Università degli studi di Padova
Facoltà di ingegneria

Corso di laurea in Ingegneria Informatica

Algoritmi su grafi e applicazioni reali

Relatore: Prof. Andrea Alberto Pietracaprina

Laureando: Andrea Pizzato

Matricola: 610771 IF

Anno accademico 2012 - 2013

Indice

Elenco delle figure

Capitolo 1

Introduzione

L'obiettivo di questa tesi è quello di approfondire l'argomento dei grafi introdotto già nel corso di Dati e Algoritmi 1, concentrandosi sui problemi di ricerca del cammino minimo(applicandolo anche all'insieme di tutte le coppie di vertici del grafo) e del minimo albero ricoprente. Inoltre, verranno analizzati alcuni algoritmi di ranking delle pagine restituite da un motore di ricerca. Infine, sarà mostrata un'analisi generale della rete sociale più famosa attualmente, Facebook, concentrandosi sulla computazione del diametro e della media distanza tra due vertici, facendo una comparazione con l'esperimento del sociologo Milgram degli anni '60. L'obiettivo è quello di rimodellare questi argomenti in modo da renderli comprensibili e magari da accendere l'interesse nelle persone che leggono questo scritto. Il tema dei grafi viene affrontato in modo marginale durante il corso e quindi è importante illustrare alcune delle infinite applicazioni che possono trovare e che trovano realmente del mondo dell'ingegneria informatica. Chi non ha mai utilizzato un navigatore, anche uno di quelli disponibili sul Web? Chi non ha mai utilizzato un computer nella stanza in cui ce ne fossero altri, cioè in presenza di una rete di calcolatori? Chi non ha mai sfruttato l'invenzione geniale del motore di ricerca? Ecco, questi sono solo alcuni degli strumenti senza cui la nostra società non potrebbe più muoversi e si paralizzerebbe. Di seguito viene riportata una piccola descrizione del contenuto dei capitoli al fine di rendere chiaro quale sarà l'argomento trattato in questo scritto. Il secondo capitolo è un capitolo introduttivo che, da un punto di vista astratto, descrive la struttura del grafo, indicandone le principali proprietà. Il terzo capitolo approfondisce il problema del capitolo precedente estendendolo a tutte le coppie di vertici del grafo, cercando una soluzione prestante. Il quarto capitolo ha l'obiettivo di esporre il problema di ricerca del cammino minimo e di minimo albero ricoprente, spiegando quali applicazioni possono trovare nel mondo dell'informazione. Il quinto capitolo permette di comprendere sotto quali principi si muovono i motori di ricerca del grafo del Web, dimostrando la loro efficienza e improntando una traccia per

un'eventuale miglioramento futuro di queste strutture. Il sesto capitolo si occupa di analizzare da un punto di vista strutturale la rete sociale Facebook, concentrandosi sui concetti di spid, diametro, distanza media, affiancando l'esperimento a quello già compiuto da Milgram nel 1960 e mettendo in luce che il mondo in cui viviamo è più ″piccolo″ di quanto si possa pensare. Si assuma durante tutta la tesina che quando si parla di complessità, con il termine log si intende il logaritmo in base 2 dell'argomento considerato.

Capitolo 2

Concetti di base

Un grafo, concettualmente, è un modo di rappresentare relazioni esistenti tra coppie di oggetti. Da un punto di vista astratto, un grafo G è un insieme V di vertici e una collezione E di coppie di vertici appartenenti a V chiamati archi. Un grafo è sparso se il numero di archi che si dipartono da un generico vertice è costante o comunque indipendente dal numero dei vertici del grafo. Un grafo è denso se il numero di archi che si dipartono da un generico vertice è una funzione lineare del numero dei vertici. Formalmente, il grafo $G(V, E)$ è denso se : $|E| = O(|V|^2)$. Per esempio quando da ogni vertice si dipartono $|V|$ archi (Grafo completo). Mentre invece il grafo $G(V, E)$ è sparso se : $|E| = O(|V|)$. Per esempio quando da ogni vertice si dipartono k archi. Gli archi possono essere diretti o non diretti . Un arco (u, v) è detto diretto da u a v se la coppia (u, v) è ordinata. Un arco (u, v) è non diretto se la coppia (u, v) non è ordinata. Se tutti gli archi di un grafo sono non diretti allora si definisce il grafo non diretto. Al contrario, un grafo diretto è un grafo che ha tutti gli archi diretti. Nel caso in cui un grafo abbia archi diretti e non diretti, il grafo si definisce misto. I due vertici uniti da un arco sono chiamati estremi. Nel caso di un arco sia diretto si definiscono origine e destinazione. Due vertici si dicono adiacenti se c'è un arco che li collega. Un arco si dice incidente su un vertice se il vertice è uno degli estremi dell'arco. Gli archi uscenti di un vertice sono gli archi che hanno quel vertice come origine. Gli archi entranti di un vertice sono gli archi diretti che hanno quel vertice come destinazione. Il grado di un vertice v, chiamato $deg(v)$ è il numero di archi incidenti di v. Il grado entrante e il grado uscente di un vertice v sono il numero di archi entranti e uscenti di v, e sono definiti $indeg(v)$ e $outdeg(v)$, rispettivamente. È importante notare che la il grafo definisce gli archi come una collezione e non un insieme, scelta motivata dal fatto che in questa possono starci più archi con gli stessi estremi (e la stessa direzione, nel caso diretto). Questi archi sono definiti archi multipli o archi paralleli. Altri archi interessanti sono quelli che collegano un vertice a se stesso. Definiamo questi ultimi self-loop. Un grafo si definisce semplice se

non ha self-loop o archi paralleli.

Proposizione 2.1. *Sia G un grafo con m archi, allora*

$$\sum_{v \ in \ G} \deg(v) = 2m$$

Proposizione 2.2. *Sia G un grafo diretto con m archi, allora*

$$\sum_{v \ in \ G} indeg(v) = \sum_{v \ in \ G} outdeg(v) = m$$

Proposizione 2.3. *Sia G un grafo semplice con m archi ed n vertici. Se G è non diretto, allora* $m \leq \frac{n(n-1)}{2}$*, e se G è diretto, allora* $m \leq n(n\text{-}1)$.

Un percorso è una sequenza di vertici alternati ad archi che inizia in un vertice e termina in un altro tale che ogni arco sia incidente al suo vertice predecessore e successore. Un ciclo è un percorso con almeno un arco che sia sia sorgente che destinazione. Si definisce semplice un percorso se ogni vertice del percorso è distinto e definiamo semplice un ciclo se ogni vertice del ciclo è distinto tranne il primo e l'ultimo. Un percorso orientato è un percorso in cui ogni arco è orientato verso una direzione. Un ciclo diretto è un ciclo in cui ogni arco è orientato verso una direzione. Un sottografo di G è un grafo H i cui vertici e archi sono sottoinsieme dei vertici e archi di G, rispettivamente. Un sottografo ricoprente di G è un sottografo di G che contiene tutti i vertici di G. Un grafo è connesso se, per ogni coppia di vertici, c'è un percorso che li collega. Se un grafo non è connesso, il suo più grande sottografo connesso si chiama componente connessa di G. Una foresta è un grafo senza cicli. Un albero è una foresta connessa, cioè un grafo connesso senza cicli. Un albero ricoprente di un grafo è un grafo ricoprente, connesso e senza cicli.

Proposizione 2.4. *Sia G un grafo non diretto con m archi ed n vertici:*

- *se G è connesso, allora* $m \geq n\text{-}1$
- *se G è un albero, allora* $m \equiv n\text{-}1$
- *se G è una foresta,* $m \leq n\text{-}1$

Dati i vertici u e v di un digrafo (cioè un grafo diretto) G , si afferma che u raggiunge v (e v è raggiungibile da u) se G ha un percorso diretto da u a v. Un digrafo si dice strettamente connesso se per ogni coppia (u, v) di vertici di G , u raggiunge v e v raggiunge u. Un ciclo diretto di G è un ciclo i cui archi sono attraversati in accordo con le rispettive direzioni. Un digrafo è aciclico se non ha cicli diretti.

Un grafo diretto aciclico o DAG (Directed Acyclic Graph) è un grafo senza cicli diretti. Alcune applicazioni in cui sono sfruttati i DAG possono essere rappresentare l'ereditarietà tra classi di un programma in linguaggio Java, i prerequisiti tra corsi di un programma di laurea o ancora i vincoli di scheduling tra processi di un progetto.

Ora, sia G un digrafo con n vertici. In teoria dei grafi un ordinamento topologico è un ordinamento lineare di tutti i vertici di un DAG. I vertici di un grafo si definiscono ordinati topologicamente se i vertici sono disposti in modo tale che ogni vertice viene prima di tutti i vertici collegati ai suoi archi uscenti. L'ordinamento topologico non è un ordinamento totale, poichè la soluzione può non essere unica. Nel caso peggiore infatti si possono avere ordinamenti topologici diversi che corrispondono a tutte le possibili permutazioni dei vertici.

Proposizione 2.5. *G ha un ordinamento topologico se e solo se è aciclico e orientato.*

Un grafo pesato è un grafo che abbia un valore numerico $w(e)$ associato ad ogni arco e, chiamato peso di e.

Capitolo 3

Shortest Path e Minimum Spanning Tree

3.1 Shortest Path

Sia G un grafo pesato. La lunghezza (o peso) di un cammino (path) è la somma dei pesi degli archi di P, dove con P si indica il percorso considerato. Cioè, se $P \equiv ((v_0, v_1), (v_1, v_2), ..., (v_{k-1}, v_k))$, allora la lunghezza di P , definita $w(P)$ è definita come $w(P) \equiv \sum_{i=0}^{k-1} w((v_i, v_{i+1}))$. La distanza da un vertice v ad un vertice u in G, definita $d(u, v)$, è la lunghezza di un cammino di lunghezza minima (o, appunto, shortest path) da u a v se questo cammino esiste. Il problema computazionale, quindi, dopo aver ricevuto in input un vertice appartenente ad un grafo, si risolve restituendo in output i shortest path da quel vertice (target) a qualsiasi altro vertice del grafo. Il problema del shortest path trova numerose applicazioni in ambiti molteplici, come per esempio la pianificazione di reti di trasporto, la connessione tra pagine del World Wide Web, la gestione di reti di calcolatori, l'archiviazione di dati o i problemi di scheduling.

3.1.1 Algoritmo di Dijkstra

Per introdurre la soluzione del problema dei shortest path si sfrutta un'algoritmo di visita del grafo chiamato breadth first search(BFS). Questo algoritmo riceve in input un vertice e restituisce in output la BFS a partire da quel vertice. La sua peculiarità è quella di suddividere i veritici in liveli. La BFS comincia dal vertice s e visita tutti i vertici che si trovano a distanza i dal vertice considerato. Ciclicamente visita i vertici a livello $i+1$ fino ad aver visitato tutti i vertici del grafo. L'idea principale nell'algoritmo di Dijkstra è quella di eseguire una serie di BFS ″pesate″ cominciando da v. In particolare, si può usare la tecnica greedy (spiegata successivamente) per sviluppare un algoritmo che iterativamente fa crescere una nube di

vertici su v, con i vertici che entrano nella nube in ordine della distanza da v. Dunque, in ogni iterazione, il vertice scelto successivamente è il vertice fuori dalla nube più vicino a v. L'algoritmo termina quando non ci sono più vertici fuori della nube, a quel punto viene resituito un shortest path da v ad ogni altro vertice di G. Questa è la descrizione procedurale dell'algoritmo di Dijkstra. Per semplificare l'implementazione dell'algoritmo, si assuma nel seguito che il grafo G fornito in input sia non diretto. Si denotino gli archi di G come coppie non ordinate di vertici (u, z). Si assuma inoltre che i pesi degli archi siano non negativi. Di importanza fondamentale nell'algoritmo di Dijkstra è la procedura di rilassamento. Sia G un grafo pesato e V l'insieme dei suoi vertici. Si definisce un'etichetta $D[u]$ per ogni vertice $u \in V$ che viene utilizzata per approssimare la distanza in G da v a u. $D[u]$ dunque memorizza la lunghezza del miglior percorso che abbiamo trovato finora da v a u. Inizialmente $D[v] \equiv 0$ e $D[u] = \infty$ per ogni $u \neq v$, ed inoltre C che è la nostra nube di vertici che inizialmente sarà vuota. Per ogni iterazione dell'algoritmo, si seleziona un vertice u non in C con la più piccola $D[u]$, e si inserisce in C. Nella prima iterazione si inserisce v in C. Una volta inserito un nuovo vertice u in C, si aggiorna l'etichetta $D[z]$ per ogni vertice z che sia adiacente a u e che sia fuori da C, per riflettere il fatto che può esserci un nuovo e migliore percorso per andare da z a v.

if $D[u]+w((u,z)) < D[z]$ **then**
$D[z] \longleftarrow D[u]+w((u.z))$

Algoritmo `Dijkstra`(T, P)
input `Un grafo non diretto semplice` G `con archi con peso non negativo, e un vertice` $v \in G$
output `Un'etichetta` $D[u]$`, per ogni vertice` $u \in G$`, tale che` $D[u]$ `sia la lunghezza del cammino minimo da` v `a` $u \in G$

$D[v] \longleftarrow$ `0;` $D[u] \longleftarrow \infty$ `per ogni vertice` $u \neq v$`;`
`Sia` Q `una priority queue che contiene tutti i vertici di` G `usando l'etichetta` D `come chiave;`
while (Q `non vuota`) **do**
`inserisci un nuovo vertice` u `nella nube`
$u \longleftarrow Q$`.removeMin();`
`for each vertice` z `adiacente a` u `tali che` $z \in Q$ `do`
`utilizza la procedura di rilassamento sull'arco u,z`
if ($D[u] + w((u,z)) <$ `D[z]`) **then**
$D[z] \longleftarrow D[u] + w((u,z))$
`cambia in` $D[z]$ `la chiave del vertice` $z \in Q$
return `l'etichetta` $D[u]$ `per ogni vertice` u

Cosa garantisce che nel momento in cui u viene inserito in C, il valore dell'etichetta $D[u]$ è proprio il valore del cammino minimo tra u e v? La correttezza dell'algoritmo di Dijkstra è garantita dalla proposizione successiva, che è garantita a sua volta dal fatto che i pesi degli archi sono non negativi.

Proposizione 3.1. *nell'algoritmo Dijkstra , ogni volta che un vertice u è inserito nella nube, l'etichetta D[u] è uguale a d(u,v), la lunghezza del più breve percorso da v a u.*

Dimostrazione
Sia $D[t] > d(v,t)$ per qualche vertice t in V, e sia u il primo vertice che l'algoritmo ha inserito dentro la nube C tale che $D[u] > d(v,u)$. C'è dunque un shortest path P da v a u (in caso contrario significa che $d(u,v) = \infty = D[u]$). Si consideri successivamente il momento in cui u viene inserito in C, e sia z il primo vertice di P che non si trova in C al momento. Sia y il predecessore di z nel percorso P (potrebbe essere $y=v$)(Si osservi l'Immagine 3.1). Sicuramente, dalla scelta di z, y si trova già in C al momento. Inoltre, $D[y]=d(v,y)$, visto che u è il primo vertice non corretto. Quando y era stato inserito in C, si era testato (e forse aggiornato) $D[z]$ in modo che si avesse a quel punto

$$D[z] \leq D[y] + w((y,z)) = d(v,y) + w((y,z)).$$

Ma visto che z è il prossimo vertice nel cammino minimo da v a u, ciò implica che

$$D[z] = d(v,z).$$

Ma si sta considerando u, non z, per controllare se inserirlo in C; di conseguenza,

$$D[u] \leq D[z].$$

È chiaro che un sotto-cammino minimo è esso stesso un shortest path. Di conseguenza, visto che z è nel percorso minimo tra v e u,

$$d(v,z) + d(z,u) = d(v,u).$$

Inoltre, $d(z,u) \geq 0$ perchè non ci sono archi con peso negativo (passo fondamentale). Dunque,

$$D[u] \leq D[z] = d(v,z) \leq d(v.z) + d(z,u) = d(v,u).$$

Ma questo contraddice la definzione di u; dunque, non può esistere un tale vertice u.

