J. F. Naughton. Estimating the size of generalized transitive closures. *In Proc. of the 15th Int. Conference on Very Large Databases*, pages 165–172, 1989.

[20] C. R. Palmer, P. B. Gibbons, and C. Faloutsos. A fast approximation of the neighbourhood function for massive graphs. *Technical Report CMUCS -01-122*, 2001.

[21] C. R. Palmer, P. B. Gibbons, and C. Faloutsos. Anf: A fast and scalable tool for data mining in massive graphs. *ACM Press*, 2002.

[22] C. R. Palmer, P. B. Gibbons, and C. Faloutsos. Data mining in large graphs. in the Workshop on Dynamic Social Network Modelling and Analysis, 2002.

[23] S. Wasserman and K. Faust. Social network analysis: Methods and applications. *Cambridge University Press*, 1994.

[24] D. J. Watts and S.H. Strogatz. Collective dynamics of 'small world' networks. *Nature 393*, 1998.

[25] Framework WebGraph. http://webgraph.dsi.unimi.it/.

[26] R. J. Wilson, N. L. Biggs, and E. K. Lloyd. *Graph Theory 1736-1936.* Oxford University Press, 1999.