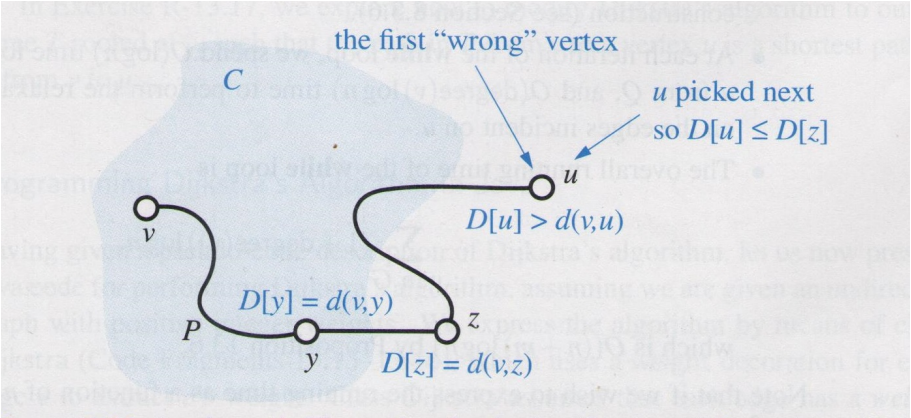


Figura 3.1: Illustrazione della dimostrazione della proposizione 3.1

Complessità algoritmo di Dijkstra

Si faccia riferimento allo pseudocodice appena proposto. Sia n il numero di vertici del grafo in input G e sia m il numero di archi del grafo in input G. Si assuma che gli i pesi degli archi possano essere sommati o confrontati in tempo costante. Si pensi di poter rappresentare la struttura grafo G con una lista di adiacenze. Questa struttura dati permette di passare da u a vertici ad esso adiacenti durante il passaggio del rilassamento in tempo proporzionale al loro numero. La coda con priorità Q è implementata con uno heap. Questo permette di estrarre il vertice u con la più piccola etichetta D in tempo $O(\log n)$.
Si immagini di avere un metodo $replacekey(e, k)$ dove e è la entry che memorizza la chiave per il vertice z. Si può dunque implementare il metodo in $O(\log n)$, poiché per trovare un vertice in uno heap si spende $O(\log n)$ mentre la sostituzione della chiave si può eseguire in tempo costante. Facendo queste assunzioni, l'algoritmo di Dijkstra ha complessità $O((n + m) \log n)$. Riferendosi al codice sopra descritto, l'analisi è la seguente:

- inserire tutti i vertici in Q col loro valore di chiave iniziale può essere fatto in $O(n \log n)$ ripetendo gli inserimenti, o in $O(n)$ con la costruzione dell'heap bottom-up;

- per ogni iterazione del while , spendiamo $O(\log n)$ per rimuovere i vertici u da Q, e $o(degree(v) \log n)$ per fare il rilassamento sui vertici incidenti u;

- il totale tempo del while è:

$$\sum_{v \ in \ G} 1 + degree(v) \log(n)$$

che corrisponde a $O((n+m)\log n)$ dalla Proposizione 2.1 Ricordando inoltre, dalla Proposizione 2.3 che $m \leq n(n\text{-}1)/2 \approx n^2$, viene immediato semplificare la complessità a $O(m \log n)$.

Nelle Immagini 3.2 e 3.3 si illustra l'esecuzione dell'algoritmo di Dijkstra.

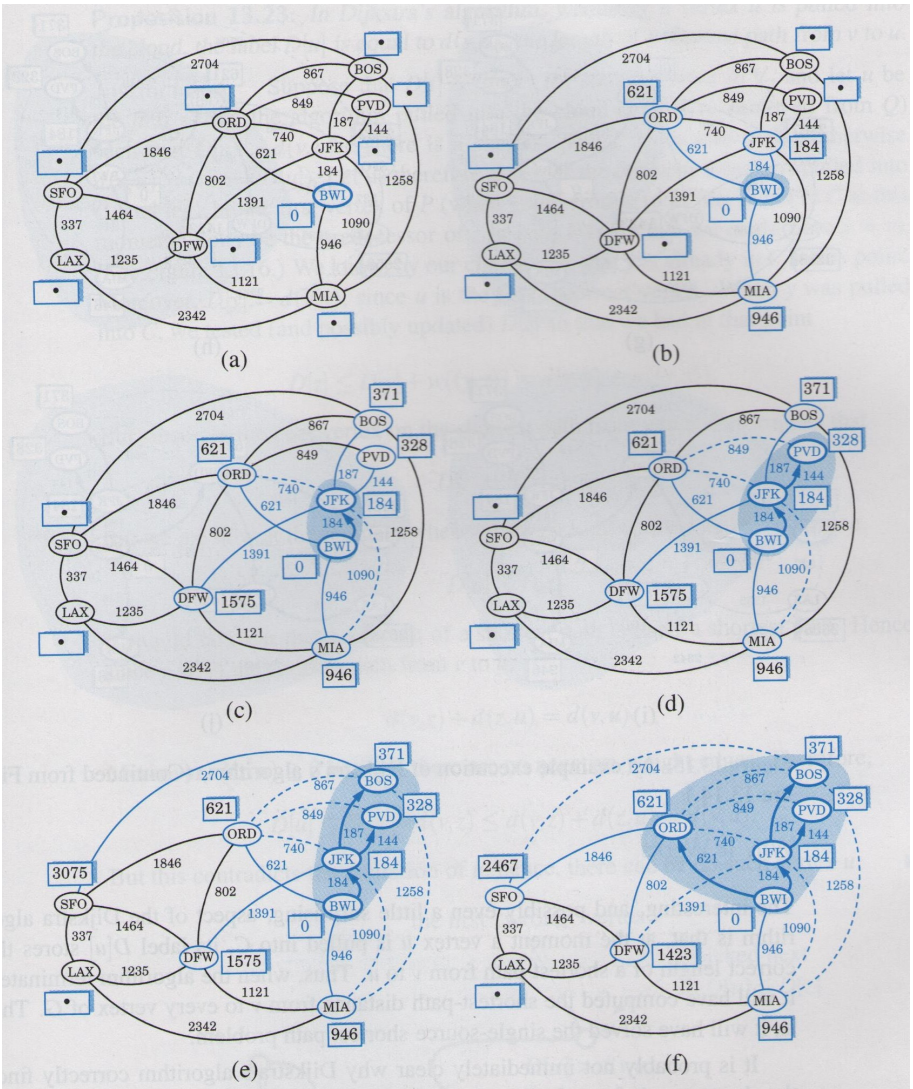


Figura 3.2: Esecuzione algoritmo di Dijkstra-1

3.1.2 Algoritmo di Bellman-Ford

L'algoritmo di Bellman-Ford calcola i shortest path da un vertice target agli altri vertici del grafo diretto e pesato e, a differenza dell'algoritmo di Dijkstra, alcuni pesi degli archi possono essere negativi anche se comunque

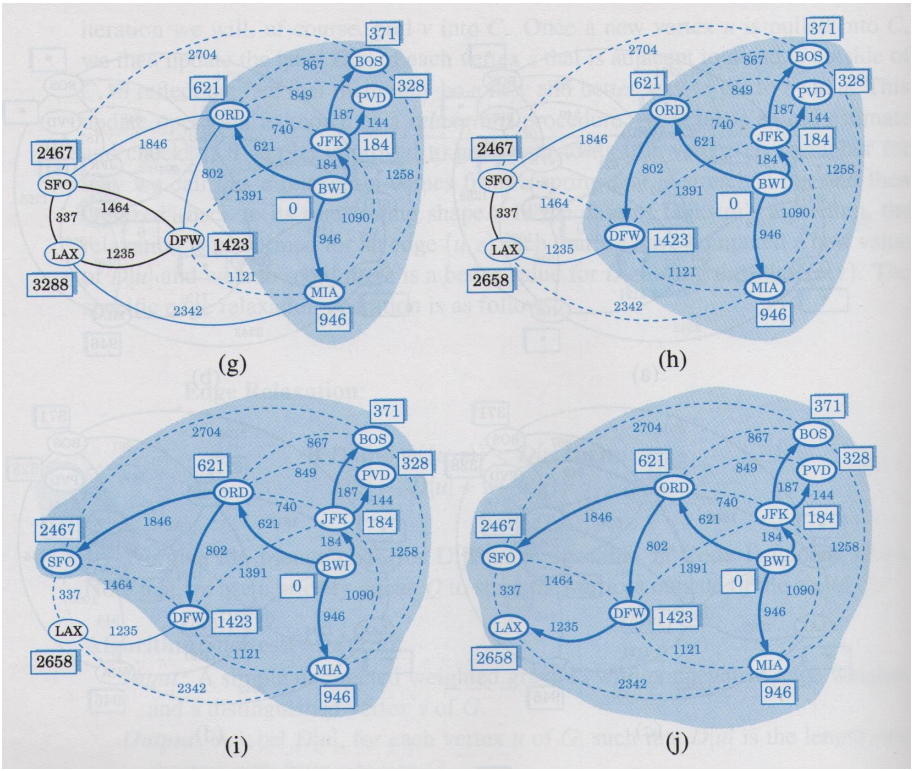


Figura 3.3: Esecuzione algoritmo di Dijkstra-2

non sono accettati cicli negativi. Dato in input un grafo diretto e pesato $G{=}(V, E)$ con un vertice sorgente s e una funzione peso $w{:}E{\to}\Re$, l'algoritmo di Bellman-Ford restituisce un valore booleano che indica se ci sono o no cicli negativi raggiungibili dal vertice sorgente. Se il valore booleano è **FALSE**, l'algoritmo indica che non esistono soluzioni. In caso contrario l'algoritmo produce i shortest path e i loro valori. Questo algoritmo è strutturalmente molto simile a quello di Dijkstra; tuttavia, mentre quest'ultimo selezionava il vertice di peso minimo tra quelli non ancora processati, con tecnica greedy, l'algoritmo di Bellman-Ford semplicemente processa tutti gli archi e lo fa n volte, dove n è il numero di vertici nel grafo. Le ripetizioni permettono alle distanze minime di propagarsi in modo preciso attraverso il grafo, poiché, in assenza di cicli negativi, il shortest path può solo visitare ciascun vertice al più una volta. Diversamente da quello con tecnica greedy, che dipende da certe assunzioni derivate dai pesi positivi fatte a priori, questo semplice approccio si applica al caso più generale.

Algoritmo `Bellman-Ford`(G, w, s)
input `Un grafo diretto` G `senza restizioni sul peso degli archi,`
`un vertice` $v \in G$ `e una funzione peso` w
output `Un valore booleano che indichi se l'algoritmo ha soluzioni e in caso affermativo`
`i shortest path e i relativi pesi dal vertice target`

for each `vertex` $v \in V[G]$ **do**
    $D[v]{\leftarrow}\infty$ `//D e' la stessa etichetta utilizzata nell'algoritmo di Dijkstra`
    $\pi[v]{\leftarrow}NIL$ `//dove` $\pi[v]$ `e' il vertice analizzato prima di v,`
`cioe' il predecessore`
$D[s]{\leftarrow}$ `0`
for `i`$\leftarrow$ **to** $|V[G]|$ `-1`
      do for each `edge` $(u, v) \in E[G]$
      `procedura di rilassamento degli archi`
  for each `edge` $(u, v) \in E[G]$
      do if $D[v] > D[u]$ `+` $w(u, v)$
      then return `FALSE`
return `TRUE`

L'algoritmo di Bellman-Ford ha una complessità temporale $O(n \cdot m)$, dove n ed m sono rispettivamente il numero di vertici e di archi del grafo, poiché il primo ciclo richiede tempo $\Theta(n)$, il secondo ″for″ richiede $|V|\Theta(E)$ e l'ultimo ciclo ″for″ ha un tempo di $O(E)$. Si deduce dunque che, nel caso non siano presenti pesi negativi, sia più efficiente l'utilizzo dell'algoritmo di Dijkstra che invece ha complessità $O(m \log n)$. Si mostra di seguito in immagine 3.4 l'esecuzione dell'algoritmo. Il vertice s è il vertice sorgente. I valori dell'etichetta D sono inseriti all'interno dei

vertici, e gli archi ombreggiati indicano i valori del predecessore: se l'arco (u, v) è ombreggiato, allora il predecessore di v $\pi[v] = u$. In questo esempio, le operazioni di rilassamento dell'arco vengono eseguite nell'ordine (t, x),(t, y),(t, z),(x, t),(y, x),(y, z),(z, x),(z, s),(s, t),(s, y). L'algoritmo di Bellman-Ford resituisce **TRUE** in questo esempio.

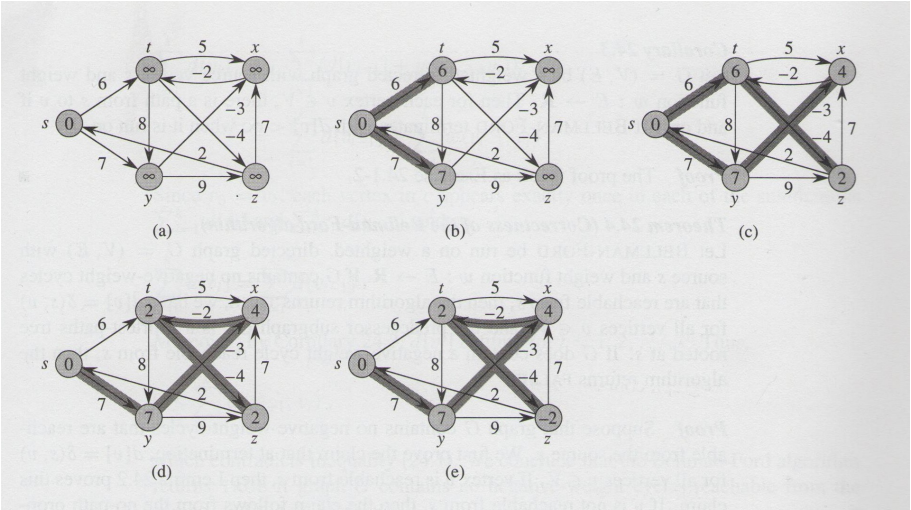


Figura 3.4: Esecuzione algoritmo di Bellman-Ford

3.2 Minimum Spanning Tree

Si supponga di voler connettere tutti i computer in un ufficio usando il minimo numero di fili. Possiamo usare un grafo G i cui vertici sono computer e i cui archi rappresentano tutte le possibile coppie (u, v) di computer dove il perso $w((v, u))$ dell'arco (v, u) è uguale alla lunghezza totale di cavo necessaria per connettere v a u. Più che un shortest path, ci serve un albero T che contenga tutti i vertici di G e che abbia il minimo peso totale di tutti gli alberi possibili. Cerchiamo di dare una definizione più formale del problema appena presentato. Dato un grafo non diretto pesato G , siamo interessati a trovare un albero T che contenga tutti i vertici in G e che minimizzi la somma

$$w(T) = \sum_{(v,u)\ in\ T}\ w(v, u)$$

Un albero come questo che contiene tutti i vertici di un grafo connesso G è detto spanning tree e nella fattispecie il problema di trovare uno spanning tree T col minor peso totale degli archi è un problema di ricerca del minimum spanning tree.

Viene ora enunciata una proposizione che sarà alla base degli algoritmi successivi in quanto garantisce che la scelta dell'arco con peso minimo. Sarà tutto più chiaro successivamente.

Proposizione 3.2. *Sia G un grafo connesso e sia $V = V_1 \cup V_2$ una partizione di vertici di G in due insiemi disgiunti non vuoti. In più, sia e un arco di G con il minimo peso tra tutti quelli con un estremo in V_1 e l'altro in V_2 .C'è un minimo albero ricoprente T che ha e come uno di questi archi.*

Dimostrazione: Sia T il minimo albero ricoprente di G. Se T non contiene l'arco e, l'aggiunta di e a T crea un ciclo. Di conseguenza, c'è qualche arco f di questo ciclo che ha un estremo in V_1 e uno in V_2. In più, dalla scelta di e, $w(e) \leq w(f)$. Se si rimuove f da $T \bigcup \{ \text{ e } \}$, otteniamo un albero ricoprente il cui peso totale degli archi non è maggiore di prima. Poiché T era in minimo albero ricoprente , lo sarà anche quello ottenuto dopo la rimozione di f.

3.2.1 Algoritmo di Kruskal

L'idea fondamentale di questo algoritmo è creare il minimum spanning tree a cluster. Inizialmente, ogni vertice è in un cluster distinto dal vertice stesso. L'algoritmo, dunque, considera ogni arco a turno, in ordine crescente di peso. Se un arco e connette due cluster differenti, allora e è aggiunto all'insieme degli archi del minimo albero ricoprente, e i due gruppi connessi da e vengono fusi in un singolo cluster. Se d'altro canto e connette due vertici che sono già nello stesso cluster, allora viene scartato. Una volta che l'algoritmo ha inserito abbastanza archi per formare uno spanning tree, cioè da fare in modo che l'insieme di vertici sia connesso, termina e restituisce questo albero che è anche il minimo.
Il fatto che sia minimo è garantito dalla proposizione 2.2.

Algoritmo `Kruskal`(G)
input `Un grafo non diretto semplice` G `con` n `vertici e` m `archi`
output `Un minimo albero ricoprente` T `per` G

for `each vertice` v `in` G **do**
  `Si definisce una nube elementare` $C(v) \longleftarrow v$
  `Inizializza una piority queue` Q `che contenga tutti gli archi di` G`,`
  `usando i pesi come chiave`
  $T \longleftarrow \emptyset$ (T `alla fine contiene gli archi`
  `del minimum spanning tree)`
  while T `ha meno di` $n - 1$ `archi` **do**
    $(u, v) \longleftarrow Q$`.removeMin()`
    `Sia` $C(v)$ `la nube contenente` v`, e sia` $C(u)$ `la nube contenente` u`.`

if $C(v) \neq C(u)$ **then**
`Aggiungi l'arco` (v, u) `a` T
`Fondere` $C(v)$ `e` $C(u)$ `in una nube unica //Passo fondamentale`
return T

Nelle Immagini 3.5, 3.6 e 3.7 si illustra l'esecuzione dell'algoritmo di Kruskal.

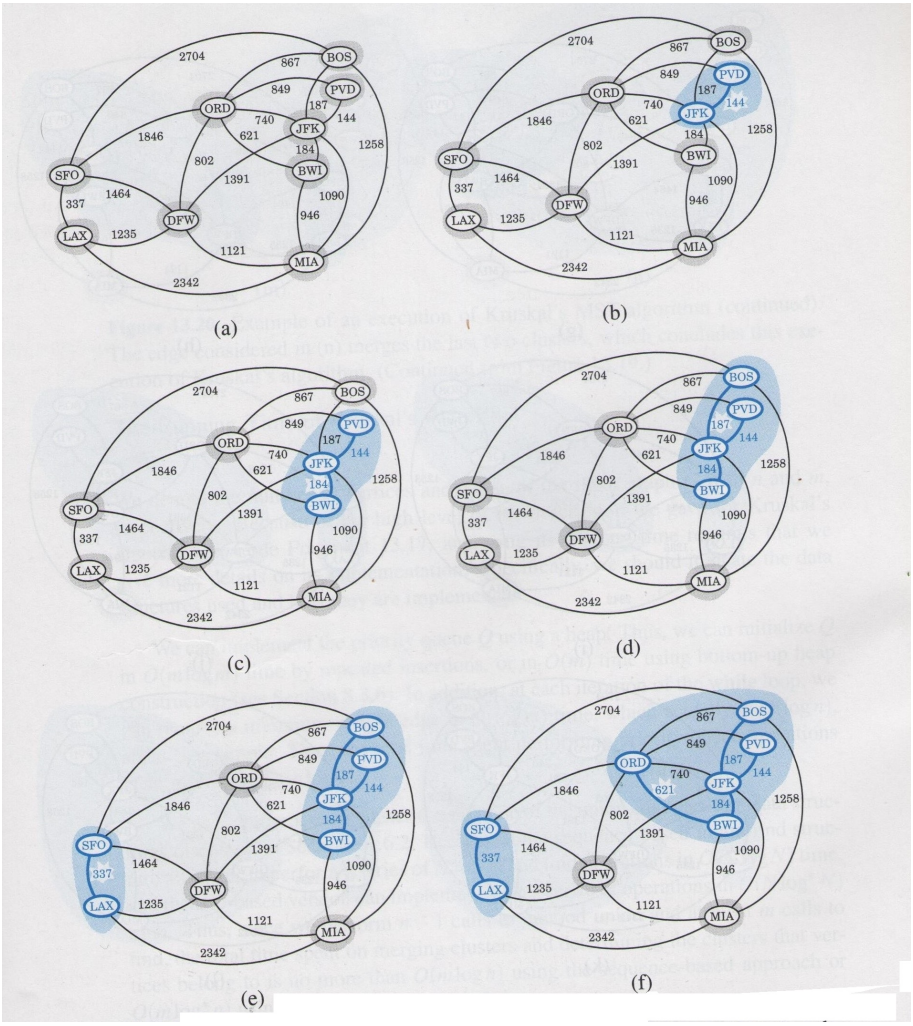


Figura 3.5: Esecuzione algoritmo di Kruskal-1

Complessità algoritmo di Kruskal

Sia G un grafo con n vertici e m archi. Sia Q una coda con priorità implementata con uno heap. Possiamo dunque inizializzare Q con $O(m \log m)$

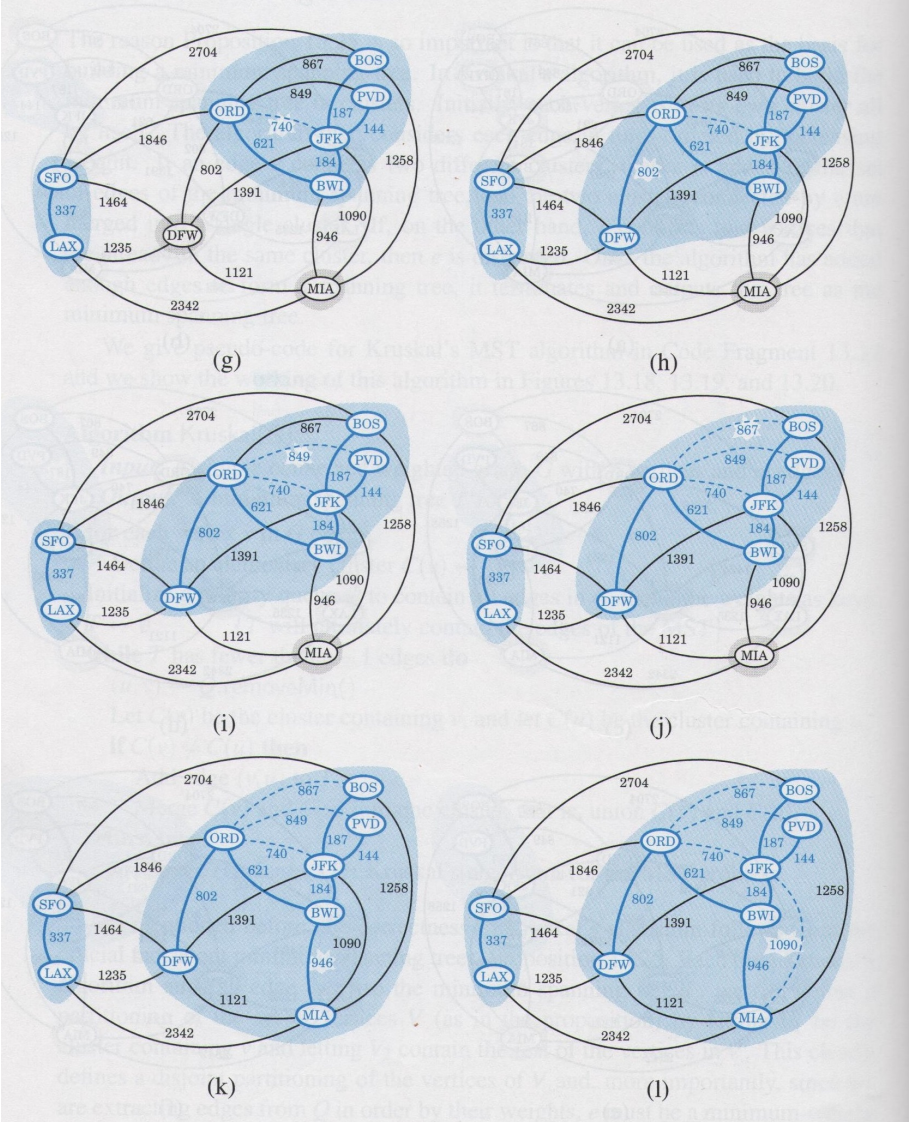


Figura 3.6: Esecuzione algoritmo di Kruskal-2

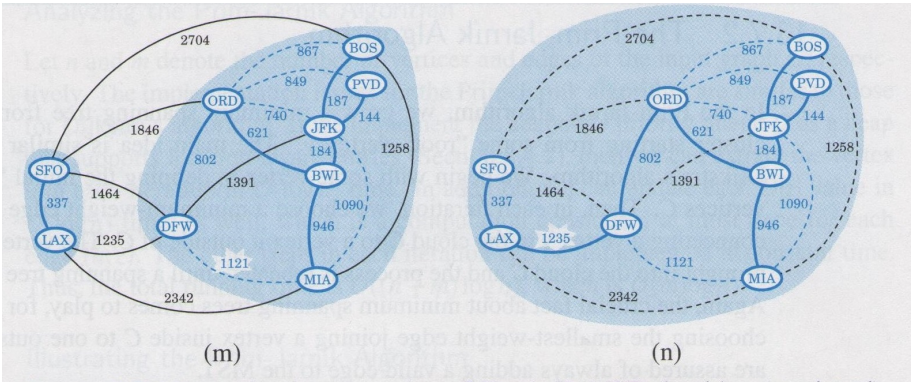


Figura 3.7: Esecuzione algoritmo di Kruskal-3

inserimenti continui o in $O(m)$ usando il bottom up heap. In ogni iterazione del ciclo while, possiamo rimuovere un arco di peso minimo in $O(\log m)$ che in realtà è $O(\log n)$ poiché G è semplice. Dunque, il tempo totale speso per le operazione fatte dalla priority queue non è più di $O(m \log n)$. Quindi, per le stesse motivazioni esplicate per algoritmo di Dijkstra, concludiamo che la complessità dell'algoritmo di Kruskal è $O((m + n) \log n)$ che può essere semplificato in $O(m \log n)$ ricordando la Proposizione 2.3.

3.2.2 Algoritmo di Prim-Jarník

Nell'algoritmo di Prim-Jarník viene creato uno spanning tree a partire da un singolo cluster cominciando da una radice v. L'idea principale è simile a quella dell'algoritmo di Dijkstra. Si comincia da v, inserendolo nella nube iniziale di vertici C. Quindi, in ogni iterazione, si sceglie un arco $e = (v, u)$ con peso minimo , connettendo un vertice v già presente nella nube C ad un vertice u fuori da C. Il vertice u è dunque inserito nella nube C e il processo continua finché non è formato l'intero albero. Il punto cruciale è sempre quello di scegliere al meglio l'arco con peso minimo, e questo viene garantito dalla Proposizione 2.2. Per implementare efficientemente questo approccio, prendiamo un altro spunto dall'algoritmo di Dijkstra. Manteniamo un'etichetta $D[u]$ per ogni vertice u fuori dalla nube C, così che $D[u]$ memorizzi il peso del miglior arco corrente per unire u alla nube C. Queste etichette ci permettono di ridurre il numero di archi che dobbiamo considerare nel decidere che vertice sarà il prossimo da inserire nella nube.

Algoritmo `PrimJarnik`(G)
input `Un grafo non diretto semplice` G `con` n `vertici e` m `archi`
output `Un minimo albero ricoprente` T `per` G

```
Si prenda qualsiasi vertice v di G
D[v] ⟵ 0
for each vertice v ≠ u do
  D[u] ⟵ ∞
  T ⟵ ∅
  Inizializza una piority queue Q con una entry ((u, null ), D[u]
  per ogni vertice u dove (u, null ) sia l'elemento e D[u]) sia la chiave )
  while Q ≠ ∅ do
    (u, e) ⟵ Q.removeMin()
    Aggiungi u e e a T
    for each vertice z adiacente a u tale che z sia in Q do
       utilizzare la tecnica di rilassamento su (u, z)
      if w((u, z)) < D[z then
      D[z] ⟵ w((u, z))
      cambia a (z, (u, z)) l'elemento del vertice z ∈ Q
      cambia a D[z] la chiave del vertice z ∈ Q
return T
```

Complessità dell'algoritmo Prim-Jarník

Siano n e m il numero di vertici e archi del grafo G. Se implementiamo la priority queue Q come uno heap allora possiamo estrarre il vertice u in ogni iterazione in $O(\log n)$. In più possiamo aggiornare ogni $D[z]$ in $O(\log n)$, essendo una procedura considerata al più per ogni arco (u, z). Le altre operazioni eseguite in ciascuna iterazione possono essere implementate in tempo costante. Dunque, il tempo totale è $O((m + n) \log n)$ come per Dijkstra, che è $O(m \log n)$.

Nelle Immagini 3.8 e 3.9 si illustra l'esecuzione dell'algoritmo di Prim-Jarník .

Gli algoritmi di Dijkstra, Kruskal, Prim-Jarník sono stati studiati in [2] come le relative immagini e le definizioni principali. Per quanto riguarda l'algoritmo di Bellman-Ford, le informazioni principali sono ricavate da [1] e [4].

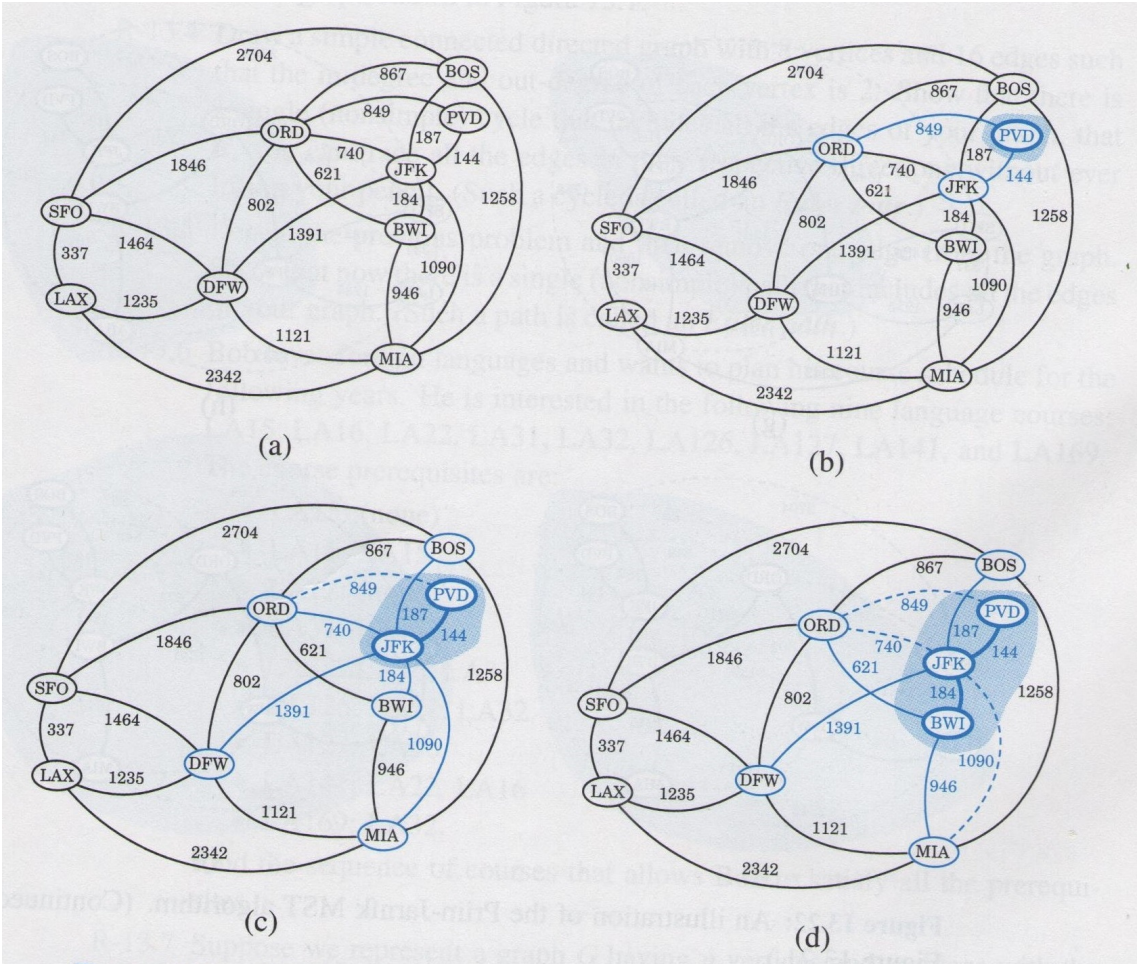


Figura 3.8: Esecuzione algoritmo di Prim-Jarník-1

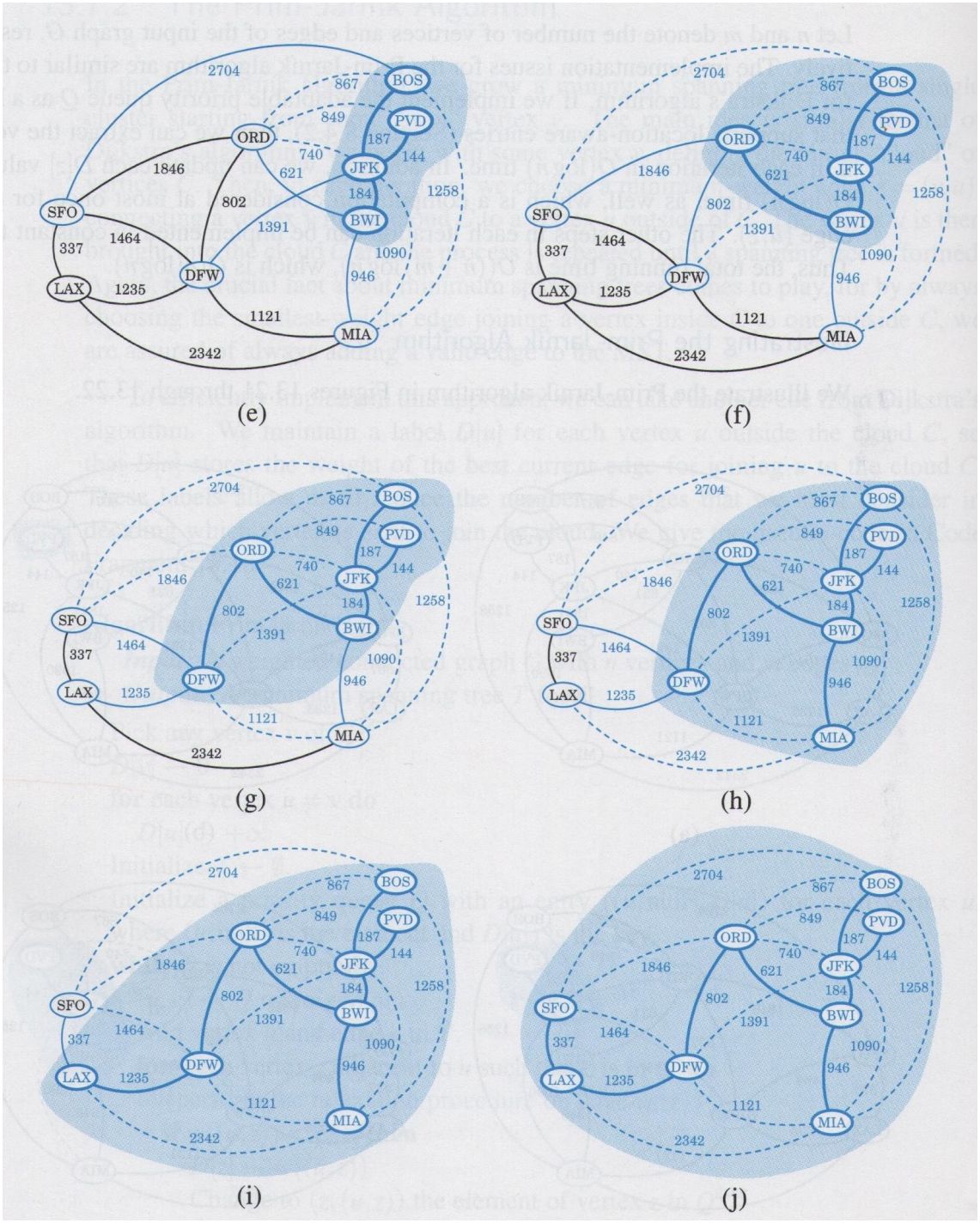


Figura 3.9: Esecuzione algoritmo di Prim-Jarník-2

Capitolo 4

All-pairs shortest path

Il problema qui presentato è quello di trovare i più brevi cammini tra tutte le coppe di vertici di un grafo. Questo problema può nascere, per esempio, nel creare una tabella delle distanza tra le coppie di città per un navigatore. Si riceve in input un grafo G pesato e viene restituito in output per ogni coppa u, v di vertici appartenenti a V , un cammino minimo da u a v. Tipicamente il risultato sarà una matrice, dove la coppia in riga u e in colonna v contiene il valore del peso del cammino minimo da u a v. Si può risolvere questo problema iterando un single source shortest path algorithm $|V|$ volte, una per ogni vertice del grafo. Se i pesi sono tutti non negativi, il tempo è $O(V * (V + E)\ \log\ V)$(si pensi all'algoritmo di Dijkstra), ipotizzando la priority queue implementata da uno heap. Se i pesi sono anche negativi possiamo usare l'algoritmo di Bellman-Ford ottenendo una complessità $O(V^2E)$ che su un grafo denso è (V^4). L'obiettivo è di cercare di trovare una soluzione più efficiente. Per convezione, assumiamo che i vertici siano numerati $1, 2, ...,$ $|V|$ in modo che l'input sia un a matrice $n \times n$ W che rappresenta i pesi degli archi di un grafo diretto a n vertici G=(V, E). Cioè, W=(w_{ij}) dove

$$w(ij) = \begin{cases} 0, \ \ se\ i = j \\ peso\ \ i, j, \ \ se\ i \neq j\ \ (i, j) \in E \\ \infty, \ \ se\ i \neq j\ \ (i, j) \notin E \end{cases} \qquad (4.1)$$

Sono ammessi archi con peso negativo, ma non cicli con peso negativo. La tabella di output è una matrice $n \times n$ D=d_{ij}, dove d_{ij} contiene il peso di un cammino minimo da i a j. Cioè, se δ_{ij} è il cammino minimo da i a j, allora sarà d_{ij}=δ_{ij} alla fine. Per risolvere questo problema si cercherà non solo il peso dei shortest path ma anche la matrice del predecessore Π=π_{ij} dove π_{ij} è NIL sia se i=j oppure se non c'è un cammino da i a j, e altrimenti π_{ij} è il predecessore di j in qualche percorso minimo da i a j. Il sottografo ricavato dalla i-esima riga della matrice Π dovrebbe essere un albero dei shortest path con radice i. Per ogni vertice i appartenete a V, definiamo sottografo predecessore di G per i il grafo $G_{\pi,i}$=$(V_{\pi,i}, E_{\pi,i})$, dove

$V_{\pi,i}$= { j ∈ V:$\pi_{ij} \neq$ NIL} $\bigcup$ {i}

and

$E_{\pi,i}$= $\{(\pi_{ij}, j) : j \in V_{\pi,i} - \{i\}\}$

4.1 Shortest path tramite moltiplicazione di matrici

Viene ora presentato un algoritmo di programmazione dinamica per risolvere il problema del all-pairs shortest path nel grafo diretto G=(V, E). I passi procedurali necessari per sviluppare un algoritmo di programmazione dinamica sono: caratterizzare una struttura per una soluzione ottima; definire ricorsivamente il valore di una soluzione ottima e trovare il valore di una soluzione ottima in un metodo bottom-up.

In questo caso specifico ogni ciclo più esterno apparirà come una moltiplicazione tra due matrici, in cui $\cdot \rightarrow +$ e $+ \rightarrow$min. Saranno presentati due algoritmi, uno con tempo di esecuzione $\Theta(V^4)$ e l'altro, più raffinato, di complessità $\Theta(V^3 \cdot \log V)$.

Lemma 4.1.1. *Dato un grafo diretto e pesato G=(V,E) con la funzione peso w:E⟶ R, sia p=⟨1...k⟩ un cammino minimo da* 1 *a k e, per ogni i e j tali che 1≤i≤j≤k, sia p_{ij}=⟨i, .., j⟩ un sotto-cammino di p dal vertice i a j. Allora p_{ij} è un cammino minimo da i a j.*

Supponiamo che il grafo G sia rappresentato da una matrice di adiacenza W=(w_{ij}). Consideriamo un cammino minimo p da i a j, e supponiamo che p contenga al più $m-1$ archi. Assumendo che non ci siano cicli con peso negativo, m è finito. Se i=j, allora p ha peso 0 e non ha archi. Se $i \neq j$, allora decomponiamo p in $i \hookrightarrow k \rightarrow j$ dove p'=⟨i...k⟩ ora contiene al più $m-1$ archi. Dal lemma 4.1.1 , p' è un cammino minimo da i a k, e quindi $\delta(i,j)$=$\delta(i,k)$+w_{kj}. Ora, sia $l_{ij}^{(m)}$ un peso minimo di ogni percorso da i a j che contiene al più m archi. Quando m=0, c'è un cammino minimo da i a j senza archi se e solo se j=i. Cioè,

$$l_{ij}^{(0)} = \begin{cases} 0, & \text{se } i = j \\ \infty, & \text{se } i \neq j \end{cases}$$

Per m≥ 1, calcoliamo $l_{ij}^{(m)}$ come il più piccolo tra i $l_{ij}^{(m-1)}$ e (il peso del cammino minimo da i a j consistendo al più in m-1 archi) e il peso minimo di ogni cammino da i a j che contiene al più m archi, ottenuto osservando tutti i possibili predecessori k di j. Si definisce ricorsivamente

$$l_{ij}^{(m)} = \min(l_{ij}^{(m-1)}, \min_{1\leq k\leq n}\left\{l_{ik}^{(m-1)} + w_{kj}\right\}) \;=\; \min_{1\leq k\leq n}\left\{l_{ik}^{(m-1)} + w_{kj}\right\},$$

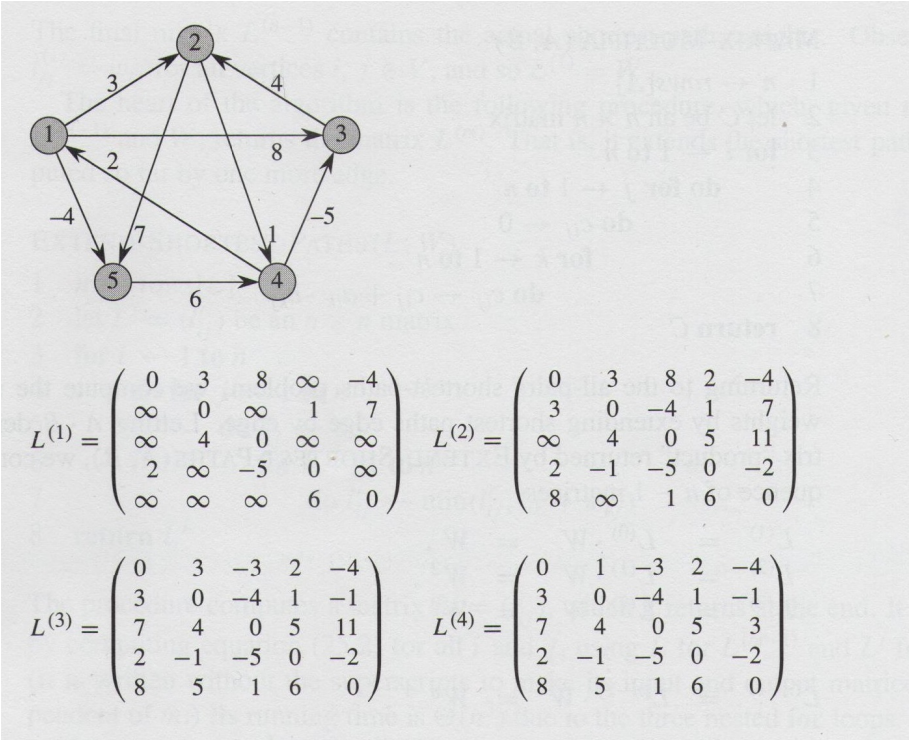
ricordando che w_{jj}=0. Se il grafo non contiene cicli a peso negativo, allora per ogni copia di vertici i e j per ogni $\delta(i,j)<\infty$, c'è un cammino minimo da i a j che sia semplice e quindi contiene al più n-1 archi. Un cammino da i a j con più di n-1 archi non può avere in peso minore del cammino minimo da i a j. I pesi del cammino minimo sono dunque dati da:
$\delta(i,j)=l_{ij}^{(n-1)}=l_{ij}^{(n)}=l_{ij}^{(n+1)}$. (4.2)
Prendendo come input la matrice $W=(w_{ij})$, calcoliamo ora una serie di matrici $L^{(1)},..,L^{(n-1)}$, dove per $m=1,..,n-1$, abbiamo $L^{(m)}=(l_{ij}^{(m)})$. La matrice finale $L^{(n-1)}$ contiene i pesi definitivi dei shortest path. Si osservi che $l_{ij}^{(1)}=w_{ij}$, per ogni $i,j\in V$ e quindi $L^{(1)}=W$. Il cuore dell'algoritmo è la seguente procedura che , data una matrice $L^{(m-1)}$ e W, restituisce la matrice $L^{(m)}$. Cioè, estende i shortest path, calcolati fino ad ora, di un arco in più. Il suo tempo di computazione è $\Theta(n^3)$ per i 3 cicli "for" innestati.

Algoritmo `Extend-shortest-paths`(L,W)
input: `matrici` $L^{(m-1)}$ `e` W
output: `matrice` $L^{(m)}$
$n \longleftarrow rows[L]$
`sia` $L' \equiv (l'_{ij})$ `una matrice` $n * n$
for $i \longleftarrow$ `1` **to** n **do**
  for $j \longleftarrow$ `1` **to** n **do**
    $l'_i j \longleftarrow \infty$
      for $k \longleftarrow$ **to** n **do**
      $l'_{ij} \longleftarrow \min\ (l'_{ij}, l'_{ik} + w_{kj})$
return L'

Calcoliamo i pesi dei cammini minimi estendendo i shortest path da arco ad arco. Si consideri il prodotto di matrici calcolato con la procedura precedente, computiamo la sequenza di n-1 matrici
$L^{(1)}=L^{(0)}*W= W$,
$L^{(2)}=L^{(1)}*W= W^2$,
$L^{(3)}=L^{(2)}*W= W^3$,
...
...
$L^{(n-1)}=L^{(n-2)}*W=W^{n-1}$
Da quanto discusso precedentemente, la matrice $L^{(n-1)}=W^{(n-1)}$ contiene i pesi dei shortest path. La seguente procedura calcola la sequenza di matrici in $\Theta(n^4)$:

Algoritmo `Slow-all-pairs-shortest-paths`(W)
input: `matrice` W `definita in 4.1`
output: `matrice` $L^{(n-1)}$ `contenente i pesi dei shortest path`
$n \longleftarrow rows[W]$
$L^{(1)} \longleftarrow W$
for $m \longleftarrow$ `2` **to** n`-1` **do**
  $L^{(m)} \longleftarrow$ `Extend-shortest-paths`$(L^{(m-1)}, W)$
return$L^{(n-1)}$

In Figura 4.1 viene mostrata l'esecuzione dell'algoritmo per il grafo lì rappresentato.



$$L^{(1)} = \begin{pmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & \infty & -5 & 0 & \infty \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \quad L^{(2)} = \begin{pmatrix} 0 & 3 & 8 & 2 & -4 \\ 3 & 0 & -4 & 1 & 7 \\ \infty & 4 & 0 & 5 & 11 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & \infty & 1 & 6 & 0 \end{pmatrix}$$

$$L^{(3)} = \begin{pmatrix} 0 & 3 & -3 & 2 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 11 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{pmatrix} \quad L^{(4)} = \begin{pmatrix} 0 & 1 & -3 & 2 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 3 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{pmatrix}$$

Figura 4.1: Grafo diretto e sequenza delle matrici L^m calcolata con Slow-all-pairs-shortest-paths

L'obiettivo comunque non è computare tutte le matrici $L^{(m)}$: l'interesse è rivolto solo alla matrice $L^{(n-1)}$. Si ricordi che per la (4.2), l'assenza di cicli a peso negativo, implica che $L^{(m)}$=$L^{(n-1)}$ per tutti gli interi $m \geq n$-1. Poiché la moltiplicazione tradizionale di matrici è associativa, lo è anche il prodotto di matrici definito dal codice dell'Extend-shortest-paths. Quindi, possiamo calcolare $L^{(n-1)}$ con solo $\lceil(\log(n-1))\rceil$ prodotti di matrici calcolando la

sequenza
$L^{(1)}$=W ,
$L^{(2)}$=W^2=$W * W$,
$L^{(4)}$=W^4=$W^2 * W^2$,
...
...
$L^{(2^{\lceil(\log(n-1))\rceil})}$=$W^{(2^{\lceil(\log(n-1))\rceil})}$=$W^{(2^{\lceil(\log(n-1)-1)\rceil})} * W^{(2^{\lceil(\log(n-1)-1)\rceil})}$
poiché $2^{(\lceil(\log(n-1))\rceil)} \geq$ n-1, il prodotto finale è uguale a $L^{(n-1)}$. La seguente procedura calcola la sequenza di matrici di interesse usando la tecnica della ripetizione quadratica.

Algoritmo `Faster-all-pairs-shortest-paths`(W)
input: `matrice` W `definita in 4.1`
output: `matrice` $L^{(m)}$ `soluzione del problema computazionale`
$n \longleftarrow rows[W]$
$L^{(1)} \longleftarrow W$
$m \longleftarrow$ `1`
while $m < n -$ `1` **do**
  $L^{(2m)} \longleftarrow$ `Extend-shortest-paths`$(L^{(m)}, L^{(m)})$
  $m \longleftarrow 2m$
return $L^{(m)}$

il tempo è $\Theta(n^3 \log n)$ poiché i $\lceil \log(n-1) \rceil$ prodotti di matrici necessita $\Theta(n^3)$ per l'algoritmo Extend-shortest paths. .

4.2 Algoritmo di Floyd-Warshall

Nello sviluppare questo alogoritmo (anch'esso di programmazione dimnamica) si consideri possibile la presenza di archi con peso negativo ma non cicli con peso negativo.

Nell'algoritmo Floyd-Warshall si usa una differente caratterizzazione della struttura del cammino minimo che è stato usato nell'algoritmo basato sulla moltiplicazione di matrici. Questo considera i vertici intermedi di un cammino minimo, dove un vertice intermedio di un cammino semplice $p = \langle 1, .., l \rangle$ è ogni vertice di p diverso da 1 o l, cioè appartenente all'insieme $\{2, ..., l-1\}$. L'algoritmo si basa sulla seguente osservazione. Si prenda un sottoinsieme dei vertici $\{1...k\}$ per qualche k<$|V|$. Per ogni coppia di vertici $i, j \in V$ si considerino tutti i cammini da i a j i cui vertici intermedi sono estratti da $\{1...k\}$ e sia p un cammino (semplice) a peso minimo tra tutti loro. L'algoritmo estrapola una relazione tra il percorso p e i percorsi minimi da i a j con tutti i vertici intermedi nell'insieme $\{1...k-1\}$. La relazione dipende dal fatto se k è un vertice intermedio del percorso p o no (si veda la figura 4.2):

- se k non è un vertice intermedio, allora i vertici intermedi del percorso p sono nell'insieme $\{1..k-1\}$. Cioè, un percorso minimo dal vertice i al vertice j come tutti i vertici intermedi in $\{1....k-1\}$ è anche un percorso minimo da i a k con tutti i vertici intermedi nell'insieme $\{1...k\}$;

- se k è un vertice intermedio del percorso, allora rompiamo p in i$\hookrightarrow p_1 \hookrightarrow$ k$\hookrightarrow p_2 \hookrightarrow$ j come mostrato in figura 4.2. Dal lemma 4.1.1, p_1 è un percorso minimo da i a k con tutti i vertici intermedi nell'insieme $\{1...k-1\}$. Poiché il vertice k non è un vertice intermedio del percorso p_1, vediamo che p_1 è un cammino minimo da i a k con tutti i vertici intermedi nell'insieme $\{1...k-1\}$. Allo stesso modo, p_2 è un cammino minimo dal vertice k al vertice j con tutti i vertici intermedi nell'insieme $\{1...k-1\}$.

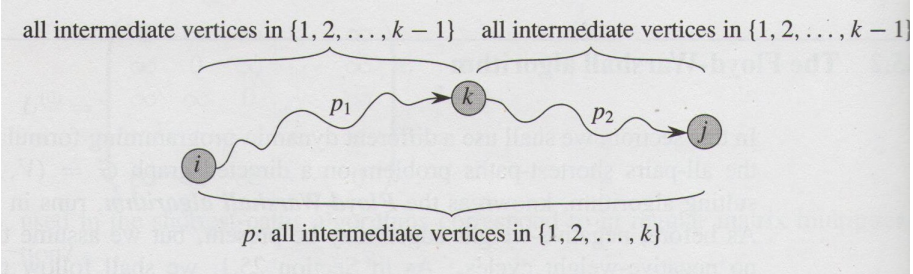


Figura 4.2: Rappresentazione dei vertici intermedi

Basata sull'osservazione precedente, si definisce una formulazione ricorsiva dei shortest path che sia differente da quella vista nel metodo della moltiplicazione di matrici. Sia $d_{ij}^{(k)}$ il peso del cammino minimo dal vertice i al vertice j per cui tutti i vertici intermedi sono nell'insieme $\{1....k\}$. Quando k=0, un percorso dal vertice i al vertice k senza vertici intermedi numerati con più di 0 , non ha vertici intermedi. Un percorso tale ha al più un arco e quindi $d_{ij}^{(0)}$=w_{ij}. Una definizione ricorsiva è

$$d_{ij}^{(k)} = \begin{cases} w_{ij}, & se\ k = 0 \\ \min(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}), & se\ k \geq 1\ \ i,j \notin E \end{cases} \qquad (4.3)$$

Poichè per ogni percorso tutti i vertici intermedi sono nell'insieme $\{1....n\}$, la matrice $D^{(n)}$=$(d_{ij}^{(n)})$ dà la soluzione finale: $d_{ij}^{(n)}$=$\delta(i,j)$ per tutti gli $i,j \in V$. Basata sulla ricorrenza (4.3), la seguente procedura può essere utilizzata per calcolare il valore di $d_{ij}^{(k)}$ al crescere del valore di k. L'input è una matrice

$n \times n$ W definita nell'equazione (4.1). La procedura restituisce la matrice $D^{(n)}$ dei pesi dei shortest path.

Algoritmo `Floyd-Warshall`(W)
input: `matrice` W `definita in 4.1`
output: `matrice` $D^{(n)}$ `contentente i pesi dei shortest path`
$n \longleftarrow rows[W]$
$D^{(0} \longleftarrow W$
for $k \longleftarrow$ `1` **to** n
  do for $i \longleftarrow$ `1` **to** n **do**
    for $j \longleftarrow$ `1` **to** n **do**
      $d_{ij}^{(k)} \longleftarrow \min\ (d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}$
return $D^{(n)}$

Il tempo di esecuzione dell'algoritmo è determinata dal triplo ″for″ innestato . Poiché ogni confronto per trovare il minimo richiede $O(1)$, l'algoritmo viene eseguito in tempo $\Theta(n^3)$. In Figura 4.3 viene mostrata l'esecuzione dell'algoritmo. Si noti, in conclusione, che grazie alle supposizioni fatte precedentemente, l'algoritmo di Floyd-Warshall sia da preferire di gran lunga rispetto a quello che sfrutta la moltiplicazione tra matrici. Questo lo si vede in termini di complessità per le analisi svolte in questo capitolo. Le informazioni relative a questo capitolo, come le immagini, sono state ricavate da [1].

$$D^{(0)} = \begin{pmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & \infty & -5 & 0 & \infty \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \quad \Pi^{(0)} = \begin{pmatrix} \text{NIL} & 1 & 1 & \text{NIL} & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 2 & 2 \\ \text{NIL} & 3 & \text{NIL} & \text{NIL} & \text{NIL} \\ 4 & \text{NIL} & 4 & \text{NIL} & \text{NIL} \\ \text{NIL} & \text{NIL} & \text{NIL} & 5 & \text{NIL} \end{pmatrix}$$

$$D^{(1)} = \begin{pmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & 5 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \quad \Pi^{(1)} = \begin{pmatrix} \text{NIL} & 1 & 1 & \text{NIL} & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 2 & 2 \\ \text{NIL} & 3 & \text{NIL} & \text{NIL} & \text{NIL} \\ 4 & 1 & 4 & \text{NIL} & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 5 & \text{NIL} \end{pmatrix}$$

$$D^{(2)} = \begin{pmatrix} 0 & 3 & 8 & 4 & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & 5 & 11 \\ 2 & 5 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \quad \Pi^{(2)} = \begin{pmatrix} \text{NIL} & 1 & 1 & 2 & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 2 & 2 \\ \text{NIL} & 3 & \text{NIL} & 2 & 2 \\ 4 & 1 & 4 & \text{NIL} & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 5 & \text{NIL} \end{pmatrix}$$

$$D^{(3)} = \begin{pmatrix} 0 & 3 & 8 & 4 & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & 5 & 11 \\ 2 & -1 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \quad \Pi^{(3)} = \begin{pmatrix} \text{NIL} & 1 & 1 & 2 & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 2 & 2 \\ \text{NIL} & 3 & \text{NIL} & 2 & 2 \\ 4 & 3 & 4 & \text{NIL} & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 5 & \text{NIL} \end{pmatrix}$$

$$D^{(4)} = \begin{pmatrix} 0 & 3 & -1 & 4 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 3 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{pmatrix} \quad \Pi^{(4)} = \begin{pmatrix} \text{NIL} & 1 & 4 & 2 & 1 \\ 4 & \text{NIL} & 4 & 2 & 1 \\ 4 & 3 & \text{NIL} & 2 & 1 \\ 4 & 3 & 4 & \text{NIL} & 1 \\ 4 & 3 & 4 & 5 & \text{NIL} \end{pmatrix}$$

$$D^{(5)} = \begin{pmatrix} 0 & 1 & -3 & 2 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 3 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{pmatrix} \quad \Pi^{(5)} = \begin{pmatrix} \text{NIL} & 3 & 4 & 5 & 1 \\ 4 & \text{NIL} & 4 & 2 & 1 \\ 4 & 3 & \text{NIL} & 2 & 1 \\ 4 & 3 & 4 & \text{NIL} & 1 \\ 4 & 3 & 4 & 5 & \text{NIL} \end{pmatrix}$$

Figura 4.3: Sequenza delle matrici $D^{(k)}$ e $\Pi^{(k)}$ calcolate con l'algoritmo Floyd-Warshall per il grafo di figura 4.1

Capitolo 5

I motori di ricerca e la loro classificazione

Il fatto di avere una grande disponibilità di documenti on-line, al giorno d'oggi, pone in risalto il bisogno di avere degli algoritmi intelligenti che, come risposta ad una ricerca, ci restituiscano un elenco preciso di quelle che potrebbero essere le nostre necessità. Questo si può fare soltanto se si hanno informazioni veritiere e accurate della significatività delle pagine dei documenti disponibili nel Web. L'operazione di richiesta del documento è resa possibile dai motori di ricerca, ovvero sistemi on-line che effettuano una raccolta e catalogano i documenti in base a criteri che andremo a studiare più approfonditamente in questo capitolo. Ovviamente, non avrebbe senso accontentarsi che ci venga restituita dopo una richiesta una lista casuale di documenti (si pensi praticamente all'azione in sè che chiunque di noi ha compiuto almeno una volta), ma si è interessati ad avere una lista ″intelligente″ e che ″capisca″ quali possono essere con maggiore probabilità i file di maggior interesse per chi sta effettuando la ricerca. L'obiettivo è quello di estrarre dall'elenco di tali documenti un sottoinsieme S che contenga relativamente pochi documenti (un centinaio al massimo), i cui documenti siano rilevanti per l'utente che ha formulato l'interrogazione e che provengano da fonti autorevoli.

Quello che ci si aspetta è che nella lista di documenti restituita in testa ci siano quelli con maggiore significatività. Ad un motore di ricerca sono perciò affidate sia l'operazione di ricerca, sia la classificazione delle pagine in base alla loro significatività o rango (rank). La lista sarà restituita in ordine di rango decrescente. Dunque, data una collezione di documenti D, si definisce una funzione $rango$: D$\longrightarrow \Re$ tale che , per ogni coppia di documenti d_1 e d_2 in D, consideriamo d_1 più rilevante di d_2 se $rango(d_1)$>$rango(d_2)$.

Inizialmente la funzione di rango veniva calcolata considerando quanto ″interessante″ fosse il contenuto della pagina scelta. Tuttavia, con la nascita delle tecnologie ipertestuali, in cui i documenti posso essere riferiti tra

di loro (si pensi alle pagine Web collegate), questi collegamenti tra le pagine vengono presi molto in considerazione per il calcolo del rango. In altre parole, la funzione di rango usa, oltre al contenuto dei documenti, anche la struttura a grafo orientato dell'intera collezione (in cui i vertici sono i documenti e gli archi orientati sono i collegamenti tra loro). Intuitivamente quanto più un documento è riferito da altri documenti, tanto più esso è significativo all'interno della collezione D: infatti, tali riferimenti sono ritenuti essere considerati come l'espressione di una forma di giudizio umano. In un certo senso, la significatività di un documento viene determinata in modo democratico, attraverso un meccanismo di 'votazione' in cui l'introduzione di un riferimento dal documento d_1 al documento d_2 assume la valenza di un voto espresso da d_1 a favore di d_2. In quest'ottica una pagina è tanto più significativa quanto maggiore è il grado entrante del vertice corrispondente. Nel caso in cui il rango di un documento sia calcolato a partire solo dal suo contenuto, quest'ultimo può essere falsificato dal suo produttore al fine di aumentare in modo indebito il rango del documento: tale attività è nota, nel caso del Web, come *search spam* e consiste, ad esempio, nel modificare elementi di una pagina quali il titolo, le parole chiave e i testi associati ai riferimenti (snippet). Per fare un esempio, si pensi anche alla ricerca sul canale Youtube di un video musicale. Talvolta la creazione da parte di altri utenti di video che nel titolo contengano parole chiave come ″official″, fuorvia totalmente la ricerca restituendo risultati non interessanti per chi l'ha effettuata. Invece, una funzione di rango basata anche sui riferimenti tra i documenti risulta di difficile manipolazione da parte di un singolo produttore, e rende quindi il rango stesso più resistente a questo tipo di search spam. In questo processo di votazione (basato esclusivamente sul grado entrante dei vertici), si deve tenere presente anche un altro aspetto, cioè che un riferimento proveniente da un documento che ne ha molti in uscita deve avere una rilevanza minore rispetto a quello di un documento che ne ha pochi, perché si suppone che nel secondo sia stata effettuata una scelta più accurata e meticolosa dei documenti da collegare. Inoltre, dobbiamo tenere conto anche del rango dei documenti, in quanto un riferimento proveniente da un documento ″autorevole″ è più influente rispetto a quello proveniente da uno meno rilevante. Attualmente, i più importanti motori di ricerca del Web utilizzano una propria funzione di rango basata sull'analisi dei collegamenti (link analysis) e, in generale, impiegano specifici programmi, detti crawler o spider, che visitano e raccolgono le pragine del web seguendo i link che le collegano, analizzano il testo in ogni pagina racconto e costruiscono un indice di ricerca dei termini che compaiono nelle pagine stesse, tengono traccia dei link tra le pagine, costruendo quindi la matrice di adiacenza del grafo del Web (o meglio, della sua porzione visitata) e infine calcolano, a partire dalla matrice di adiacenza, il rango delle pagine raccolte(o di un loro sottoinsieme) secondo modalità che possono variare da motore a motore.

Verranno ora presi in considerazione due approcci che stanno alla base

dei metodi odierni per il calcolo del rango: si consideri che questi non sono tra loro alternativi, ma piuttosto complementari. Il primo approccio, detto *PageRank* e utilizzato da Google insieme ad altri metodi, calcola il rango di tutte le pagine raccolte: le pagine restituite in seguito a un'interrogazione vengono poi mostrate all'utente in ordine decrescente di rango. Il secondo approccio, detto $HITS$ (Hypertext Induced Topic Selection o selezione degli argomenti indotta dagli ipertesti) e utilizzato inizialmente da Clever dell'IBM, ha in parte ispirato successivi motori di ricerca come Ask,il quale usa ExpertRank: l'approccio $HITS$ identifica prima un opportuno insieme D di pagine che, in qualche modo, solo collegate all'interrogazione e di queste solamente ne calcola il rango che viene utilizzato per decidere l'ordine con cui mostrare le pagine in D. Osserviamo che la collezione D (dominio della funzione rango) è costituita da tutte le pagine raccolte dai crawler nel caso di *PageRank*, mentre tale collezione è formata dalle sole pagine collegate all'interrogazione nel caso dell'$HITS$: in quest'ultimo caso, la pagina può avere valori di rango differenti a seconda dell'interrogazione richiesta ovvero una solo parola diversa nella richiesta può individuare un insieme D di documenti differente e quindi una risposta difforme. Si può dedurre di conseguenza che i due metodi possono essere combinati utilizzando $HITS$ per affinare i risultati di *PageRank*.

5.1 Significatività delle pagine con PageRank

Il punto fondamentale è che una funzione di rango basata sul sistema di votazione e da applicare al grafo de Web rispetta due principi fondamentali: il rango di una pagina è direttamente proporzionale al rango delle pagine che la riferiscono e inversamente proporzionale al grado uscente di tali pagine.

Il sistema di votazione prende in considerazione tutte le pagine con almeno una preferenza, e per questo motivo l'astensione dal voto viene considerata come se si desse una preferenza uniforme a tutte le pagine del Web. In termini di matrice delle adiacenze del Web, ciò vuole dire che le righe con tutti 0 (corrispondenti alle pagine senza link di uscita) diventano righe con tutti 1. A queste osservazioni dobbiamo aggiungere quella che il tipico navigatore (surfer) non sempre si muove all'interno del Web seguendo un link, ma può capitare che si utilizzi la barra dei comandi di un browser per spostarsi direttamente su una pagina non necessariamente collegata a quella attuale (si pensi al comando home o alla frecce ″pagina successiva″ o ″pagina precedente″): si supponga che questo avvenga con probabilità $1-\alpha$ con $0 < \alpha < 1$. Per ogni pagina i, indichiamo nel seguito con $E(i)$ l'insieme delle pagine che riferiscono i e con $uscita(i)$ il grado in uscita di i. Il *PageRank* modella le suddette affermazioni definendo matematicamente la seguente funzione rango per ogni parola j:

$$rango(j) = (1-\alpha)+\alpha \cdot \sum_{i \in E(j)} \frac{rango(i)}{uscita(i)}.$$

Osservando tale formula si può dedurre che con probabilità α la pagina j viene raggiunta seguendo un link da una delle pagine che la riferiscono, mentre con probabilità 1-α essa viene raggiunta digitando direttamente il suo indirizzo nella barra dei comandi. Sia A la matrice di adiacenza del grafo del Web, ovvero $A[i][j]$=1 se e solo se esiste un link dalla pagina i alla pagina j per $0 \leq i$, $j \leq n-1$, dove n indica il numero di pagine (si noti che $\forall\ i$ deve esistere almeno un valore di j tale che $A[i][j] = 1$, ovvero una pagina deve essere collegata ad almeno un'altra del Web): quindi, $i \in E(j)$ se e solo se $A[i][j]$=1. Il calcolo della funzione rango secondo l'equazione è esplicitato nel codice dell'algoritmo PageRank, che per prima cosa calcola il grado in uscita di ogni vertice e inizializza un array uni-dimensionale di n elementi, detto vettore rango, per il quale al termine dell'esecuzione vale $X[j]$=$rango(j)$ per $0 \leq j \leq n-1$. Il ciclo successivo calcola il rango delle pagine usando l'equazione sopra: poiché tale equazione è ricorsiva, non è sufficiente una sua sola applicazione, ma il calcolo deve essere effettuato in maniera iterativa, in modo che i valori calcolati a un passo divengano i valori in ingresso del passo successivo, fino a raggiungere il risultato finale. In particolare, il quarto ciclo for esegue un passo di tale calcolo iterativo: notiamo che, poichè $A[i][j]$=1 se $i \in E(j)$ e $A[i][j]$=0 altrimenti, il ciclo for più interno corrisponde al calcolo del valore $\sum_{i \in E(j)} \frac{rango(i)}{uscita(i)}$ presente nell'equazione. Quando il ciclo do-while termina, il valore di X che viene restituito soddisfa l'equazione: pertanto, se il codice termina calcola il valore di $PageRank$.

Algoritmo `PageRank`(A)
input: A `matrice delle adiacenze di un grafo` G `con` n `vertici`
output: X`, il valore di PageRank`
for $j \leftarrow$ `0` **to** n
  $uscita[j]$ = `0 ;`
for $i \leftarrow 0$ **to** n **do**
  $uscita[j] \leftarrow uscita[j]$ `+` $A[j][i]$
  $X[j]$= `1;`
do while$(X \neq Y)$`{`
    for $j \leftarrow$`0` **to** n
      $Y[j] = X[j]$`;`
    for $j \leftarrow$`0` **to** n `{`
      $X[j]$ = `0;`
      for $i \leftarrow$`0` **to** n
      $X[j]$=$X[j] + A[i][j] \cdot Y[i]$ `//uscita[i];`
      $X[j]$=$(1-\alpha) + \alpha \cdot X[j]$`; } }`
return X`;`

Per quanto riguarda la complessità del codice, ogni iterazione del ciclo do-while richiede tempo $O(n^2)$: quindi, la complessità temporale è $O(bn^2)$

dove b indica il numero di iterazioni che sono eseguite. In linea di principio, b potrebbe essere un valore infinito. In quanto i valori X e Y calcolati a ogni iterazione potrebbero oscillare senza mai convergere allo stesso valore: inoltre, la convergenza potrebbe dipendere dai valori con cui si analizza X. La convergenza ad un valore finito verrà dimostrata nel paragrafo 5.3. Una naturale interpretazione del codice consiste nell'immaginare una pagina fornita inizialmente di un ″credito″ di significatività di default (pari a 1 nel codice): successivamente, ogni pagina decide di distribuire in parti uguali il suo credito alle pagine da essa riferite, ricevendo in cambio un contributo da ciascuna pagina che la riferisce. Anche se da un lato aprire all'esterno una

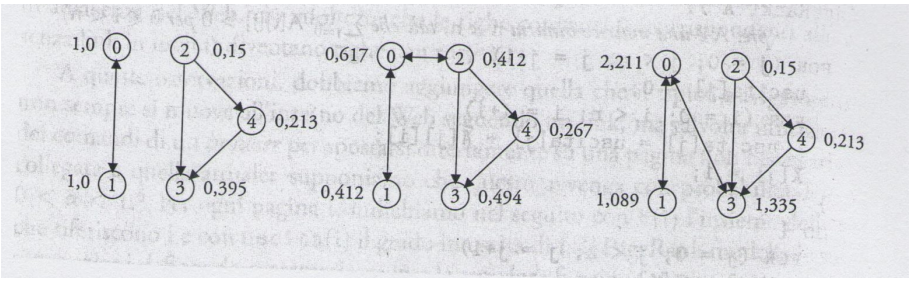


Figura 5.1: Apertura di PageRank

pagina, includendo in essa dei link in uscita, causa una perdita di rango, dall'altro una tale apertura può incrementare il numero di riferimenti alla pagina stessa e, quindi, aumentare il flusso di rango in entrata che, in linea di principio, potrebbe compensare e superare quello in uscita. Consideriamo l'esempio mostrato nella parte sinistra della Figura 5.1, in cui il sito Web formato dalle due pagine 0 e 1 è isolato rispetto al resto del mondo: notiamo che in questa situazione il rango totale del sito formato dalle due pagine viene equamente distribuito tra di esse, in quanto le due pagine si riferiscono vicendevolmente. Volendo aprire tale sito all'esterno, possiamo decidere di farlo in diversi modi. Ad esempio possiamo decidere di collegare la pagina 0 alla pagina 2 e viceversa: in tal caso, il sito globalmente perde rango, passando da 2 a poco più di 1. Se, però, decidiamo di collegare la pagina 0 alla pagina 3 e viceversa, come mostrato nella parte destra della Figura 5.1, la situazione cambia drasticamente: a pagina 0 arriva a un rango superiore a 2 e anche la 1 aumenta leggermente il rango, così che il rango totale passa da 2 a 3,3. In altre parole, gestendo l'apertura verso l'esterno in modo opportuno, il rango delle proprie pagine può migliorare significativamente. Un'altra interessante osservazione consiste nel fatto che aprire un sito verso l'esterno può essere accompagnato da una ristrutturazione del sito stesso, in modo da diminuire la quantità di rango che viene perso a causa dell'apertura. Consideriamo la situazione mostrata nella parte sinistra della Figura 5.2, in cui la pagina potrebbe essere una tipica pagina contenete un

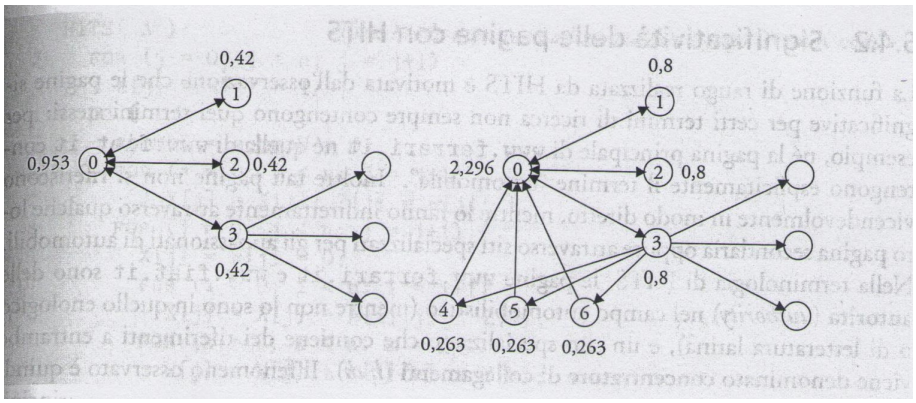


Figura 5.2: Effetti della ristrutturazione di un sito Web sul valore di PageRank

elenco di riferimenti a pagine esterne al sito formato dalle pagine 0,1,2 e 3. Come possiamo vedere, il rango iniziale del sito si riduce significativamente a causa della pagina 3: il rango totale, dopo l'apertura verso l'esterno, è divenuto pari a circa 2,213 con una perdita del 45%. In questo caso, possiamo progettare una ristrutturazione del sito aggiungendo delle pagine di recensione delle pagine esterne a cui la pagina 3 fa riferimento e che a loro volta fanno riferimento alla pagina 0 (che funge da home page del sito), come mostrato nella parte destra della figura: così facendo il rango che fuoriesce dal sito si riduce significativamente e il rango totale passa da un potenziale di 7 a un valore attuale pari a circa 5,485, con una perdita del 22%.
Risulta evidente da questi esempi che imparare a gestire in modo eticamente corretto ed intelligente le pagine Web è di enorme rilevanza in ambito informatico. Nel mondo della pubblicità, ad esempio, la visibilità , cioè avere il rango della pagina alto, è di importanza fondamentale. A causa dei rischi di falsificazione derivati da alcune strategie di manipolazione del valore di PageRank, il concetto di rango di una pagina Web è diventato sempre meno affidabile. Un altro svantaggio del PageRank è che esso tende a favorire pagine più vecchie: intuitivamente, una pagina nuova, anche se molto interessante, non avrà all'inizio molti riferimenti in entrata, a meno che non sia parte di un sito già esistente e con un insieme di pagine fortemente connesse tra di loro.

5.2 Significatività delle pagine con HITS

La funzione di rango realizzata da *HITS* è motivata dall'osservazione che le pagine significative per certi termini di ricerca non sempre contengono quei termini stessi: per esempio, né la pagina principale di www.ferrari.it né quel-

la di www.fiat.it contengono esplicitamente il termine ″automobile″. Inoltre tali pagine non si riferiscono vicendevolmente in modo diretto, mentre lo fanno indirettamente attraverso qualche loro pagine secondaria. Nella terminologia di $HITS$, le pagine www.ferrari.it e www.fiat.it sono delle autorità (authority) nel campo automobilistico, e un sito specializzato che contiene dei riferimenti a entrambe viene denominato concentratore di collegamenti (hub). Il fenomeno osservato è quindi quello del mutuo rafforzamento in termini di significatività, secondo i seguenti principi:

- un concentratore è tanto più significativo quanto lo sono le autorità a cui si riferisce e quindi il peso del primo è direttamente proporzionale al peso delle ultime;
- un'autorità è tanto più significativa quanto lo sono i concentratori che la riferiscono e quindi il peso della prima è direttamente proporzionale al peso degli ultimi.

HITS classifica le pagine in base ai principi sopra esposti: ogni pagina è simultaneamente un'autorità e un concentratore, ovviamente con peso molto variabile. A tal fine, le seguenti due funzioni di rango sono utilizzate per una data collezione D di documenti:

$rangoA(j)$ misura il peso in D della pagina j intesa come autorità;

$rangoC(j)$ misura il peso in D della pagine j intesa come concentratore.

Da quanto osservato sopra, possiamo derivare le equazioni ricorsive che definiscono le suddette funzioni per la pagina j nella collezione D, indicando con $E(j) \subset D$ l'insieme delle pagine che riferiscono j e con $U(j) \subset D$ quelle riferite da j, all'interno della collezione stessa:

$rangoA(j) = \sum_{i \in E(j)} rangoC(i)$

$rangoC(j) = \sum_{k \in U(j)} rangoA(k)$

Analogamente al $PageRank$ possiamo effettuare il calcolo di tali funzioni mediante un procedimento iterativo, assegnando un valore iniziale pari a $1/n$ a ciascuna pagina j e computando di volta in volta il valore di rango, per mantenere l'invariante che $\sum_{j \in D} rangoA(j) = \sum_{j \in D} rangoC(j) = 1$.

Algoritmo `Hits`(A)
input: A `matrice delle adiacenze di un grafo` G `con` n `vertici`
output: `valori del rango definiti per` $HITS$
for $j \leftarrow$ `0` **to** n
  $X[j] = W[j] = 1$;

```
do while(X≠Y) ∨ (W[j]≠Z) {
  sommaX=sommaY=0; //sommaX e sommaY sono le somme dei valori del rango
    for j←0 to n
      Y[j] = X[j]; Z[j]=W[j];
    for j←0 to n {
      X[j] = W[j] = 0;
      for i←0 to n
      X[j]=X[j] + A[i][j] * Z[i] ;
      for k←0 to n
      W[j]=W[j] + A[j][k] * Y[i] ;
      sommaX = sommaX + X[j];
      sommaW = sommaW + W[j]; }
    for j←0 to n
      X[j] = X[j] //sommaX; W[j]=W[j] //sommaW; }
return < X, W >;
```

Per semplificare la notazione, il codice suppone che il sottoinsieme delle pagine del grafo del Web che compongono la collezione D siano state nuovamente numerate da 0 a $n-1$ e che A sia la matrice di adiacenza del sottografo del Web indotto dalle pagine in D: quindi, $i \in E(j)$ se e solo se $A[i][j]=1$ e $k \in U(j)$ se e solo se $A[j][k]=1$. Il codice inizializza due vettori di rango di n elementi tali che al termine dell'esecuzione vale $X[j]=rangoA(j)$ e $W[j]=rangoC(j)$ per $0 \leq j \leq n-1$. Il ciclo successivo calcola le due funzioni di rango usando le equazioni precedenti: poiché tali equazioni sono ricorsive, i valore calcolati a un passo diventano i valori in ingresso del passo successivo fino a raggiungere il risultato finale. Le righe del ciclo do while che calcolano i valori $sommaX$ e $sommaW$ servono a normalizzare i valori così calcolati: quando il ciclo do-while termina(ovvero i nuovi valori coincidono con quelli calcolati precedentemente) i valori di X e W che vengono restituiti dall'algoritmo soddisfano necessariamente l'equazioni precedenti. Pertanto se il codice termina calcola esattamente le funzioni di rango definite per $HITS$. Analogamente a $PageRank$, ogni iterazione del ciclo do while richiede tempo $O(n^2)$: quindi, la complessità temporale è $O(bn^2)$ dove b indica il numero di iterazioni che sono eseguite. Anche in questo caso, la convergenza ad un valore finito di b verrà provata nella Sezione 5.3. A questo punto, è interessante discutere come viene costruita la collezione D di documenti su cui applicare $HITS$, mostrando anche come esso può integrarsi con altri sistemi di rango. Ipotizziamo di eseguire un'interrogazione utilizzando un motore di ricerca con la propria funzione di rango ottenendo così un elenco di risultati: prendiamo i primi t in ordine di rango formando un insieme S di partenza (t=200 nella proposta originale di $HITS$). Per ottenere D, estendiamo S con le pagine che appartengono al vicinato di quelle in S: $HITS$ aggiunge a S tutte le pagine in $U(j)$ per $j \in S$ e un opportuno sottoinsieme delle pagine in $E(j)$ per $j \in S$. Il metodo $HITS$, come quello $PageRank$,

presenta alcuni svantaggi che in qualche modo limitano la sua affidabiltà come implementazione del concetto di rango di una pagine Web. Anzitutto, poiché il valore di $HITS$ dipende dall'interrogazione, la costruzione dell'insieme D e il calcolo descritto nel codice di prima devono essere eseguiti ogni volta al momento dell'interrogazione (con ovvie conseguenze dal punto di vista delle prestazioni). Inoltre, $HITS$ è soggetto a meccanismi di search spam al pari di $PageRank$. Dal punto di vista del produttore di una pagina Web, è infatti facile aggiungere link in uscita e quindi aumentare in tal modo il punteggio di concentratore della pagina stessa: poiché i due punteggi di HITS sono tra loro indipendenti, questo può portare a un aumento del ponteggio di autorità della pagina. Inoltre, cambiamenti locali alla struttura dei collegamenti possono risultare in punteggi drasticamente diversi, in quanto la collezione D è piccola in confronto all'intero Web. Infine, $HITS$ presenta un cosiddetto problema di ″deriva del soggetto″ (topic drift), in quanto è possibile che costruendo l'insieme D includiamo una pagina non precisamente focalizzata sull'argomento dell'interrogazione ma con un alto punteggio di autorità: il rischio è che questa pagina e quelle a essa vicine possano dominare la lista ordinata che viene restituita all'utente, spostando così l'attenzione su documenti non corrispondeti alla richiesta eseguita.

5.3 Convergenza del calcolo iterativo di PageRank

Per dimostrare la convergenza del valore di b (cioè il numero di iterazioni del ciclo) della complessità degli algoritmo che calcolano il valore di PageRank e $HITS$ si fa uso di un famoso risultato di algebra lineare, detto teorema di Perron-Frobenius(si veda [8]). Ciò è dovuto al fatto che, come mostriamo in questo paragrafo, possiamo interpretare il codice come un procedimento iterativo per il calcolo della soluzione di un sistema di equazioni lineari: il teorema di Perron-Frobenius ci consente di dimostrare che tale soluzione esiste ed è unica e che viene calcolata da tale procedimento iterativo. Indichiamo con $X^{(k)}$ il valore del vettore di rango X al termine della k-esima iterazione del ciclo do-while del codice per $k \geq 1$, e con $X(0)$ il suo valore iniziale (ovvero, formato da tutti 1). In base alle istruzioni eseguite dal codice all'interno del ciclo, possiamo riformulare l'equazione come

$$X^{(k)}[j]=(1\text{-}\alpha)+\alpha \cdot \sum_{i=0}^{n-1} \frac{A[i][j]}{uscita[i]} \cdot X^{(k-1)}[i] \qquad (5.3.1)$$

per $0 \leq j < n$ e $k \geq 1$. Osserviamo che per ogni k$\geq$0, $\sum_{j=0}^{n-1} X^{(k)}=n$. Ciò è vero per k=0. Supponendo che lo sia per ogni $h<k$ e applicando l'equazione (5.3.1) abbiamo che

$$\begin{aligned}\sum_{j=0}^{n-1} X^{(k)}[j] &= (1-\alpha)n + \alpha \sum_{j=0}^{n-1}\sum_{i=0}^{n-1} (\frac{A[i][j]}{uscita[i]} X^{(k-1)}[i]) \\ &= (1-\alpha)n + \alpha \sum_{j=0}^{n-1} X^{(k-1)}[i] \sum_{i=0}^{n-1} \frac{A[i][j]}{uscita[i]} \\ &= (1-\alpha)n + \alpha \sum_{j=0}^{n-1} X^{(k-1)}[i]) \\ &= (1-\alpha)n + \alpha n = n\end{aligned}$$

Dove la terza uguaglianza è dovuta al fatto che $uscita[i]{=}\sum_{j=0}^{n-1} A[i][j]$ per ogni $0{\leq}i{<}n$ mentre la quarta uguaglianza segue dall'ipotesi induttiva che $\sum_{i=0}^{n-1} X^{(k-1)}[i]{=}n$. Mostriamo ora che l'insieme delle equazioni, una per ogni valore di j, costituisce un sistema di equazioni lineari, che può essere espresso mediante moltiplicazione di matrici. A tal fine, definiamo la matrice M di dimensione $n \times n$ tale che

$$M[j][i]{=}\ \tfrac{1-\alpha}{n}\ +\ \alpha\ \tfrac{A[i][j]}{uscita[i]}\ ;$$

per $0{\leq}j,i{<}n$. possiamo verificare che

$$X^{(k)}{=}M{\times}\ X^{(k-1)}$$

in quanto $\sum_{j=0}^{n-1} X^{(k)}[j]{=}n$ per ogni $k{\geq}\ 0$. Quindi, $X^{(k)}{=}M^k X^{(0)}$: pertanto, il codice comnverge se esiste un k tale che $M^k{=}M^{k-1}$. Il teorema di Perron-Frobenius ci permette di affermare che ciò è asintoticamente vero i quanto gli elementi di M sono tutti positivi: per ogni $\epsilon{>}0$ esiste k $\epsilon{\geq}1$ tale che $\left|M^k[j][i] - M^{(k-1)}[j][i]\right|{<}\epsilon$ per $0{\leq}i,\ j{<}n$. Per questo motivo, la condizione di terminazione del codice deve essere espressa in funzione di un grado di precisione epsilon che deve essere fornito in ingresso insieme alla matrice A. Inoltre, lo stesso teorema garantisce che la soluzione calcolata non dipende dalla scelta di $X^{(0)}$. Il teorema di Perron-Frobenius potrebbe essere applicato anche al codice del calcolo di $HITS$, considerando comunque che la soluzione calcolata può dipendere dai valori assegnati inizialmente a X e W. Le informazioni relative a questo capitolo sono state ricavate da [5] e [8], come anche le immagini.

Capitolo 6

Approfondimento: Four degrees of separation

Frigyes Karithy, nel suo saggio ″Lamcszemek″ del 1929 suggerì che due persone qualsiasi sono distanziate da al massimo sei link di amicizia nel mondo. Per quanto riguarda lo studio delle reti sociali, il sociologo Milgram pose le basi con il suo esperimento degli anni '60. Egli cercò di valutare la distanza media tra una coppia qualunque di persone su un grafo mediante un processo di invio e ricezione di messaggi. Nell'esperimento, una lettera doveva viaggiare da un mittente in Nebraska a un destinatario nel Massachussets e come richiesta specifica, doveva essere inviata ad un conoscente con l'obiettivo di avvicinare ad ogni passo il più possibile il messaggio al destinatario (nel caso in cui il destinatario ovviamente non fosse tra le persone conosciute): il conoscente considerato, e tutti i possibili intermediari, dovevano operare secondo la stessa specifica richiesta, fino alla consegna della lettere ad destinatario finale. Milgram scoprì che il medio numero di intermediari nel cammino delle lettere sta tra 4.4 e 5.7, in relazione al gruppo di persone considerato. Addentrandoci nell'argomento di nostro interesse, per quanto riguarda la rete sociale Facebook(729 milioni di user, 69 miliardi di collegamenti di amicizia)la distanza media che è stata osservata è di 4.74, corrispondendo a 3.74 intermediari. Più generalmente, è stata studiato la distribuzione di distanza di Facebook e di qualche sottografo geografico di interesse, osservando anche la loro evoluzione nel tempo. Uno degli obiettivi nello studio della distribuzione di distanza è l'identificazione di interessanti parametri statistici che possono essere usati per parlare propriamente di social network a partire da altre reti complesse come un grafo Web. Più generalmente, la distribuzione di distanza è uno degli aspetti globali che permette di rifiutare modelli probabilistici permettendo di fornire definizioni non ipotetiche. Si consideri il parametro denominato spid(shortest-paths index of dispersion) che, gestendo la distanza tra due vertici qualsiasi come una variabile aleatoria, ne misura il rapporto σ^2/ν (Si veda [7]). Quest'ulti-

mo misura la dispersione della distribuzione di distanza, generalmente essere minore di 1 (sotto-dispersione) per le reti sociali, ma più grande di 1 (sovra-dispersione) per il grafi Web. Dunque, una delle domande interessanti può essere: qual è lo spid di Facebook? I dati ottenuti sono i seguenti: la distanza di Facebook è 4.74 mentre lo spid del grafo è solo 0.09, in accordo con la congettura precedente che le reti sociali hanno, in effetti, lo spid minore di 1. Per realizzare l'analisi della struttura del grafo di Facebook, sono state applicate alcune tecniche di compressione che hanno dimostrato un incremento della località. Queste tecniche verranno presentate successivamente. Riprendiamo per un momento l'esperimento descritto precedentemente del professor Milgram. In poche parole, i risultati ottenuti dall'esperimento di Milgram erano i seguenti: solo il 22% delle catene è stato completato; il numero medio di intermediari in queste catene era di 5.2, con una grande differenza rispetto dal gruppo di Boston (4.4); dall'altro lato, la popolazione del Nebraska richiedeva 5.7 intermediari in media. Le conclusioni finali erano che la lunghezza del cammino medio è molto più piccola di quello che ci si aspettava, e che la zona geografica influisce molto. Si noti che Milgram stava misurando la lunghezza media di un cammino instradato su una rete sociale, che è in realtà solo un limite superiore sulla distanza media (infatti le persone dell'esperimento non dovevano necessariamente inviare le lettere a conoscenti lungo un cammino minimo). In un certo senso, i risultati ottenuti sono molto più interessanti di quello che sembrano, poiché non solo provano che il mondo è piccolo, ma che chi ci vive dentro può fare vedere la sua piccolezza. Una differenza tra l'esperimento mostrato in questo scritto e quello di Milgram è che la nozione di amicizia di Facebook è difficilmente comparabile all'idea di amicizia nella vita; in particolare, non ci si può aspettare, in generale, che tutti i contatti di Facebook si conoscano personalmente (come invece richiedeva Milgram). Questo fatto può artificialmente ridurre le lunghezze dei percorsi, ma anche il contrario è vero: visto che ci saranno molti conoscenti di primo grado che non sono su Facebook, allora qualche cammino minimo non sarà scrutabile. Questi due fenomeni si equiparano vicendevolmente. Ciònondimeno, una comparazione più stretta tra questo esperimento e quello di Milgram è molto difficile. Ci si limiti alla parte dell'esperimento di Milgram che riguarda la stima della distribuzione di distanza. Il più grande esperimento simile a quello presentato qui è quello in cui gli studiosi hanno considerato un grafo di comunicazione con 180 milioni di vertici e 1.3 miliardi di archi estratti dalla rete di Micrsoft Messenger; loro scoprirono che la distanza media e di 6.6 (5.6 intermediari, cioè circa 6 gradi di separazione). Si noti, comunque, che un grafo di comunicazione ha un arco tra due persone solo se loro hanno comunicato durante un periodo di osservazione specifico di un mese, e quindi non inseriscono nei link di amicizia quelli che non hanno comunicato tra di loro.

6.1 Definizioni e Strumenti

La *funzione di vicinanza* $N_G(t)$ di un grafo G restituisce per ogni $t \in N$ il numero di coppie di vertici (x, y) tali che y sia raggiungibile da x in al più t passi. Fornisce dati su quanto velocemente la sfera attorno ad ogni vertice si espande. Dalla funzione di vicinanza è possibile derivare la distanza di distribuzione, che dà per ogni t l'insieme delle coppie raggiungibili alla distanza esatta di t. Sebbene non verrà descritto in modo meticoloso (non essendo argomento di questa tesina), è bene dichiarare che si utilizzerà $HyperANF$ (dove ANF sta per approximate neighbourhood function, si veda [7]), un algoritmo in grado di approssimare velocemente la funzione di vicinanza di grafi molto grandi.

6.2 Esperimenti

I grafi qui analizzati sono grafi di utenti di Facebook che erano attivi nel Maggio 2011; un utente attivo è un utente che si è connesso entro gli ultimi 28 giorni. La decisione di restringere gli studi a utenti attivi ci permette di eliminare account che sono stati abbandonati. Non sono considerate inoltre le pagine (per esempio di persone famose) alle quali le persone possono mettere ″mi piace″ e si ricorda che uno user non può avere più di 5000 amicizie. Si è deciso di estendere gli esperimenti in due direzioni: temporali e regionali. Cioè si analizzano l'intero grafo Facebook (fb), il sottografo USA (us), quello italiano (it) e svedese (se). Si analizza inoltre l'insieme dei sottografi italiano e svedese (itse) per controllare se combinare due reti regionali ma distanti può significativamente cambiare la distanza media, come aveva fatto anche Milgram nel suo esperimento. Per ogni grafo noi calcoliamo la distribuzione di distanza da 2007 ad oggi servendosi di molteplici esecuzioni di $HyperANF$.

6.2.1 Setup

Le computazioni sono state eseguite su una macchina con un 24-core con 72 GB di memoria e un TB di spazio su disco. Il primo obiettivo era quello di comprimere il grafo Facebook in modo da ottenere performance più interessanti e in modo da risparmiare spazio su disco. Un grafo è ben compresso quando esibisce somiglianza (vertici con indici vicini hanno liste simili di successori) e località (liste di successori hanno piccoli campi vuoti). Poiché il nostro tempo di calcolo è grandemente ridotto dalla compressione, siamo interessati a stabilire quando può essere indotta maggior località in un grafo di questa grandezza permutando in modo appropriato il grafo usando il metodo di compressione Layered Labelled Propagation (LLP, si veda [6]). Questo metodo prevede il raggruppamento a più livelli e combinando i cluster ottenuti al fine di ordinare i vertici del grafo si ricava un aumento della

località e della somiglianza. È interessante sapere che applicare LLP ai grafi di questo esempio ha richiesto 10 giorni di calcolo al nostro hardware.

6.2.2 Esecuzione

Le esecuzioni di *HyperANF* nel corrente grafo di Facebook usava 32 registri, e quindi lo spazio per i raggruppamenti era circa di 27GB. Come velocità approssimativa, una singola esecuzione di LLP sul grafo corrente di Facebook ha richiesto circa 13.5 ore. Le Tabelle 6.1 e 6.2 mostrano rispettivamente il numero di vertici e di link di amicizia del dataset e la distanza media. Si può notare come all'aumentare degli anni i link di amicizia sono diventanti di numero maggiore considerando il numero di utenti. Allo stesso modo si può notare(in Figura 6.5) che la media distanza si è piano piano avvicinata al valore finale di 4.74. Durante la rapidissima crescita di Facebook negli anni, i nostri grafi mostrano un veloce decremento della distanza media, che comunque appare ora stabilizzarsi. Comunque, la densità del grafo continua a diminuire. Vediamo cioè il cosiddetto fenomeno di piccolo mondo all'opera: una più piccola frazione di archi che connette gli utenti, ma nondimeno una distanza media minore.

	it	se	itse	us	fb
Original	14.8 (83%)	14.0 (86%)	15.0 (82%)	17.2 (82%)	20.1 (86%)
LLP	10.3 (58%)	10.2 (63%)	10.3 (56%)	11.6 (56%)	12.3 (53%)

Figura 6.1: Numero di bit per link e rapporto di compressione per i grafi correnti

	it	se	itse	us	fb
2007	1.31	3.90	1.50	119.61	99.50
2008	5.88	46.09	36.00	106.05	76.15
2009	50.82	69.60	55.91	111.78	88.68
2010	122.92	100.85	118.54	128.95	113.00
2011	198.20	140.55	187.48	188.30	169.03
current	226.03	154.54	213.30	213.76	190.44

Figura 6.2: Grado medio dei dataset

Lower bounds from HyperANF runs

	it	se	itse	us	fb
2007	41	17	41	13	14
2008	28	17	24	17	16
2009	21	16	17	16	15
2010	18	19	19	19	15
2011	17	20	17	18	35
current	19	19	19	20	58

Exact diameter of the giant component

current	25	23	27	30	41

Figura 6.3: Limite inferiore del diametro di ciascun grafo ed esatti valori per il maggior componente connesso(>99.7%) dei grafi correnti

	it	se	itse	us	fb
2007	159.8 K (105.0 K)	11.2 K (21.8 K)	172.1 K (128.8 K)	8.8 M (529.3 M)	13.0 M (644.6 M)
2008	335.8 K (987.9 K)	1.0 M (23.2 M)	1.4 M (24.3 M)	20.1 M (1.1 G)	56.0 M (2.1 G)
2009	4.6 M (116.0 M)	1.6 M (55.5 M)	6.2 M (172.1 M)	41.5 M (2.3 G)	139.1 M (6.2 G)
2010	11.8 M (726.9 M)	3.0 M (149.9 M)	14.8 M (878.4 M)	92.4 M (6.0 G)	332.3 M (18.8 G)
2011	17.1 M (1.7 G)	4.0 M (278.2 M)	21.1 M (2.0 G)	131.4 M (12.4 G)	562.4 M (47.5 G)
current	19.8 M (2.2 G)	4.3 M (335.7 M)	24.1 M (2.6 G)	149.1 M (15.9 G)	721.1 M (68.7 G)

Figura 6.4: Numero di vertici e link di amicizia dei dataset

	it	se	itse	us	fb
2007	10.25 (±0.17)	5.95 (±0.07)	8.66 (±0.14)	4.32 (±0.02)	4.46 (±0.04)
2008	6.45 (±0.03)	4.37 (±0.03)	4.85 (±0.05)	4.75 (±0.02)	5.28 (±0.03)
2009	4.60 (±0.02)	4.11 (±0.01)	4.94 (±0.02)	4.73 (±0.02)	5.26 (±0.03)
2010	4.10 (±0.02)	4.08 (±0.02)	4.43 (±0.03)	4.64 (±0.02)	5.06 (±0.01)
2011	3.88 (±0.01)	3.91 (±0.01)	4.17 (±0.02)	4.37 (±0.01)	4.81 (±0.04)
current	3.89 (±0.02)	3.90 (±0.04)	4.16 (±0.01)	4.32 (±0.01)	4.74 (±0.02)

Figura 6.5: Distanza media

	it	se	itse	us	fb
2007	32.46 (±1.49)	3.90 (±0.12)	16.62 (±0.87)	0.52 (±0.01)	0.65 (±0.02)
2008	3.78 (±0.18)	0.69 (±0.04)	1.74 (±0.15)	0.82 (±0.02)	0.86 (±0.03)
2009	0.64 (±0.04)	0.56 (±0.02)	0.84 (±0.02)	0.62 (±0.02)	0.69 (±0.05)
2010	0.40 (±0.01)	0.50 (±0.02)	0.64 (±0.03)	0.53 (±0.02)	0.52 (±0.01)
2011	0.38 (±0.03)	0.50 (±0.02)	0.61 (±0.02)	0.39 (±0.01)	0.42 (±0.03)
current	0.42 (±0.03)	0.52 (±0.04)	0.57 (±0.01)	0.40 (±0.01)	0.41 (±0.01)

Figura 6.6: Variazione della distanza di distribuzione

	it	se	itse	us	fb
2007	3.17 (±0.106)	0.66 (±0.016)	1.92 (±0.078)	0.12 (±0.003)	0.15 (±0.004)
2008	0.59 (±0.026)	0.16 (±0.008)	0.36 (±0.028)	0.17 (±0.003)	0.16 (±0.005)
2009	0.14 (±0.007)	0.14 (±0.004)	0.17 (±0.004)	0.13 (±0.003)	0.13 (±0.009)
2010	0.10 (±0.003)	0.12 (±0.005)	0.14 (±0.006)	0.11 (±0.004)	0.10 (±0.002)
2011	0.10 (±0.006)	0.13 (±0.006)	0.15 (±0.004)	0.09 (±0.003)	0.09 (±0.005)
current	0.11 (±0.007)	0.13 (±0.010)	0.14 (±0.003)	0.09 (±0.003)	0.09 (±0.003)

Figura 6.7: Spid

6.2.3 Spid (Shortest-Paths Index of Dispersion)

Lo spid è l'indice di dispersione della distribuzione di distanza. Ricordiamo che reti con lo spid maggiore di 1 sono considerate di tipo ″Web″, mentre reti con spid minore di 1 sono considerati propriamente reti sociali. La intuizione dietro lo spid è che vere reti sociali favoriscono connessioni brevi, mentre nel Web lunghe connessioni sono comuni. Per rispondere alla domanda fatta all'inizio di questo scritto, lo spid di facebook è mostrato in Figura 6.7.

6.2.4 Diametro

HyperANF non può fornire un preciso risultato del diametro: comunque, una sua esecuzione indica necessariamente un lower bound per il diametro del grafo. Notiamo dalla Tabella 6.3 che il lower bound è minore nelle singole nazioni che nell'intero grafo di Facebook: cioè, ci sono persone che sono significativamente più distanti nel mondo in generale che in una singola nazione. Per confermare questa informazione, decidiamo di cercare di calcolare il vero diametro direttamente, anche se questo è un problema molto oneroso dal punto di vista computazionale: per grafi molto grandi, algoritmi basati su matrici non sono fattibili a causa della richiesta di spazio, ed eseguire un semplice algoritmo che compie n volte la BFS non è fattibile in termini di tempo. È stato quindi implementato un algoritmo ad hoc. L'idea è la seguente: si consideri un vertice x, e troviamo (tramite BFS) un vertice y il più lontano possibile da x. Si trovi dunque un vertice z il più lontano possibile da y: $d(y, z)$ è un lower bound del diametro, ed è il vero diametro

del grafo se questo è un albero. Consideriamo ora un vertice c a metà tra y e z: tale vertice è nel mezzo del grafo, così se h è l'eccentricità di c (la distanza del vertice più distante da c) ci aspettiamo che $2h$ sia un buon upper bound per il diametro. Se i nostri limiti superiore e inferiore si incontrano, l'algoritmo termina. Altrimenti, si esegue nuovamente l'algoritmo utilizzando come punto di partenza i vertici a distanza esattamente h da c. Chiaramente, se M è il massimo delle eccentricità dei vertici nel confine, $\max\{2(h-1), M\}$ è un nuovo limite superiore, e M è un nuovo limite inferiore. Quindi continuando a iterare questo procedimento si spera che i due valori di limite inferiore e superiore si incontrino. Questa implementazione utilizza una BFS multicore: la coda di vertici a distanza d è segmentata in piccoli blocchi gestiti da ogni singolo core. Alla fine di un ciclo, abbiamo calcolato la coda di vertici alla distanza $d+1$. È molto importante sottolineare che questo tipo di algoritmo viene applicato alla giant component del grafo preso in considerazione, ovvero alla più grande componente connessa, per evitare di imbattersi in casi in cui alcune coppie di vertici non siano connesse: si supponga ad esempio che due persone stringano amicizia tra di loro e con nessun altro. Questa è una delle situazioni da evitare. L'implementazione sopra presentata, comunque, era in grado di scoprire il diametro del grafo degli Usa in 20 minuti (chiaramente grazie alla compressione LLP). Il diametro di Facebook ha richiesto 10 ore di calcolo su una macchina con una RAM di 1TB(in realtà, 256GB sarebbero stati sufficienti, sempre grazie alla compressione LLP). I valori riportati nella Tabella 6.3 confermano quello che avevamo scoperto prima usando i dati approssimati forniti dalla lunghezza delle esecuzioni di $HyperANF$, e suggerisce che mentre la distribuzione ha una distanza media bassa ed è abbastanza concentrata, ci sono comunque coppie di vertici che sono molto più lontane. Si noti che nel caso del grafo di fb, il diametro della gigantesca componente è veramente più piccolo del limite fornito dalle runs di $HyperANF$.

6.3 Osservazioni finali

È stata considerata la più grande rete sociale mai creata da numerosi punti di vista. È stato analizzato il grafo completo di Facebook e altri grafi ottenuti restringendo geograficamente o temporalmente i link coinvolti usando $HyperANF$. La distanza media di Facebook è 4.74, cioè 3.74 gradi di separazione. Lo spid di Facebook è 0.09 molto minore di uno come aspettato per una rete sociale. Le reti ridotte geograficamente hanno una distanza media minore, come succedeva nell'esperimento di Milgram(e come ci aspetteremmo nella realtà). Complessivamente, questi risultati aiutano a dipingere Facebook per la rete sociale che in realtà è come ci si aspettava, cioè un grafo a piccolo-mondo. Quando Milgram pubblicò i suoi risultati, in realtà ha offerto due interpretazioni opposte di cosa ″sei gradi di separazione″ davvero

significasse. Da un lato, ha osservato che una tale distanza è considerevolmente più piccola di quella che una persona naturalmente penserebbe. Ma allo stesso tempo, Milgram notò che il risultato poteva anche essere interpretato come se le persone fossero in media sei ″mondi″ distanti, sei gruppi di conoscenze, sei strutture di distanza. Da questa più oscura prospettiva, è rassicurante vedere che le scoperte qui riportate mostrano che le persone sono in realtà solo distanti quattro mondi, e non sei: quando si considera un'altra persona del mondo, un amico di un amico conosce un amico del loro amico, in media. Le informazioni relative a questo capitolo sono state ricavate da [3],[6] e [7].

Capitolo 7

Conclusioni

La tesina vuole essere una dimostrazione di come, ricercando la soddisfazione delle richieste dell'essere umano, si possa sfruttare la teoria dedicata ai grafi per giungere alla scoperta di strumenti senza i quali oggigiorno non sarebbe possibile stare al passo con l'evoluzione della società. Lo sviluppo di algoritmi su grafi è diventato un campo estremamente sviluppato in cui è molto interessante mettersi in gioco e cercare di diventare concorrenziali ed esclusivi. Un grande riscontro di questa affermazione lo si ha quando si utilizza un qualsiasi browser, un navigatore satellitare, un social network. Moltissime persone hanno a che fare con questi strumenti ogni giorno, in quanto sono finalizzati a rendere le loro attività quotidiane più semplici da affrontare. Rendere questi ″mezzi″ prestanti ed efficaci è di assoluto interesse per l'informatica. In conclusione, è importante ricordare che l'informatica è uno strumento, di cui ci si deve servire per rendere la vita più semplice, per chi con essa ci lavora e per chiunque altro.

Bibliografia

[1] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein. *Intoduction to Algorithms* Second Edition, McGraw Hill/MIT Press, 2001.

[2] Michael T. Goodrich, Romerto Tamassia. *Data Structures Algorithms in Java* Fourth Edition, John Wiley Sons, Inc. 2006.

[3] Lars Backstrom, Paolo Boldi, Marco Rosa, Johan Ugander, Sebastiano Vigna. *Four Degrees of Separation.* Websci 2012, Evanston Illinois, USA. http://law.dsi.unimi.it

[4] http://it.wikipedia.org/wiki/Algoritmo$_di_Bellman-Ford$

[5] Pierluigi Crescenzi, Giorgio Gambosi e Roberto Grossi. *Strutture di dati e algoritmi. Progettazione, analisi e visualizzazione.* Pearson, January 2006.

[6] Paolo Boldi, Marco Rosa, Sebastiano Vigna, Massimo Santini. *Layered Label Propagation: A MultiResolution Coordinate-Free Ordering for Compressing Social Networks.* 2011, Hyderabad, India.

[7] Paolo Boldi, Marco Rosa, Sebastiano Vigna. *HyperANF: Approximating the Neighbourhood Function of Very Large Graphs on a Budget.* January, 2011.

[8] http://it.wikipedia.org/wiki/Teorema$_di_Perron-Frobenius$